FACULTY OF ENGINEERING
AND ARCHITECTURE

Model Reduction in Deep Active Inference

Samuel Wauthier

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors

Prof. Bart Dhoedt, PhD* - Tim Verbelen, PhD** - Bram Vanhecke, PhD***

* Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

** VERSES, the Netherlands

*** Fakultät für Physik, Universität Wien, Austria

March 2025

GHENT
UNIVERSITY

ISBN 978-94-6355-962-1
NUR 984
Wettelijk depot: D/2025/10.500/22

Members of the Examination Board

Chair

Prof. Hennie De Schepper, PhD, Ghent University

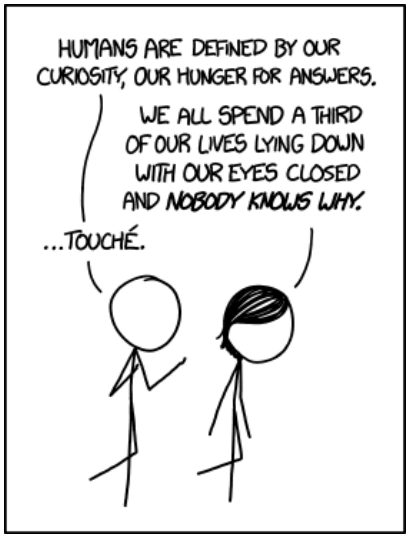
Other members entitled to vote

Cedric De Boom, PhD, Data Minded
Prof. Jutho Haegeman, PhD, Ghent University
Tom Lefebvre, PhD, Ghent University
Prof. Pieter Simoens, PhD, Ghent University

Supervisors

Prof. Bart Dhoedt, PhD, Ghent University
Tim Verbelen, PhD, VERSES, the Netherlands
Bram Vanhecke, PhD, Universität Wien, Austria

Answers



Stanford sleep researcher William Dement said that after 50 years of studying sleep, the only really solid explanation he knows for why we do it is 'because we get sleepy'.

xkcd.com #1345

Preface

"This thesis is the denouement of years of hard work." Many years ago, I used these words in the preface of my Master's thesis. I wrote them thinking that academic degree would be the last I would ever obtain. The resulting preface had a rather *end-of-an-era* feel to it. I, probably like many university students, assumed life would become more mundane from that point on. I was wrong. Life has a way of keeping things interesting, if you let it.

This thesis was made possible through the help and support of many people. I like to think of it as a great collaboration. In the words of the great Isaac Newton: "If I have seen further, it is by standing on the shoulders of giants". We often forget how much our achievements are enabled by others and forget to be grateful in the process. This achievement is held up by four great pillars, four groups of people, that deserve acknowledgment.

The first pillar consists of probably the most important contributors to this dissertation: my promoters. It speaks for itself when I say that these words would never have been written without them. Bart and Tim, thank you for granting me this incredible opportunity and allowing me to minimize my free energy. Bram, thank you for revitalizing my research and expanding my horizons.

The second pillar is my family. María and Avi, there are no words to express how much you've done for me, and how much you've had to put up with me. Thank you for sticking with me through thick and thin. Mom and dad, thank you for always being there whenever I needed you. You were never more than a phone call away. Yasmine, you are awesome. Thank you for always being so caring and a special thank you for all the thesis covers you've made. Parents- and sisters-in-law, muchas gracias por toda la ayuda y por habernos recibido siempre con tanta calidez. Aunties, uncles, cousins, just your names would already take up an entire page, but know that I'm grateful for all the help I've received, all the get-togethers, and all your words of encouragement.

The third pillar consists of all my friends and colleagues who've provided me with inspiration and much needed distraction over the years.

Crucially, I want to thank all my (ex-)colleagues whose journeys lined up with mine at some point during these five years. Cedric, you guided my

research through the harshest of COVID-19. Sam, thanks to you, I never had to be alone in the office. Jeroen, I will never forget how you repaid me threefold for those Dinosaur cookies. Pieter, we[1] destroyed those Italians in foosball. Ozan, going for walks and venting to you during COVID helped keep me motivated. Toon, our gym sessions were mostly filled by talking and a bit of exercise. Pietro, going to conferences with you was a blast. Stefano, we spent countless hours chatting about life and research over coffee. Daria, I enjoyed sending you pictures of cake when you missed another cake day. To the colleagues from the office next door: Wei-Cheng, Mattijs, Thorsten, Sander, Alvaro, Amrapali, Casper, Ciem, Vishisht; the last 2.5 years would never have been so enjoyable without you. Thank you all for the welcoming atmosphere, the endless coffee breaks and the precious memories. I will miss you.

I also want to thank all my friends that have inspired and supported me, even before I started this PhD. Dieter, you are a cornerstone of my life. Our friendly rivalry as teenagers shaped a large part of my life. My group of friends from university with an unfortunate name: Alexis, Ami, Bob, Bram, Jeroen, Simon, Sofie, and Vincent[2]; thanks for all the laughs and interesting discussions. You guys always push me to greater heights. My friends from school: Camiel, Connie, Dimi, Esmée, Joos, and Manu; thank you for always believing in me, even when I failed in the past. Knowing you think so highly of me has always motivated me.

To all the people whose names have not appeared, but have supported me over the years: I appreciate you too!

The fourth and final pillar is singular value decomposition (SVD). Honestly, this method carries three quarters of this dissertation. I consider it an essential tool in any scientist's toolbox. It has been my trusty sidekick since the start of my PhD. Thanks, SVD!

To everyone, cheers and happy reading!

Gent, February 26, 2025
Samuel T. Wauthier

[1] I guess I helped a little bit?
[2] In alphabetical order to not cause jealousy.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARE	Action Respecting Embedding
ARM	Augment-REINFORCE-Merge
BMR	Bayesian Model Reduction
CNN	Convolutional Neural Network
DMRG	Density Matrix Renormalization Group
ELBO	Evidence Lower BOund
EPS	Entangled Plaquette State
FMDP	Finite Markov Decision Process
GECO	Generalized ELBO with Constrained Optimization
GMRES	Generalized Minimal RESidual
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
iDMRG	infinite Density Matrix Renormalization Group
ISMETA	I'm So Meta, Even This Acronym
KL Divergence	Kullback–Leibler Divergence
LDA	Linear Discriminant Analysis
LLE	Locally Linear Embedding
LSTM	Long Short-Term Memory
MDP	Markov Decision Process

MERA	Multi-scale Entanglement Renormalization Ansatz
ML	Machine Learning
MPO	Matrix Product Operator
MPS	Matrix Product State
MSE	Mean Squared Error
NLL	Negative Log-Likelihood
NMF	Negative Matrix Factorization
PCA	Principal Component Analysis
PEPS	Projected Entangled Pair State
POMDP	Partially Observable Markov Decision Process
QLASS	Q-Learning with Adaptive State Segmentation
RGB	Red Green Blue
RGM	Renormalizing Generative Model
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SBS	String-Bond State
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
TDVP	Time-Dependent Variational Principle
TN	Tensor Network
TT	Tensor Train
TTN	Tree Tensor Network
uMPS	uniform Matrix Product State
VAE	Variational Auto-Encoder
VUMPS	Variational Uniform Matrix Product State

Samenvatting
– Summary in Dutch –

Artificiële intelligentie (AI) heeft zich sinds haar ontstaan spectaculair ontwikkeld. De continue groei in de grootte van AI-modellen en de toenemende eisen op het gebied van rekenkracht hebben echter vragen doen rijzen over de schaalbaarheid en duurzaamheid van AI-modellen, met name bij het streven naar artificiële algemene intelligentie (AGI).

Dit proefschrift onderzoekt de uitdagingen die gepaard gaan met modelcomplexiteit en stelt oplossingen inzake modelreductie voor binnen het kader van diepe actieve gevolgtrekking, een benadering die inzichten uit neurowetenschappen en AI integreert om intelligente agenten te creëren die in staat zijn tot adaptief gedrag door van hun omgeving te leren.

Actieve gevolgtrekking verwijst naar het proces waarbij levende organismen hun verrassingswaarde (of variationele vrije energie) minimaliseren om homeostase te behouden en effectief met hun omgeving te interageren. In AI is actieve gevolgtrekking uitgebreid naar diepe actieve gevolgtrekking, waarbij neurale netwerken worden gebruikt om de omgeving te modelleren. Het onderzoek in dit proefschrift richt zich op het opschalen van modellen voor diepe actieve gevolgtrekking en verkent methoden om hun complexiteit te verminderen en tegelijkertijd minimaal in te boeten op prestaties.

Diepe actieve gevolgtrekking maakt gebruik van neurale netwerken om generatieve modellen te leren die toekomstige toestanden en observaties van een agent voorspellen op basis van eerdere acties en observaties. Deze modellen worden steeds complexer naarmate ze grotere hoeveelheden gegevens verwerken en hun interne toestandsruimten aanpassen aan de omgeving. Maar, naarmate de dimensie van deze toestandsruimten toeneemt, nemen ook de reken- en geheugeneisen toe, wat deze aanpak ongeschikt maakt voor toepassingen in ware tijd, zoals robotica, waar rekenkracht vaak beperkt is.

Het belangrijkste doel van dit onderzoek is het ontwikkelen van technieken die de complexiteit van deze modellen verminderen. Dit omvat het vinden van de kleinst mogelijke dimensie van het model die nog steeds nauwkeurig toestanden en observaties kan voorspellen, waardoor de rekenlast wordt verminderd terwijl de effectiviteit van de AI-agent behouden blijft.

In dit proefschrift speelt het concept van slaap een centrale rol in de benadering van modelreductie. Slaap verwijst in deze context naar het biologische proces van synaptisch snoeien, dat wil zeggen het vermogen van de hersenen

om overbodige neurale verbindingen te elimineren, waardoor hun efficiëntie wordt geoptimaliseerd terwijl essentiële functionaliteit behouden blijft. Door biologische slaap na te bootsen, kan de AI-agent complexiteit verminderen en tegelijkertijd minimaal inboeten op prestaties. De slaapmethode is vooral nuttig in robottoepassingen, waar agenten vaak een beperkt geheugen en verwerkingskracht hebben.

Een van de methoden die in dit onderzoek is ontwikkeld is een offline reductietechniek voor latente ruimten. Dit houdt in dat een diepe actieve gevolgtrekking-model wordt getraind met een groot aantal latente ruimtedimensies, waarna singulierewaardenontbinding (SVD) wordt toegepast om overbodige dimensies te identificeren en te verwijderen. De methode werkt door het evalueren van het belang van elke dimensie in de latente ruimte via de singuliere waarden die worden verkregen uit SVD. Dimensies met kleine singuliere waarden worden als minder informatief beschouwd en worden gesnoeid. Na dit snoeiproces wordt het model opnieuw getraind met het gereduceerde aantal latente ruimtedimensies. De offline methode vereenvoudigt de modellen na het trainen, waardoor ze efficiënter worden.

De online reductiemethode voor latente ruimten pakt de beperkingen van de offline aanpak aan door de modelcomplexiteit dynamisch te verminderen tijdens de training. In plaats van dimensies te snoeien na de voltooiing van de training, maakt de online methode gebruik van een gatingmechanisme op basis van L_0-regularisatie om het aantal latente ruimtedimensies dynamisch aan te passen naarmate de agent leert. Hierdoor kan het model zijn complexiteit aanpassen aan de gegevens die het tegenkomt, wat de prestaties in realtime optimaliseert.

Hoewel de online methode sneller en flexibeler is dan de offline methode, heeft het zijn nadelen. Het kan namelijk leiden tot een iets lagere nauwkeurigheid in vergelijking met de offline methode die door grondiger snoeien betere prestaties behaalt. Desondanks is de online-aanpak geschikt voor toepassingen waarbij de trainingstijd een cruciale factor is en waar de agent zich snel moet aanpassen aan veranderende omgevingen.

Naast offline en online reductie van latente ruimten onderzoekt het proefschrift het gebruik van tensor netwerken (TN) als alternatief voor traditionele neurale netwerken voor diepe actieve gevolgtrekking. Tensor netwerken bieden een manier om het aantal parameters in een model dynamisch aan te passen op basis van de complexiteit van de omgeving, wat vooral voordelig is in omgevingen met veel dimensies als geheugen en rekenmiddelen beperkt zijn.

Het gebruik van matrixproducttoestanden (MPS), een soort TN dat toegespitst is op eendimensionale systemen, maakt flexibele schaalvergroting van modelparameters mogelijk. Deze aanpak zorgt ervoor dat het model voldoende capaciteit behoudt om te leren en zich aan te passen, terwijl het tegelijkertijd rekenkundig efficiënt is. Het onderzoek toont aan dat op MPS-gebaseerde agenten optimaal kunnen handelen in verschillende omgevingen, zoals de T-maze en frozen lake.

De verdere uitbreiding naar uniforme matrixproducttoestanden (uMPS) verbetert het modelleringsvermogen van MPSen voor omgevingen met een oneindige tijdschaal en behoudt het mechanisme voor schaalvergroting. Bovendien brengt ze een interessante eigenschap met zich mee. Namelijk, ze breidt ook het voorspellingsvermogen van het generatieve model uit naar potentieel oneindige tijdschaal.

Het onderzoek dat in dit proefschrift wordt gepresenteerd, levert verschillende belangrijke bijdragen aan het vakgebied van diepe actieve gevolgtrekking en AI. Het introduceert nieuwe methoden voor modelreductie, zowel offline als online, waardoor intelligente agenten efficiënt kunnen presteren in omgevingen met beperkte rekenmiddelen. De integratie van tensor netwerken opent nieuwe mogelijkheden voor het toepassen van actieve gevolgtrekking in omgevingen met veel dimensies.

Vooruitblikkend stelt het proefschrift verschillende evoluties voor toekomstig onderzoek voor. Hieronder valt modeluitbreiding, waarbij een gereduceerd model opnieuw wordt uitgebreid wanneer de agent in contact komt met nieuwe informatie die moet worden geïntegreerd. Daarnaast brengt de TN-formulering van actieve gevolgtrekking een heel nieuwe waaier aan onderzoeksvragen met zich mee.

Kortom, dit proefschrift helpt de techniek in diepe actieve gevolgtrekking vooruit door innovatieve technieken voor modelreductie te presenteren die complexiteit en prestaties in balans houden. Door de grootte van latente toestandsruimten en modellen in het algemeen te reduceren, maken de voorgestelde methoden efficiëntere en schaalbaardere AI-modellen mogelijk die kunnen worden ingezet in realtime, middelenbeperkte omgevingen. Het werk legt een solide basis voor toekomstig onderzoek naar het creëren van adaptieve, intelligente systemen die niet alleen krachtig, maar ook praktisch zijn voor toepassingen in de echte wereld.

Summary

Artificial Intelligence (AI) has evolved dramatically since its inception. However, the continuous growth in AI model sizes and computational demands has raised concerns regarding the scalability and sustainability of AI models, particularly in achieving Artificial General Intelligence (AGI). This dissertation investigates the challenges of model complexity and proposes solutions for model reduction within the framework of deep active inference, an approach that integrates neuroscience and AI to create intelligent agents capable of adaptive behavior through learning from their environments.

Active inference refers to the process by which living organisms minimize surprisal (or variational free energy) to maintain homeostasis and effectively interact with their surroundings. In the field of AI, active inference has been extended to deep active inference, where neural networks are employed to model the beliefs that agents form about their environment. The research presented in this dissertation addresses the issue of scaling deep active inference models and explores methods for reducing their complexity while minimally compromising performance.

Deep active inference relies on neural networks to learn generative models that predict the future states and observations of an agent based on past actions and observations. These models grow increasingly complex as they process large amounts of data and adjust their internal state spaces to reflect the environment. However, as the dimensionality of these state spaces grows, so do the computational and memory requirements, making it infeasible for use in real-time applications such as robotics, where resources are often limited.

The primary focus of this research is to develop techniques aimed at reducing the complexity of these models. This involves identifying the smallest possible dimensionality of the model that still allows it to accurately predict states and observations within its environment, thereby reducing the computational burden while maintaining the effectiveness of the AI agent.

The concept of sleep plays a central role in the dissertation's approach to model reduction. Sleep, in this context, refers to the biological process of synaptic pruning, i.e., the brain's ability to eliminate redundant neural connections, optimizing its efficiency while retaining essential functionality. By mimicking biological sleep, the AI agent can reduce complexity with minimal sacrifices to performance. The sleep method is particularly useful for robotic applications, where agents often have limited resources.

One of the methods developed in this research is an offline latent space reduction technique. It involves training a deep active inference model with a high number of latent space dimensions and then applying singular value decomposition (SVD) to identify and remove redundant dimensions. The method works by scoring the importance of each dimension in the latent space through the singular values obtained from SVD. Dimensions with small singular values are deemed less informative and are pruned. After this pruning process, the model is retrained with the reduced number of latent space dimensions. The offline method simplifies the models post-training, making them more efficient after the learning process.

Another approach, an online latent space reduction method, addresses the limitations of the offline approach by reducing model complexity dynamically throughout the training process. Rather than pruning dimensions after training is complete, the online method uses a gating mechanism based on L_0 regularization to dynamically adjust the number of latent space dimensions as the agent learns. This allows the model to adapt its complexity according to the data it encounters, thereby optimizing performance in real-time.

While the online method is faster and more adaptable than the offline method, it comes with certain trade-offs. Specifically, it may result in slightly lower accuracy compared to the offline method, which achieves better performance. Despite this, the online approach is well-suited for applications where training time is a critical factor and where the agent needs to adjust to changing environments on the fly.

In addition to offline and online latent space reduction, the dissertation delves into the use of tensor networks (TNs) as an alternative to traditional neural networks for deep active inference. Tensor networks provide a way to dynamically adjust the number of parameters in a model based on the complexity of the environment. This adaptability makes them particularly advantageous in high-dimensional environments when memory and computational resources are constrained.

The choice of matrix product states (MPS), a type of TN specialized for one-dimensional systems, allows for flexible scaling of model parameters. This approach ensures that the model retains sufficient capacity to learn and adapt, while also being computationally efficient. The research demonstrates that MPS-based agents are capable of acting optimally in various environments, such as the T-maze and frozen lake.

Further extension to uniform matrix product states (uMPS) enhances the modeling capability of MPS for environments with an infinite time horizon. This approach preserves the parameter scaling mechanism that is crucial for maintaining computational efficiency as models grow in complexity. Moreover, it introduces an interesting property: it also extends the predictive capability of the generative model, allowing it to make predictions over potentially infinite time horizons.

The research presented in this dissertation offers several key contributions to the field of deep active inference and AI. It introduces novel methods for

reducing model complexity both offline and online, ensuring that intelligent agents can perform efficiently in environments with limited computational resources. Moreover, the integration of tensor networks opens new avenues for applying active inference in high-dimensional environments.

Looking ahead, the dissertation proposes several directions for future research. Firstly, there is model expansion, where a reduced model is once again expanded when the agent comes into contact with new information that must be integrated. Secondly, there is the novel TN formulation of active inference which presents many new research questions.

In conclusion, this dissertation advances the state of the art in deep active inference by presenting innovative techniques for model reduction that balance complexity and performance. By reducing the size of latent state spaces and models in general, the proposed methods allow for more efficient and scalable AI models that can be deployed in real-time, resource-limited environments. The work lays a solid foundation for future research in creating adaptive, intelligent systems that are not only powerful but also practical for real-world applications.

— I'm tired.

Everyone

1 Introduction

Artificial intelligence has been around for much longer than many people imagine. The beginning of this chapter provides a short history on this topic. This also serves to place the research presented in this dissertation in a broader context and highlights the importance of reducing the size of models in the current AI landscape.

Another purpose of this chapter is to familiarize the reader with active inference and what it entails. More importantly, it explains how model reduction presents itself in living organisms.

Ultimately, I present the goals and contributions of this dissertation, as a roadmap for the remainder of this book. This includes an overview of all the work that was published during the course of the research, highlighting key milestones and findings that have shaped the trajectory of my PhD.

1.1 Artificial (general) intelligence

Even before the term "artificial intelligence" (AI) was coined by John McCarthy in 1956 [38], humans had been imagining and dreaming of artificially created life. People told stories of Talos in Greek antiquity, humanoid automatons in early Chinese lore, golems in Jewish folklore [39], and brazen heads in Christian legend [22]. After the establishment of the field of modern AI in the 1950s, researchers began making predictions about when we would achieve an AI that matches human-level intelligence across a multitude of cognitive tasks. In 1965, Herbert A. Simon predicted that "machines will be capable, within twenty years, of doing any work a man can do" [74], while in 1967, Marvin Minksy said that "within a generation... the problem of creating 'artificial intelligence' will substantially be solved" [12]. Over the course of the next 40 years, vain promises made by AI researchers lead to a collapse in the 1970s, a revival in the 1980s, and a second collapse in the late 1980s in public confidence regarding human-level AI, and AI in general [11]. These so-called AI winters came with severe funding cuts and lead to a shift in focus on "narrow AI", AI that is limited to solving a single task [69].

Around the turn of the century, the term "artificial general intelligence" (AGI) came into use [26, 23], a new name for AI able to match humans on multiple cognitive tasks, as interest in human-level AI gained traction once again [80]. Since then, research into AGI has picked up again, and AGI is now a subject of considerable debate [1]. Prominent researchers are attempting to predict a critical time point for AGI once more. Geoffrey Hinton has stated that it might take less than 30 years [43], while NVIDIA's CEO, Jensen Huang, has said that AI would be able to pass any test as well as humans within five years [49]. Now, many leading figures have expressed concerns regarding a potential existential risk that comes with artificial superintelligence [28, 67, 8], referring to the notion that such an intelligence could grow out of control and/or misalign with human goals and ethics, and endanger humanity.

Regardless of the opinions of public figures, it is a fact that AI has achieved impressive feats in the last decade. Landmark achievements have lead to AI gaining tremendous popularity and making world news several times these past few years. Some of these were: AlexNet winning the ImageNet Large Scale Visual Recognition Challenge in 2012 [69], every time Deepmind's AlphaGo defeated a champion in the board game "Go" [27, 53, 42, 73], the rise of generative pre-trained transformers (GPT) [65] and ChatGPT [56], and developments in generative AI in general [13, 41]. In fact, the growth of AI has been so steady that multiple high-profile individuals, Stephen Hawking and Elon Musk among others, signed an open letter calling attention to the

societal impacts of AI already in 2015 [19].

Despite all the hype, researchers are still unable to agree on what constitutes AGI [48]. The most well-known test of intelligence is the Turing test, famously introduced by Alan Turing in 1950 [77]. In this test, a human judge would have to evaluate written natural language conversations between a human and a computer. If the judge was unable to tell apart the human and the computer, the computer would pass the test. Since then, many tests have been imagined and developed as a response to the weaknesses of preceding tests [24, 50, 37, 44].

Evidently, this has not stopped scientists from attempting to build AI applicable to a wide range of cognitive tasks [47]. Symbolic AI is historically one of the first approaches to AGI [20]. This family of AI is based on symbolic representation of knowledge and reasoning. Another common approach is connectionist AI [5], which uses artificial neural networks to model human cognition. An emerging approach is that of embodied AI [57]. Embodied AI is based on the concept of embodied cognition [71], which postulates that many aspects of an organism's cognition are shaped by the exchange of sensory observations and interactions with the environment.

Finally, beyond the risks and ethical concerns, recent developments in AI have revealed another trend: AI models have been growing sharply in number of parameters (Fig. 1.1) and exponentially in training compute (Fig. 1.2) [3]. This trend presents several troubling issues. For one, current notable AI models require large amounts of memory. For instance, GPT-4 is estimated to run on 1.8 trillion parameters [70]. Assuming 32-bit precision, this corresponds to $\sim$7 TB of memory. For comparison, the Google Pixel 9, hailed as "the smartest smartphone" — the most AI-integrated smartphone, offers storage capacities of 128 GB or 256 GB [25]. For another, current notable AI models require large amounts of power. GPT-4 is estimated to have been trained on 10 000 to 25 000 A100 graphics processing units (GPUs) for three months [52] and runs on clusters of 128 A100 GPUs [70]. Given that a single A100 GPU consumes up to 400 W [51], the energy demands are immense. OpenAI's CEO, Sam Altman, has disclosed that the cost of training GPT-4 exceeded 100 million USD [32, 3].

It is evident that models of this scale are only practical for deployment in data centers, as the memory and power requirements far exceed what is available for mobile or consumer-grade devices. These financial and resource constraints suggest that only large corporations can afford to operate models of this magnitude, further consolidating power within a small group of tech giants [3]. Furthermore, these challenges do not account for the environmental impacts associated with the energy consumption and resource usage of such large-scale AI models [3].

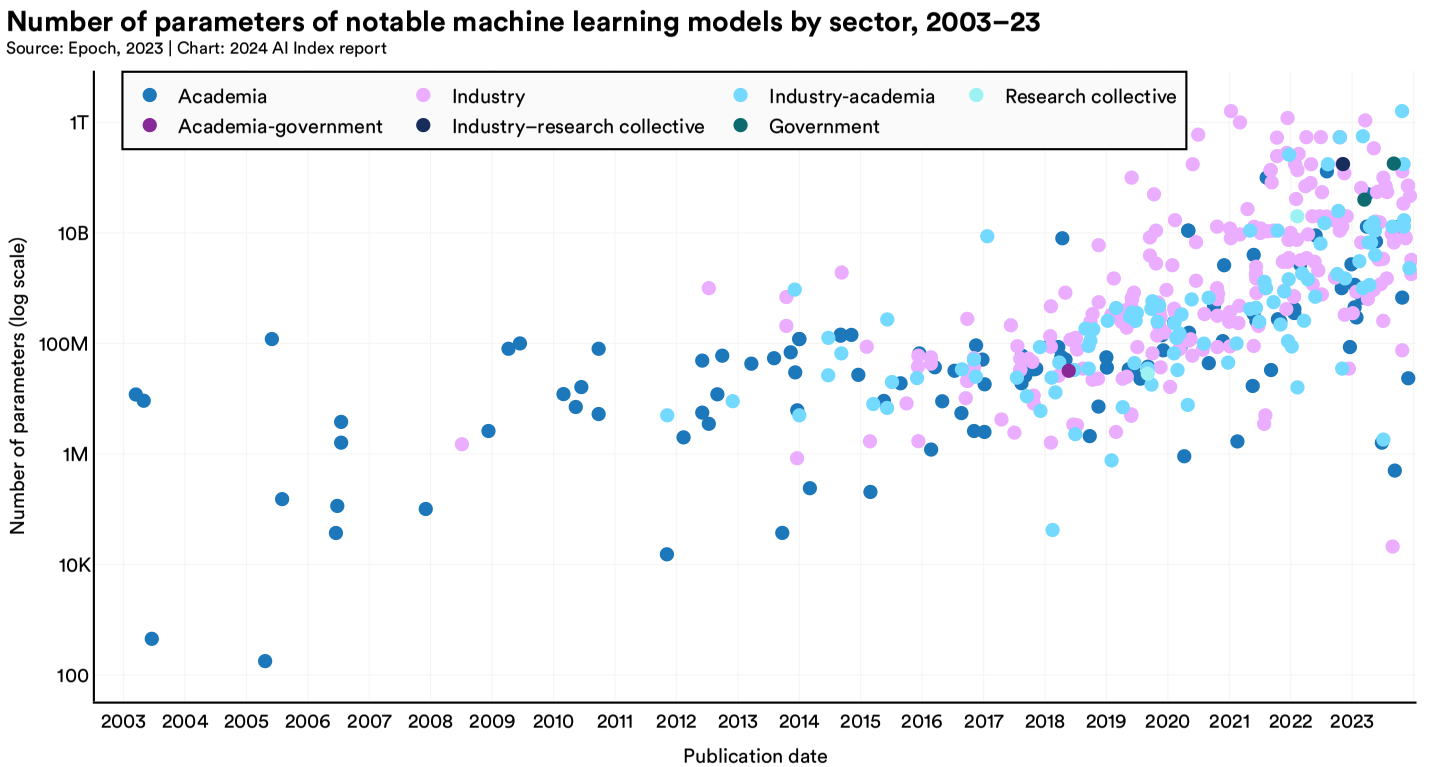


Figure 1.1: Evolution of number of model parameters in the last 20 years per sector. [3]

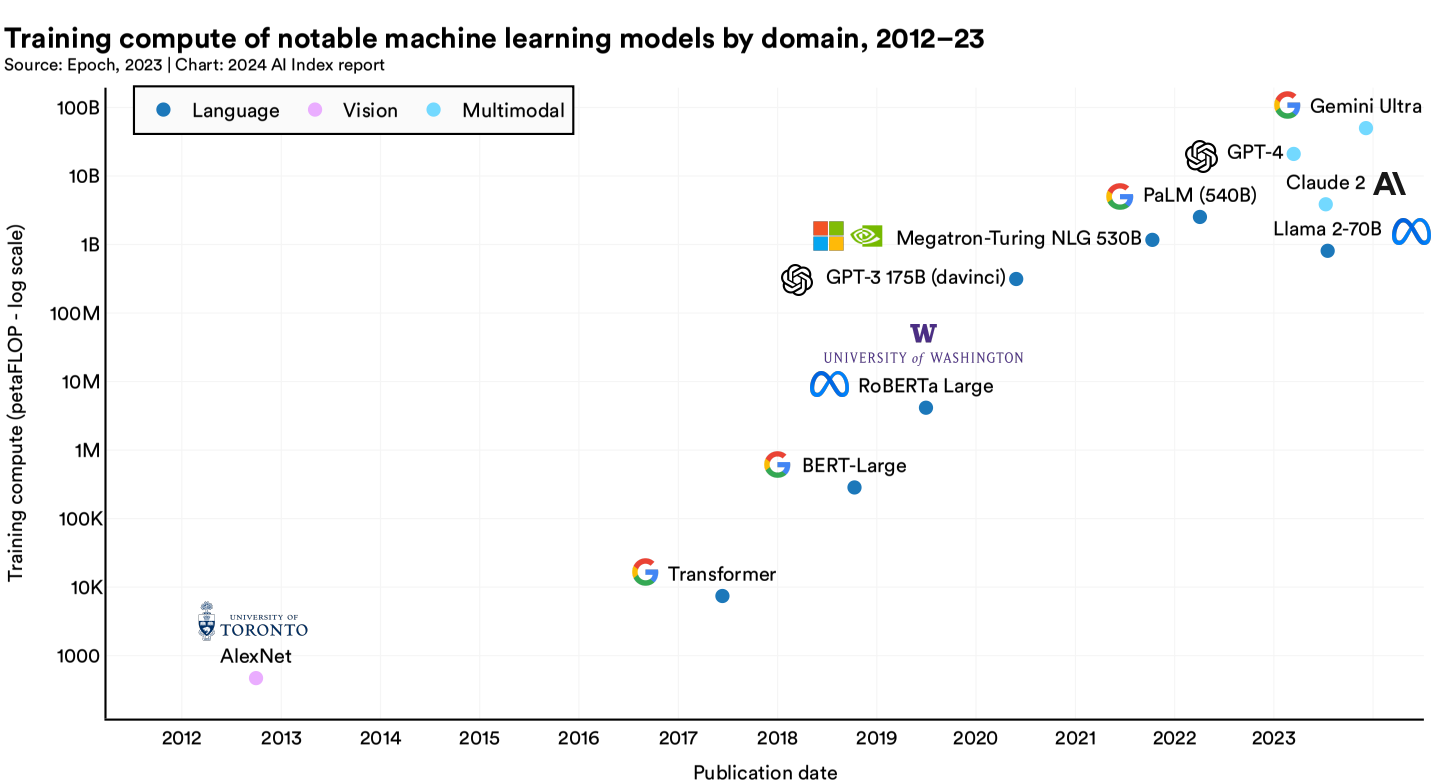


Figure 1.2: Evolution of training compute in the last 10 years. [3]

If the aforementioned trend continues, the number of parameters could increase substantially by the time artificial general intelligence (AGI) is realized. Embedding AGI into robotic systems or autonomous vehicles would require either substantial onboard computational hardware or offloading models to the cloud, which introduces latency concerns. Clearly, to achieve AGI, current approaches are not sustainable. The path forward will require advancements in model efficiency and a reduction in model size.

1.2 Active inference

A theory that has recently been gaining popularity in the field of AI is that of active inference. First and foremost, active inference is a theory of behavior and learning in neuroscience [15]. It strives to answer the question: "How do living organisms persist while engaging in adaptive exchanges with their environment?" [60] To this end, active inference posits a continual action-perception loop between a living system and its environment (Fig. 1.3) [60], i.e., the agent learns by interacting with the environment, while the environment provides the agent with sensory stimuli. Due to this, it can also function as an embodied approach to AGI.

While later chapters present active inference in a technical context, this chapter provides a more biologically focused explanation. It explores how active inference manifests in living systems, offering a more intuitive and biologically driven explanation before transitioning to the more formalized discussions that follow. It must be stated that active inference has proven to be a notably difficult subject and often requires a while to understand. For the interested reader, multiple attempts have been made in the literature at a coherent description [15, 61].

In order to stay alive, living systems maintain homeostasis [6], i.e., they maintain a careful balance between all their physical and chemical conditions. Despite being immersed in an environment which is constantly pushing the conditions out of balance, homeostasis can be maintained through physiological negative feedback mechanisms [6]. For example, while the outdoor temperature changes, the human body regulates its temperature by sweating (to lose heat) or by increasing its metabolic rate (to increase heat production). Further, living systems can help maintain homeostasis by acting on the environment. Humans can take action in order to better regulate their temperature by moving to colder areas on the planet, or by putting on a jumper. What's more, humans plan ahead in an attempt to make temperature changes more predictable and less surprising by predicting the weather and building houses.

Active inference frames the ability to persist within an interactive envi-

Figure 1.3: Interaction between a living system and its environment illustrating the action-perception loop. [Generated with DeepAI.]

ronment as the minimization of surprisal. Crudely said, organisms adapt by trying to avoid surprising states, e.g., such as having too high or low a body temperature. To that end, living systems have a preferred state (which may change over time) [60], a state of being which is considered optimal, e.g., a set point in physiology, such as a body temperature of around 37 °C [6]. States may be less or more surprising depending on how much they deviate from the preferred state [60].

In order to adapt to the environment, an active inference agent receives sensory observations. For example, humans can see, hear, feel, etc. the world around them. Importantly, agents receive imperfect information from the environment. Organisms cannot observe the whole world at once. This is true on a large scale: a person in Belgium cannot see everything that is going on in Spain; but also on a smaller scale: you cannot see through the wall in front of you[1]. Thus, active inference agents are equipped with an internal representation of the world, a so-called generative model [59]. The generative model reflects the agent's beliefs about itself and the state of the world, as well as how the world works, i.e., the dynamics [59]. The agent forms and updates its beliefs based on the observations it receives from the environment [59]. For example, imagine a person believing it is sunny and stepping outside to find it raining. This person would be surprised and update their belief about the weather from sunny to rainy.

The agent's internal model can be described mathematically through probability theory. Here, the agent's beliefs about the world are encoded

[1] Unless there is a window in it.

as probability distributions. The agent holds beliefs about the state of the world s, the things it can observe o, and the actions it can take a. It may do so for multiple moments in time $t = 1, \ldots, T$. In summary, the agent holds beliefs about states $s_{0:T} = \{s_0, s_1, \ldots, s_T\}$, observations $o_{1:T} = \{o_1, o_2, \ldots, o_T\}$, and actions $a_{1:T} = \{a_1, a_2, \ldots, a_T\}$, in the form of probabilities $P(o_{1:T}, s_{0:T}, a_{1:T})$.

The agent's environment is typically modeled as a Markov chain. This means that the next state of the environment only depends on the current state. Further, this allows us to factorize

$$P(o_{1:T}, s_{0:T}, a_{1:T}) = \underbrace{P(a_{1:T})}_{\text{policy}} \underbrace{P(s_0)}_{\substack{\text{initial} \\ \text{state}}} \prod_{t=1}^{T} \underbrace{P(s_t|s_{t-1}, a_t)}_{\text{transition}} \underbrace{P(o_t|s_t)}_{\text{likelihood}} . \tag{1.1}$$

In this mathematical language, the minimization of surprisal equates to the minimization of the negative logarithm of the evidence [59]:

$$-\log P(o_{1:T}) . \tag{1.2}$$

This minimization generally proceeds through belief updates by means of Bayes' rule:

$$P(s_{0:T}, a_{1:T}|o_{1:T}) = \frac{P(o_{1:T}|s_{0:T}, a_{1:T})P(s_{0:T}, a_{1:T})}{P(o_{1:T})} , \tag{1.3}$$

given the prior $P(s_{0:T}, a_{1:T})$ and likelihood $P(o_{1:T}|s_{0:T}, a_{1:T})$ distributions. However, it is usually assumed that computation of the evidence $P(o_{1:T})$ is intractable [59]. Therefore, active inference defines a proxy, called the variational free energy F [59], an upper bound on the surprisal,

$$\begin{aligned} -\log P(o_{1:T}) &\leq \mathrm{E}_{Q(s_{0:T}, a_{1:T})} \left[\log Q(s_{0:T}, a_{1:T}) - \log P(o_{1:T}, s_{0:T}, a_{1:T})\right] \\ &\coloneqq F[Q, o_{1:T}] , \end{aligned} \tag{1.4}$$

by approximating the posterior $P(s_{0:T}, a_{1:T}|o_{1:T})$ with a parameterized posterior $Q(s_{0:T}, a_{1:T})$. Since it is an upper bound, minimizing the variational free energy also minimizes the surprisal, and this minimization is performed by finding the best parameterization of Q.

At the same time, the generative model allows the agent to form beliefs about the future [59]. It encodes beliefs about the consequences of the agent's actions, and as such, allows planning ahead [58]. For instance, if the person in the previous example had been uncertain about the weather, they might have anticipated the rain and brought a raincoat, or they might have tried to resolve their uncertainty by checking the forecast. Additionally, depending on how sophisticated the generative model is, it allows for complex

behavior [58]. For example, if the agent is able to plan far enough ahead, it may allow for short periods of surprisal to ensure subsequent longer periods of unsurprisal.

Mathematically, an active inference agent plans its actions by computing the expected free energy of the actions that bring it into its preferred state of being. In other words, the agent scores the actions it can perform by how much variational free energy it expects each action will cost when attempting to reach the preferred state. The preferred state[2] can be expressed in terms of a distribution of preferred observations $P^*(o)$, which describes the sensory observations that the agent would experience if it were in its preferred state of being. The expected free energy from the current time τ until some time T in the future is given by [46]

$$G = \mathrm{E}_{Q(s_{\tau:T}, a_{\tau:T}) P(o_{\tau:T} | s_{\tau:T}, a_{\tau:T})} \left[Q(s_{\tau:T}, a_{\tau:T}) - P^*(o_{\tau:T}, s_{\tau:T}) \right] , \quad (1.5)$$

where $P^*(o_{\tau:T}, s_{\tau:T}) = P^*(o_{\tau:T}) \sum_{a_{\tau:T}} Q(s_{\tau:T}, a_{\tau:T})$. To facilitate computation, two ways of expanding this expression have been proposed: the standard [15] and the sophisticated [17] schemes. Regardless of which scheme is used, the actions that the agent performs are those that minimize the expected free energy G.

As active inference agents accumulate more information, their generative models may grow increasingly complex, incorporating increasingly detailed representations of their environment. To manage the growing complexity, it is important that living systems employ some way of simplifying the generative model, like the processes involved in memory processing and synaptic strength renormalization during sleep.

1.3 Sleep

Researchers are still unsure whether sleep is universal [72, 10], i.e., whether all animals sleep. The fact of the matter is that the answer to this conundrum depends on the definition of sleep [10, 64, 62]. Although animals do not sleep the same way humans do, nearly all animals exhibit some form of sleep or rest state [10]. The duration, patterns, and depth of sleep vary widely across species [10, 62]. For example, while a brown bat may sleep for about 20 hours a day, a giraffe might only need a few hours. Despite these differences, sleep plays a critical role in maintaining health and functionality in all animals [10].

Sleep serves many purposes in animals [64, 45]. While the main function of sleep is still unknown, researchers have discovered some potential functions

[2] Realistically, the agent does not have a single preferred state, but a range of states that are preferred. Moreover, some states may be more preferred than others.

of sleep in humans [45]. Some of these functions relate to metabolic function, the immune system and neurodevelopment. For example, during sleep, the brain processes and consolidates memories, which helps in learning and cognitive functions [45]. A key function of sleep is synaptic pruning [34], a process where the brain reduces or eliminates weaker neural connections while strengthening more essential ones. Pruning synaptic connections allows the brain to remain efficient by clearing out unnecessary data.

Active inference employs a process similar to synaptic pruning during sleep called Bayesian model reduction [18, 16]. Bayesian model reduction is an evidence-based technique for eliminating redundant parameters from a statistical model [18]. In other words, Bayesian model reduction reduces the complexity of an active inference agent's generative model, while maintaining predictive accuracy.

This method works by comparing the current model to reduced models, where reduction amounts to the use of alternative priors that switch off certain parameters [18]. Mathematically, given the current prior $P(s_{0:T}, a_{1:T})$ and an alternative prior $\tilde{P}(s_{0:T}, a_{1:T})$, the evidence for the reduced model is given by [18]

$$\tilde{P}(o_{1:T}) = P(o_{1:T}) \int \sum_{a_{1:T}} P(s_{0:T}, a_{1:T}|o_{1:T}) \frac{\tilde{P}(s_{0:T}, a_{1:T})}{P(s_{0:T}, a_{1:T})} \mathrm{d}s_{0:T} \,, \tag{1.6}$$

where, in practice, the posterior $P(s_{0:T}, a_{1:T}|o_{1:T})$ will be replaced by the approximate posterior $Q(s_{0:T}, a_{1:T})$. With this, the difference in variational free energy realized by switching to the reduced model can be readily computed through [18]

$$\Delta F = -\log \tilde{P}(o_{1:T}) + \log P(o_{1:T}) \,. \tag{1.7}$$

The key feature of this method is that it does not require computing the alternative posterior $\tilde{Q}(s_{0:T}, a_{1:T})$ for the reduced models. Instead, the reduced models can be obtained directly from the current model. The difference ΔF can be used as a criterion for scoring reduced models, such that the model that yields the largest decrease in variational free energy is kept.

Sleep-like mechanisms and pruning models are important for AGI, and AI in general. AI systems benefit from a reduction in complexity in an important way. Smaller models require less computational resources in terms of both compute and memory. This is especially significant when deploying AI and AGI on robots, where power and memory are limited. Moreover, complexity reduction may lead to better generalizability, allowing models to perform better on unseen data. Integrating sleep-inspired processes into AGI systems could be key to achieving long-term sustainability and adaptability.

1.4 Deep active inference

One of the drawbacks of putting active inference into practice is the requirement to predefine the agent's internal representation of the world [7]. The earlier-mentioned formulation assumes that each internal state is disentangled and well-defined, meaning that each state has a unique interpretation. In that case, the state space, the space spanned by states s, is a vector space where each dimension represents a unique concept.

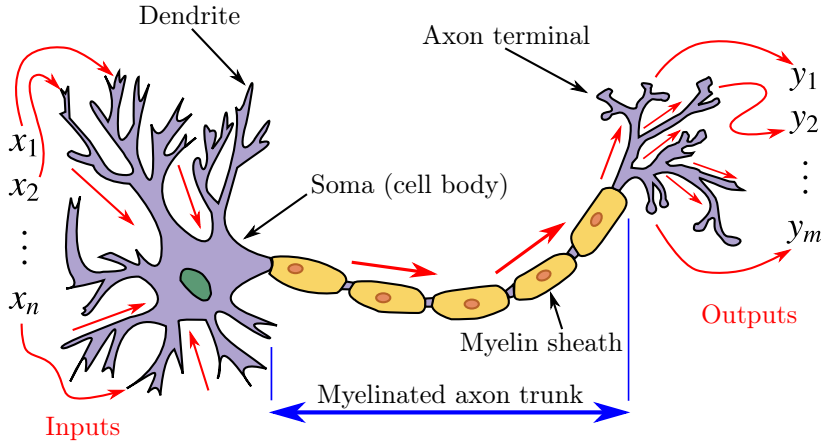
In reality, it is not straightforward to define such a state space. For example, consider a person looking at an animal: the person's internal representation of this animal may not only encode which animal it is looking at, but also which color it is, what it is doing, where it is located, where it is going, etc. For this reason, the number of use cases covered in the literature by active inference is still fairly limited. Many experiments involve toy models and grid worlds and serve more as a proof-of-concept. Moreover, the models used in these experiments are relatively small in terms of parameters, especially when compared to popular AI models nowadays.

A possible solution is to learn the states from data instead. Classically, active inference is simply defined through stochastic matrices, which makes it difficult to learn from observations, although attempts have been made [75]. To simplify learning the state space, the internal model of the agent can be designed using existing paradigms in machine learning.

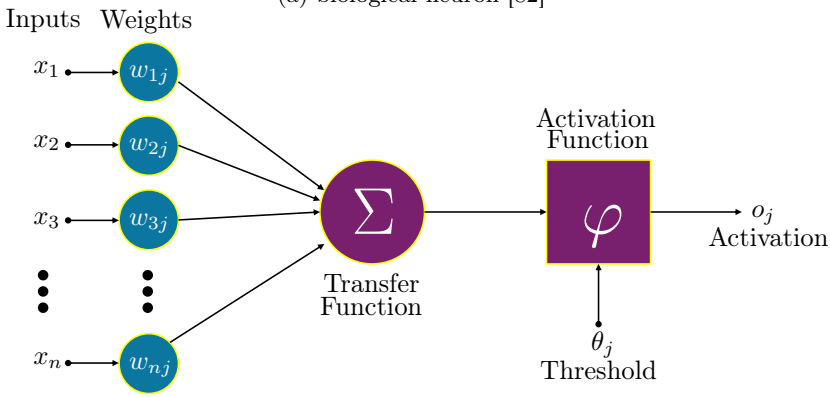
1.4.1 Artificial neural networks

Over the course of the last few years, artificial neural networks (ANNs) have become ubiquitous in AI. The number of types of ANNs has increased rapidly as AI has grown in popularity. Some of the best-known networks include: multilayer perceptrons (MLP) [40, 68], convolutional neural networks (CNNs) [33], autoencoders [4], Hopfield networks [30], long short-term memory (LSTM) [29], and transformers [78]; which are used in a large range of applications, from generative AI to autonomous driving.

ANNs were inspired by the workings of the brain in animals [40, 84]. Artificial neurons can be seen as a very basic model of biological neurons in the brain (see Fig. 1.4). Here, the inputs correspond to the neurotransmitters or electrical signals passing through the synapse, received by the dendrites of the postsynaptic cell. The weights are comparable to synaptic weights. The transfer function operates as the soma in the sense that it aggregates the signals received from the dendrites and propagates it through the axon and to axonal terminals. The activation function acts as the threshold potential and determines whether the neuron fires. Finally, activations, or outputs, correspond to the signal emitted by axonal terminals of the presynaptic



(a) biological neuron [82]



(b) artificial neuron [83]

Figure 1.4: Comparison between a biological and an artificial neuron.

cell. By connecting and arranging artificial neurons in layers it is possible to construct basic ANNs called MLPs (see Fig. 1.5).

ANNs can learn to perform a task by adjusting the weights carried by the artificial neurons. This usually occurs through a process called backpropagation [36, 35]. Loosely speaking, this process assesses the effect of each weight on the error obtained after performing the task. The weights can then be adjusted so that the error decreases the next time the task is performed.

Standard deep active inference involves replacing certain parts of the generative model by ANNs. In Çatal et al. [7], it involves parameterizing the prior transition $P(s_t|s_{t-1}, a_t)$, likelihood $P(o_t|s_t)$ and posterior transition

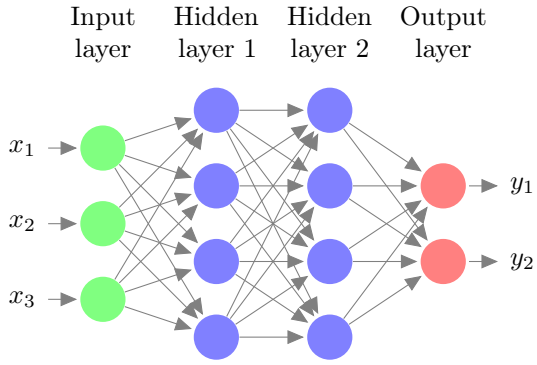


Figure 1.5: Graphical representation of a multilayer perceptron (MLP) with 4 layers: an input layer, two hidden layers, and an output layer.

$P(s_t|s_{t-1}, a_t, o_t)$, e.g., by using MLPs or CNNs, which can then be trained using backpropagation.

Crucially, this requires specifying the number of neurons in the networks before training. It is typically not possible to predict how many neurons (or which network architecture) will be necessary for a certain task. Since adding and removing neurons from ANNs is not straightforward, it is useful to devise methods that enable the shrinking and/or growing of these networks based on how much information must be retained after or during training. Sleep-inspired mechanisms may provide a solution.

1.4.2 Tensor networks

Tensor networks (TNs) are models originally developed for use in quantum many-body physics, the study of systems of many interacting quantum particles [54, 9]. There, they represent the quantum state of the system, a mathematical description of all the information that is known about the system. Since about a decade, TNs have been slowly entering the field of AI [79].

While the literature on TNs in AI is still limited, their use in this field is interesting for several reasons. Firstly, they were developed to deal with the curse of dimensionality in quantum many-body physics, where the degrees of freedom grow exponentially with the number of particles. This is comparable to machine learning settings, where the degrees of freedom grow exponentially with the number of input variables. TNs restrict the degrees of freedom in a tractable way. Secondly, although not much is known yet about TNs in the context of AI, TNs have been around for a relatively long time in physics [14]. Due to this, much is already known about the capacity and

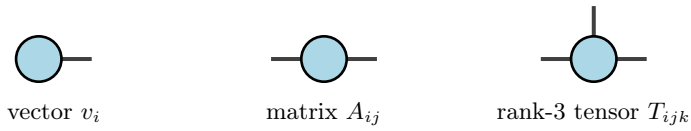


Figure 1.6: Examples of nodes in tensor diagram notation.

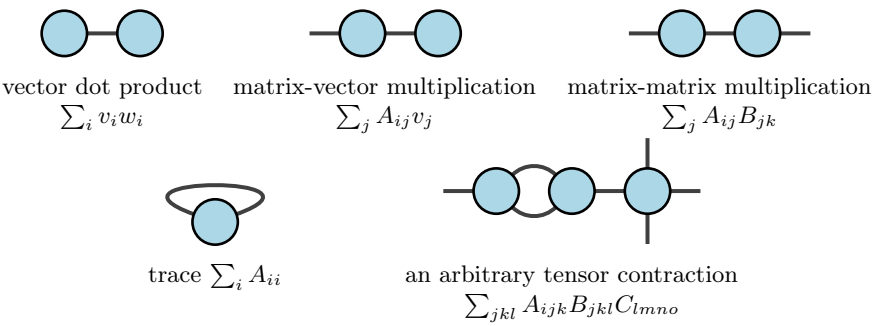


Figure 1.7: Examples of contractions in tensor diagram notation.

expressivity of TNs [21]. Lastly, the interpretability and explainability of TNs can be helpful in the path towards more explainable AI [31, 76, 66, 2].

Like neural networks, tensor networks have a useful graphical notation. As the name suggests, TNs consist of tensors. These can be regarded as a generalization of vectors and matrices. If one thinks of a vector as an object with one index v_i, and a matrix as an object with 2 indices A_{ij}, a tensor is an object with an arbitrary number of indices $T_{ijkl\ldots}$. Furthermore, tensors can be connected through contractions. These, in turn, can be regarded as a generalization of typical matrix operations, such as matrix-vector multiplication $\sum_j A_{ij} v_j$, matrix-matrix multiplication $\sum_j A_{ij} B_{jk}$, and the trace $\sum_i A_{ii}$. That is, a contraction is simply the summation over an index. In graphical notation, tensors are nodes with a leg for each index (see Fig. 1.6), and contractions are edges (see Fig. 1.7). This notation provides a comprehensible overview for tensor network architectures, as evidenced by the last example in Fig. 1.7, where the graphical notation is manifestly easier to read than the mathematical notation.

Arguably the simplest type of TN is the matrix product state (MPS), or tensor train (TT) [14, 63, 55, 9]. In physics, this network is generally used for 1-dimensional strongly correlated quantum systems. For example, an MPS can be used to model the 1-dimensional transverse-field Ising model, a 1-dimensional quantum version of the famous Ising model, which consists

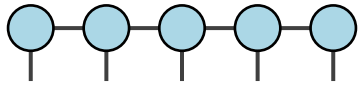


Figure 1.8: A matrix product state (MPS) with five tensors.

of spin-$\frac{1}{2}$ particles[3] sitting on a 1D lattice that can interact with their left and right neighbors. An MPS is a TN with tensors arranged in a chain-like fashion (see Fig. 1.8). The free indices correspond to the physical degrees of freedom of the system and usually represent different sites within the system, e.g, different lattice sites (i.e., different particles) in the Ising model. The contracted indices are virtual degrees of freedom called bonds. The dimension of these bonds generally depends on the correlation between the different sites, e.g., strongly interacting particles will lead to larger bond dimensions.

The MPS architecture is powerful for several reasons. For one, an MPS can model any system as long as the bond dimensions are large enough [54]. Although this may mean that its bond dimensions become exponentially large, under certain conditions, an MPS can be efficiently approximated to almost arbitrary accuracy by an MPS with finite bond dimensions [54]. For another, despite having been developed for quantum systems, MPSs are a class of models that allows efficient computation on classical computers. Indeed, efficient algorithms that leverage the architecture of the network exist for virtually every operation [81], from the calculation of expectation values of observables to truncation of the bond dimensions.

Since quantum states are inherently probabilistic and can be used for probabilistic modeling, MPSs are a potential candidate for the generative model in active inference. This involves modeling the joint probability $P(o_{1:T}, a_{1:T})$ as an MPS. Marginal and conditional probabilities can then be derived from this joint probability. Note that the states $s_{0:T}$ are not included in the joint probability, since the MPS itself adopts the role that the states would fulfill[4]. The learnable parameters in this model are simply the elements of the tensors and can be trained through computational optimization, such as stochastic gradient descent (SGD). Training the MPS entails updating one tensor at a time by sweeping back and forth across the chain.

This type of model offers the advantage of facilitating both the reduction and expansion of the model in a straightforward manner. The bond dimensions serve as a mechanism for efficiently adding or removing model parameters. Increasing the bond dimensions expands the model's capacity, while decreasing them reduces its complexity. As a result, the initial selec-

[3]When observing these particles, they either have spin 'up' or 'down'.
[4]This will become clear in later chapters.

tion of the number of parameters becomes less important, as the model can dynamically adjust to the specific requirements of the given task.

1.5 Research Contributions

The research included in this book started as a way to both demystify active inference and deploy active inference models on robots. The initial goal was to disentangle the latent space of deep active inference agents in order to investigate what is encoded. An encounter with singular value decomposition (SVD) eventually led to a technique for latent space and model reduction. This gave birth to a series of 'we can do better'-moments and grew into what is now a collection of techniques for latent space and model reduction.

This work emerged from existing research in deep active inference, i.e., active inference where the generative model is learned using deep artificial neural networks (ANNs). At the onset of this doctorate, the combination of active inference with deep learning was the primary focus of our research group. While active inference usually requires handcrafted state spaces, deep active inference aims to learn the state space from data using ANNs. The flexibility of ANNs offered a promising approach to building more adaptive and data-driven models.

Since the ultimate goal of deep active inference is to create embodied intelligent agents, the need for these models to eventually be physically embedded in robots became evident. This led to the realization that reducing the size and complexity of the models was essential to minimizing hardware requirements, as well as to promote generalizability. Thus, reducing model size without compromising performance became a central challenge.

Ultimately, this dissertation evolved into a pursuit of optimizing active inference models by finding the best-performing models with the fewest parameters. The culmination of this work offers new pathways for creating more efficient, scalable, and adaptable embodied agents. By focusing on model reduction techniques, this research contributes to the broader goal of enabling intelligent systems that are not only powerful but also efficient enough to be deployed on more accessible and less resource-intensive hardware. The research questions that this dissertation addresses are multifold:

1. How can the complexity of existing deep active inference models be reduced?

2. Can model complexity be reduced using online methods?

3. Can complexity reduction be facilitated by employing a different model?

4. How scalable are the proposed methods?

These and other questions have been answered in the following chapters. Chapter 2 proposes an offline technique for latent space reduction based on singular value decomposition (SVD). Chapter 3 compares the previous technique with an online method based on Generalized ELBO with Constrained Optimization (GECO), L_0 regularization and the Augment-REINFORCE-Merge (ARM) gradient estimator. Chapter 4 presents a novel implementation of the active inference framework based on quantum physics-inspired matrix product states (MPS). This implementation allows for dynamic adaptation of model size during training. Chapter 5 lays the groundwork for an infinite time horizon extension to the MPS-based method, as well as quantum mechanics-like operators for the calculation of the expected free energy. Finally, Chapter 6 draws overarching conclusions about the presented work and proposes potential avenues for future research.

Chapter 2: An Offline Method for Latent Space Reduction

The method developed in this chapter builds on our work presented in Çatal et al. [7]. It involves initializing a network with a high latent space dimensionality and progressively pruning unnecessary dimensions using singular value decomposition (SVD). This approach mimics the process of synapse pruning during sleep in biological systems. Two environments from the OpenAI Gym were used for experiments: the mountain car environment demonstrated that the network's free energy does not decrease beyond a certain number of dimensions, identifying an optimal latent space size; and the car racing environment showed how latent space dimensionality impacts the reconstruction of observed environments, with fewer dimensions leading to less accurate reconstructions. Using this method it is possible to train a model with many latent space dimensions and then gradually reduce them through the "sleep" process without compromising model performance.

Chapter 3: An Online Method for Latent Space Reduction

The method developed in this chapter also builds on our work presented in [7]. It is designed to optimize the dimensionality of the latent space during the training process, rather than after it has completed. This method addresses the drawbacks of the offline method, which requires multiple training loops and retraining after each pruning step. It relies on a gating mechanism based on L_0 regularization, the Augment-REINFORCE-Merge

(ARM) gradient estimator, and accuracy control via Generalized ELBO with Constraint Optimization (GECO). The method was evaluated using three different environments: two simulated environments (mountain car, car racing) and a real-world scenario (robot navigation). These environments were chosen to represent a range of difficulties and types of observations, from simple one-dimensional data to complex pixel-based inputs. While the online method reduces training time, it might lead to a slight drop in performance compared to the offline method, which achieves higher accuracy through longer training.

Chapter 4: Active Inference with Matrix Product States

This chapter explores the integration of tensor networks, specifically matrix product states (MPS), with active inference. The dynamic determination of the number of parameters is a key advantage of using MPS. It allows the model to adapt to the complexity of the environment without requiring manual tuning of parameters. Moreover, it ensures that the model maintains enough capacity to learn the environment, while also being efficient in terms of computation. The proposed method was tested on the T-maze and Frozen Lake environments. Results indicate that the MPS-based agents successfully learn to act optimally. While the dynamic determination of parameters provides flexibility and adaptability, it comes with trade-offs regarding scalability.

Chapter 5: Active Inference with Uniform Matrix Product States

This last chapter further explores the use of matrix product states (MPS) in active inference and provides the theoretical basis for two expansions to the previous work. Firstly, the MPS is substituted by a translation-invariant MPS, called a uniform MPS (uMPS). uMPS are particularly useful for studying infinite systems, making them ideal for modeling environments with an infinite time horizon. The uMPS implementation addresses a key limitation of the MPS implementation, which was constrained by a finite time horizon. Secondly, quantum mechanics-like operators are used to compute the expected free energy. These operators are significantly simpler to compute and would allow for efficient calculation of the expected free energy for an arbitrary amount of steps in the future.

1.6 Publications

The research results obtained throughout this doctorate have been published in scientific journals and presented at a series of international conferences and workshops. The following list provides an overview of these publications.

1.6.1 Journal Publications

[1] O. Çatal, **S. Wauthier**, C. De Boom, T. Verbelen, and B. Dhoedt, *Learning Generative State Space Models for Active Inference*. Published in Frontiers in Computational Neuroscience 14:574372, 2020.

[2] **S. T. Wauthier**, O. Çatal, C. De Boom, T. Verbelen, and B. Dhoedt, *Model Reduction Through Progressive Latent Space Pruning in Deep Active Inference*. Published in Frontiers in Neurorobotics 16:795846, 2022.

[3] **S. T. Wauthier**, T. Verbelen, B. Dhoedt, and B. Vanhecke, *Planning with tensor networks based on active inference*. Published in Machine Learning: Science and Technology 5:045012, 2024.

1.6.2 Conference Publications

[1] O. Çatal, **S. Wauthier**, T. Verbelen, C. De Boom, and B. Dhoedt, *Deep Active Inference for Autonomous Robot Navigation*. Published in Proceedings of the Bridging AI and Cognitive Sciences workshop, International Conference on Learning Representations (ICLR), 2020.

[2] **S. T. Wauthier**, O. Çatal, C. De Boom, T. Verbelen, and B. Dhoedt, *Sleep: Model Reduction in Deep Active Inference*. Published in Proceedings of the International Workshop on Active Inference, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), 2020.

[3] **S. T. Wauthier**, P. Mazzaglia, O. Çatal, C. De Boom, T. Verbelen, and B. Dhoedt, *A learning gap between neuroscience and reinforcement learning*. Published in Proceedings of the How Can Findings About The Brain Improve AI Systems? workshop, International Conference on Learning Representations (ICLR), 2021.

[4] C. De Boom, **S. Wauthier**, T. Verbelen, and B. Dhoedt, *Dynamic Narrowing of VAE Bottlenecks Using GECO and L_0 Regularization*. Published in Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2021.

[5] **S. T. Wauthier**, B. Vanhecke, T. Verbelen, and B. Dhoedt, *Learning Generative Models for Active Inference Using Tensor Networks*. Published in Proceedings of the International Workshop on Active Inference, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), 2022.

[6] **S. T. Wauthier**, B. Vanhecke, T. Verbelen, and B. Dhoedt, *Tensor networks for active inference with discrete observation spaces*. Published in Proceedings of the Machine Learning and the Physical Sciences workshop, Neural Information Processing Systems (NeurIPS), 2022.

1.7 References

[1] AGI Society, "Conference Series on Artificial General Intelligence," Conference series. https://www.agi-conference.org/, 2024. Last accessed on 2024-08-22.

[2] Aizpurua, B., Palmer, S., and Orus, R., "Tensor Networks for Explainable Machine Learning in Cybersecurity." 2024. arXiv:2401.00867 [cs.LG].

[3] Artificial Intelligence Index, "Artificial Intelligence Index Report," tech. rep., Stanford Institute for Human-Centered Artificial Intelligence, 2024. https://aiindex.stanford.edu/report/.

[4] Bank, D., Koenigstein, N., and Giryes, R., *Autoencoders*, pp. 353–374. Springer International Publishing, 2023.

[5] Berkeley, I. S. N., "The Curious Case of Connectionism," *Open Philosophy* **2** no. 1, (Jan. 2019) 190–205.

[6] Betts, J. G., Young, K. A., Wise, J. A., Johnson, E., Poe, B., Kruse, D. H., Korol, O., Johnson, J. E., Womble, M., and DeSaix, P., *Anatomy and Physiology 2e*. OpenStax, Houston, Texas, Apr. 2022. https://openstax.org/books/anatomy-and-physiology-2e/pages/preface.

[7] Çatal, O., Wauthier, S., De Boom, C., Verbelen, T., and Dhoedt, B., "Learning Generative State Space Models for Active Inference," *Frontiers in Computational Neuroscience* **14** (Nov. 2020) 574372.

[8] Center for AI Safety, "Statement on AI Risk," Open letter. https://www.safe.ai/work/statement-on-ai-risk, June 2023. Last accessed on 2024-08-22.

[9] Cirac, J. I., Pérez-García, D., Schuch, N., and Verstraete, F., "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems," *Reviews of Modern Physics* **93** no. 4, (Dec. 2021) 045003.

[10] Cirelli, C. and Tononi, G., "Is Sleep Essential?" *PLoS Biology* **6** no. 8, (Aug. 2008) e216.

[11] Crevier, D., *AI: The Tumultuous History of the Search for Artificial Intelligence*. BasicBooks, 1993.

[12] Darrach, B., "Meet Shakey, the First Electronic Person," *Life Magazine* (Nov. 1970) 58–68.

[13] "The race of the AI labs heats up," *The Economist* (Jan. 2023) . https://www.economist.com/business/2023/01/30/the-race-of-the-ai-labs-heats-up.

[14] Fannes, M., Nachtergaele, B., and Werner, R. F., "Finitely correlated states on quantum spin chains," *Communications in Mathematical Physics* **144** no. 3, (Mar. 1992) 443–490.

[15] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O'Doherty, J., and Pezzulo, G., "Active inference and learning," *Neuroscience & Biobehavioral Reviews* **68** (Sep. 2016) 862–879.

[16] Friston, K., Parr, T., and Zeidman, P., "Bayesian model reduction." 2019. arXiv:1805.07092 [stat.ME].

[17] Friston, K., Da Costa, L., Hafner, D., Hesp, C., and Parr, T., "Sophisticated Inference," *Neural Computation* **33** no. 3, (Mar. 2021) 713–763.

[18] Friston, K. J., Lin, M., Frith, C. D., Pezzulo, G., Hobson, J. A., and Ondobaka, S., "Active Inference, Curiosity and Insight," *Neural Computation* **29** no. 10, (Oct. 2017) 2633–2683.

[19] Future of Life Institute, "Research Priorities for Robust and Beneficial Artificial Intelligence: An Open Letter," Open letter. https://futureoflife.org/open-letter/ai-open-letter/, Oct. 2015. Last accessed on 2024-08-22.

[20] Garnelo, M. and Shanahan, M., "Reconciling deep learning with symbolic artificial intelligence: representing objects and relations," *Current Opinion in Behavioral Sciences* **29** (Oct. 2019) 17–23.

[21] Glasser, I., Sweke, R., Pancotti, N., Eisert, J., and Cirac, I., "Expressive power of tensor-network factorizations for probabilistic modeling," in *Advances in Neural Information Processing Systems*, Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., eds., vol. 32. Curran Associates, Inc., 2019.

[22] Godwin, W., *Lives of the Necromancers: Or, An Account of the Most Eminent Persons in Successive Ages who Have Claimed for Themselves, Or to Whom Has Been Imputed by Others, the Exercise of Magical Power*. The golden library. Chatto and Windus, 1876.

[23] Goertzel, B., "Who coined the term "AGI"?" Blog post. https://web.archive.org/web/20181228083048/http:

//goertzel.org/who-coined-the-term-agi/, 2011. Last accessed on 2024-08-22.

[24] Goertzel, B., "What counts as a conscious thinking machine?" *New Scientist* (Sep. 2012) . https://www.newscientist.com/article/mg21528813-600-what-counts-as-a-conscious-thinking-machine/.

[25] Google, "Pixel 9 Tech Specs," Store page. https://store.google.com/us/product/pixel_9_specs, 2024. Last accessed on 2024-08-23.

[26] Gubrud, M., "Nanotechnology and International Security," in *Fifth Foresight Conference on Molecular Nanotechnology*. Nov. 1997.

[27] Hassabis, D., "AlphaGo: using machine learning to master the ancient game of Go," Blog post. https://blog.google/technology/ai/alphago-machine-learning-game-go/, Jan. 2016. Last accessed on 2024-08-22.

[28] Hawking, S., Russell, S., Tegmark, M., and Wilczek, F., "Stephen Hawking: 'Transcendence looks at the implications of artificial intelligence - but are we taking AI seriously enough?'," *The Independent (UK)* (May 2014) . https://www.independent.co.uk/news/science/stephen-hawking-transcendence-looks-at-the-implications-of-artificial-intelligence-but-are-we-taking-ai-seriously-enough-9313474.html.

[29] Hochreiter, S. and Schmidhuber, J., "Long Short-Term Memory," *Neural Computation* **9** no. 8, (Nov. 1997) 1735–1780.

[30] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences* **79** no. 8, (Apr. 1982) 2554–2558.

[31] Khrulkov, V., Novikov, A., and Oseledets, I., "Expressive power of recurrent neural networks," in *International Conference on Learning Representations*. 2018.

[32] Knight, W., "OpenAI's CEO Says the Age of Giant AI Models Is Already Over," *WIRED* (Apr. 2023) . https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/.

[33] LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," *Nature* **521** no. 7553, (May 2015) 436–444.

[34] Li, W., Ma, L., Yang, G., and Gan, W.-B., "REM sleep selectively prunes and maintains new synapses in development and learning," *Nature Neuroscience* **20** no. 3, (Jan. 2017) 427–437.

[35] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G., "Backpropagation and the brain," *Nature Reviews Neuroscience* **21** no. 6, (Apr. 2020) 335–346.

[36] Linnainmaa, S., "Taylor expansion of the accumulated rounding error," *BIT* **16** no. 2, (June 1976) 146–160.

[37] Marcus, G., Rossi, F., and Veloso, M., "Beyond the Turing test," Conference workshop. `https://web.archive.org/web/20230210010147/https://www.math.unipd.it/~frossi/BeyondTuring2015/`, 2015. Last accessed on 2024-08-23.

[38] McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C., "A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE," Project proposal. `http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html`, 1955. Last accessed on 2024-08-22.

[39] McCorduck, P., *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. AK Peters Ltd, 2004.

[40] McCulloch, W. S. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics* **5** no. 4, (Dec. 1943) 115–133.

[41] "What is generative AI?" *McKinsey & Company* (Apr. 2024) . `https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai`.

[42] Metz, C., "Google's AlphaGo Continues Dominance With Second Win in China," *WIRED* (May 2017) . `https://www.wired.com/2017/05/googles-alphago-continues-dominance-second-win-china/`.

[43] Metz, C., "'The Godfather of A.I.' Leaves Google and Warns of Danger Ahead," *The New York Times* (May 2023) . `https://www.nytimes.com/2023/05/01/technology/ai-google-chatbot-engineer-quits-hinton.html`.

[44] Mikhaylovskiy, N., *How Do You Test the Strength of AI?*, pp. 257–266. Springer International Publishing, 2020.

[45] Miletínová, E. and Bušková, J., "Functions of Sleep," *Physiological Research* **70** no. 2, (Mar. 2021) 177–182.

[46] Millidge, B., Tschantz, A., and Buckley, C. L., "Whence the Expected Free Energy?" *Neural Computation* **33** no. 2, (Feb. 2021) 447–482.

[47] Mucci, T. and Stryker, C., "Getting ready for artificial general intelligence with examples," *IBM Blog* (Apr. 2024) . https://www.ibm.com/blog/artificial-general-intelligence-examples/.

[48] Muehlhauser, L., "What is AGI?" *Machine Intelligence Research Institute* (Aug. 2013) . https://intelligence.org/2013/08/11/what-is-agi/.

[49] Nellis, S., "Nvidia CEO says AI could pass human tests in five years," *Reuters* (Mar. 2024) . https://www.reuters.com/technology/nvidia-ceo-says-ai-could-pass-human-tests-five-years-2024-03-01/.

[50] Nilsson, N. J., "Human-Level Artificial Intelligence? Be Serious!" *AI Magazine* (2005) . https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/General%20Essays/AIMag26-04-HLAI.pdf.

[51] "NVIDIA A100 TENSOR CORE GPU," tech. rep., NVIDIA Corporation, 2020. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf.

[52] NVIDIA, "GTC March 2024 Keynote with NVIDIA CEO Jensen Huang," Keynote speech. https://www.youtube.com/watch?v=Y2F8yisiS6E&list=TLGGFIbdOwQMZx4yMzA4MjAyNA, 2024. Last accessed on 2024-08-23.

[53] Ormerod, D., "AlphaGo defeats Lee Sedol 4–1 in Google DeepMind Challenge Match," *Go Game Guru* (Mar. 2016) . https://web.archive.org/web/20160317095008/https://gogameguru.com/alphago-defeats-lee-sedol-4-1/.

[54] Orús, R., "Tensor networks for complex quantum systems," *Nature Reviews Physics* **1** no. 9, (Aug. 2019) 538–550.

[55] Oseledets, I. V., "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing* **33** no. 5, (Jan. 2011) 2295–2317.

[56] Ouyang, L., Wu, J., *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, Koyejo, S., Mohamed, S., Agarwal, A., Belgrave,

D., Cho, K., and Oh, A., eds., vol. 35, pp. 27730–27744. Curran Associates, Inc., 2022.

[57] Paolo, G., Gonzalez-Billandon, J., and Kégl, B., "Position: A Call for Embodied AI," in *Proceedings of the 41st International Conference on Machine Learning*, Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., eds., vol. 235 of *Proceedings of Machine Learning Research*, pp. 39493–39508. PMLR, 21–27 jul 2024.

[58] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*, ch. The High Road to Active Inference, pp. 41–62. In [61], Mar. 2022.

[59] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*, ch. The Low Road to Active Inference, pp. 15–40. In [61], Mar. 2022.

[60] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*, ch. Overview, pp. 3–14. In [61], Mar. 2022.

[61] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, Mar. 2022.

[62] Pennisi, E., "The simplest of slumbers," *Science* **374** no. 6567, (Oct. 2021) 526–529.

[63] Perez-Garcia, D., Verstraete, F., Wolf, M., and Cirac, J., "Matrix product state representations," *Quantum Information and Computation* **7** no. 5 & 6, (July 2007) 401–430.

[64] R. Zielinski, M., T. McKenna, J., and W. McCarley, R., "Functions and Mechanisms of Sleep," *AIMS Neuroscience* **3** no. 1, (2016) 67–104.

[65] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I., "Improving language understanding with unsupervised learning," 2018. OpenAI.

[66] Ran, S.-J. and Su, G., "Tensor Networks for Interpretable and Efficient Quantum-Inspired Machine Learning," *Intelligent Computing* **2** (Jan. 2023) 0061.

[67] Roose, K., "A.I. Poses 'Risk of Extinction,' Industry Leaders Warn," *The New York Times* (May 2023) . `https://www.nytimes.com/2023/05/30/technology/ai-threat-warning.html`.

[68] Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review* **65** no. 6, (1958) 386–408.

[69] Russell, S. J. and Norvig, P., *Artificial Intelligence: A Modern Approach*. Pearson, 4 ed., 2021.

[70] Schreiner, M., "GPT-4 architecture, datasets, costs and more leaked," *THE DECODER* (2023) . https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/.

[71] Shapiro, L. and Spaulding, S., "Embodied Cognition," in *The Stanford Encyclopedia of Philosophy*, Zalta, E. N. and Nodelman, U., eds. Metaphysics Research Lab, Stanford University, Fall 2024 ed., 2024. https://plato.stanford.edu/archives/fall2024/entries/embodied-cognition/.

[72] Siegel, J. M., "Do all animals sleep?" *Trends in Neurosciences* **31** no. 4, (Apr. 2008) 208–213.

[73] Silver, D., Schrittwieser, J., *et al.*, "Mastering the game of Go without human knowledge," *Nature* **550** no. 7676, (Oct. 2017) 354–359.

[74] Simon, H., *The Shape of Automation for Men and Management*. Harper Torchbooks. The Academy Library. Harper & Row, 1965.

[75] Smith, R., Schwartenbeck, P., Parr, T., and Friston, K. J., "An active inference approach to modeling structure learning: concept learning as an example case,".

[76] Tangpanitanon, J., Mangkang, C., Bhadola, P., Minato, Y., Angelakis, D. G., and Chotibut, T., "Explainable natural language processing with matrix product states," *New Journal of Physics* **24** no. 5, (May 2022) 053032.

[77] Turing, A. M., "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind* **LIX** no. 236, (Oct. 1950) 433–460.

[78] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I., "Attention is All you Need," in *Advances in Neural Information Processing Systems*, Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., eds., vol. 30. Curran Associates, Inc., 2017.

[79] Wang, M., Pan, Y., Xu, Z., Yang, X., Li, G., and Cichocki, A., "Tensor Networks Meet Neural Networks: A Survey and Future Perspectives." 2023. arXiv:2302.09019 [cs.LG].

[80] Wang, P. and Goertzel, B., "Introduction: Aspects of Artificial General Intelligence," in *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006*, pp. 1–16. IOS Press, 2007.

[81] White, S. R., "Density matrix formulation for quantum renormalization groups," *Physical Review Letters* **69** no. 19, (Nov. 1992) 2863–2866.

[82] "Neuron3.svg," Wikimedia Commons, by Egm4313.s12 (Prof. Loc Vu-Quoc). `https://commons.wikimedia.org/w/index.php?curid=72801384`, 2018. Own work, CC BY-SA 4.0. Last accessed on 2025-02-10.

[83] "Artificial neuron structure.svg," Wikimedia Commons, by Funcs. `https://commons.wikimedia.org/w/index.php?curid=148910507`, 2024. Own work, CC0. Last accessed on 2025-02-10.

[84] Wikipedia contributors, "Artificial neuron — Wikipedia, The Free Encyclopedia," `https://en.wikipedia.org/w/index.php?title=Artificial_neuron&oldid=1274681232`, 2025. [Online; accessed 20-February-2025].

—It is a common experience that a problem difficult at night is resolved in the morning after the committee of sleep has worked on it.

John Steinbeck

2

An offline method for latent space reduction

The research in this chapter was originally intended to be a stepping stone towards further research into the structure of learned latent spaces in active inference. The use of artificial neural networks does not guarantee that every latent space dimension encodes a unique concept in the agent's environment, which is often referred to as disentanglement. Using singular value decomposition, it is possible to identify directions in the latent space that are linearly uncorrelated, and are therefore expected to contain concepts which are unrelated.

In this work, singular value decomposition is instead used to find directions that contain little to no variation, directions in which the state changes very little no matter what the agent sees and does. If such directions are found, it suggests that a latent space can be constructed without them. In other words, the latent space can be reduced.

Sleep: Model Reduction in Deep Active Inference

S. T. Wauthier • O. Çatal • C. De Boom • T. Verbelen • B. Dhoedt
Published in Active Inference. IWAI 2020. Communications in Computer and Information Science, vol 1326. Springer, Cham.

Abstract

Sleep is one of the most important states of the human mind and body. Sleep has various functions, such as restoration, both physically and mentally, and memory processing. The theory of active inference frames sleep as the minimization of complexity and free energy in the absence of sensory information. In this paper, we propose a method for model reduction of neural networks that implement the active inference framework. The proposed method suggests initializing the network with a high latent space dimensionality and pruning dimensions subsequently. We show that reduction of latent space dimensionality decreases complexity without increasing free energy.

2.1 Introduction

Sleep is a phenomenon that occurs in most animals [10]. It is a topic of intensive research as it has been shown to be important for both the mind [18, 12] and the body [14]. In particular, sleep and learning have been connected in many hypotheses [17, 3], as well as mental health [4] and memory [20].

Active inference is a theory of behaviour and learning that originated in neuroscience [8]. The basic assumption is that intelligent agents attempt to minimize their variational free energy. Variational free energy — named for its counterpart in statistical physics i.e. Helmholtz free energy — is also known as the evidence lower bound (ELBO) in variational Bayesian methods.

Since its conception, active inference has been explored in multiple subfields of neuroscience and biology [5, 6, 11] and eventually found its way into the field of computer science [19, 15, 2]. In particular, Ueltzhöffer [19] and Çatal *et al.* [2] have made developments in *deep active inference*, i.e. the use of deep neural networks to implement active inference.

Recent work [13, 9, 7] has pointed out the relation between the function of removing redundant connections during sleep and Bayesian model reduction

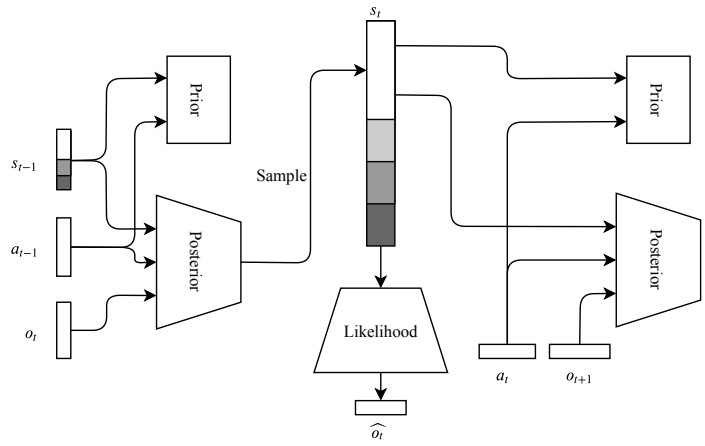


Figure 2.1: Information flow of neural networks. The posterior network takes in previous state and action, and current observation. The prior network takes in previous state and action. The likelihood network takes in current state. The state s_t illustrates that dimensions may be pruned, if they are unused.

(BMR) in active inference, i.e. complexity minimization through elimination of redundant parameters. In this work, we propose a method for reducing complexity in the deep active inference framework. We evaluate the method through simulation experiments.

2.2 Deep active inference

Currently, using deep neural networks in active inference to learn state spaces, in addition to policy and posterior, is becoming increasingly popular, which contrasts with active inference on discrete state spaces as described in [9]. In this approach, the dimensionality of the state space is a hyperparameter, i.e. it must be specified before training and cannot change along the way. Here, we briefly introduce the method provided by Çatal *et al.* [2].

Assuming the policy π may be broken up into a sequence of actions $\mathbf{a}_t$ and the current state depends on the previous action instead of the policy, a generative model with observations $\mathbf{o}_t$ and states $\mathbf{s}_t$ is defined as

$$P(\tilde{\mathbf{o}}, \tilde{\mathbf{s}}, \tilde{\mathbf{a}}) = P(\mathbf{s}_0)P(\tilde{\mathbf{a}}) \prod_{t=1}^{T} P(\mathbf{o}_t|\mathbf{s}_t)P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), \tag{2.1}$$

where $\tilde{\mathbf{x}} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$.

Deep neural networks are used to parameterize the prior, likelihood and approximate posterior distributions: $p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$, $p_\phi(\mathbf{o}_t|\mathbf{s}_t)$ and $q_\xi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \mathbf{o}_t)$, respectively. With this, minimization of free energy consists of minimizing the loss function

$$L(\theta, \phi, \xi; \mathbf{o}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) = \\ \mathrm{D}_{\mathrm{KL}}\left(q_\xi(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \mathbf{o}_t) \,||\, p_\theta(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})\right) - \log p_\phi(\mathbf{o}_t|\mathbf{s}_t). \quad (2.2)$$

Prior, likelihood and posterior distributions are chosen to be multivariate normal distributions. As opposed to the standard VAE, optimization is done over sequences in time. Additionally, empirical priors are learned, instead of using fixed priors. A chart on the information flow can be found in Figure 2.1.

2.3 Latent space dimensionality reduction and sleep

The size of the latent space vector $\mathbf{s}$ is an important hyperparameter. On the one hand, this must be large enough to explain observations in the generative model. On the other hand, it must be kept minimal to reduce complexity as to minimize the required resources, such as memory and power (both computational and electrical). In general, one does not know the optimal size of $\mathbf{s}$. A typical way of finding a well-performing value is a hyperparameter sweep. Parameter sweeps, however, are resource intensive and require many unnecessary training runs. Therefore, we propose a method for dimensionality reduction in the deep active inference framework.

The basic idea is to prune dimensions in the latent space vector $\mathbf{s}$. A popular method for inspecting informative dimensions of a vector space is singular value decomposition (SVD). This technique is used to factorize an $n \times m$ matrix $\boldsymbol{A}$ into three matrices $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^*$, where a common geometrical interpretation is that the decomposition gives 2 rotation matrices $\boldsymbol{U}$ and $\boldsymbol{V}^*$, and a scaling matrix $\boldsymbol{S}$.

Algorithm 2.1 lines out the method in the form of pseudo-code. Let n be the dimensionality of the latent space. We sample a latent space vector from m different sequences to construct the column vectors of a matrix $\boldsymbol{A}$. The column space of $\boldsymbol{A}$, denoted $\mathcal{C}(\boldsymbol{A})$, forms a subspace of the latent space. Applying SVD to $\boldsymbol{A}$ gives the scaling matrix $\boldsymbol{S}$. The values on the diagonal of $\boldsymbol{S}$ are the singular values of $\boldsymbol{A}$, and suggest a size for the dimensions of $\mathcal{C}(\boldsymbol{A})$ after rotation with $\boldsymbol{U}$. Dimensions with small singular values are assumed to be unused. To this end, we define a threshold α for which dimensions corresponding to singular values smaller than α can be pruned. We repeat

Algorithm 2.1: Sleep

input : A trained model model with dimensionality n
The number of repetitions N and number of sequences m
A threshold α
output : The new latent space dimensionality ν

```
while i < N do
    A ← []                                              // make a matrix
    while j < m do
        a ← GenerateSequence(model)                     // generate a new
                                                           sequence
        v ← Sample(a)                                   // sample a latent space vector
        A ← [A, v]                                      // insert vector as a new column in
                                                           matrix
        j ← j + 1
    S ← SVD(A)                                          // apply SVD to matrix
    c_i ← #(S_kk > α) for 0 < k < n                     // count sv's over
                                                           threshold
    c ← [c, c_i]                                        // add number to list of outcomes
    i ← i + 1
ν ← Avg(c)                                              // average over all outcomes
```

this procedure N times — by generating m new sequences each time — and average the number of pruned dimensions, in order to obtain a relatively robust outcome.

It is important to stress, here, that SVD does not allow one to find *which* dimensions can be pruned. Instead, it is used to converge to the optimal *number* of dimensions. SVD provides the size of dimensions of the column space of $\boldsymbol{A}$, i.e. $\mathcal{C}(\boldsymbol{A})$, described in a basis of the latent space after a rotation with $\boldsymbol{U}$. The actual basis vectors are a linear combination of the rotated basis vectors. In other words, having a zero dimension[1] in rotated latent space, does not necessarily mean there is one in latent space. However, it does indicate that it is possible to reduce dimensionality by choosing a different rotation, since it shows that there is an orientation of the basis vectors which requires less dimensions to describe the column space. Returning to the model, by retraining with a lower dimensionality n, we essentially force the model to learn the latent space with a different orientation which requires less dimensions.

We have dubbed the method *sleep*, since it replicates synapse pruning, as well as Bayesian model reduction. From an active inference perspective, the proposed method is analogous to BMR in that it considers a generative

[1] That is, having a dimension with small singular value.

model with a large number of latent factors and optimizes this number post hoc [16]. In other words, both the goals of the proposed method and BMR are to consider alternative models which may give simpler explanations for the same observations. That said, in both cases, the balance between accuracy and complexity is crucial, i.e. accuracy should not suffer due to simplicity. Indeed, the measure for this trade-off is free energy.

Since latent space in deep active inference is learned using deep neural networks, there is no guarantee that each latent space dimension represents an individual feature. Without knowing what is contained in latent space, it is not possible to target specific parameters to turn off as in BMR. Because of this, algorithm 2.1 must be succeeded by retraining to obtain a reduced model. In this sense, the earlier analogy is incomplete, since BMR allows one to obtain the reduced model parameters from the full model, i.e. it allows one to find *which* dimensions can be pruned.

In the end, the purpose of the sleep method is to reduce complexity whenever an application (e.g. a robot running a deep active inference implementation) has downtime. The overall sequence of events, then, proceeds as follows. Start with a large value for the latent space dimensionality and train the model. Deploy the model on the application. Each time there is downtime in the application (e.g. the robot is charging), reduce the model by sleeping and retraining. Continue this pattern of sleeping and retraining until the model cannot reduce any further.

2.4 Experimental setup

Experiments were performed using two environments from the OpenAI Gym [1]. The first experiment employs a modified version of the MountainCar environment, where noise is added to the observation and only the position can be observed. The goal of this environment is to drive up a steep mountain using an underpowered car that starts in a valley. The car is underpowered in the sense that it cannot produce enough force to go against gravity and drive up the mountain in one go. It must first build up enough momentum by driving up the side(s) of the valley. In this experiment, we know upfront that the model only needs 2 dimensions in latent space: position and velocity. Details about the neural networks used for this experiment can be found in 2.B.1.

The second experiment employs the CarRacing environment. The goal of this environment is to stay in the middle of a race track using a race car. The car and track are viewed from a top-down perspective. The car must steer left and right to stay on track. Compared to the MountainCar, the CarRacing environment utilizes more complicated dynamics and produces

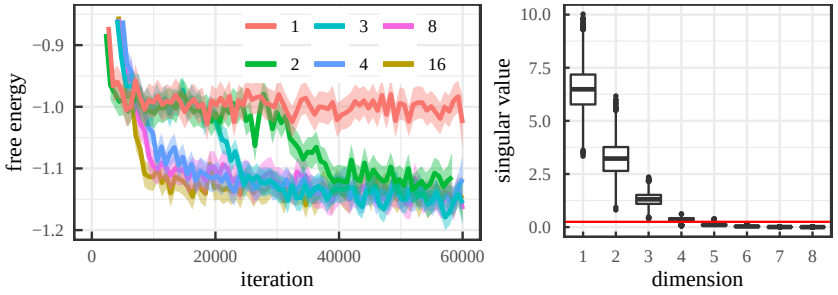


Figure 2.2: (Left) Free energy during training of MountainCar for different state space sizes. Curves show smoothed data (LOESS, span 0.02) with 95% standard error bands. (Right) Boxplot of singular values while sleeping at 8 latent space dimensions ($N = 10^4$).

higher dimensional observations. Examples of the environments can be found in 2.A. Details about the neural networks used for this experiment can be found in 2.B.2.

2.5 Results

Fig. 2.2 shows the evolution of the free energy of MountainCar during training with a fixed number of latent space dimensions (see 2.C.1 for a similar figure for CarRacing). It suggests that free energy decreases as more state space dimensions are added. However, it also shows that free energy does not visibly decrease beyond a certain number of dimensions. For the MountainCar, we see that the free energy does not decrease for more than 2 dimensions, while for the CarRacing (2.C.1), we see that the free energy does not decrease for more than 4 dimensions. In essence, there appears to be a critical value of the latent space size. For latent spaces larger than this critical value, the free energy does not reduce. This critical value corresponds to the optimal value for the dimensionality with respect to the accuracy/complexity trade-off.

Fig. 2.3 demonstrates how latent space dimensionality affects reconstruction and how too few dimensions can lead to aspects of the environment not being learned. It shows reconstructions of the CarRacing track for latent space dimensions of 32, 16, 8, 4, 2 and 1 (top to bottom with ground truth in the top sequence). Note how the curvature of the track is not accurately reconstructed through 1 dimension, especially at early time steps. Also, 2 dimensions still seem to lack accuracy (see curvature in second time step). Furthermore, note how the feedback bar is incorrectly encoded by dimensions lower than 4.

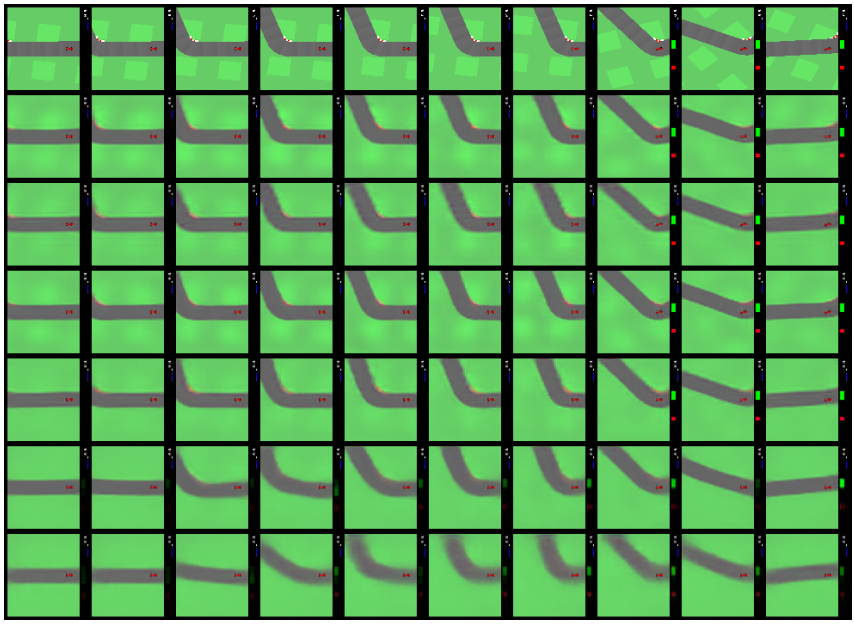


Figure 2.3: Reconstructions of CarRacing track over time with different latent space dimensions. From top to bottom: ground truth, 32, 16, 8, 4, 2, 1.

Fig. 2.2 also shows a boxplot for the singular values obtained for the MountainCar with 8 latent space dimensions for $N = 10^4$ iterations (plots for different latent space dimensions can be found in 2.C.2). The red line shows a threshold $\alpha = 0.25$. The figure suggests that there is a difference in sizes in the latent space dimensions. Indeed, the first four singular values are on average larger than α, while the remaining values are on average smaller than α. This indicates that certain dimensions are very small[2], therefore, contain less information, and may be pruned subsequently.

Fig. 2.4 illustrates the algorithm put into practice with different sleep cycles for the CarRacer with threshold set at $\alpha = 0.25$, where we started with 16 latent space dimensions. In this example, we initiated sleep every 5×10^4 training iterations and checked if dimensionality could be reduced. If so, we pruned and restarted training with lower dimensionality, else we continued training for 5×10^4 iterations, until reduction was possible. We stopped the process after 7 sleep cycles. As expected, the sleep sequence manages to reduce the complexity of the model, without impacting the free energy negatively.

[2]This means that the variance of the data within these dimensions is small.

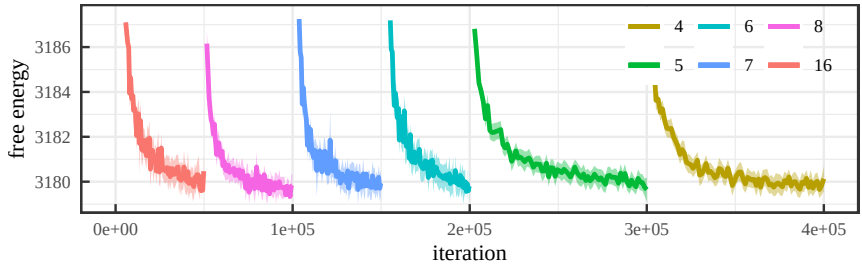


Figure 2.4: Free energy over 7 sleep cycles of CarRacer. Setting threshold $\alpha = 0.25$ gives the reduction: $16 \to 8 \to 7 \to 6 \to 5 \to 4$, after which it cannot reduce further. Curves show smoothed data (LOESS, span 0.02) with 95% standard error bands.

When compared to Fig. 2.8 in 2.C.2, the previous result is exactly as expected. Following the steps described there, the state space can effectively be pruned down to 4 dimensions. Observe that if we were to repeat the experiment for the MountainCar, Fig. 2.7 in 2.C.2 shows that setting the threshold at $\alpha = 0.25$ would return a state space dimensionality of 2.

2.6 Conclusion

Our results show that it is possible to train a deep active inference model by setting a large number of latent space dimensions and subsequently sleeping until minimal complexity is reached. However, the method proposed in this paper is not optimal. A few caveats remain. First of all, the current method requires retraining. After applying SVD, the entire model must be retrained from scratch. Second of all, there exist limitations to SVD. For instance, SVD does not take into account nonlinear transformations. Therefore, relations between different dimensions may remain and the optimal dimensionality may never be reached.

In future work, we will investigate the effects of sleeping at regular intervals during training. For example, we may sleep after every 10^4 time steps to check if we can already reduce the latent space. Another option we will investigate, is to prune both unnecessary dimensions and weights. This way, we may be able to maintain the trained neural network, while reducing complexity. In addition, we want to experiment with different methods for dimensionality reduction, such as nonlinear methods. Another option to be explored is to learn and set unused dimensions to 0 during training.

Acknowledgments

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme.

2.7 References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym." 2016. arXiv:1606.01540 [cs.LG].

[2] Çatal, O., Nauta, J., Verbelen, T., Simoens, P., and Dhoedt, B., "Bayesian policy selection using active inference." 2019. arXiv:1904.08149 [cs.LG].

[3] Fattinger, S., de Beukelaar, T. T., Ruddy, K. L., Volk, C., Heyse, N. C., Herbst, J. A., Hahnloser, R. H. R., Wenderoth, N., and Huber, R., "Deep sleep maintains learning efficiency of the human brain," *Nature Communications* **8** no. 1, (May 2017) 15405.

[4] Freeman, D., Sheaves, B., *et al.*, "The effects of improving sleep on mental health (OASIS): a randomised controlled trial with mediation analysis," *The Lancet Psychiatry* **4** no. 10, (Oct. 2017) 749–758.

[5] Friston, K., Mattout, J., and Kilner, J., "Action understanding and active inference," *Biological Cybernetics* **104** no. 1–2, (Feb. 2011) 137–160.

[6] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O'Doherty, J., and Pezzulo, G., "Active inference and learning," *Neuroscience & Biobehavioral Reviews* **68** (Sep. 2016) 862–879.

[7] Friston, K., Parr, T., and Zeidman, P., "Bayesian model reduction." 2018. arXiv:1805.07092 [stat.ME].

[8] Friston, K. J., Daunizeau, J., and Kiebel, S. J., "Reinforcement Learning or Active Inference?" *PLoS ONE* **4** no. 7, (July 2009) e6421.

[9] Friston, K. J., Lin, M., Frith, C. D., Pezzulo, G., Hobson, J. A., and Ondobaka, S., "Active Inference, Curiosity and Insight," *Neural Computation* **29** no. 10, (Oct. 2017) 2633–2683.

[10] Joiner, W. J., "Unraveling the Evolutionary Determinants of Sleep," *Current Biology* **26** no. 20, (Oct. 2016) R1073–R1087.

[11] Kirchhoff, M., Parr, T., Palacios, E., Friston, K., and Kiverstein, J., "The Markov blankets of life: autonomy, active inference and the free energy principle," *Journal of The Royal Society Interface* **15** no. 138, (Jan. 2018) 20170792.

[12] Krause, A. J., Simon, E. B., Mander, B. A., Greer, S. M., Saletin, J. M., Goldstein-Piekarski, A. N., and Walker, M. P., "The sleep-deprived human brain," *Nature Reviews Neuroscience* **18** no. 7, (May 2017) 404–418.

[13] Li, W., Ma, L., Yang, G., and Gan, W.-B., "REM sleep selectively prunes and maintains new synapses in development and learning," *Nature Neuroscience* **20** no. 3, (Jan. 2017) 427–437.

[14] Newman, A. B., Nieto, F. J., Guidry, U., Lind, B. K., Redline, S., Shahar, E., Pickering, T. G., and Quan, S. F., "Relation of Sleep-disordered Breathing to Cardiovascular Disease Risk Factors: The Sleep Heart Health Study," *American Journal of Epidemiology* **154** no. 1, (July 2001) 50–59.

[15] Oliver, G., Lanillos, P., and Cheng, G., "Active inference body perception and action for humanoid robots." 2019. `arXiv:1906.03022 [cs.RO]`.

[16] Smith, R., Schwartenbeck, P., Parr, T., and Friston, K. J., "An Active Inference Approach to Modeling Structure Learning: Concept Learning as an Example Case," *Frontiers in Computational Neuroscience* **14** (May 2020) .

[17] Stickgold, R., Hobson, J. A., Fosse, R., and Fosse, M., "Sleep, Learning, and Dreams: Off-line Memory Reprocessing," *Science* **294** no. 5544, (Nov. 2001) 1052–1057.

[18] Tsuno, N., Besset, A., and Ritchie, K., "Sleep and Depression," *The Journal of Clinical Psychiatry* **66** no. 10, (Oct. 2005) 1254–1269.

[19] Ueltzhöffer, K., "Deep active inference," *Biological Cybernetics* **112** no. 6, (Oct. 2018) 547–573.

[20] Walker, M. P. and Stickgold, R., "Sleep, Memory, and Plasticity," *Annual Review of Psychology* **57** no. 1, (Jan. 2006) 139–166.

2.A OpenAI Gym examples

Fig. 2.5 shows snapshots of the MountainCar and CarRacing environments from the OpenAI Gym [1]. Note that observations in the MountainCar environment consist of position and velocity values[3], while CarRacing provides RGB pixels[4].

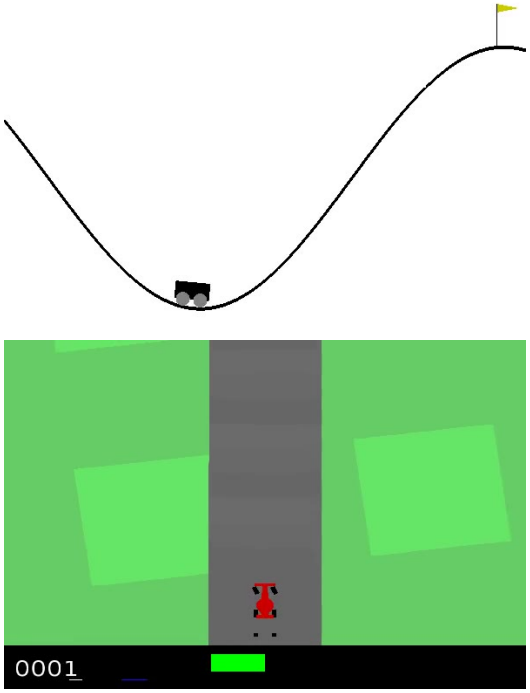


Figure 2.5: (Top) Example of MountainCar environment [1]. (Bottom) Example of CarRacing environment [1].

[3]The observations are encoded as real values. The car can perform 3 actions: accelerate to the left, don't accelerate, and accelerate to the right. The actions are encoded as a single integer-valued variable.

[4]The observations are 96×96 pixels. The car can perform 5 actions: do nothing, steer left, steer right, accelerate and brake. The actions are encoded as a single integer-valued variable.

2.B Neural network definitions

2.B.1 Mountain car

Table 2.1 shows the neural architecture of the network used in the MountainCar experiments.

Table 2.1: Specifications of the MountainCar neural network with s latent space dimensions.

	Layer	Neurons/Filters	Activation Function
Posterior	Linear	20	Leaky ReLU
	Linear	$2 \times s$	Leaku ReLU
Likelihood	Linear	20	Leaky ReLU
	Linear	2	Leaky ReLU
Prior	Linear	20	Leaky ReLU
	Linear	$2 \times s$	Leaky ReLU

2.B.2 Car racing

Table 2.2 shows the neural architecture of the network used in the CarRacing experiments. All filters have 3×3 kernel, as well as stride and padding of 1.

Table 2.2: Specifications of the CarRacing neural network with s latent space dimensions.

	Layer	Neurons/Filters	Activation Function
Posterior	Convolutional	8	Leaky ReLU
	Convolutional	16	Leaky ReLU
	Convolutional	32	Leaky ReLU
	Convolutional	64	Leaky ReLU
	Convolutional	128	Leaky ReLU
	Convolutional	256	Leaky ReLU
	concat	N/A	N/A
	Linear	$2 \times s$	Leaku ReLU
Likelihood	Linear	$128 \times 2 \times 9$	Leaky ReLU
	Convolutional	128	Leaky ReLU
	Convolutional	64	Leaky ReLU
	Convolutional	32	Leaky ReLU
	Convolutional	16	Leaky ReLU
	Convolutional	8	Leaky ReLU
	Convolutional	3	Leaky ReLU
Prior	LSTM cell	128	Leaky ReLU
	Linear	$2 \times s$	Softplus

2.C Additional plots

2.C.1 Free energy during training

Fig. 2.6 shows the evolution of the free energy during training for CarRacing similar to the left plot in Fig. 2.2. Note how the free energy does not visibly decrease when using more than 4 dimensions.

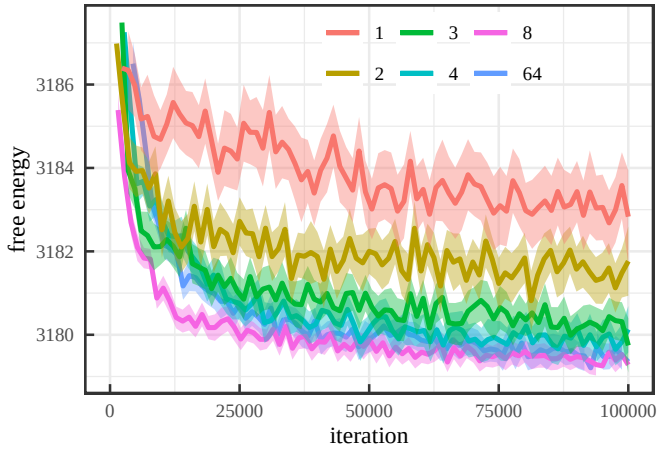


Figure 2.6: Free energy during training of CarRacer for different state space sizes. Curves show smoothed data (LOESS, span 0.02) with 95% standard error bands.

2.C.2 Boxplots for different latent space dimensions

Fig. 2.7 shows boxplots for the singular values obtained for the MountainCar with different latent space dimensions for 10^4 iterations, while Fig. 2.8 shows the same for the CarRacing. The red line in each plot indicates the threshold $\alpha = 0.25$.

One can do the following mental exercise. Choose a boxplot and count the amount of dimensions that are above threshold on average. This number will be the new dimensionality. Go to the boxplot for that dimensionality and, again, count the amount of dimensions. Repeat this until the dimensionality does not reduce further. Using this process, we can see that the MountainCar will not reduce below 2 and the CarRacing will not reduce below 4.

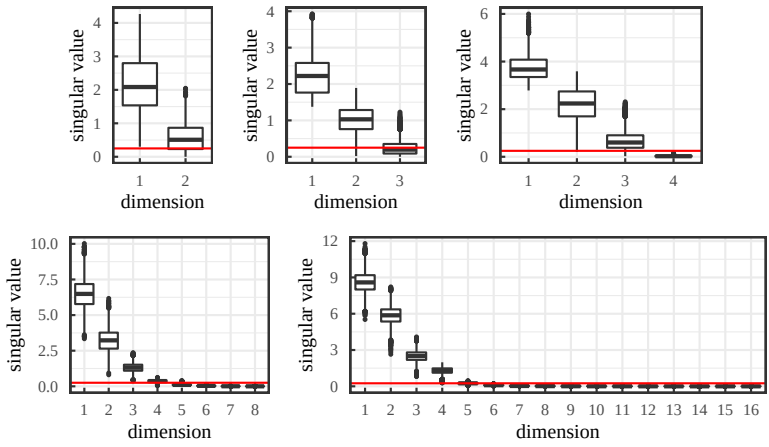


Figure 2.7: Boxplots of singular values while sleeping at different latent space dimensions for the MountainCar ($N = 10^4$).

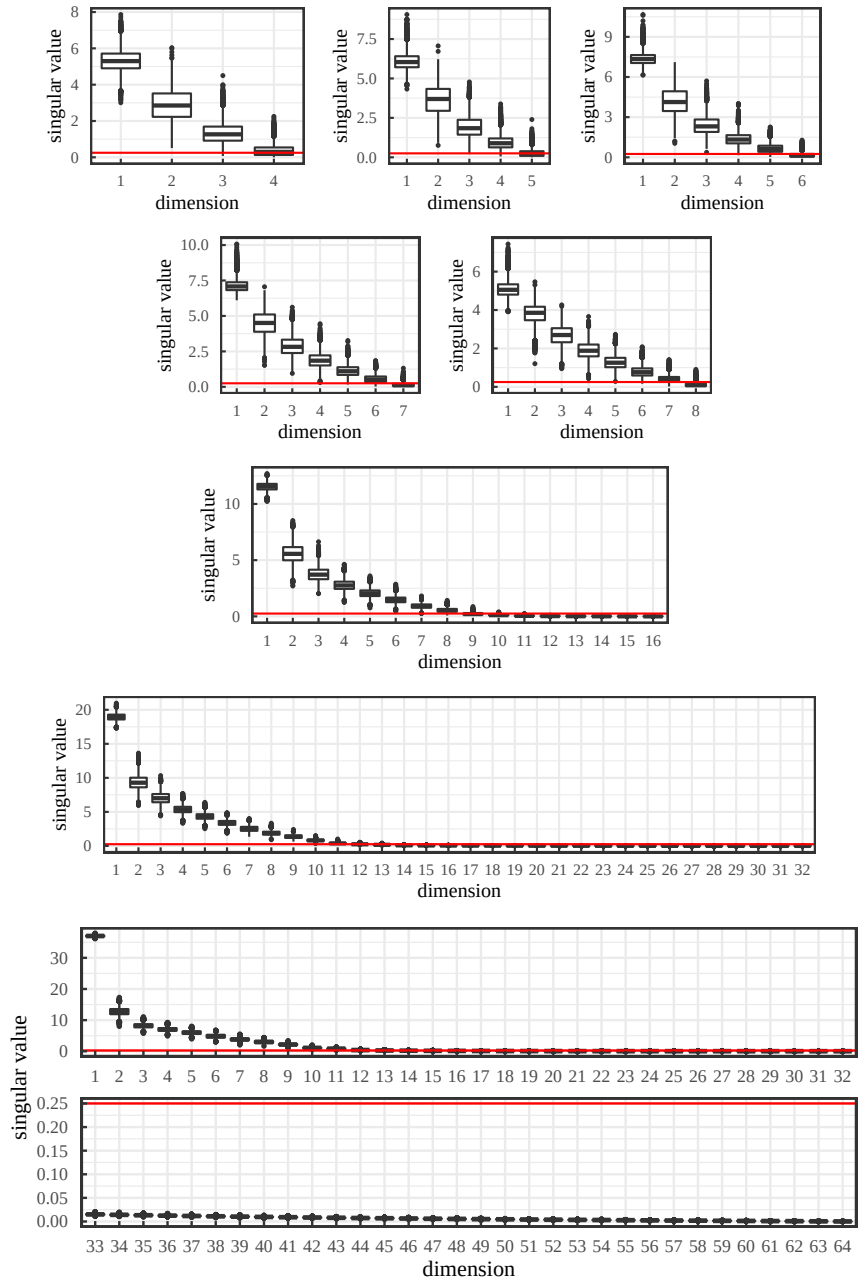


Figure 2.8: Boxplots of singular values while sleeping at different latent space dimensions for the CarRacing ($N = 10^4$).

— Man is a genius when he is dreaming.

Akira Kurosawa

3

An online method for latent space reduction

The natural follow-up to the offline method in the previous chapter is an online method. Ironically, the research for this method was performed over online calls at the height of the COVID-19 pandemic. It resulted in the technique presented in this chapter, which employs a gating mechanism to restrict the number of dimensions in the latent space.

In the end, this technique is again based on ANNs. One can ask oneself whether ANNs really lend themselves to the idea of model reduction. On the one hand, they were inspired by biological neural networks, where connection pruning is ubiquitous, giving the impression that complexity reduction may be straightforward. On the other hand, how to prune connections in the mathematical objects that make up ANNs is less than trivial, as evidenced by the entire field of sparse neural networks that was born from this very question. Subsequent chapters will step away from ANNs.

In order to compare the offline method with the proposed online method, this chapter briefly revisits the offline method. Following new insights obtained after the previous chapter, it introduces some slight changes with respect to the algorithm. These insights were the result of additional research and a deepened understanding of the algorithm. While there is no practical difference between the offline methods laid out in the previous and the current chapter, there are some clarifications and refinements.

The most notable differences are in the construction of the matrices $\boldsymbol{A}$. Firstly, the previous chapter mentions construction through column vectors. In this chapter, this is changed to row vectors, to be more in line with how data matrices are usually implemented. However, since singular values are invariant under transposition, this has no effect on the outcome.

Secondly, the previous chapter mentions using generic $m \times n$ matrices, while this chapter specifies using square matrices. Previously, the use of square matrices was considered to be a computational detail, since the algorithm should work for any shape of $\boldsymbol{A}$. While this remains true, it is an unavoidable fact that the outcome of the algorithm depends on the choice of threshold, and the shape of $\boldsymbol{A}$ determines how this threshold should be chosen. To see this, consider that, for an arbitrary matrix $\boldsymbol{X}$ of shape $n \times n$ and a matrix $\boldsymbol{Y} = \begin{pmatrix} X & r_{n+1} & \cdots & r_m \end{pmatrix}^T$ of shape $m \times n$ with $m > n$ constructed by appending the arbitrary row vectors $r_{n+1}, \ldots, r_m$ to $\boldsymbol{X}$, we have that $\sigma_i(\boldsymbol{X}) < \sigma_i(\boldsymbol{Y})$ for $i \leq n$, where the singular values of $\boldsymbol{X}$ and $\boldsymbol{Y}$, i.e., $\sigma_i(\boldsymbol{X})$ and $\sigma_i(\boldsymbol{Y})$, are ordered from largest to smallest. In other words, longer matrices generally have larger singular values. Thus, although both the shape of $\boldsymbol{A}$ and the threshold can be freely chosen, there is a certain interplay between the two that should be taken into account. Longer matrices require larger thresholds for the algorithm to yield the same reduction, such that choosing an appropriate threshold becomes less straightforward. The use of square matrices facilitates the interpretation of the threshold, since this lends itself to the typical scaling matrix interpretation, and in turn, facilitates choosing the threshold.

Model Reduction through Progressive Latent Space Pruning in Deep Active Inference

S. T. Wauthier • C. De Boom • O. Çatal • T. Verbelen • B. Dhoedt
Published in Frontiers in Neurorobotics 2022, 16:795846.

Abstract

Although still not fully understood, sleep is known to play an important role in learning and in pruning synaptic connections. From the active inference perspective, this can be cast as learning parameters of a generative model and Bayesian model reduction, respectively. In this paper, we show how to reduce dimensionality of the latent space of such a generative model, and hence model complexity, in deep active inference during training through a similar process. While deep active inference uses deep neural networks for state space construction, an issue remains in that the dimensionality of the latent space must be specified beforehand. We investigate two methods that are able to prune the latent space of deep active inference models. The first approach functions similar to sleep and performs model reduction post hoc. The second approach is a novel method which is more similar to reflection, operates during training and displays 'aha' moments when the model is able to reduce latent space dimensionality. We show for two well-known simulated environments that model performance is retained in the first approach and only diminishes slightly in the second approach. We also show that reconstructions from a real world example are indistinguishable before and after reduction. We conclude that the most important difference constitutes a trade-off between training time and model performance in terms of accuracy and the ability to generalise, via minimization of model complexity.

3.1 Introduction

While the role of sleep in animals still contains a lot of mystery [45, 35], it has been linked to many phenomena, such as restorative processes in the brain [32] and memory processing [52, 5, 58]. In particular, sleep and learning appear to be deeply intertwined [38, 60, 34]. Recent work has indicated that the removal of redundant neural connections during sleep

[42] can be compared to minimization of complexity through elimination of redundant parameters during Bayesian model reduction (BMR) in Bayesian approaches to brain function [31, 22, 18]. Removal of redundant connections while strengthening others should promote learning [42].

Artificial agents used for learning specific tasks are often based on the formalism of Markov decision processes (MDPs) [64, 47, 27]. In this formalism, the complexity of the environment determines the complexity of the latent space (a.k.a. state space), as the number of dimensions grows rapidly with the number of possible states the agent can find itself in. A large state space increases computational costs and can lead to overfitting [59]. It is possible to reduce the state space through various methods, such as state clustering or segmentation (e.g. Q-learning with adaptive state segmentation [49]), state vector transformation (e.g. basis iteration for reward based dimensionality reduction [57]) and state space reconstruction (e.g. action respecting embedding [6]) [59]. State space reduction can improve generalization, as well as reduce computational complexity and learning time.

In cases where the agent cannot fully observe the underlying state of the world, the formalism is generalized into a partially observable Markov decision process (POMDP) and decisions cannot be made based on the current state, but must be made based on the current belief about the state. For example, when the agent's observations consist of images, there are a number of variables which are not exactly known, such as the agent's own velocity, the velocity of objects in the images, the existence of objects outside the field of view of the camera, etc. The agent can, however, infer some of these variables based on the information in the images and, as a result, form beliefs about these variables. Which variables are relevant depends strongly on the task at hand. Therefore, it is inefficient to always attempt to track every possible variable. A possible solution exists in the form of feature learning to extract a smaller size feature space, which can then be used as state space. While deep neural networks have been shown to be rather good at encoding high-dimensional (observation) spaces into low-dimensional (feature) spaces [30], the issue remains that the size of the low-dimensional space must be specified. This space should be small enough to promote generalization and reduce complexity, yet large enough to maintain model accuracy.

In this paper, we focus on pruning the latent space in deep active inference. Active inference is a theory of behavior and learning that has been gaining attention in the last decade [16, 37, 46, 55]. The theory adopts the Bayesian brain hypothesis and, as such, frames sleep as Bayesian model reduction. Recently, methods have been developed through which the latent space is learned by deep neural networks. These methods are known as deep

active inference [62, 8] and have been shown to be able to solve multiple environments [9], i.e. the mountain car problem, the OpenAI Gym car racing environment [7] and a robot navigation environment.

We present two methods for latent space pruning in the deep active inference framework. First, we discuss the off-line algorithm from our earlier work [65], which acts on a model which has previously been trained. Second, we present an on-line algorithm that prunes dimensions on-the-fly during training on sequences of observations and actions. We explore some of the properties of the latter and compare both methods in terms of performance and show that both methods are able to effectively prune dimensions in the latent space. Furthermore, we show that the off-line method requires a longer training time, but maintains performance, while the on-line method requires a shorter training time, but leads to a slight drop in performance.

This work is structured as follows. In section 3.2, we provide an overview of related work. In section 3.3, we briefly summarize (deep) active inference and describe our methods for pruning. We proceed with a description of the environments used in the experiments. In section 3.4, we explain the experiments and display the results, which we discuss in section 3.5. In section 3.6, we conclude this work and examine future prospects.

3.2 Related work

The earliest use of the term 'active inference', as it is used now, in relation to the free energy principle [14] dates back to Friston et al. [21]. Since then, it has been expanded upon in a number of ways and applied in a multitude of scenarios. Beside continued work on behavior and learning [15, 16], active inference has been described as a process theory [17] and as a hierarchical model [23]. Furthermore, active inference has been employed in hermeneutics [20], in a theory of allostasis [3], to augment traditional reinforcement learning approaches [61], and in humanoid robot control [50]. In particular, our methods relate to developments in deep active inference [62, 8, 9], where the goal is to step away from predefined state spaces by learning through artificial neural networks.

Thus far, sleep has yet to be covered extensively in the context of active inference. Hobson and Friston [31] review the purpose of sleep in terms of free energy minimization. Similarly, the premise that the brain's generative model is actively refined during sleep is explored by Hobson et al. [33]. Friston et al. [22] detail the parallels between Bayesian model reduction and certain mechanisms associated with sleep or 'Aha' moments. Specifically, it refers to the removal of redundant connections to minimize complexity. Further, sleep has been tied to concept learning, where Bayesian model

reduction merges different states into one and reduces complexity [56].

In comparison, artificial neural network pruning has a relatively long history. As a means to improve generalization and reduce hardware and storage requirements, LeCun et al. [41] suggested a method for removing unimportant weights from a neural network using second derivatives. In response, Hassibi and Stork [29] proposed a method using the inverse Hessian matrix from training data and structural information of the neural network. In recent years, with the popularity of deep neural networks, the discussion has been reopened. Various techniques have been developed to compress large models and reduce the number of parameters. Gong et al. [25] examine information theoretical vector quantization methods. Han et al. [28] introduced a 3-step method involving pruning, trained quantization and Huffman coding. Other notable contributions include "soft weight-sharing" [63], variational dropout [48], L_0-norm-based methods [44, 43] and Dirichlet pruning [1].

Many dimensionality reduction techniques exist for the purpose of transforming a high-dimensional space into a low-dimensional space. Traditional dimensionality reduction techniques include singular value decomposition (SVD) and the related method principal component analysis (PCA) [51], linear discriminant analysis (LDA) [11], non-negative matrix factorization (NMF) [40], etc. In this paper, we also investigated SVD as state dimensionality reduction technique after training. The downside is that this requires retraining afterwards. A well-known neural network-based method is that of the autoencoder [39], which inspired the more recent variational autoencoder (VAE) [36]. However, also in this case the dimensionality of the latent bottleneck is a hyperparameter that needs to be tuned by the experimenter. Model reduction methods related to Markov decision processes can be classified into three different approaches. Clustering and segmentation approaches attempt to partition the state space into a finite number of partitions, examples include Q-learning with adaptive state segmentation (QLASS) [49] and extended ϵ-reduction for hierarchical reinforcement learning [2]. In our case, observations can contain continuous features, which would require an infinite number of states. State vector transformation approaches project a high-dimensional space onto a low-dimensional space, such as basis iteration for reward based dimensionality reduction [57], locally linear embedding (LLE) [54] and dimensionality reduction by learning an invariant mapping (DrLIM) [26]. Apart from DrLim, these methods require recomputation of the embedding for each unknown datapoint, and rely on predetermined computable distance metrics. DrLim, on the other hand, requires training a neural network. State space reconstruction methods build a low-dimensional representation of the data, e.g. multidimensional scaling [4] and action re-

specting embedding (ARE) [6]. Multidimensional scaling, like PCA and SVD, generates linear embeddings. ARE is not able to embed unknown datapoints. A similar approach to the off-line sleep method performs reduction on the state space in deep Q-networks through PCA [24].

3.3 Methods

3.3.1 Active inference

Consider an agent, natural or artificial, inhabiting an environment, e.g. a living room or a warehouse. Such an agent interacts with its environment in two ways: it can observe the state of the environment and it can perform actions within the environment. For example, a mouse can use its eyes to see where it is and use its mouth to eat cheese, or a roomba can see the shape of the room using its sensors and clean up dirt using its brushes. However, due to limitations on the sensors of the agent, such as noise and range limits, an agent can never know the exact state of the environment, e.g. the mouse cannot see what is going on outside of its field of view and the roomba cannot know the shape of the room if its view is blocked by a large piece of furniture. Therefore, it must always make decisions based on incomplete information. This concept is formalized as a partially observable Markov decision process (POMDP).

Active inference postulates that a natural agent maintains an internal model of the world [16]. That is, an agent holds beliefs about the world. Mathematically, this is represented by a generative model $P(\tilde{\boldsymbol{o}}, \tilde{\boldsymbol{s}}, \pi)$ over possible observations $\boldsymbol{o}$, states $\boldsymbol{s}$ and policies π,

$$P(\tilde{\boldsymbol{o}}, \tilde{\boldsymbol{s}}, \pi) = P(\boldsymbol{s}_0)P(\pi)\prod_{t=1}^{T} P(\boldsymbol{o}_t|\boldsymbol{s}_t)P(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \pi), \tag{3.1}$$

where the tilde notation indicates a variable sequence over time, i.e. $\tilde{x} = (x_1, x_2, x_3, \ldots, x_T)$ and policies are sequences of actions $\boldsymbol{a}_t = \pi(t)$. This generative model is continually updated to include evidence from new observations.

Active inference adheres to the free energy principle in that action selection occurs on the basis of variational free energy [16]. In other words, an agent selects its actions in such a way that the actions minimize a free energy functional

$$F = \mathbb{E}_{Q(\boldsymbol{s},\pi)}[\log Q(\boldsymbol{s}, \pi) - \log P(\tilde{\boldsymbol{o}}, \tilde{\boldsymbol{s}}, \pi)] \tag{3.2}$$

$$= \underbrace{\mathrm{D}_{\mathrm{KL}}(Q(\boldsymbol{s}, \pi) \parallel P(\boldsymbol{s}, \pi))}_{complexity} - \underbrace{\mathbb{E}_{Q(\boldsymbol{s},\pi)}[\log P(\tilde{\boldsymbol{o}}|\tilde{\boldsymbol{s}}, \pi)]}_{accuracy}, \tag{3.3}$$

where Q denotes the approximate posterior which emerges in variational Bayesian methods and D_{KL} denotes the Kullback-Leibler (KL) divergence. Minimizing this functional corresponds to minimizing the complexity of accurate explanations. Particularly, it constitutes a trade-off between model complexity and accuracy.

Crucially, an agent will also aim to minimize its free energy in the future, and hence select policies that it believes will yield a low free energy. This leads to the notion of expected free energy over a certain policy

$$G(\pi) = \sum_{\bar{t}} G(\pi, \bar{t}) \tag{3.4}$$

$$G(\pi, \bar{t}) = \mathbb{E}_{Q(\boldsymbol{o}_{\bar{t}}, \boldsymbol{s}_{\bar{t}}|\pi)}[\log Q(s_{\bar{t}}|\pi) - \log P(\boldsymbol{o}_{\bar{t}}, \boldsymbol{s}_{\bar{t}}|\pi)] \tag{3.5}$$

$$= \underbrace{\mathrm{D}_{\mathrm{KL}}(Q(\boldsymbol{s}_{\bar{t}}|\pi) \parallel P(\boldsymbol{s}_{\bar{t}}))}_{expected\ cost} + \underbrace{\mathbb{E}_{Q(\boldsymbol{s}_{\bar{t}}|\pi)}[H(\log P(\boldsymbol{o}_{\bar{t}}|\boldsymbol{s}_{\bar{t}}))]}_{expected\ ambiguity}, \tag{3.6}$$

where we have used that $Q(\boldsymbol{o}_{\bar{t}}, \boldsymbol{s}_{\bar{t}}|\pi) \approx P(\boldsymbol{o}_{\bar{t}}|\boldsymbol{s}_{\bar{t}})Q(\boldsymbol{s}_{\bar{t}}|\pi)$. Expected free energy again decomposes into two terms: expected cost and expected ambiguity. Expected cost is the divergence between predicted state distribution and the preferred state distribution $P(\boldsymbol{s}_{\bar{t}})$, i.e. the states the agent wants to be in. Expected ambiguity is the accuracy expected under predicted outcomes. This means that an agent will select policies that realize preferred outcomes and resolve ambiguity.

3.3.2 Deep active inference

Belief state spaces in active inference are typically constructed manually, and/or taken to be identical to the state space of the generative process of the modeled environment when known to the experimenter [12]. In other words, every aspect of the state space is specified by hand. However, handcrafting a state space is typically only feasible for low-dimensional problems. The task becomes more difficult for high-dimensional problems, e.g. in the case of high-dimensional sensor input such as RGB images, and problems where the dynamics are difficult to model by hand. For that reason, recent work has focused on learning state space representations using artificial neural networks.

In this work, we adopt the model as implemented by Çatal et al. [9]. With this framework, the generative model in eq. (3.1) is slightly reformulated. The policy π is broken up into actions $\boldsymbol{a}_t$, so that

$$P(\tilde{\boldsymbol{o}}, \tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}) = P(\boldsymbol{s}_0)P(\tilde{\boldsymbol{a}}) \prod_{t=1}^{T} P(\boldsymbol{o}_t|\boldsymbol{s}_t)P(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}). \tag{3.7}$$

Deep neural networks are used to parameterize the approximate posterior $q_{\boldsymbol{\theta}}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_t)$, prior $p_{\boldsymbol{\phi}}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$ and likelihood $p_{\boldsymbol{\psi}}(\boldsymbol{o}_t|\boldsymbol{s}_t)$, where we have introduced the parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$. In combination with eq. (3.2), minimization of free energy is realized using the loss function

$$\mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}) = \mathrm{D}_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_t) \parallel p_{\boldsymbol{\phi}}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})) - \log p_{\boldsymbol{\psi}}(\boldsymbol{o}_t|\boldsymbol{s}_t), \tag{3.8}$$

which corresponds to the free energy of time step t. Here, the expectation from eq. (3.2) is carried out through minibatches as in Kingma and Welling [36] and can therefore be dropped from eq. (3.8). In practice, the free energy is averaged out over all time steps, $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_t$, before adjusting the weights of the neural networks. Importantly, approximate posterior, prior, and likelihood distributions are modeled as multivariate normal distributions. The information flow in the network is visualized in Figure 3.1.

This architecture makes it possible to engage in planning as shown by Çatal et al. [9]. In brief, this requires generating imaginary rollouts using the learned prior model. Trajectories are then scored using expected free energy (as in eq. (3.4)). In practice, planning occurs through iterative Monte Carlo sampling. For each policy π, J state trajectories of K time steps are sampled using the prior model. The expected free energy for each policy is, then, estimated using [19]

$$\begin{aligned}\widehat{G}_t(\pi) = &\sum_{\bar{t}=t+1}^{t+K} \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\mu_{\widehat{s}_{\bar{t}}}, \sigma_{\widehat{s}_{\bar{t}}}) \parallel P(s_{\bar{t}})\right) + \frac{1}{\rho} H\left(\mathcal{N}(\mu_{\widehat{o}_{\bar{t}}}, \sigma_{\widehat{o}_{\bar{t}}})\right) \\ &+ \sum_{\pi'} \sigma\left(-\gamma \widehat{G}_{t+K}\left(\pi'\right)\right) \widehat{G}_{t+K}\left(\pi'\right),\end{aligned} \tag{3.9}$$

where $\mu_{\widehat{s}_{\bar{t}}}$ and $\sigma^2_{\widehat{s}_{\bar{t}}}$ are the batch mean and variance of the state samples $\widehat{s}_{\bar{t}}$, $\mu_{\widehat{o}_{\bar{t}}}$ and $\sigma^2_{\widehat{o}_{\bar{t}}}$ are the batch mean and variance of the corresponding observation samples $\widehat{o}_{\bar{t}} \sim p_\psi(\,\cdot\,|\widehat{s}_{\bar{t}})$, σ is a softmax function, and ρ is a hyperparameter which allows tuning the precision of prior preferences over states in the future. The last term effectively implements a deep tree search over policies, where each path is effectively an accumulation of expected free energy over actions. This construction has the same format as a Bellman recursion, but, in this instance, it is recursion of expected free energy functionals of (Bayesian) beliefs about latent states and policies as opposed to value functions of states and policies.

Despite the parameterization of the posterior and prior, the issue remains that the size of the latent space must be specified. The size of the latent space is an important hyperparameter, since it can have critical consequences regarding model performance and resource usage. The free energy functional (eq. (3.2)) emphasizes the importance of the accuracy-complexity trade-off

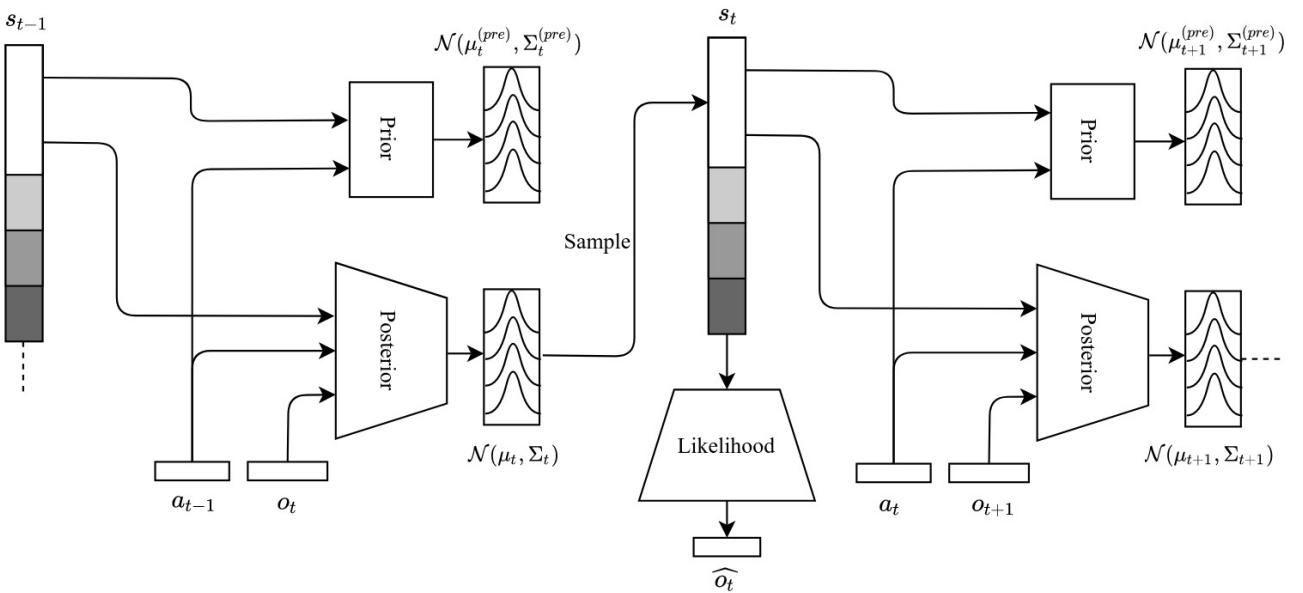


Figure 3.1: Information flow between neural networks in the deep active inference framework. The posterior network takes the current observation $\boldsymbol{o}_t$, previous state $\boldsymbol{s}_{t-1}$ and previous action $\boldsymbol{a}_{t-1}$, and returns a multivariate normal posterior distribution from which the current state is sampled. The prior takes the previous state $\boldsymbol{s}_{t-1}$ and previous action $\boldsymbol{a}_{t-1}$, and returns a multivariate normal prior distribution. The likelihood takes the current state $\boldsymbol{s}_t$ and returns a reconstructed observation $\hat{\boldsymbol{o}}_t$. The process is recurrent and is repeated until the end of the sequence. States $\boldsymbol{s}_{t-1}$ and $\boldsymbol{s}_t$ are shaded to reflect the fact that the dimensionality of the latent space is a hyperparameter and may vary.[a]

[a]The '(pre)' superscripts serve to distinguish the parameters of the prior from the parameters of the posterior.

in active inference. While the latent space dimensionality should be large enough to encode all the relevant information for the task at hand contained in the observations and actions, it should also be small enough to have low memory usage and promote generalization. This is especially important during planning, since this step is done in latent space and requires drawing multiple samples and evaluating multiple paths. The dimensionality is, therefore, a trade-off, and identifying the critical value is not trivial and often depends on the application at hand. Due to this, there is no one-size-fits-all value for the size of the latent space, and a sensible value must be found through different means. In practice, this often implies trial and error or a parameter sweep; both of which are suboptimal, since they require a lot of resources and unnecessary training loops. In the following sections, we describe two methods to resolve the issue.

3.3.3 Off-line sleep

A method that is able to prune the latent space is the sleep algorithm proposed by Wauthier et al. [65]. It allows the agent to initialize the model with a large number of latent space dimensions and, subsequently, sleep until the model can no longer reduce. The approach resembles biological sleep in the sense that it minimizes model complexity post hoc and in the absence of observations. Additionally, in the active inference framework, it can be compared with BMR, which provides an analytic approach to removing redundant model parameters or latent space dimensions. However, in the current implementation, the model must be retrained after each "reduction". Importantly, this is an off-line method, as the reduction happens when the model is not training.

The off-line sleep method is based on singular value decomposition (SVD). Geometrically, singular values in SVD can be understood as the lengths of the semi-axes of the ellipsoid containing the data. The idea is that small singular values correspond to small semi-axes and, therefore, can be pruned, as these dimensions are not informative. Algorithm 3.1 sketches the workings of the method. We start by selecting a large initial number of latent space dimensions ν. This number will be reduced through the ensuing iteration. We train the model with ν latent space dimensions for E epochs using a data set consisting of sequences of actions and observations. After training, we generate $N\nu$ sequences of latent vectors from the model using the data set. From each sequence, we sample a vector to populate a square matrix $\boldsymbol{A}_i$, i.e. each row is a latent space vector, in such a way that we obtain N square matrices. We perform SVD on each $\boldsymbol{A}_i$ and find the number of singular values that are larger than the threshold value α. Then, we compute a new latent space dimensionality by averaging the number of remaining singular

values over the N matrices and round off to the closest integer value. Next, we retrain the model with the new latent space dimensionality and repeat the process until the number of dimensions can no longer be reduced.

Note that algorithm 3.1 does not need to be executed in a single run. When training a model for a certain application, it is possible to pause after having trained the model a first time (line 3) and to continue whenever the application experiences downtime. This allows the model to be used from the start and be optimized in subsequent sleep runs.

Crucially, SVD does not allow one to know *which* dimensions can be pruned at each iteration. The method only indicates *how many* dimensions are informative. Geometrically, in general, the semi-axes of the earlier-mentioned ellipsoid are rotated with respect to the basis vectors of the latent space. As a result, singular values do not correspond to specific dimensions of the latent space and it is not possible to distinguish individual latent dimensions to prune. For this reason, retraining is a necessity. To alleviate this drawback, we now present an on-line method, which optimizes the number of latent dimensions as part of the training process.

3.3.4 On-line sleep

In response to the problem that the model must be retrained in off-line sleep, we present an on-line method for latent space pruning. In this method, weights are pruned on-the-fly during training. This type of model reduction is very similar to reflection, where the agent is allowed to reflect on its understanding of the observations during training. This results in "aha"

Algorithm 3.1: Off-line sleep on sequences of actions and observations

input : Data set $\mathcal{D} = \{(o_1, a_1), \ldots, (o_d, a_d)\}$
Number of repetitions N
Threshold α
output : A pruned model

1 select initial latent space dimensionality ν
2 **repeat**
3 train model with ν latent space dimensions for E epochs
4 generate $N\nu$ sequences of latent vectors from model
5 populate N square matrices $\boldsymbol{A}_i$ by sampling one state vector from each sequence
6 perform SVD on all $\boldsymbol{A}_i$ and apply threshold α to singular values
7 $\nu \leftarrow$ rounded average of remaining number of singular values
8 **until** ν *no longer decreases*

moments: in a sense, moments where the agent realizes the world can be explained more simply. As we will show, this can be seen as sudden jumps in latent space dimensionality.

In brief, in order to reduce the number of states, we gate each state dimension with a gate parameter that follows a Bernoulli distribution. During optimization, we try to close as many of these gates as possible through L_0 regularization. By co-optimizing the Bernoulli parameters in this manner, the model can learn how many states are required to solve a given task. At first glance, this poses two issues: first, the Bernoulli parameters make the gradient intractable and, second, without some way of limiting the reduction, the model will simply try to reduce the dimensionality to zero. The first issue can be resolved by making use of the Augment-REINFORCE-Merge (ARM) gradient estimator [66]. The second issue can be resolved through generalized ELBO with constraint optimization (GECO) [53], as shown by De Boom et al. [13]. GECO introduces a Lagrange multiplier λ on the accuracy and a threshold hyperparameter τ on accuracy which controls λ. The accuracy term in the loss function (second term in eq. (3.8)) will receive more weight during optimization as long as the desired level of accuracy has not been reached. Once the threshold has been reached, more weight will be given to the complexity term. In other words, the model must reach a certain level of accuracy before it can reduce its complexity.

One of the key features of working in the active inference framework is that data sets consist of sequences of actions and observations. This has a number of consequences for the implementation of the on-line sleep algorithm. For instance, whereas in a variational autoencoder (VAE) [36] as demonstrated by De Boom et al. [13], the gates are sampled with each forward pass through the posterior network, in deep active inference, the gates remain unchanged with each pass through the posterior as long as they relate to the same sequence. In other words, the gates are sampled once at the beginning of the sequence and are kept constant throughout the rest of the sequence.

Furthermore, without accounting for sequence length, loss values would increase with the length of the sequence. Therefore, it is necessary to average over sequence length in both the reconstruction error and KL divergence to compare sequences of different length. A direct result is that the interpretation of the threshold τ changes slightly, as it is now a threshold on the time-averaged reconstruction error.

Finally, note that the prior in a VAE is a simple multivariate standard normal, while in deep active inference, the prior is parameterized. Consequently, the posterior is no longer "pulled" towards a standard normal, and the KL divergence must be computed between the posterior and prior

networks.

To summarize, the loss function at time step t becomes

$$\begin{aligned}\mathcal{L}_t(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}) = \\ \frac{1}{\sum_i z_i} \mathrm{D}_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \odot \boldsymbol{z} | \boldsymbol{s}_{t-1} \odot \boldsymbol{z}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_t) \parallel p_{\boldsymbol{\phi}}(\boldsymbol{s}_t \odot \boldsymbol{z} | \boldsymbol{s}_{t-1} \odot \boldsymbol{z}, \boldsymbol{a}_{t-1})) \\ - \lambda \left(- \log p_{\boldsymbol{\psi}}(\boldsymbol{o}_t | \boldsymbol{s}_t \odot \boldsymbol{z}) - \tau \right),\end{aligned} \tag{3.10}$$

where the gates $\boldsymbol{z} = \mathbf{1}_{[\boldsymbol{u} < S(\boldsymbol{\zeta})]}$ with $\boldsymbol{u} \sim \prod_{i=1}^{n} \mathcal{U}_{[0,1]}$ are sampled once per sequence, $S(\zeta_i)$ correspond to the Bernoulli parameters, λ is the Lagrange multiplier and τ is the accuracy threshold. Further, $\odot$ denotes the Hadamard product and $S(x) = (1 + \exp(-kx))^{-1}$ is the sigmoid function with parameter k, where we set $k = 7$ as in Li and Ji [43]. We, once again, average over the sequence length and add the L_0 regularization term, to obtain the loss function

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_t + \sum_{i=1}^{n} S(\zeta_i). \tag{3.11}$$

Note that, similar to De Boom et al. [13], we average the KL term over the number of open gates to ensure that having more states does not lead to a higher KL term, as this would lead to an additional influence on the dimensionality. Subsequently, the gradient for the gate parameters ζ_i becomes

$$\begin{aligned}\nabla_{\boldsymbol{\zeta}}^{\mathrm{ARM}} \mathcal{L} = \frac{\lambda}{T} \sum_{t=1}^{T} \left(- \log p_{\boldsymbol{\psi}}(\boldsymbol{o}_t | \boldsymbol{s}_t \odot \bar{\boldsymbol{z}}) + \log p_{\boldsymbol{\psi}}(\boldsymbol{o}_t | \boldsymbol{s}_t \odot \boldsymbol{z}) \right) \left(\boldsymbol{u} - \frac{1}{2} \right) \\ + \sum_{i=1}^{n} \nabla_{\zeta_i} S(\zeta_i),\end{aligned} \tag{3.12}$$

where $\bar{\boldsymbol{z}} = \mathbf{1}_{[\boldsymbol{u} > S(\boldsymbol{\zeta})]}$.

Algorithm 3.2 sketches the workings of the on-line sleep method. As a result of gating and GECO, pruning of latent dimensions occurs within a single training loop and does not require multiple training loops. Initially, the model starts out with 95% of its gates open. It is then trained using eq. (3.11) *without regularization term* until the reconstruction error (accuracy) reaches the threshold value τ. Once the threshold has been reached, gates can be opened and closed. Practically, this means that, at this point, we start computing the gradient in eq. (3.12) and add the regularization term in eq. (3.11). Training thereupon continues until the model has trained a total of E epochs.

Algorithm 3.2: On-line sleep on sequences of actions and observations

input : Data set $\mathcal{D} = \{(o_1, a_1), \ldots, (o_d, a_d)\}$
output : A pruned model

initialize model with 95% of gates open
repeat
  optimize loss function without regularization term
until *threshold* τ *reached*
repeat
  compute ARM gradient for gate parameters (eq. (3.12))
  optimize loss function with regularization term (eq. (3.11))
until *trained for a total of* E *epochs*

3.4 Results

3.4.1 Environments

The methods presented in this paper are evaluated using a number of different environments. The environments were selected to include a varying degree of difficulty. These include: the mountain car environment (Figure 3.2**A**), the car racing environment (Figure 3.2**B**) and a robot navigation environment in the lab on a turtlebot (Figure 3.2**C**). Here, the mountain car provides simple 1D observations, while the other environments provide pixel-based observations. Furthermore, while the mountain car and car racing environments are simulations, the robot navigation environment is real-life.

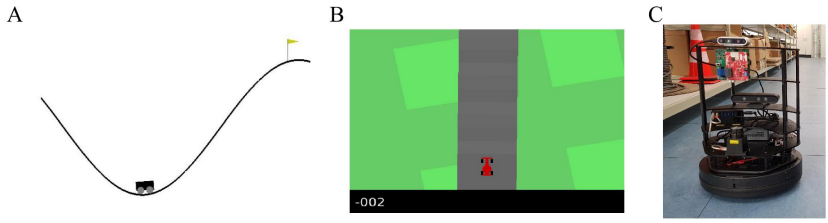


Figure 3.2: **(A)** In the mountain car environment a car can throttle left and right and needs to reach the hill top. **(B)** The carracer environment provides top-down pixel observations to race a car over a circuit. **(C)** The robot navigation scenario contains first person camera images and velocity control actions for a turtlebot driving around in a lab environment.

3.4.1.1 Mountain car

The mountain car environment is a classic, simple 1D environment. It consists of an underpowered car that starts in a valley and must drive up a steep mountain. Importantly, the car cannot drive up the mountain in one go and must build up momentum to reach the top of the mountain. The only actions it has, are accelerate left or right, or do nothing.

The original OpenAI gym environment [7] returns observations with the position and velocity of the car. In our experiments, we employed a modified version in which only the position can be observed and noise is added to the observations, turning the problem into a POMDP. Due to the simplicity of the environment, we expect to only need 2 latent space dimensions to solve the problem, i.e. position and velocity. This allows for efficient evaluation of the proposed algorithms, since the expected dimensionality is already known.

The data set is generated before training using a random agent and consists of 100 sequences of length 200. When passing the data to the model during training, we randomly extract a subsequence of 100 consecutive steps out of each sequence. That way the model learns the dynamics of the environment. Similarly, the test data set was generated using a random agent and consists of 32 sequences of length 200.

3.4.1.2 Car racing

The car racing environment [7] consists of a top-down view of a race car on a racetrack. The goal is for the race car to remain on the racetrack. In our implementation, the car can accelerate and turn right or left. The environment returns observations as 96×96 pixel images. In other words, observations are high-dimensional and the dynamics are more complicated than in the mountain car environment. In this case, the optimal number of latent dimensions cannot be known in advance.

In this case, the data set consists of 7 prerecorded sequences of differing lengths: 568, 723, 482, 521, 669, 647 and 676. When passing the data to the model during training, we randomly extract a subsequence of 15 consecutive steps out of each sequence. The test data set consists of a prerecorded sequence of length 635.

3.4.1.3 Robot navigation

Beside the previous environments, experiments were done on the mobile robot data set provided by Çatal et al. [10]. The data set consists of camera, lidar and radar recordings of the industrial IoT lab at UGent. For our purposes, we only require the camera recordings which contain sequences of

240×320 pixel images. Again, this is a high-dimensional environment with complicated dynamics and the optimal number of latent dimensions is not known beforehand. Moreover, the data originate from the real world, albeit a controlled environment, and are not obtained from simulation.

3.4.2 Mountain car

By virtue of the simplicity of the mountain car environment, we use this environment to inspect whether our methods are able to converge to the optimal dimensionality, i.e. 2 dimensions. Effective reduction, for both off-line and on-line sleep, depends on the chosen threshold, α and τ, respectively.

Artificial neural networks are instantiated as in Çatal et al. [9]. That is, prior, posterior and likelihood networks consist of fully connected neural networks with two hidden layers which each contain 20 neurons. Importantly, the size of the output layer of the prior and posterior networks equals the size of the latent space, and is therefore selected by the model reduction algorithm. This also applies to the input layer of the likelihood network. Baseline and off-line sleep runs are trained with a batch size of 64 sequences of length 100 using the loss function in eq. (3.8) with an additional factor 2 on the complexity term and an additional regularization term with weight 0.001 comprised of the KL divergence between the posterior and a standard normal distribution. On-line sleep runs are trained with a batch size of 32 sequences of length 100 using the loss function in eq. (3.11). All minimization occurs through the Adam optimizer with learning rate 0.001. Additional

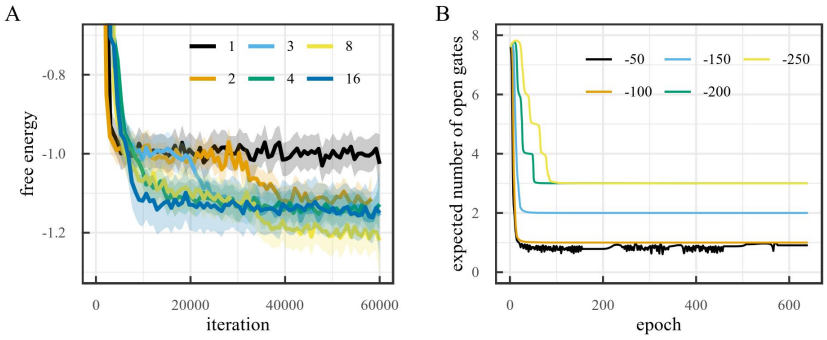


Figure 3.3: **(A)** Free energy during training on the mountain car environment for different latent space sizes (LOESS smoothed, span 0.02). **(B)** Number of dimensions obtained with on-line sleep on the mountain car environment for different thresholds with initial number of dimensions 8.

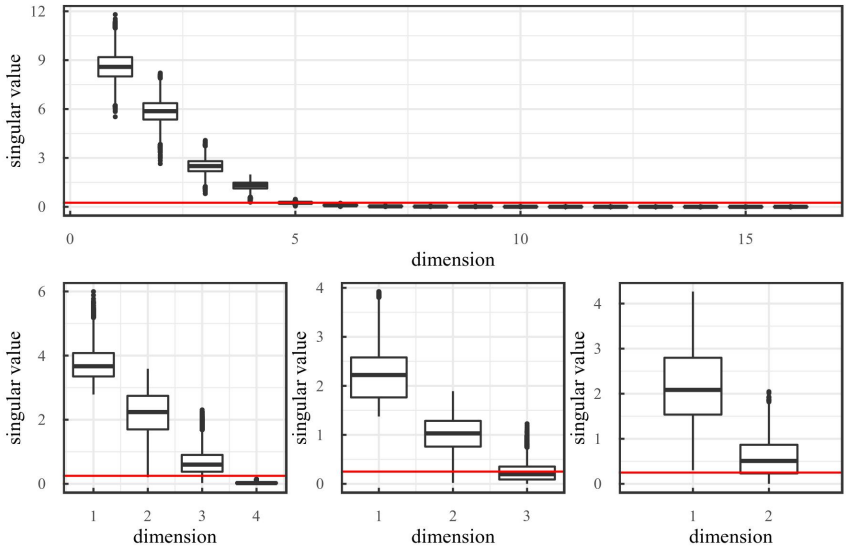


Figure 3.4: Boxplots of singular values after off-line sleep on the mountain car environment for different dimensionality. The red line indicates a threshold $\alpha = 0.25$. These figures show that sleeping with the chosen threshold would result in the reduction: $16 \rightarrow 4 \rightarrow 3 \rightarrow 2$.

details are described in Çatal et al. [9].

As mentioned in section 3.4.1.1, the environment can be solved using two latent dimensions i.e. position and velocity. As a result, we expect to find a lower free energy when the model has more than one latent dimensions, since a latent space with only one dimension can only encode the position, but never the velocity, while a latent space with more than one dimension can encode position, velocity and (possibly) noise on the observations. Figure 3.3**A** shows the free energy during training for the mountain car environment for different latent space dimensionalities. The figure shows that, after 60000 training iterations, the free energy for models with one latent dimension is indeed significantly higher than for models with more than one dimension. Note that the free energy for models with more than one latent dimension does not decrease significantly when increasing the number of dimensions. Small differences in free energy are likely due to differences in how noise is encoded. If the noise characteristics are better encoded, this can lead to a slightly lower free energy.

Figure 3.4 shows singular values from off-line sleep after training on the mountain car environment. The figure also shows that performing off-line sleep with threshold $\alpha = 0.25$ results in a reduction from 16 dimensions to 2

dimensions through $16 \rightarrow 4 \rightarrow 3 \rightarrow 2$.

Figure 3.3**B** shows the number of dimensions obtained for the mountain car environment for different values of the threshold τ when the initial number of dimensions is 8. It also shows how final dimensionality depends on the threshold. This makes sense, when the threshold is set too low, the model will attempt to obtain a very high accuracy by modeling noise, increasing the dimensionality. When the threshold is set too high, not enough information will be encoded in the latent space and the dimensionality will continue to reduce. Note that for $\tau = -50$, the algorithm continues to attempt to reduce the dimensionality to 0, but since this would lead to a very low accuracy, the dimensionality shoots back up to 1.

3.4.3 Car racing

Since the car racing environment provides high-dimensional observations, it is a useful environment to assess the model's performance. We assess the evolution of the model's performance both during and after training.

Again, neural networks are instantiated as in Çatal et al. [9]. The posterior network consists of a fully connected neural network, where feature vectors are extracted from observations by convolutional layers and concatenated to action and state vectors. Consequently, the likelihood network consists of a deconvolutional neural network. Contrary to the architecture for the car racing environment in Çatal et al. [9], the prior consists of an LSTM with 128 features to allow for more temporal depth in the prior transition model. As in the case with the mountain car, the size of the output layer of the prior and posterior networks and the size of the input layer of the likelihood network equal the size of the latent space. More details on the network and architecture can be found in Supplementary Material.

To assess model performance, the preferred state is defined by taking an observation in which the car is located in the middle of the track and translating it to state space (see Figure 3.5). In practice, the environment is

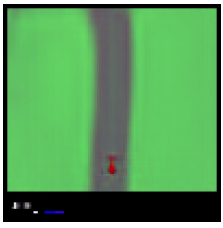


Figure 3.5: Reconstruction of preferred state used for car racing environment evaluations.

always initialized with the car in the middle of the road, therefore, the first frame can be taken as preferred observation. Performance is then evaluated through an active inference agent with this preferred state and $\rho = 0.0001$.

Additional details on neural network architecture and model performance assessment are described in Çatal et al. [9].

3.4.3.1 Off-line sleep

Figure 3.6 shows the free energy during training for the car racing environment. To make sure the latent space retains all information after pruning, it is important that the free energy remain minimal. Note that the free energy does not decrease for latent space dimensionality larger than 5. That is, adding dimensions, when the dimensionality is larger than or equal to 5, does not reduce the free energy. Inversely, removing dimensions, when the dimensionality is less than or equal to 5, increases the free energy. This suggests that the optimal latent space dimensionality is 5, since this is the smallest value for which the free energy remains minimal. As a result, this is the critical value which the algorithm should attempt to achieve.

Figure 3.7 demonstrates the workings of the off-line sleep algorithm. After training for 2400 epochs, the model performs SVD in an attempt to reduce the dimensionality. If the dimensionality can be reduced, the excessive dimensions are pruned and the model retrains with a reduced number of dimensions, otherwise training stops.

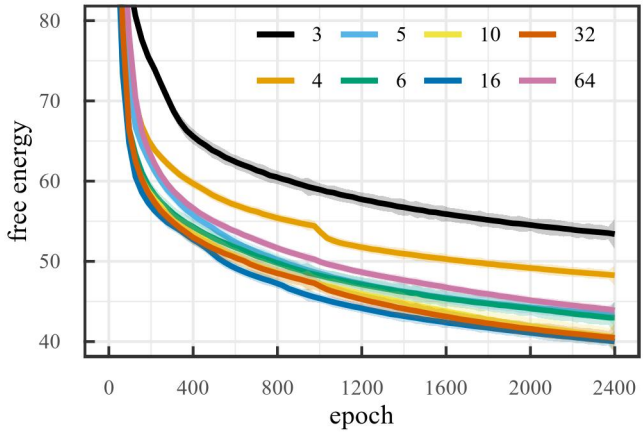


Figure 3.6: Free energy during training on the car racing environment for baselines runs with different latent space sizes (LOESS smoothed, span 0.02).

3.4.3.2 On-line sleep

Since the regularization term consists of the sum of the Bernoulli parameters of each gate, this term also indicates how many gates are expected to remain open. Therefore, we use this number to indicate the latent space dimensionality in the on-line sleep case.

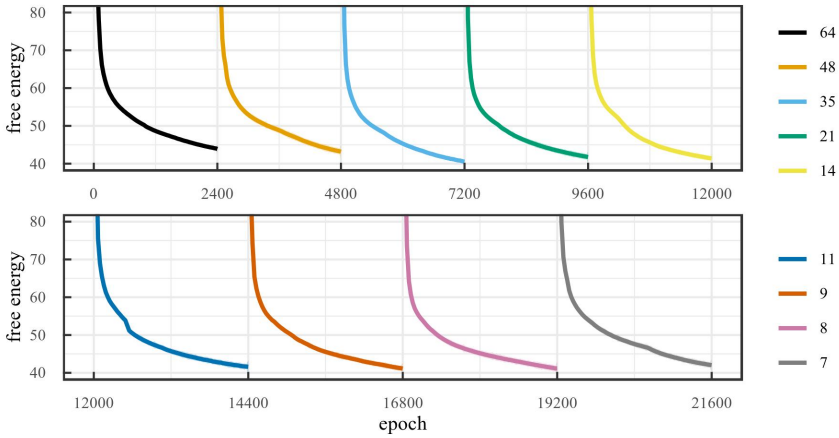


Figure 3.7: Free energy during training over an off-line sleep process with $\alpha = 0.1$ for the car racing environment (LOESS smoothed, span 0.02). This run required 9 cycles of off-line sleep and converged to 7 dimensions, where the model was trained for 2400 epochs each cycle.

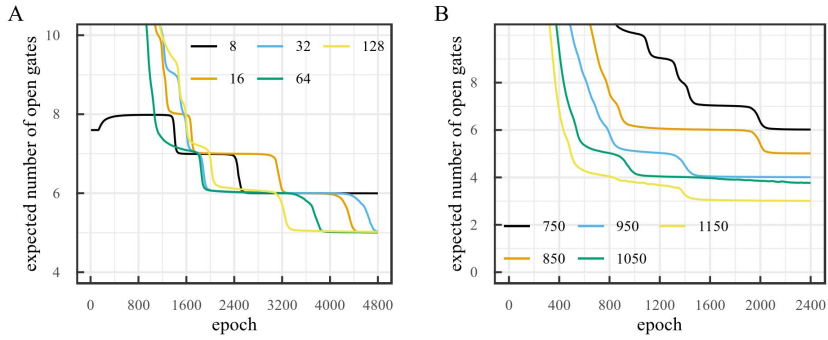


Figure 3.8: Number of dimensions obtained with on-line sleep on the car racing environment **(A)** for different initial number of states with threshold $\tau = 800$ over 4800 epochs and **(B)** for different thresholds with initial number of dimensions 32 over 2400 epochs.

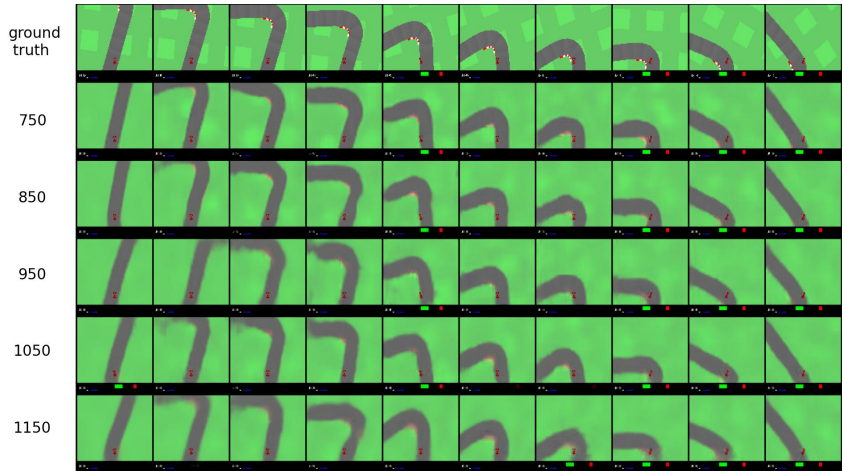


Figure 3.9: Reconstructed observations from the car racing environment after on-line sleep (32 initial dimensions, 2400 epochs) with varying threshold τ (from top to bottom: ground truth, 750, 850, 950, 1050, 1150).

Since initialization is important in the convergence of neural networks, it is important to assess the impact of different initial values on the final dimensionality. Two hyperparameters are crucial here: initial dimensionality and threshold value τ. Firstly, we investigate the effect of initial dimensionality on final dimensionality when τ is fixed. Figure 3.8**A** shows the number of dimensions obtained for the car racing environment for different initial dimensionalities when the threshold is fixed to 800. The figure shows that after 4800 epochs the number of gates that remain open is 5 for most runs when $\tau = 800$. This suggests that final dimensionality is largely independent of initial dimensionality. The chosen threshold was based on the reconstruction error obtained after training the model without pruning. For this, we determined at which point the reconstructions became good for the human eye. It is likely that the reconstructions are good enough to evaluate policies before they become good for the human eye. However, the chosen threshold works well for our intents and purposes.

Secondly, we investigate the effect of the threshold value τ on final dimensionality. Figure 3.8**B** shows the number of dimensions obtained for the car racing environment for different values of the threshold τ when the initial number of dimensions is 32. This figure shows the dependency of the number of dimensions on the reconstruction threshold. A lower threshold leads to a higher dimensionality and vice-versa. Furthermore, Figure 3.9 illustrates how the reconstructed observations change depending on threshold τ. A higher threshold tends to lead to a slightly blurrier image and less

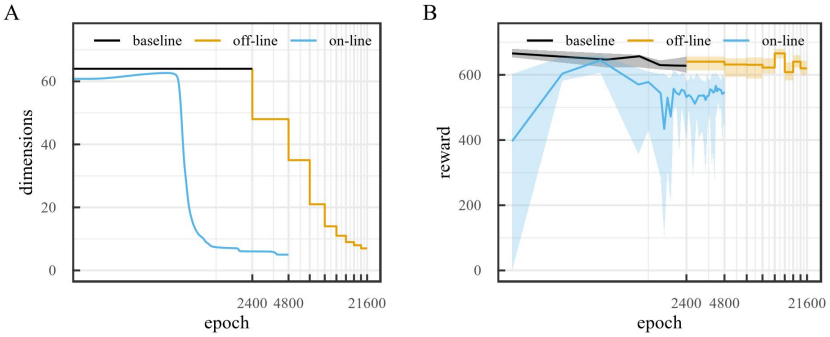


Figure 3.10: Evolution of **(A)** number of states and **(B)** median reward obtained (over 20 rollouts) over number of epochs for baseline, off-line sleep ($\alpha = 0.1$) and on-line sleep ($\tau = 800$) runs with 64 states initial latent space dimensions. Error bars show 25% and 75% quantiles. For off-line sleep, each step of the reward curve shows the median reward obtained at the end of the training run for the corresponding dimensionality, i.e. after training for 2400 epochs at the corresponding dimensionality.

accurate image features.

To evaluate how the model performs, we used the reward obtained from the car racing environment. This reward increases with $1000/R$ with every new track tile visited, where R is the total number of track tiles, and decreases by 0.1 per frame. If the car exits the playing field, it loses 1000 reward per frame instead. If the reward drops below 0, we consider the rollout as failed and set the reward to 0. We used a rollout length of 300 time steps. Figure 3.10**A** compares the evolution of the number of latent space states during baseline runs, off-line sleep with $\alpha = 0.1$ and on-line sleep with $\tau = 800$ for 64 initial latent space dimensions. Both off-line and on-line sleep are able to effectively reduce the dimensionality of the latent space. However, on-line sleep manages to reduce the latent space to 5 dimensions over 4800 epochs, while off-line sleep requires 21600 epochs to reduce to 7 dimensions. Furthermore, Figure 3.10**B** compares the evolution of the reward during the same runs. Importantly, for off-line sleep, each step of the reward curve shows the average reward obtained at the end of the training run for the corresponding dimensionality, i.e. after training for 2400 epochs at the corresponding dimensionality. The performance of the off-line sleep run remains in range of the baseline run. This makes sense, as off-line sleep can simply be seen as a concatenation of baseline runs with decreasing dimensionalities. In comparison, the performance of the on-line sleep run diminishes slightly during training. The lower performance can be

attributed to the fact that the model needs to adapt to less dimensions each time the dimensionality reduces.

3.4.4 Robot navigation

To score performance in this real-world application, we compare the reconstructed observations from full (baseline) and reduced models as a proxy for model evidence (c.f., classification accuracy based upon posterior predictive densities).

The architecture is similar to the one used for the car racing environment. Details on the layers can be found in the Supplementary Material. Again, the size of the output layer of the prior and posterior networks and the size of the input layer of the likelihood network equal the size of the latent space. Models are trained with a batch size of 8 sequences of length 10. Loss function minimization occurs through the Adam optimizer with learning rate 0.0001.

Figure 3.11 shows the number of dimensions obtained with on-line sleep on the robot navigation environment with $\tau = 12000$ and an initial dimensionality of 128. Over 600 epochs, the number of dimensions reduces to 15. Again, the threshold was based on the reconstruction error obtained after training the model without pruning. We determined at which point the reconstructions became good for the human eye.

Reconstructed observations are shown in Figure 3.12. Here, the top

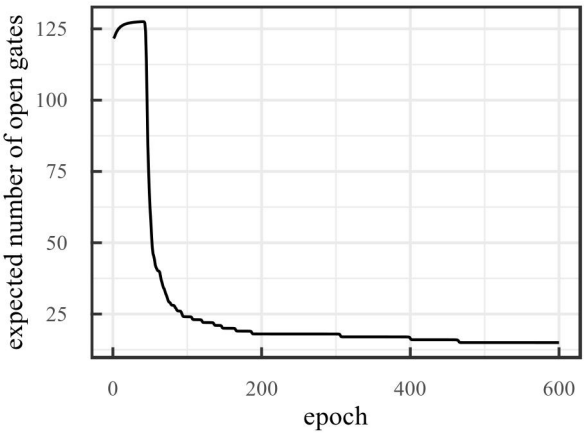


Figure 3.11: Number of dimensions obtained with on-line sleep on the robot navigation environment with threshold $\tau = 12000$. After 600 epochs, reduction stopped at 15 dimensions.

Figure 3.12: Reconstructed observations on the robot navigation environment. The top row contains ground truth observations, the second row contains reconstructions for a model trained with 128 latent space dimensions (achieving a reconstruction error of 9460), and the third row shows reconstructions for a model trained with on-line sleep with $\tau = 12000$ (see Figure 3.11).

row shows ground truth observations, the second row shows reconstructions for a model trained with 128 latent space dimensions (which reaches a reconstruction error of 9460), and the third row shows reconstructions for the model trained with on-line sleep (at $\tau = 12000$) displayed in Figure 3.11. No significant differences are visible, suggesting that while on-line sleep is able to reduce the number of latent space dimensions, it does not significantly reduce the quality of the reconstructions. In other words, on-line sleep enables us to obtain a quality of reconstructions similar to baseline using fewer latent space dimensions.

3.5 Discussion

The results show that both methods are able to effectively prune dimensions in the latent space of a deep active inference model. However, there is an important trade-off. While the off-line sleep method requires multiple training runs to converge, on-line sleep only requires a single training run. On the other hand, while the performance can reduce slightly during on-line sleep, each off-line sleep cycle trains for a specific latent space dimensionality and therefore maintains performance.

Performance evaluations show that there is indeed a slight decline in

performance when using on-line sleep compared to baseline training runs. This decline can be attributed to the small number of epochs that the model trains for a certain dimensionality. While baseline runs have 2400 epochs to optimize for a given dimensionality, on-line sleep runs continually reduce dimensionality and, therefore, must continually adjust to the dimensionality. This could potentially be resolved by, once again, training without regularization and without updating the gate parameters after the model has finished training.

Results from on-line sleep show that the final dimensionality is robust with respect to initial dimensionality settings of the algorithm. Any sufficiently large initial state space dimensionality will lead to the same final dimensionality, given that the model has trained long enough. This result makes sense, since the initial dimensionality does not determine which or how much information should be encoded in the latent space. Instead, it simply provides the model with a larger initial latent space in which to encode information. Excess dimensions are then pruned by the algorithm. Conversely, final dimensionality does depend on the chosen threshold value. Indeed, the reconstruction threshold determines how much detail from the observations should be retained. A low threshold ensures more detail is retained, which in turn leads to more dimensions, and vice-versa. Based on these results, a possible course of action would be to first train a model without sleep and a sufficiently large dimensionality in order to find a good reconstruction threshold, and finally, train a model with on-line sleep using the previously obtained threshold and a sufficiently large dimensionality.

Moreover, in this work, the measure for adapting the Lagrange multiplier is the level of accuracy or reconstruction error. It may be possible to use other measures. For example, in the case of the car racing environment, a more sensible approach to threshold selection could be to adapt the threshold based on the performance of the model, in such a way that the model will only reduce if the performance can be maintained. That is, as long as the performance does not reach the same level as before reduction, it should not reduce further. This is a research route that will be explored in future work.

The number of epochs is also important for the final dimensionality, because the dimensionality depends on the number of epochs. One could inquire until what point models should be trained. In the end, the purpose of on-line sleep is to be able to learn a task and prune dimensions simultaneously so as not to require spending (much) more time finding an optimal latent space dimensionality through parameters sweeps or off-line sleep. Observing that final dimensionality is largely independent of initial dimensionality, it may be good practice to spend around the same number of epochs as a baseline run would require to converge.

Additionally, it is important to discuss the role that the data (and the environment) play. In any type of learning, the data are an integral part of the final result. For example, in deep learning applications, overrepresentation of a certain aspect of the data can lead to biases in the model. In our case, the data have an effect on the final dimensionality. For the car racing example, final dimensionality may depend on the complexity of the observations given to the model. Observations that contain more complex tracks could lead to more dimensions being necessary.

Off-line sleep can be compared to more traditional dimensionality reduction techniques, e.g. principal component analysis (PCA), non-negative matrix factorization (NMF), etc., in the sense that it reduces the state space *post hoc* by evaluating linear combinations and correlations between dimensions. Unfortunately, this method can only be applied after the model has been trained, which can cause the reduction process to take a considerable amount of time, since the model must be retrained. On-line sleep, instead, is able to perform model reduction while the model is training. This removes the need to retrain.

Reduction methods for MDPs mentioned in the literature are typically designed to maintain the structure of the state space after reduction. However, since our methods learn the state space through neural networks, it is not necessary to maintain this structure. Indeed, in the case of off-line sleep, a simple SVD suffices, since the model must retrain after reduction. In fact, any method that removes dimensions from the state space, i.e. prunes connections in the neural network, requires that the network be retrained (or further trained), which defeats the purpose of maintaining structure. In addition, since the structure of the state space changes during training, most methods cannot be used during training and must be applied post hoc. In the case of on-line sleep, the model is trained during reduction, which means it adjusts itself to the number of remaining dimensions on-the-fly.

Crucially, the main increase in difficulty in the environments throughout this work consists of the complexity of the observations. The complexity increased from 1-dimensional observations for the mountain car to high-dimensional rendered images for car racing, and high-dimensional real world images for the robot. In addition to complexity in observation space, one could consider increasingly complex tasks that need to be executed, i.e. increasing action spaces. In this case, an optimal dimensionality and structure of the state (and action) space could have an effect on the agent performance, for example in the context of hierarchical reinforcement learning [2]. However, this comparison will be further explored in future work.

Finally, in this work, we investigate the latent space size in relation to active inference. However, deep active inference introduces neural networks,

where the size of each layer must also be specified. In the future, we will work on applying the gating mechanism to layers in the neural network.

Although not pursued in this work, the use of Bayesian model reduction also finesses the problem of sharp minima in neural networks by automatically minimising model complexity. This is a subtle issue in the sense that most approaches to eluding local minima involve adding extra parameters to destroy fixed points on (e.g. variational free) energy landscapes. However, having escaped local minima, it is then necessary to prune the network to ensure generalisation.

3.5.1 Bayesian model reduction

Bayesian model reduction is a relatively new procedure that can be regarded as a special case of Bayesian model selection; namely, selecting the model with the greatest marginal likelihood or model evidence. The particular benefit of BMR is that the evidence for a reduced model can be evaluated analytically, given the evidence and posteriors of a parent or full model. This means that models can be scored in terms of their evidence quickly and efficiently, without any need for retraining, having learned the full model. This efficient form of model selection rests on casting reduced models as models in which certain parameters (i.e. connections) are removed using precise shrinkage priors. It can be regarded as a generalisation of the Savage-Dickie ratio for automatic model selection (c.f., automatic relevance detection). Commonly used equations for BMR can be found in [18] for a variety of distributions over model parameters (ranging from Gaussian densities to Dirichlet distributions).

To leverage the efficiency of BMR, it is necessary to have a posterior over the model parameters that need pruning. This presents a slight problem for deep active inference, because we have replaced the parameters of a generative model with neural networks with learnable weights that are treated as point estimators, with no uncertainty. In vanilla treatments of active inference, the approximate posterior covers latent states, policies and parameters. Conversely, in deep active inference we only have posteriors over the latent states and policies. To finesse this problem, we have equipped the parameters with switching or gating variables that play the role of sufficient statistics of a simple posterior over model parameters (i.e. the connection weights).

We can now conjecture that the reduced variational free energy of the implicit posterior over model parameters, under different reduced models, can be approximated with the GECO; namely, a generalised ELBO with constraint optimisation. If this conjecture is true, it suggests that the online sleep procedure described above is, the Bayes optimal way to prune networks.

As with all BMR procedures, there remains the delicate issue of how much data to acquire (i.e. the duration of the training periods), before running BMR. This is usually dictated by the trade-off mentioned above, in terms of the speed of optimisation, relative to the amount of time it takes. An upper limit on the amount of training or data is clearly provided by the times over which the model (i.e. generative process) does not change. Heuristically, a lower limit depends on how much evidence is required to commit to a simpler model.

3.6 Conclusion

In this paper, we have examined and compared two methods that are able to reduce the latent space in deep active inference models: an off-line method that reduces a trained model post hoc and reapplied multiple times, and an on-line method which applies a gate to each dimension in the latent space and can prune dimensions on-the-fly during training.

Experiments on the mountain car environment showed that both methods were able to achieve the optimal number of dimensions depending on the chosen threshold. Results from the car racing environment showed that the off-line method was able to retain performance, but used approximately five times more epochs to converge, while the on-line method showed slightly lower performance. Finally, the robot navigation environment showed that even in real-world settings on-line sleep was able to reduce dimensionality.

3.6.1 Future work

Aside from the ideas already discussed in section 3.5, we would like to explore one more point of interest: model expansion. This work showed that it is possible to reduce latent space dimensionality. A question that arises is what happens when the agent obtains entirely new observations. One would expect the state space to expand, since it must accommodate for new information. Future work will call attention to this topic.

3.7 Additional Requirements

Conflict of Interest Statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Author Contributions

SW and CD worked out the mathematical basis for the experiments. SW, CD, TV and BD conceived the experiments. SW performed the experiments. All authors supervised the experiments. All authors contributed to the article and approved the submission.

Funding

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program. OÇ was funded by a Ph.D. grant of the Flanders Research Foundation (FWO).

Data Availability Statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

3.8 References

[1] Adamczewski, K. and Park, M., "Dirichlet Pruning for Convolutional Neural Networks," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, Banerjee, A. and Fukumizu, K., eds., vol. 130 of *Proceedings of Machine Learning Research*, pp. 3637–3645. PMLR, 13–15 apr 2021.

[2] Asadi, M. and Huber, M., "State Space Reduction For Hierarchical Reinforcement Learning," in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, Barr, V. and Markov, Z., eds., pp. 509–514. AAAI Press, May 2004.

[3] Barrett, L. F., Quigley, K. S., and Hamilton, P., "An active inference theory of allostasis and interoception in depression," *Philosophical Transactions of the Royal Society B: Biological Sciences* **371** no. 1708, (Nov. 2016) 20160011.

[4] Borg, I. and Groenen, P., *Modern Multidimensional Scaling: Theory and Applications*. Springer Series in Statistics. Springer New York, 2 ed., 2005.

[5] Born, J. and Wilhelm, I., "System consolidation of memory during sleep," *Psychological Research* **76** no. 2, (May 2011) 192–203.

[6] Bowling, M., Ghodsi, A., and Wilkinson, D., "Action respecting embedding," in *Proceedings of the 22nd international conference on Machine learning - ICML '05*, ICML '05, pp. 65–72. ACM Press, 2005.

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym." 2016. `arXiv:1606.01540 [cs.LG]`.

[8] Çatal, O., Nauta, J., Verbelen, T., Simoens, P., and Dhoedt, B., "Bayesian policy selection using active inference." 2019. `arXiv:1904.08149 [cs.LG]`.

[9] Çatal, O., Wauthier, S., De Boom, C., Verbelen, T., and Dhoedt, B., "Learning Generative State Space Models for Active Inference," *Frontiers in Computational Neuroscience* **14** (Nov. 2020) 103.

[10] Çatal, O., Jansen, W., Verbelen, T., Dhoedt, B., and Steckel, J., "LatentSLAM: unsupervised multi-sensor representation learning for localization and mapping." 2021. `arXiv:2105.03265 [cs.RO]`.

[11] Cohen, P., Cohen, P., West, S. G., and Aiken, L. S., *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates Publishers, 2003.

[12] Da Costa, L., Parr, T., Sajid, N., Veselic, S., Neacsu, V., and Friston, K., "Active inference on discrete state-spaces: A synthesis," *Journal of Mathematical Psychology* **99** (Dec. 2020) 102447.

[13] De Boom, C., Wauthier, S., Verbelen, T., and Dhoedt, B., "Dynamic Narrowing of VAE Bottlenecks Using GECO and L_0 Regularization." 2021. `arXiv:2003.10901 [cs.LG]`.

[14] Friston, K., Kilner, J., and Harrison, L., "A free energy principle for the brain," *Journal of Physiology-Paris* **100** no. 1–3, (July 2006) 70–87.

[15] Friston, K., Rigoli, F., Ognibene, D., Mathys, C., Fitzgerald, T., and Pezzulo, G., "Active inference and epistemic value," *Cognitive Neuroscience* **6** no. 4, (Mar. 2015) 187–214.

[16] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O'Doherty, J., and Pezzulo, G., "Active inference and learning," *Neuroscience & Biobehavioral Reviews* **68** (Sep. 2016) 862–879.

[17] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., and Pezzulo, G., "Active Inference: A Process Theory," *Neural Computation* **29** no. 1, (Jan. 2017) 1–49.

[18] Friston, K., Parr, T., and Zeidman, P., "Bayesian model reduction." 2019. `arXiv:1805.07092 [stat.ME]`.

[19] Friston, K., Da Costa, L., Hafner, D., Hesp, C., and Parr, T., "Sophisticated Inference," *Neural Computation* **33** no. 3, (Mar. 2021) 713–763.

[20] Friston, K. J. and Frith, C. D., "Active inference, communication and hermeneutics," *Cortex* **68** (July 2015) 129–143.

[21] Friston, K. J., Daunizeau, J., and Kiebel, S. J., "Reinforcement Learning or Active Inference?" *PLoS ONE* **4** no. 7, (July 2009) e6421.

[22] Friston, K. J., Lin, M., Frith, C. D., Pezzulo, G., Hobson, J. A., and Ondobaka, S., "Active Inference, Curiosity and Insight," *Neural Computation* **29** no. 10, (Oct. 2017) 2633–2683.

[23] Friston, K. J., Rosch, R., Parr, T., Price, C., and Bowman, H., "Deep temporal models and active inference," *Neuroscience & Biobehavioral Reviews* **90** (July 2018) 486–501.

[24] Ge, M. and Ouyang, R., "State-Space Reduction in Deep Q-Networks," GitHub. `https://github.com/rlouyang/stat234-writeup/blob/master/final_project.pdf`, Apr. 2018.

[25] Gong, Y., Liu, L., Yang, M., and Bourdev, L., "Compressing Deep Convolutional Networks using Vector Quantization." 2014. `arXiv:1412.6115 [cs.CV]`.

[26] Hadsell, R., Chopra, S., and LeCun, Y., "Dimensionality Reduction by Learning an Invariant Mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, vol. 2, pp. 1735–1742. IEEE, 2006.

[27] Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M., "Dream to Control: Learning Behaviors by Latent Imagination." 2020. `arXiv:1912.01603 [cs.LG]`.

[28] Han, S., Mao, H., and Dally, W. J., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." 2016. `arXiv:1510.00149 [cs.CV]`.

[29] Hassibi, B. and Stork, D., "Second order derivatives for network pruning: Optimal Brain Surgeon," in *Advances in Neural Information Processing Systems*, Hanson, S., Cowan, J., and Giles, C., eds., vol. 5. Morgan-Kaufmann, 1992.

[30] Hinton, G. E. and Salakhutdinov, R. R., "Reducing the Dimensionality of Data with Neural Networks," *Science* **313** no. 5786, (July 2006) 504–507.

[31] Hobson, J. and Friston, K., "Waking and dreaming consciousness: Neurobiological and functional considerations," *Progress in Neurobiology* **98** no. 1, (July 2012) 82–98.

[32] Hobson, J. A., "Sleep is of the brain, by the brain and for the brain," *Nature* **437** no. 7063, (Oct. 2005) 1254–1256.

[33] Hobson, J. A., Hong, C. C.-H., and Friston, K. J., "Virtual reality and consciousness inference in dreaming," *Frontiers in Psychology* **5** (Oct. 2014) 1133.

[34] Holz, J., Piosczyk, H., Landmann, N., Feige, B., Spiegelhalder, K., Riemann, D., Nissen, C., and Voderholzer, U., "The Timing of Learning before Night-Time Sleep Differentially Affects Declarative and Procedural Long-Term Memory Consolidation in Adolescents," *PLoS ONE* **7** no. 7, (July 2012) e40963.

[35] Joiner, W. J., "Unraveling the Evolutionary Determinants of Sleep," *Current Biology* **26** no. 20, (Oct. 2016) R1073–R1087.

[36] Kingma, D. P. and Welling, M., "Auto-Encoding Variational Bayes." 2022. `arXiv:1312.6114 [stat.ML]`.

[37] Kirchhoff, M., Parr, T., Palacios, E., Friston, K., and Kiverstein, J., "The Markov blankets of life: autonomy, active inference and the free energy principle," *Journal of The Royal Society Interface* **15** no. 138, (Jan. 2018) 20170792.

[38] Korman, M., Raz, N., Flash, T., and Karni, A., "Multiple shifts in the representation of a motor sequence during the acquisition of skilled performance," *Proceedings of the National Academy of Sciences* **100** no. 21, (Oct. 2003) 12492–12497.

[39] Kramer, M. A., "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal* **37** no. 2, (Feb. 1991) 233–243.

[40] Lawton, W. H. and Sylvestre, E. A., "Self Modeling Curve Resolution," *Technometrics* **13** no. 3, (Aug. 1971) 617–633.

[41] LeCun, Y., Denker, J., and Solla, S., "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, Touretzky, D., ed., vol. 2. Morgan-Kaufmann, 1989.

[42] Li, W., Ma, L., Yang, G., and Gan, W.-B., "REM sleep selectively prunes and maintains new synapses in development and learning," *Nature Neuroscience* **20** no. 3, (Jan. 2017) 427–437.

[43] Li, Y. and Ji, S., "L_0-ARM: Network Sparsification via Stochastic Binary Optimization," in *Machine Learning and Knowledge Discovery in Databases*, Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M., and Robardet, C., eds., pp. 432–448. Springer International Publishing, 2020.

[44] Louizos, C., Welling, M., and Kingma, D. P., "Learning Sparse Neural Networks through L_0 Regularization," in *International Conference on Learning Representations*. 2018.

[45] Mignot, E., "Why We Sleep: The Temporal Organization of Recovery," *PLoS Biology* **6** no. 4, (Apr. 2008) e106.

[46] Millidge, B., Tschantz, A., Seth, A. K., and Buckley, C. L., "On the Relationship Between Active Inference and Control as Inference," in

Active Inference, Verbelen, T., Lanillos, P., Buckley, C. L., and De Boom, C., eds., pp. 3–11. Springer International Publishing, 2020.

[47] Mnih, V., Kavukcuoglu, K., *et al.*, "Human-level control through deep reinforcement learning," *Nature* **518** no. 7540, (Feb. 2015) 529–533.

[48] Molchanov, D., Ashukha, A., and Vetrov, D., "Variational Dropout Sparsifies Deep Neural Networks." 2017. `arXiv:1701.05369 [stat.ML]`.

[49] Murao, H. and Kitamura, S., "Q-Learning with adaptive state segmentation (QLASS)," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. "Towards New Computational Principles for Robotics and Automation"*, CIRA-97, pp. 179–184. IEEE Comput. Soc. Press, 1997.

[50] Oliver, G., Lanillos, P., and Cheng, G., "An Empirical Study of Active Inference on a Humanoid Robot," *IEEE Transactions on Cognitive and Developmental Systems* **14** no. 2, (June 2022) 462–471.

[51] Pearson, K., "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2** no. 11, (Nov. 1901) 559–572.

[52] Potkin, K. T. and Bunney, W. E., "Sleep Improves Memory: The Effect of Sleep on Long Term Memory in Early Adolescence," *PLoS ONE* **7** no. 8, (Aug. 2012) e42191.

[53] Rezende, D. J. and Viola, F., "Taming VAEs." 2018. `arXiv:1810.00597 [stat.ML]`.

[54] Roweis, S. T. and Saul, L. K., "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science* **290** no. 5500, (Dec. 2000) 2323–2326.

[55] Sajid, N., Ball, P. J., Parr, T., and Friston, K. J., "Active Inference: Demystified and Compared," *Neural Computation* **33** no. 3, (Mar. 2021) 674–712.

[56] Smith, R., Schwartenbeck, P., Parr, T., and Friston, K. J., "An Active Inference Approach to Modeling Structure Learning: Concept Learning as an Example Case," *Frontiers in Computational Neuroscience* **14** (May 2020) 41.

[57] Sprague, N., "Basis iteration for reward based dimensionality reduction," in *2007 IEEE 6th International Conference on Development and Learning*, vol. 401, pp. 187–192. IEEE, July 2007.

[58] Stickgold, R. and Walker, M. P., "Sleep-dependent memory triage: evolving generalization through selective processing," *Nature Neuroscience* **16** no. 2, (Jan. 2013) 139–145.

[59] Sýkora, O., "State-space Dimensionality Reduction in Markov Decision Processes," in *WDS'08 Proceedings of Contributed Papers: Part I — Mathematics and Computer Sciences*, Šafránková, J. and Pavlů, J., eds., pp. 165–170. Matfyzpress, June 2008.

[60] Tononi, G. and Cirelli, C., "Sleep function and synaptic homeostasis," *Sleep Medicine Reviews* **10** no. 1, (Feb. 2006) 49–62.

[61] Tschantz, A., Millidge, B., Seth, A. K., and Buckley, C. L., "Reinforcement Learning through Active Inference." 2020. `arXiv:2002.12636 [cs.LG]`.

[62] Ueltzhöffer, K., "Deep active inference," *Biological Cybernetics* **112** no. 6, (Oct. 2018) 547–573.

[63] Ullrich, K., Meeds, E., and Welling, M., "Soft Weight-Sharing for Neural Network Compression." 2017. `arXiv:1702.04008 [stat.ML]`.

[64] Watkins, C. J. C. H., "Learning from delayed rewards,". PhD thesis, King's College, Cambridge, United Kingdom, 1989.

[65] Wauthier, S. T., Çatal, O., De Boom, C., Verbelen, T., and Dhoedt, B., "Sleep: Model Reduction in Deep Active Inference," in *Active Inference*, Verbelen, T., Lanillos, P., Buckley, C. L., and De Boom, C., eds., pp. 72–83. Springer International Publishing, 2020.

[66] Yin, M. and Zhou, M., "ARM: Augment-REINFORCE-Merge Gradient for Stochastic Binary Networks," in *International Conference on Learning Representations*. 2019.

3.A Supplementary Tables and Figures

3.A.1 Neural architectures

Table 3.1: Neural architectures for the car racing (CR) and robot navigation (RN) environments.

Network	Layer	CR neurons/filters	RN neurons/filters
Posterior	Convolutional	8	8
			16
		16	32
			32
		32	64
			64
		64	128
			128
		128	256
	Concat		
	Linear	128	128
	Linear	$2 \times s$	$2 \times s$
Likelihood	Linear	128	128
	Linear	$128 \times 3 \times 3$	$256 \times 4 \times 5$
	Convolutional	128	256
			128
		64	128
			64
		32	64
			32
		16	32
			16
		8	8
			8
Prior	LSTM cell	128	128
	Linear	$2 \times s$	$2 \times s$

Convolutional layers have stride 1, padding 1 and filter size 3×3. Upsampling in the likelihood occurs through nearest neighbor interpolation after each convolutional layer. The "Concat" step creates a vector by concatenating the observation feature vector obtained from the convolutional pipeline with an action vector and latent space vector. All Linear and Convolutional layers have leaky ReLU activation functions. Concat has no activation. Variance output (denoted by $2\times$) uses a softplus activation.

—We are such stuff
As dreams are made on, and our little life
Is rounded with a sleep.

Shakespeare

4

Active inference with matrix product states

Discussing research with friends and colleagues in different fields can lead to interesting collaborations. As such, the idea for the work in this chapter was first conceived during an arbitrary gym session with friends near the end of the pandemic lockdowns. The realization that tensor networks could function as generative models and that they could automatically adjust their size was key to the integration with active inference. This interdisciplinary synergy highlights the unexpected yet powerful outcomes that can arise from casual conversations across diverse fields of study.

This chapter presents an implementation of active inference in terms of a specific type of tensor network known as the matrix product state. Matrix product states are a vital element in modern quantum many-body physics and have reached multiple disciplines over the years. Their power stems from their ability to efficiently handle high-dimensional data, while boasting manageable computational costs.

While matrix product states are a relatively simple generative model, they lay the groundwork for the future application of more advanced tensor networks in active inference. For example, the model in this chapter only works for a limited number of time steps. In the next chapter, this will be extended to an infinite number of time steps by switching to a specific kind of matrix product state.

Planning with tensor networks based on active inference

S. T. Wauthier • T. Verbelen • B. Dhoedt • B. Vanhecke
Published in Machine Learning: Science and Technology 2024

Abstract

Tensor networks have seen an increase in applications in recent years. While they were originally developed to model many-body quantum systems, their usage has expanded into the field of machine learning. This work adds to the growing range of applications by focusing on planning by combining the generative modeling capabilities of matrix product states and the action selection algorithm provided by active inference. Their ability to deal with the curse of dimensionality, to represent probability distributions, and to dynamically discover hidden variables make matrix product states specifically an interesting choice to use as the generative model in active inference, which relies on "beliefs" about hidden states within an environment. We evaluate our method on the T-maze and Frozen Lake environments, and show that the tensor network-based agent acts Bayes optimally as expected under active inference.

4.1 Introduction

Tensor networks (TNs) were originally developed to represent quantum states in many-body quantum physics [10, 52, 14, 71, 50], and more recently have found their way into computer science [53] and machine learning (ML) [73]. As a result, the range of applications in the field of ML has been steadily growing. A comprehensive list of past research into ML with TN was compiled by Wang et al. [73]. For example, Stoudenmire and Schwab [65] and Cheng et al. [9] trained tensor networks in a supervised manner to perform image classification on the MNIST handwritten digits data set [40]. Furthermore, different types of tensor networks have been used for the generative modeling of the MNIST data set, such as matrix product states (MPS), a.k.a. tensor train (TT), [29, 66], tree tensor networks (TTN) [8] and projected entangled pair states (PEPS) [72].

More related to the current work, Guo et al. [27] investigated the use of matrix product operators (MPO) for probabilistic modeling, while Stokes and Terilla [64] investigated the use of MPS for the same purpose. Similarly,

Glasser et al. [26] investigated how graphical models can map to generalized tensor networks and demonstrated this through the use of entangled plaquette states (EPS) and string-bond states (SBS) on image and audio classification tasks. Mencia Uranga and Lamacraft [47] used a continuous MPS for the generative modeling of raw audio. Meanwhile, Miller et al. [49] used a uniform MPS for the generative modeling of text data. Further, Hur et al. [35] proposed an approach to generative modeling using MPS through tensor sketching, while Peng et al. [55] extended this approach to hierarchical TNs.

In reinforcement learning (RL), Liu and Fang [44] used MPS for variational Q-learning. Gillman et al. [22] and Howard [34] demonstrated similar TN approaches to policy optimization for finite Markov decision processes (FMDP), where the latter also showed its value for multi-agent RL. Similarly, Mahajan et al. [46] provided a tensorized formulation of the Bellman equation in multi-agent RL. Metz and Bukov [48] used MPS for the control of many-body quantum systems based on RL, much like Lu and Ran [45]. Further, Gillman et al. [23] provided a method to combine tensor networks with actor-critic techniques. Finally, although this work is situated largely within AI planning, it differentiates itself from earlier work through probabilistic modeling of action-observation sequences with action selection based on active inference, as opposed to learning some RL policy.

The use of tensor networks in machine learning may be interesting for several reasons. Foremost, TNs were developed to deal with the curse of dimensionality in many-body quantum systems, where the quantum state is given by a tensor with as many indices as there are particles. The number of degrees of freedom thus grows exponentially with the number of particles. Tensor networks take on this problem by restricting the exponentially large space to a tiny manifold of states satisfying an 'entanglement area-law'. This can be understood as a locality of the entanglement in the system, where a 'typical' state would have volume-law entanglement [31] and there would be no sense of locality. Specifically for MPS, it has been proven that they describe extremal eigenvectors of local gapped Hermitian operators in one dimension [30, 60, 3, 4, 2] and they have also been shown to efficiently capture low energy physics [78, 69, 28]. Moreover, it has been shown that local correlations, i.e. exponential decay, imply such an area-law [5] in any dimension, and hence permit an efficient tensor network description in that case. Such states could also be useful to machine learning, as relevant data points are often contained within a smaller subspace of the configuration space where the type of correlations is restricted. For example, in computer vision, images composed of randomly generated pixels are generally not useful to a classification or segmentation algorithm. Instead one would

expect nearby pixels to be highly correlated, and far away pixels to be weakly correlated. Similarly, in planning, only physically possible sequences are relevant, and in particular events from the distant past will typically have far less effect on planning the future than the recent past.

Other advantages are that TNs offer a distributed representation of the data set and only involve linear operations. Due to this, computations can easily be performed in parallel. Furthermore, there is a lot of available research on expressivity [37, 25] and interpretability [67, 57, 1] of TNs, leading to a relatively good understanding compared to neural networks. In fact, often tensor networks are used to characterize neural networks [11, 24, 37, 7, 61]. We also note that MPS can be used to study Markov decision problems [63].

Despite the theoretical advantages, the use of TNs in ML is relatively new and many of the algorithms still suffer from teething problems. As addressed by Liu et al. [43], tensor network-based models generally do not achieve the state-of-the-art performance of neural network-based models in practice. Much of the research done on ML using TNs can be regarded as proof-of-concept work and is verified using simple benchmarks, such as MNIST for classification.

Nevertheless, since TNs have already been shown to be able to model sequences in the form of text [49], they may also prove useful in the field of AI planning where sequences consist of actions and observations [74]. As shown in this work, combining TNs and active inference leads to a straightforward algorithm for planning.

Active inference is a theory of behavior and learning in neuroscience, where the autonomous agent is presumed to learn a generative model of the world and select actions by minimizing a quantity called the variational free energy [19]. The agent's beliefs about the environment in which it is located are encoded within this generative model in the form of probabilities over actions and observations. The agent's next action is the action which is expected to yield the lowest variational free energy in the future. Parr et al. [54] provide an excellent in-depth exposition on active inference and its formulations.

Tensor networks form a natural candidate to be used as the generative model in active inference for several reasons. Firstly, a similar argument regarding the earlier mentioned fact that TNs can also address the curse of dimensionality in ML can be made in favor of active inference. The aforementioned local correlations are comparable with the action-observation sequences allowed by the environment where one would expect only weaker correlations between actions and observations separated far in time. In active inference, action-observation sequences occupy only a fraction of the total

action-observation space and the strength of correlation typically depends on temporal separation. Secondly, TNs were developed to represent quantum states, which represent the knowledge of a quantum system, where measurements are fundamentally probabilistic. Comparably, an active inference agent's internal model of the world contains the agent's knowledge about its environment in the form of beliefs, generally represented by probability distributions. Thirdly, the number of parameters in a TN can be dynamically determined during training based on information requirements. This ability is akin to dynamically discovering latent variables and is especially valuable for autonomous agents when computational resources are limited. It can be compared to the minimization of statistical complexity through the pruning of connections which occurs during sleep in biological agents [33, 21, 41]. Lastly, it has been proposed that cells require quantum coherence for information processing [59, 15], with recent empirical evidence suggesting the occurrence of entanglement in the brain mediated by brain function [36]. This implies that control models in biological systems, i.e. the models in charge of planning, must support the possibility of quantum computation [16]. TNs naturally provide the ability to account for this. More so, it has been shown that control models in active inference can always be represented as a tensor network in certain systems [16, 17].

Although TNs have already been shown to be able to model sequences, generating sequences and planning optimal paths are two very different tasks. While a TN might be well able to generate paths that are consistent with the environment, there is no guarantee that any of these paths is optimal. Active inference guarantees that the path is optimal within the agent's belief system [54].

This paper contributes to the growing body of work in machine learning using tensor networks. In particular, we provide the first account of AI planning using tensor networks by making use of the ability of matrix product states to model sequences and the action selection algorithm provided by active inference. From a machine learning perspective, this constitutes a model-based approach to learning and planning. Inversely, from an active inference perspective, this corresponds to modeling the agent's generative model through a tensor network. In addition, we introduce an update scheme which alternates between single-site and two-site updates, and allows dynamic variation of bond dimensions, while avoiding exploding bond dimensions as well as speeding up computation.

In section 4.2, we introduce the basic concepts that constitute this work. These include matrix product states (section 4.2.1), generative modeling with tensor networks (section 4.2.2), and active inference (section 4.2.3). In section 4.3, we present our methods. We demonstrate the results of the

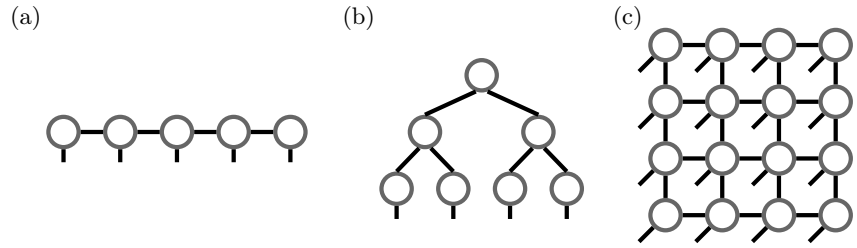


Figure 4.1: Popular types of tensor networks: (a) a matrix product state (MPS), a 1D array of tensors, (b) a tree tensor network (TTN), a set of tensors arranged in a tree structure, and (c) a projected entangled pair state (PEPS), a 2D array of tensors.

experiments in section 4.4 and provide a discussion on scalability, and data and exploration in section 4.5. We conclude this work with section 4.6 by summarizing our findings and presenting possible paths for future work.

4.2 Background

4.2.1 Matrix product states

At its core, a tensor network is a contraction of individual tensors, typically shown as a graph, i.e. a 'network', where the nodes represent tensors and the edges represent summation over an index. Such a network can have many architectures. Popular types of networks include matrix product states (MPS) [14, 38, 39, 10], tree tensor networks (TTN) [62, 52], which are tensor networks without loops in their graphical representation and can therefore be treated with similar ease as MPS, and projected entangled pair states (PEPS) [70, 52, 10], which are tensor networks designed for two dimensional quantum systems and hence their graphical representation is also manifestly two dimensional (Figure 4.1). In particular, the loops in the PEPS make them significantly more challenging to work with.

Arguably the simplest form of tensor network is the MPS, also known as tensor train (TT) [56]. In simple terms, an MPS is the result of decomposing a tensor with N indices into N tensors of smaller order. For example, the tensor T_{abcd} with four indices can be decomposed into four tensors:

$$T_{abcd} = \sum_{\alpha_1\alpha_2\alpha_3} A^{(1)}_{a\alpha_1} A^{(2)}_{\alpha_1 b\alpha_2} A^{(3)}_{\alpha_2 c\alpha_3} A^{(4)}_{\alpha_3 d} \ , \tag{4.1}$$

or graphically:

. (4.2)

The indices α_i, known as bond indices, are internal to the network. The dimensionality of the bond indices, the *bond dimension*, must be large enough for the equality in Eq. (4.2) to hold. For an MPS to be able to exactly represent an arbitrary tensor with N d-dimensional indices, the network's bonds must have dimension $d^{N/2}$ [18] in the center. In practice, the bonds are truncated [13] so the above equality may be only approximately true. The bond dimensions are thus hyperparameters and can be chosen depending on how much entanglement there is in the state. Note that an MPS should only be used in cases where this approximation can be efficient and accurate, i.e. in case one may expect an area law entanglement [12].

4.2.2 Generative modeling with tensor networks

A generative model is a statistical model of the joint probability distribution $P(X)$ of a set of random variables $X = (X_1, X_2, \ldots, X_n)$. Such a model makes it possible to generate outcomes $x = (x_1, x_2, \ldots, x_n)$ by sampling from the distribution. In addition, it allows the computation of marginal and conditional probabilities by performing the appropriate summations.

As stated earlier, quantum states can be thought of as representing the knowledge of a system. In other words, they represent the probability of observing a certain outcome (after performing a measurement on a system). Such a quantum state can be mathematically described through a wave function $\Psi(x)$ on a set of real variables $x = (x_1, x_2, \ldots, x_n)$. From there, the probability of observing x is given by the Born rule:

$$P(X = x) = \frac{|\Psi(x)|^2}{Z} \ , \tag{4.3}$$

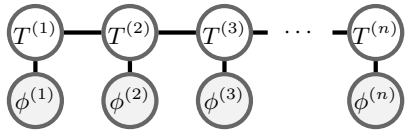
with normalization $Z = \sum_{\{x\}} |\Psi(x)|^2$, where the summation runs over the set of all possible realizations of the values of x. The Born rule shows how a quantum state can serve as a generative model [42].

When approximating a quantum state through an MPS, the wave function is parameterized as follows:

$$\Psi(x) = \sum_{\alpha_1,\ldots,\alpha_{n-1},\beta_1,\ldots,\beta_n} T^{(1)}_{\beta_1\alpha_1} T^{(2)}_{\alpha_1\beta_2\alpha_2} T^{(3)}_{\alpha_2\beta_3\alpha_3} \cdots T^{(n)}_{\alpha_{n-1}\beta_n} \phi^{(1)}_{\beta_1}(x_1)\phi^{(2)}_{\beta_2}(x_2)\phi^{(3)}_{\beta_3}(x_3)\cdots\phi^{(n)}_{\beta_n}(x_n) \ . \tag{4.4}$$

Here, each $T^{(i)}$ is a tensor of order 3 (aside from the tensors at the extremities, which are of order 2), and each $\phi^{(i)}$ is a feature map, which maps the input value x_i to a higher-dimensional feature space. In graphical notation, this becomes

$$\Psi(x) = \quad . \tag{4.5}$$



In brief, the wave function $\Psi(x)$ is decomposed into a series of tensors $T^{(i)}$ and input vectors $\phi^{(i)}$.

While the tensors $T^{(i)}$ can simply be regarded as arrays of parameters that can be learned during training, the feature maps $\phi^{(i)}$ serve another important purpose. To build the parallel with quantum mechanics, a feature map $\phi^{(i)}$ must ensure that each possible value of the input x_i can be regarded as a distinct eigenvalue (of an observable) with its corresponding eigenvector. In other words, applying the feature map $\phi^{(i)}$ to the set of possible values of the input x_i must yield a set of linearly independent vectors.

To allow for a generative interpretation of the tensor network, the set of vectors obtained through feature maps must be orthogonal and each vector must be of unit norm [65]. These conditions ensure that this set forms an orthonormal basis of the feature space. The dimensionality of this space is referred to as the *local dimension*. For example, a convenient choice of feature map is the one-hot encoding: if $x_i \in \{0, 1, 2\}$, the encoding yields $(1, 0, 0)$ for 0, $(0, 1, 0)$ for 1, and $(0, 0, 1)$ for 2. These vectors form the basis of a 3-dimensional feature space. In general, for an input x_i which can assume n discrete values, $\phi^{(i)}$ will map each of these values onto an orthonormal basisvector, such that the feature space becomes n-dimensional.

Given a data set, the MPS can be trained through a density matrix renormalization group (DMRG)-like [75, 77] method by "sweeping" back and forth across the MPS, updating tensors sequentially with respect to a given loss function. Since our tensor network will serve as a generative model, we minimize the negative log-likelihood (NLL), i.e. we maximize the model evidence directly [29]. This way, the network will attempt to learn the underlying probability distribution of the data set during training.

One possible way of updating the tensor elements is performing *single-site updates*, i.e. updating a single tensor at a time. For example, starting from the left-most tensor, compute the gradient of the loss function and update the tensor elements. Then, move to the adjacent tensor and repeat. Once the right-most tensor is reached, do the same in the opposite direction. Continue sweeping back and forth until convergence.

Alternatively, Stoudenmire and Schwab [65] describe a *two-site update*

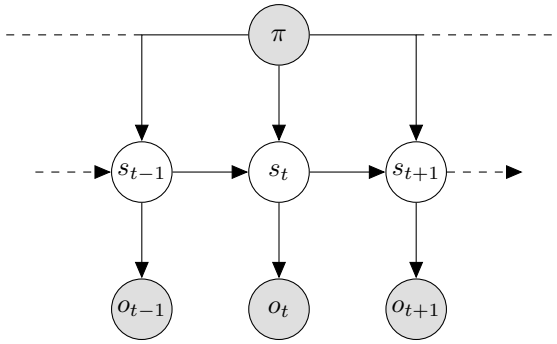


Figure 4.2: Bayesian network corresponding to the generative model $P(\tilde{o}, \tilde{s}, \pi)$ in discrete time active inference. The model is defined in terms of sequences of observations o_t and hidden states s_t evolving over time, and a policy $\pi(t) = a_t$ with a_t the selected action at time t.

scheme, in which two tensors are updated at the same time. Crucially, this allows the bond dimensions to vary during training depending on how many dimensions are required to encode the information in the data set. In other words, the number of parameters in the MPS can vary during training. In short, by merging and decomposing adjacent tensors through singular value decomposition (SVD), it is possible to increase or reduce the bond dimensions based on the number of singular values. The cutoff value ε determines the number of singular values that will be retained (cf. [65]). Singular values that are larger than ε times the largest singular value are retained, while the rest is discarded. As such, bond dimensions are generally initialized at a low number, so that the tensor network starts out with a low number of parameters and grows based on how many dimensions are needed.

As described by Han et al. [29], it is possible to obtain different marginal and conditional probabilities using the above formulation.

4.2.3 Active inference

Active inference posits that each autonomous agent is equipped with an internal model of the world, a generative model P of how sensory observations are generated, which encodes beliefs about these sensory observations o_t, hidden variables s_t and the policy π it is following [54]. Importantly, the policy returns an action as a function of time, $a = \pi(t)$, in contrast with standard formulations where the policy is a function of the state, $a = \pi(s)$ [19]. As such, one can also write down the policy as a sequence of actions a_t.

An active inference agent updates its internal model or selects actions in an attempt to minimize its surprisal, a.k.a. information content or self-

information, [54]

$$\mathfrak{I}(\tilde{o}) = -\log P(\tilde{o}) \ , \tag{4.6}$$

with $\tilde{o} = (o_1, o_2, \ldots, o_\tau)$, where we have used the $\sim$-notation indicating a sequence over time up to some time τ. In discrete time active inference, the generative model is often compared to a partially observable Markov decision process (POMDP) [19]. In that case, the generative model $P(\tilde{o}, \tilde{s}, \pi)$ factorizes as in Fig. 4.2,

$$P(\tilde{o}, \tilde{s}, \pi) = P(\pi)P(s_0)P(o_0|s_0)\prod_{t=1}^{\tau} P(s_t|s_{t-1}, \pi)P(o_t|s_t) \ , \tag{4.7}$$

and the agent instead minimizes its variational free energy

$$F[Q, \tilde{o}] = E_{Q(\tilde{s},\pi)}[\log Q(\tilde{s}, \pi) - \log P(\tilde{o}, \tilde{s}, \pi)] \tag{4.8}$$

$$= \underbrace{\mathrm{D}_{\mathrm{KL}}(Q(\tilde{s}, \pi) \,||\, P(\tilde{s}, \pi \,|\, \tilde{o}))}_{\text{divergence}} \underbrace{- \log P(\tilde{o})}_{\text{surprisal}} \ , \tag{4.9}$$

where $Q(\tilde{s}, \pi)$ is an approximation to the posterior distribution $P(\tilde{s}, \pi \,|\, \tilde{o})$, not to be confused with the Q-function with similar notation often used in RL. The use of the approximate Q is motivated by the fact that the evidence $P(\tilde{o})$, and therefore the posterior distribution $P(\tilde{s}, \pi \,|\, \tilde{o}) = \frac{P(\tilde{o} \,|\, \tilde{s}, \pi)P(\tilde{s}, \pi)}{P(\tilde{o})}$, is often intractable due to the requirement to marginalize the generative model over $\tilde{s}$ and π. Note that the variational free energy (Eq. (4.9)) is an upper bound on the surprisal (Eq. (4.6)) and becomes equal to the surprisal when the divergence becomes zero, i.e. when $Q(\tilde{s}, \pi) = P(\tilde{s}, \pi \,|\, \tilde{o})$.

The active inference agent selects actions which are expected to minimize its free energy in the future. To this end, the agent maintains a prior $P(\tilde{o} \,|\, C)$ which specifies the conditions that it would like to experience, i.e. its preferred observations. Here, C indicates the agent's preferences. Action selection proceeds by computing the expected variational free energy for all possible policies [54]

$$G(\pi) = \mathrm{E}_{\hat{Q}(\tilde{o},\tilde{s} \,|\, \pi)}[\log Q(\tilde{s} \,|\, \pi) - \log P(\tilde{o}, \tilde{s} \,|\, \pi, C)] \tag{4.10}$$

$$\approx \underbrace{-\mathrm{E}_{\hat{Q}(\tilde{o} \,|\, \pi)}[\mathrm{D}_{\mathrm{KL}}(\hat{Q}(\tilde{s} \,|\, \tilde{o}, \pi) \,||\, Q(\tilde{s} \,|\, \pi))]}_{\text{information gain}} \underbrace{-\mathrm{E}_{\hat{Q}(\tilde{o} \,|\, \pi)}[\log P(\tilde{o} \,|\, C)]}_{\text{expected surprisal}} \ , \tag{4.11}$$

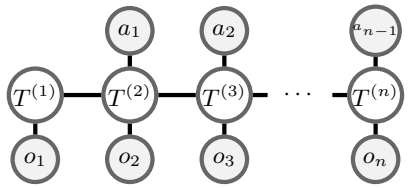
where we have used $\hat{Q}(\tilde{o}, \tilde{s} \,|\, \pi) \coloneqq P(\tilde{o} \,|\, \tilde{s})Q(\tilde{s} \,|\, \pi)$ and $P(\tilde{s} \,|\, \tilde{o}, \pi) \approx Q(\tilde{s} \,|\, \tilde{o}, \pi)$. After which, the agent follows the policy with the lowest expected variational free energy.

4.3 Methods

4.3.1 Adapting the MPS for action selection

When employing a tensor network as a generative model in an active inference agent, the network represents the beliefs held by the agent. Roughly speaking, in active inference, the generative model is the joint distribution of sequences of actions a_t and observations o_t. As a result, each time step t is associated with a tensor $T^{(t)}$. Furthermore, we may split the external leg of each tensor in Eq. (4.5) by redefining indices, such that at each tensor $T^{(t)}$, there is a leg for an action feature map $\phi_{\mathrm{a}}^{(t)}$ and an observation feature map $\phi_{\mathrm{o}}^{(t)}$. Moreover, since the action and observation spaces do not change over time, we have $\phi_{\mathrm{x}}^{(t)}(x_k) = \phi_{\mathrm{x}}(x_k)$ for all t and $x \in \{a, o\}$.

When simplifying the graphical notation by using the nodes  to refer to $\phi_{\mathrm{x}}(x_k)$, the network becomes

$$\Psi(\tilde{a}, \tilde{o}) =$$  , (4.12)

where the length n of the network depends on the environment. Since we define that action a_t leads to observation o_{t+1}, action a_0 would be superfluous and tensor $T^{(1)}$ does not receive an action.

4.3.2 Updating the MPS

Both update methods mentioned in section 4.2.2 are viable and can be used interchangeably or simultaneously (by alternating between the two). However, all the models in this work were optimized using batched stochastic gradient descent (SGD). This was done to accommodate for future applications where the data set size is too large to fit into memory in its entirety. Note that this differs from DMRG, although the implementation largely overlaps. The SGD update will predictably give rise to a far slower and noisier convergence than the DMRG update (which involves an eigenvalue problem), and this must be accounted for.[1]

As a matter of fact, allowing the bond dimensions to perpetually change during two-site updates with SGD may be detrimental to training stability. Taking steps in random directions may cause the bond dimensions to change size with every step. In turn, if the bond dimensions are constantly changing

[1]The two-site update is described in Appendix B.

size, the model may have difficulties generalizing. For example, imagine a freshly initialized model which has not yet learned. If the bond dimensions are allowed to vary from the start of training, the model is free to add new dimensions for new data without generalizing first. In a training setting with a high number of batches, this may lead to rapidly increasing bond dimensions. While in theory, over time the model will learn to generalize and fix itself, in practice, large bond dimensions may cause computational issues. In that case, it may be a good idea to try to generalize on-the-fly and add dimensions as the need arises. While it is possible to set an upper bound on the bond dimensions, for a general environment, it is difficult to anticipate the number of required dimensions, i.e. the number of required bond dimensions is highly dependent on the application. Therefore, it may also be difficult to set a nonrestrictive upper bound. Although this problem does not arise for the environments employed in this work, it should be taken into account for the general case.

A simple solution to this issue is to perform single-site updates and only perform two-site updates occasionally (e.g. when the current configuration has more or less converged). As a practical matter, we suggest to set the initial bond dimensions to a number close to where one would expect the bonds dimensions to end up, such that fewer bond dimension changes are required in the end. In this work, we set the initial bond dimensions to the observation dimension, since we expected the bonds to prioritize encoding information about the positions over the actions.

As an added benefit, the use of single-site updates reduces training time [76]. Two-site updates require merging and decomposing adjacent tensors. This effectively means squaring the local physical dimension at one site. While the local dimension is typically small in quantum-many body settings (often 2 or 3 for qubits or qutrits, respectively), the complexity would only increase by a small prefactor compared to the one-site scheme. In the context of this paper it can be significantly bigger, leading to a large difference in computational complexity that can be most clearly perceived in the SVD of the merged tensors. For reference, SVD on a single tensor has a time complexity of $\mathcal{O}(dD^3)$, while SVD on the merged tensor has a time complexity of $\mathcal{O}(d^3D^3)$, where d is the local dimension and D is the bond dimension.

Finally, the use of SGD also affects the possible choices for the cutoff ε on the singular values. An ε of order $1/\sqrt{N}$ with N the size of the dataset is generally advised, since this is also how the size of the random errors in the gradient and cost function scale.

4.3.3 Active inference with an MPS

Using an MPS, allows the agent to minimize surprisal (Eq. (4.6)) directly, rather than through variational Bayesian methods. This is because marginal probabilities become tractable when using MPS due to efficient contraction schemes. As a result, the agent will instead select actions which minimize its expected surprisal

$$\mathfrak{J}(\pi) = -\mathrm{E}_{P(\tilde{o}\,|\,\pi)}[\log P(\tilde{o}\,|\,C)] \ . \tag{4.13}$$

In practice, computing the expected surprisal for each possible policy is computationally expensive. Using the sophisticated inference [20] scheme may alleviate this problem to a certain extent. Since this scheme builds a tree over actions and observations in the future, it requires less computations than greedily computing the expected surprisal for every possible sequence. Moreover, branches can be pruned before they have been fully computed when they show signs of having high expected surprisal, or when the transition probability $P(o_{t+1}\,|\,o_{<t+1}, a_{<t+1})$ is low.

Sophisticated inference works by computing an aggregated expected surprisal for each possible action at time t, instead of the expected surprisal for each policy. Aggregation is achieved by summing the expected surprisal of subsequent actions and weighing each action by its expected surprisal as follows:

$$\begin{aligned}\mathfrak{J}(o_t, a_t) = &\underbrace{-\mathrm{E}_{P(o_{t+1}\,|\,\tilde{o}_{<t+1},\tilde{a}_{<t+1})}[\log P(o_{t+1}\,|\,C)]}_{\text{expected surprisal of next action}} \\ &+ \underbrace{\mathrm{E}_{P(a_{t+1}\,|\,o_{t+1})P(o_{t+1}\,|\,\tilde{o}_{<t+1},\tilde{a}_{<t+1})}[\mathfrak{J}(o_{t+1}, a_{t+1})]}_{\text{expected surprisal of subsequent actions}} \ ,\end{aligned} \tag{4.14}$$

$$P(a_t\,|\,o_t) = \sigma[-\mathfrak{J}(o_t, a_t)] \ . \tag{4.15}$$

Details on how the relevant probabilities can be obtained from the trained MPS, can be found in Appendix 4.A.

4.4 Experiments

The method described in the previous sections was tested on environments with discrete actions and observations. A simple, yet important environment for active inference is the T-maze, as presented by Friston et al. [19]. This environment tests whether an agent is able to resolve ambiguity in order to ensure obtaining a reward, as opposed to guessing and obtaining a reward only a fraction of the time. It is useful to demonstrate the workings of action selection through active inference.

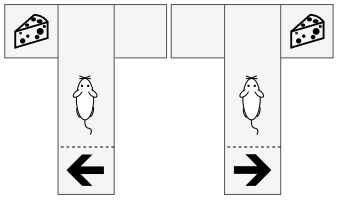


Figure 4.3: Possible instances of the T-maze environment as presented by Friston et al. [19]. The agent (mouse) starts off in the center. The reward (cheese) is located in either the left branch (left instance) or right branch (right instance). The cue reveals the location of the reward: a left-pointing arrow indicates the reward is on the left, a right-pointing arrow indicates the reward is on the right.

A slightly more complex environment is the Frozen Lake from OpenAI Gym [6]. This environment has a larger observation space and, theoretically, an infinite time horizon. We use it to investigate how our method scales to more complex environments.

4.4.1 T-maze

The T-maze environment consists of a T-shaped maze which contains an agent, e.g. a mouse, and a reward, e.g. some cheese (Figure 4.3). The maze is made up of four locations: the center, the right branch, the left branch, and the cue location. The agent starts off in the center and must attempt to reach the reward, which is placed in either the left branch or the right branch, by moving between the different locations. If the agent enters either the left or the right branch, it is trapped and cannot leave. Unfortunately for the agent, the initial state of the world is uncertain. As a result, the agent initially does not know which type of world it is in: a world with the reward in the left branch, or a world with the reward in the right branch. In other words, it does not know its context. However, by going to the cue location, the agent can discover the location of the reward and observe its context. This resolves the ambiguity in the environment and allows the agent to select the branch which contains the reward.

The pymdp [32] package provides an implementation of such a T-maze. In this implementation, the environment provides an observation with three modalities at every time step: the location, the reward, and the context. The location indicates where the agent is currently located and can be one of four possible values: center, right, left, and cue. The reward can take on three possible values: no reward, win, and loss. The “win” and “loss” observations indicate that the agent either received the reward or failed to

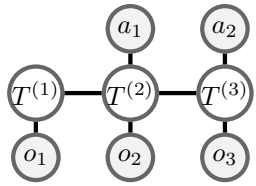
obtain the reward, respectively, while the "no reward" observation indicates that the agent has not won or lost yet. As a result, "no reward" can only be observed in the center and cue location, while "win" and "loss" can only be observed in the left and right branches. Furthermore, the context indicates the state of the world and can take on two possible values: left and right. Whenever the agent is located in the "center", "left" or "right" positions, the context observation will be randomly selected from "left" or "right" uniformly. The context observation will only yield the correct context with probability 1 when the agent is in the cue location. Lastly, the possible actions that the agent can take include: center, right, left, and cue, corresponding to which location the agent wants to go.

The above implementation was slightly modified to better reflect the environment designed by Friston et al. [19]. Firstly, the environment's horizon was set to 2, such that the agent is only allowed to perform two subsequent actions. In other words, the number of steps was limited to two.

4.4.1.1 Model

To accommodate the T-maze environment, the model (Eq. (4.12)) was limited in time as follows:

$$\Psi(\tilde{a},\tilde{o}) = \quad , \tag{4.16}$$



where the length of the network was equal to the initial time step plus the environment's horizon. Here, one-hot encodings were the feature maps of choice. This means the observation feature map o_t has $4 \times 3 \times 2 = 24$ dimensions, one for each possible combination of position, reward and context, and the action feature map a_t has 4 dimensions. Note that this network only has two bonds which varied in dimensionality during training.

4.4.1.2 Hyperparameters

The model described above relies on a number of hyperparameters. Firstly, important factors that could affect the performance of the model were cutoff ε and the data set.

Cutoff Eight different cutoffs ε for the singular values of the bonds were compared: 0.2, 0.1, 0.05, 0.04, 0.03, 0.02, 0.01 and 0.005.

Data set The data sets used in this experiment consist of sample trajectories. Five different types of data set were compared:

- four randomly generated data sets of size 100, 204, 1000 and 10000, respectively, where each sequence consisted of random actions and observations. These data sets were generated anew for each model. Note that the data may contain duplicates.

- an engineered data set which includes all possible sequences within the environment exactly once. This data set contains exactly 204 sequences. Denoted in the figures by "(204)".

As a result of their respective sizes, the data sets of size 1000 and 10000 will contain many duplicates. However, they are also more likely to capture the underlying probability distributions of the generative process.

Importantly, each model was trained for a total of 50000 sweeps with a batch size of 100 and a learning rate of 10^{-3}. In practice, for data sets of sizes 100, 204, 1000 and 10000, this translates to 50000, 25000, 5000 and 500 epochs, respectively.

Finally, we evaluated different initialization strategies, more specifically initialization with complex numbers with positive real part, and initialization to unity with standard complex normal noise. While having some effect on convergence behavior at train time, no effect on results was seen.

4.4.1.3 Results

To evaluate the performance of each model, we measured the success rate over 100 runs using the action selection scheme presented in Section 4.3.3. The preferred distribution $P(o_t \mid C)$ remained constant for every t and was factorized into preferred distributions over the three modalities: position, reward and context. Technically, the hyperparameter C can be freely chosen or learned [58]. Since the choice of weights raises a research question in and of itself, we simply selected a set of weights which worked for our purposes. Here, the weights for win and loss were fixed to the values in [19]:

$$P(\text{modality} \mid C_{\text{modality}}) = \sigma(C_{\text{modality}}) \ , \quad \text{modality} \in \{\text{position}, \text{reward}, \text{context}\} \tag{4.17}$$

$$C_{\text{position}} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \ , \ C_{\text{reward}} = \begin{bmatrix} 0 & 3 & -3 \end{bmatrix} \ , \ C_{\text{context}} = \begin{bmatrix} 0 & 0 \end{bmatrix} \ , \tag{4.18}$$

where σ is the softmax function, such that each possible value of each of the modalities has an associated weight. These distributions represent the agent's indifference towards its position and context, and its desire to obtain

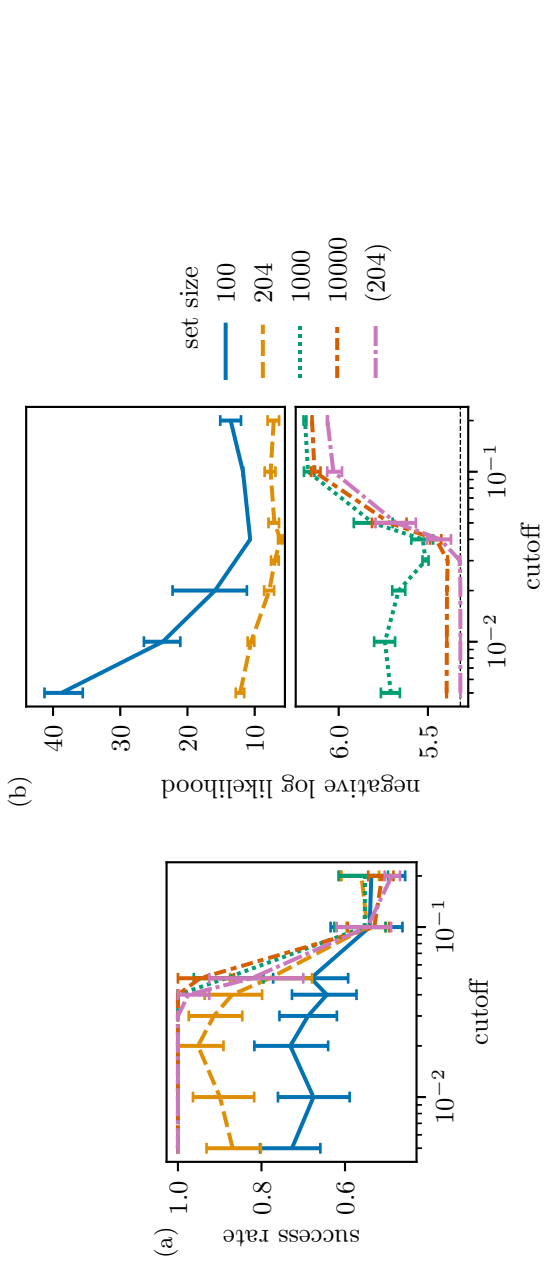


Figure 4.4: (a) Success rate as a function of cutoff and data set size. Each point reflects the average across 20 models of the success rate over 100 runs. (b) Average negative log likelihood over all possible sequences as a function of cutoff and data set size. Each point reflects the average across 20 models. The dashed line indicates the theoretical minimum of $-\log(1/204) \approx 5.32$ for this data set. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.4.1.2.

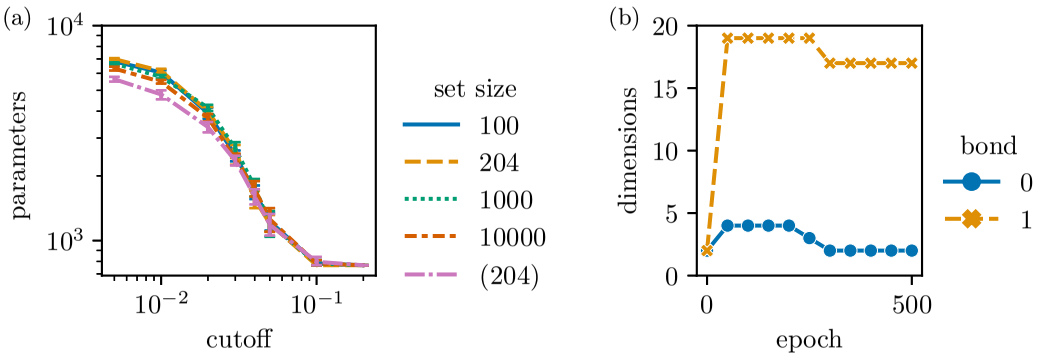


Figure 4.5: (a) Average number of parameters as a function of cutoff and data set size. Each point reflects the average across 10 models. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.4.1.2. (b) Evolution of bond dimensions during training for a model with $\varepsilon = 0.01$ trained on the engineered data set. The bond dimensions increased as the model learned new information and decreased as the model learned to generalize. Epochs beyond 500 were omitted, since there was no more change.

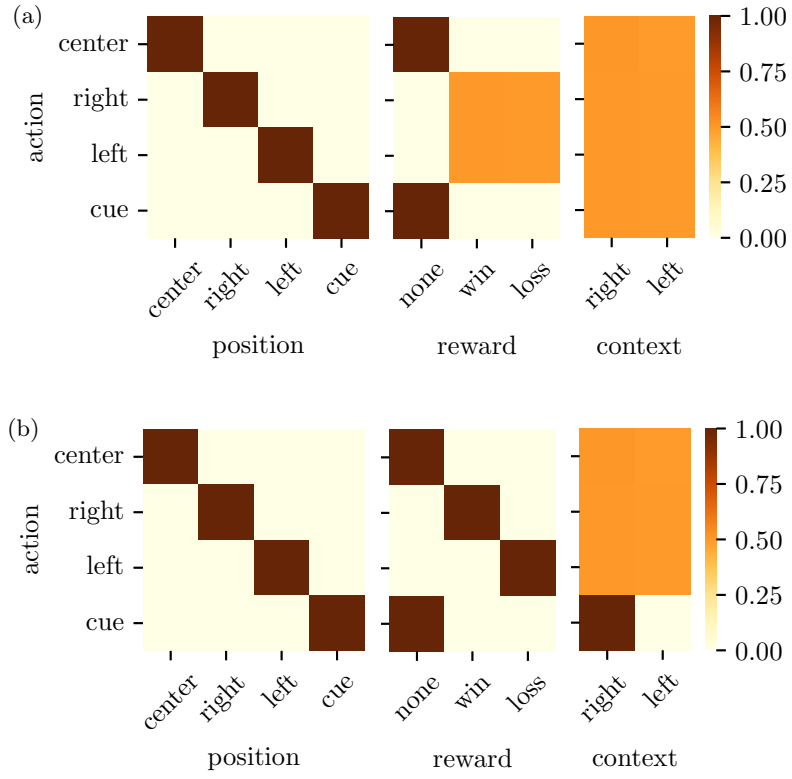


Figure 4.6: Belief shift (a) before and (b) after having seen the cue "right". Computed for a model with $\varepsilon = 0.01$ trained on the engineered data set.

a reward (weighted by 3) and not fail (weighted by -3). In other words, the agent expects to be obtaining the reward $e^3 \approx 20$ times more than to be obtaining a neutral outcome. During a run, a success (agent obtained the reward) was indicated by 1, while a failure (agent chose the wrong branch or no branch at all) was indicated by 0. Note that an agent which chooses a branch at random would obtain a success rate of 0.5.

Previous work [74] demonstrated belief shifts and action selection under sophisticated active inference (Eq. (4.11)) when training on the engineered data set with a cutoff of 0.1 using only two-site updates. In this section, we thoroughly investigate the performance for different hyperparameters (cf. Section 4.4.1.2), demonstrate the belief shifts and action selection under the surprisal minimization scheme in Section 4.3.3 and investigate performance when using single-site updates.

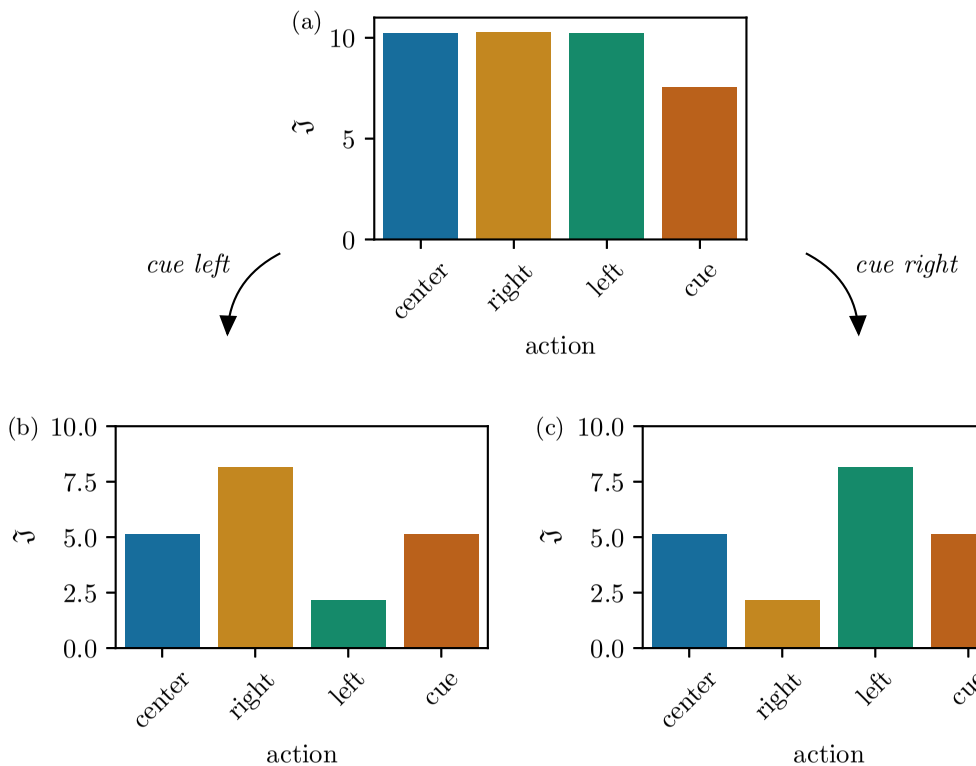


Figure 4.7: Expected surprisal: (a) for the first action, (b) for the second action after having seen the cue "left", and (c) for the second action after having seen the cue "right". Computed for a model with $\varepsilon = 0.01$ trained on the engineered data set.

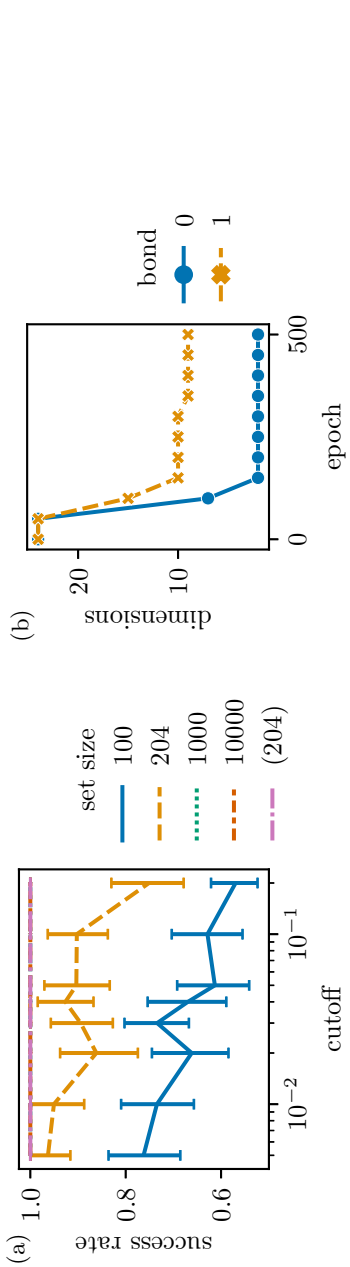


Figure 4.8: (a) Success rate obtained using single-site updates as a function of cutoff and data set size. Each point reflects the average across 20 models of the success rate over 100 runs. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.4.1.2. (b) Evolution of bond dimensions during training with single-site updates for a model with $\varepsilon = 0.01$ trained on the engineered data set. Epochs beyond 500 were omitted, since there was no more change.

Performance We first show the performance when the model is trained with only two-site updates and without single-site updates. Figure 4.4a shows the average success rate obtained across 20 models. Error bars represent the between-model 95% confidence intervals computed through bootstrapping. The figure suggests that smaller cutoffs ε lead to higher success rates. Indeed, we expect that a model with a smaller cutoff retains more information. Further, small randomly generated data sets lead to lower success rates, even at smaller cutoffs. This is likely due to the small randomly generated data sets not containing enough samples to completely capture the dynamics of the environment. As a consequence, these models never learned useful dynamics that lead to the reward with certainty. Note that the (small) engineered data set follows the general trend of the larger data sets and did attain a success rate of 100% at smaller cutoffs. A trend which continues throughout the rest of the experiments.

A model's ability to capture the underlying distribution of the data can be evaluated through the likelihood. Data points which appear in the data set should have a high likelihood, while data points which are not in the data set should have a low likelihood. As such, a model's average likelihood over all possible data points (in this case, sequences) should be high, or equivalently, its negative log likelihood should be low. Figure 4.4b shows the average negative log likelihood evaluated over all possible sequences. The theoretical minimum for this data set is $-\log(1/204) \approx 5.32$, since every sequence is equally likely. In general, the negative log likelihood decreases with smaller cutoffs. Once again, the models perform worse for small data set sizes, since small randomly generated data sets are less likely to contain all possibles sequences. Furthermore, if the randomly generated data set is not large enough, smaller cutoffs can become detrimental. This is likely because the model starts to overfit as more parameters become available (through the smaller cutoff). Note that models trained on the engineered data set approach the theoretical minimum at smaller cutoffs.

Again, we expect that a model with a smaller cutoff retains more information. In this case, information comes in the form of parameters. Logically, decreasing the cutoff leads to more singular values being retained after SVD, which in turn leads to more parameters in the model. This effect is shown in Figure 4.5a.

Since the number of parameters in the model is proportional to the bond dimensions, we can see how the model size changes during training. Figure 4.5b shows how the number of bond dimensions evolved during training for a specific model with $\varepsilon = 0.01$ trained on the engineered data set. The bond dimensions first increased greatly from the initial value of 2 as the model learned all the information, after which the bond dimensions

decreased again as the model learned to generalize better. Moreover, bond 0 was smaller than bond 1, which indicates that observation o_2 contained more information than observation o_1. Indeed, observation o_1 was always the initial position, while observation o_2 could be any position.

Beliefs When zooming in on a single well-performing model, one expects to see some shifts in the probability distributions provided by the tensor network before and after having seen the cue. In other words, one expects to see a belief shift in the active inference agent. More concretely, while the agent must be certain of its position both before and after having seen the cue, it must be uncertain of the reward it will receive before having seen the cue, but certain after having seen the cue. Figure 4.6 shows this belief shift for a model trained with $\varepsilon = 0.01$ trained on the engineered data set. The vertical axis indicates the next action, such that each row is a probability distribution over a certain modality of the next observation. In other words, each row indicates the agent's belief about the next observation given the next action. For example, before having seen the cue, the agent expects that, if it were to perform the action "right", it will almost certainly end up in the right branch, it will obtain a reward or a loss with approximately 50% chance each and it will observe the context right or left with 50% chance each. On the other hand, after having seen the cue "right", the agent expects that, if it were to perform the same action, it will almost certainly end up in the "right" branch, but it will obtain the reward almost certainly and it will again observe the context right or left with 50% chance each.

While the results for the context observation in this example may seem counter-intuitive, keep in mind that the agent can only observe the context whenever it is in the cue location. The figure shows the agent's beliefs about the *next observation* and does not directly reflect the agent's beliefs about the state of the environment. Before having seen the cue, the agent believes it can observe either context with a 50% chance. When standing in the cue location, the agent knows which context it will observe if it remains in the cue location. When moving towards another location, the agent knows it will once again observe either context with a 50% chance, since only the cue position gives a coherent context observation.

The expected surprisal (Eq. (4.14)) can provide some insight on how the agent behaves given the preferences in Eq. (4.18) and how it reaches the reward. Figure 4.7a shows the expected surprisal per action for the first action. Since the expected surprisal for going to the cue is lowest, the agent will go to the cue location. Having observed the cue "left" or "right", the expected surprisal per action for the second action now takes the form of Figures 4.7b and 4.7c, respectively. In these cases, it is clear the agent will

(a) default 4×4 (b) custom 3×3

Figure 4.9: The Frozen Lake environment [6].

choose to go to the right and left branch, respectively. Indeed, this is the optimal sequence of actions which optimizes its reward, and the expected behavior for the agent.

Single-site updates Figure 4.8a shows the success rates obtained when using single site updates. In this case, two-site updates were performed every 10th epoch. This way, the model received 9 epochs to converge and 1 epoch to adjust the bond dimensions. The initial bond dimension was set to the observation dimension, i.e. 24. These simple changes drastically improved success rates for higher cutoffs when compared with Figure 4.4a.

Figure 4.8b shows the evolution of the bond dimensions when using single-site updates. In this case, the bond dimensions remain relatively large for a short time until the model learns to generalize. Note that the bond dimensions were initialized at 24 this time, since it was known from the models using only two-site updates that the bond dimensions would quickly grow in the beginning. One can see from this figure that the model learned to generalize sooner than in the no single-site updates case (cf. Figure 4.5b). Additionally, the largest bond dimension was smaller (9 dimensions after 500 epochs) than in the no single-site updates case (17 dimensions after 500 epochs). Indeed, this suggests that the model has more difficulties generalizing when the bond dimensions are constantly allowed to vary.

4.4.2 Frozen Lake

The default Frozen Lake environment consists of a 4×4-grid which contains an agent e.g. an elf, a reward e.g. a present and some holes (Figure 4.9a). The agent always starts in the top left corner and must reach the reward in the bottom right corner without falling into a hole. If it falls into a hole, it is trapped and cannot leave. The agent can move left, down, right

or up one step at a time. However, the floor is slippery and, as a result, the agent has a chance to move perpendicular to the intended direction. For example, if the agent wants to move down, it has a $\frac{1}{3}$ probability of moving down, a $\frac{1}{3}$ probability of moving left and a $\frac{1}{3}$ probability of moving right[2].

It is possible to design custom environments, such as the 3×3-grid in Figure 4.9b, where the rules are the same as for the default environment. Moreover, it is possible to make the floor non-slippery, such that the environment becomes deterministic.

4.4.2.1 Model

For the Frozen Lake, the model (Eq. (4.12)) was adapted to the environment in a similar way as was done for the T-maze. The minimum length of the network was set to equal the initial time step plus the minimum number of steps required to reach the goal (4 for the custom environment, 6 for the default). However, we also introduced a hyperparameter which controls the number of additional steps beyond that for reasons that will become clear in the next section. Once again, one-hot encodings were the feature maps of choice. This means the observation feature map o_t has 9 dimensions for the 3×3-grid and 16 dimensions for the 4×4-grid, and the action feature map a_t has 4 dimensions.

4.4.2.2 Hyperparameters

Since the T-maze experiments suggest that a larger randomly generated data set leads to better performance and it is not feasible to engineer a data set for the Frozen Lake environment, we fixed the data set size to 100096 for this experiment. This number was chosen in order to make sure that a randomly generated data set on the 4×4 environment would contain enough samples in which the goal is reached. Moreover, this number is the first multiple of 256 larger than 10^5. Each model was trained with a batch size of 256 and a learning rate of 10^{-3}.

Important factors that could affect the performance of the model were cutoff ε and MPS length. The latter can affect performance because sequences in the data set are randomly generated. Longer sequences have a greater chance of making it to the reward position, which is important for the model to learn how to reach the reward.

Cutoff Four different cutoffs ε were compared: 0.1, 0.05, 0.03, and 0.01.

[2]If the agent hits the edge of the grid, it will not move, although it counts as having performed an action.

Length Different lengths of MPS were compared:

- for the non-slippery 3×3 environment: 5, 6, and 7.
- for the slippery 3×3 environment: 8, 10, and 12.
- for the non-slippery 4×4 environment: 7, 8, and 9.
- for the slippery 4×4 environment: 12.

Once again, we did not observe an effect of initialization strategy on results.

There were only a few exceptions to the hyperparameters described above. The learning rate was reduced to 5×10^{-4} whenever training became too unstable. This was only the case for the models with a specific initialization, cutoff 0.1 and MPS length 9 on the non-slippery 4×4 environment. In addition, the data set size for the model trained on the slippery 4×4 environment was increased by approximately a factor of 10, to 1000192 (another multiple of 256), such that it would contain more samples which reached the goal.

4.4.2.3 Results

To evaluate the performance of each model, we measured the success rate using the action selection scheme presented in Section 4.3.3. Here too, the preferred distribution $P(o_t \,|\, C)$ remained constant for every t, such that for the 3×3 environment (cf. Fig. 4.9b)

$$P(o \,|\, C) = \sigma(C) \tag{4.19}$$

$$C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & 20 \end{bmatrix} \tag{4.20}$$

and for the 4×4 environment (cf. Fig. 4.9a)

$$P(o \,|\, C) = \sigma(C) \tag{4.21}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -5 & 0 & -5 \\ 0 & 0 & 0 & -5 \\ -5 & 0 & 0 & 20 \end{bmatrix} \tag{4.22}$$

where σ is the softmax function and C encodes the agent's preference for each position within the environment. Once again, since the choice of weights raises a research question in and of itself, we simply selected a set of weights which worked for our purposes. Here, the weights were adjusted compared to

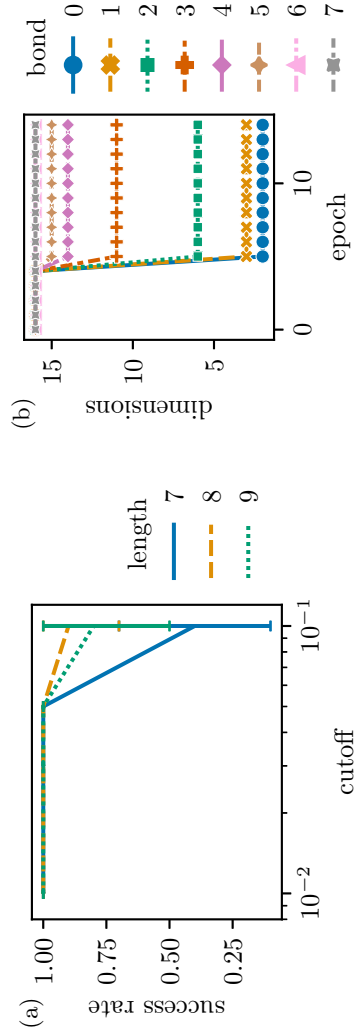


Figure 4.10: (a) Success rate for the non-slippery 4×4 environment as a function of cutoff and MPS length. Each point reflects the model average across 3 models. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.4.2.2. (b) Evolution of bond dimensions during training for a model with $\varepsilon = 0.03$ and MPS length 9 on the non-slippery 4×4 environment.

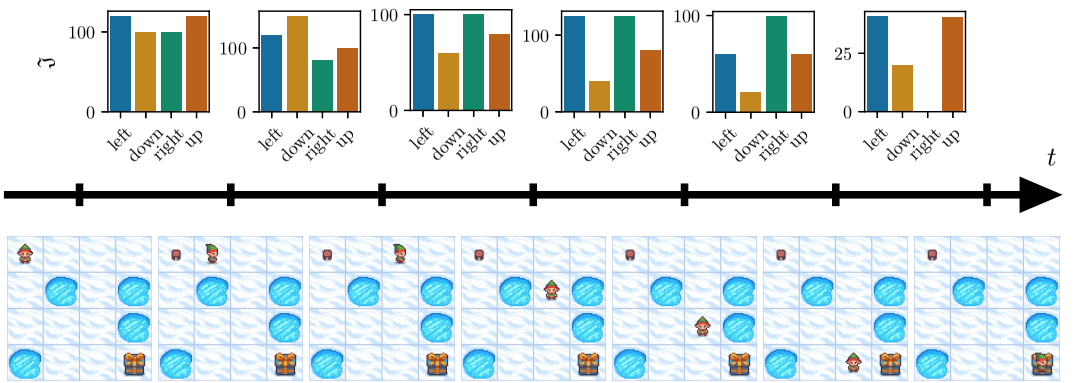
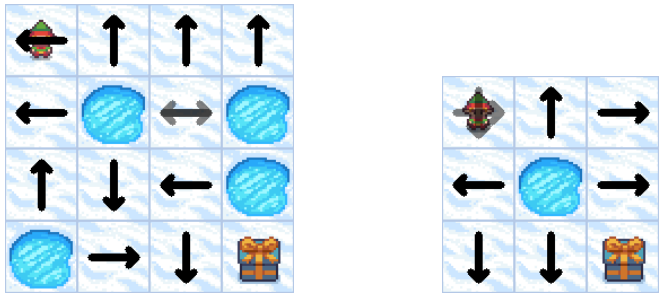


Figure 4.11: An example of an agent navigating the Frozen Lake environment. The agent's model was trained using $\varepsilon = 0.03$ and an MPS length of 9. The bottom row shows the observation at each time step, while the top row shows the expected surprisal per action computed after each observation.



(a) default 4×4 (b) custom 3×3

Figure 4.12: Optimal policies for the slippery Frozen Lake environments obtained through dynamic programming [51]. Policies are optimal in the sense that they provide the shortest path while having the lowest possible probability of falling into a hole. A run terminates when the agent reaches a hole or the reward, therefore actions in these positions have been omitted. Overlapping actions indicate both actions are equally viable.

the T-maze experiments, because of the larger position space in the Frozen Lake compared to the reward space in the T-maze and the larger horizon in the Frozen Lake. The weights represents the agent's fear of falling in holes (weighted by -5) and desire to reach the reward (weighted by 20). During a run, a success (agent obtained the reward) was indicated by 1, while a failure (agent was unable to reach the reward) was indicated by 0.

The single-site update method was used by default with two-site updates performed every 5 epochs. Using only two-site updates caused rapidly increasing bond dimensions. This was especially problematic on the slippery environments, e.g. the largest bond went from 2 to over 120 in a single epoch in the 4×4 environment. Training required either very long training times with small batches or very large memory (a batch of 256 two-site gradient tensors with a bond dimension of 64 on the 4×4 environment requires more than 68 GB). Therefore, results using only two-site updates will not be reported.

Non-slippery 3×3 The success rate for all models on the non-slippery 3×3 environment was 100% regardless of hyperparameters. Since the models were trained using single-site updates and the environment was very simple, even models with a large cutoff were able to achieve 100% success rate.

Non-slippery 4×4 Figure 4.10a shows the success rate for models with different hyperparameters on the non-slippery 4×4 environment. Error bars represent the 95% confidence intervals computed through bootstrapping.

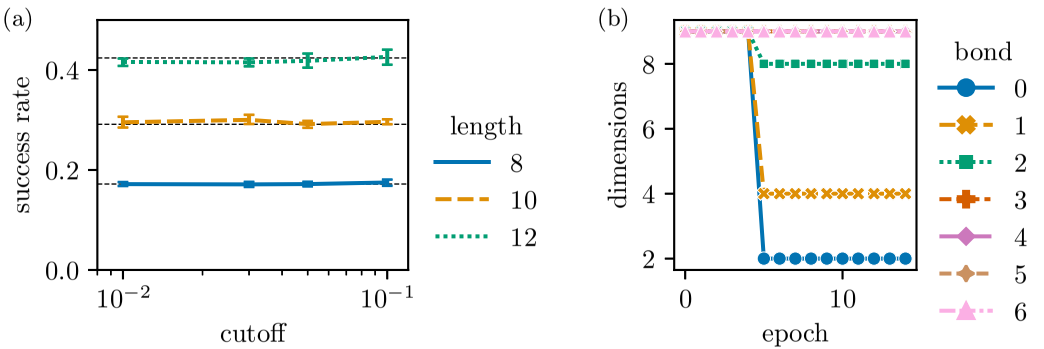


Figure 4.13: (a) Success rate obtained for the slippery 3×3 environment as a function of cutoff and MPS length. Each point reflects the average across 5 models of the success rate across 1000 runs. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.4.2.2. Horizontal lines indicate success rates for agents running the optimal policies (figure 4.12b) truncated at horizons corresponding to the lengths of the MPS, obtained through simulation. (b) Evolution of bond dimensions during training for a model with $\varepsilon = 0.03$ and MPS length 8 on the slippery 3×3 environment.

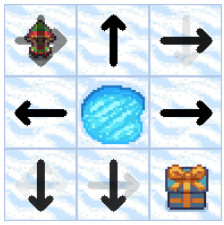


Figure 4.14: Policy obtained for the slippery 3×3 Frozen Lake environment for a model with $\varepsilon = 0.01$ and MPS length 12, by counting how often an action was taken at each position. The transparency of the arrow indicates the frequency of the action (less transparency is more frequent).

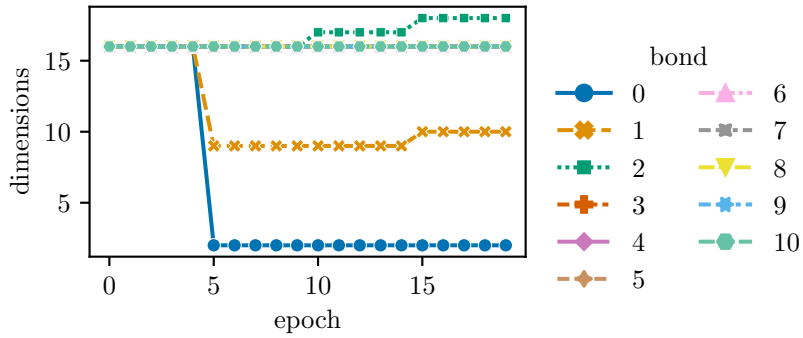


Figure 4.15: Evolution of bond dimensions during training for a model with $\varepsilon = 0.03$ and MPS length 12 on the slippery 4×4 Frozen Lake environment.

Once again, since the models were trained using single-site updates and the environment was very simple, even models with larger cutoffs were able to achieve 100% success rate, although some models failed when the cutoff was 0.1. The success rates for models with cutoff 0.1 may suggest that a longer MPS length leads to higher success rates. However, too few data were gathered to confirm this.

Figure 4.10b shows the evolution of the bond dimensions for a model with cutoff 0.03 and MPS length 9. The model starts with all bond dimensions equal to the observation dimension, i.e. 16. After training for 4 epochs using single-site updates, the model performs one epoch of two-site updates, which results in reduced bond dimensions for some of the bonds. Interestingly, the bond dimensions increase in order of bond position within the MPS. This is expected, since the agent always starts in the same top left position within the environment and is able to reach more positions as time goes by.

Figure 4.11 shows an example of how an agent can act within the non-

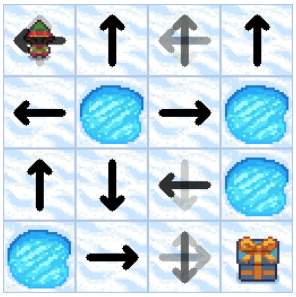


Figure 4.16: Policy obtained for the slippery 4×4 Frozen Lake environment for a model with $\varepsilon = 0.03$ and MPS length 12, by counting how often an action was taken at each position. The transparency of the arrow indicates the frequency of the action (less transparency is more frequent).

slippery 4×4 Frozen Lake environment. The agent starts out in the top left corner as usual. The top row shows the evolution of the expected surprisal over time as the agent takes actions. The bottom row shows the environment at each time step. Ultimately, the agent reaches the reward by assuming the action with the lowest expected surprisal at every step.

Slippery 3×3 Evaluating the success rate of an agent with a finite horizon on an environment with a potentially infinite time horizon can be tricky. In this case, we compared the success rate obtained by the surprisal agent with the success rate obtained by an agent which follows the optimal policy displayed in figure 4.12b truncated to the length of the MPS. The latter is optimal in the sense that it has the highest probability of reaching the goal in a small amount of steps while having the lowest probability of falling into a hole. Figure 4.13a shows the success rate obtained for models with different hyperparameters on the slippery 3×3 environment. As expected, the success rate increases with the length of the MPS. A longer MPS allows the agent to perform more steps in the environment. On the other hand, cutoff does not seem to have any effect on the success rate. Note that we do find different states with different bond dimensions for different values of the cutoff, the difference simply has no bearing on the success rate. The surprisal agents reach the success rate of the optimal agents.

Figure 4.13b shows the evolution of the bond dimensions for a model with cutoff 0.03 and MPS length 8. Similar to the previous case, the model starts with all bond dimensions equal to the observation dimension, i.e. 9. Once again, after training for 4 epochs using single-site updates, the model performs one epoch of two-site updates, which results in reduced bond dimensions for some of the bonds. Also here, the bond dimensions increase

in order of bond position within the MPS.

Figure 4.14 provides an example of the policy followed by one of the models with $\varepsilon = 0.01$ and MPS length 12. This policy was constructed by counting how often an action was taken at each position. Note that while this is very close to the optimal policy (figure 4.12b), there are a few slight discrepancies. These can be attributed to both the finite horizon of the model and the weight of the reward in the preferred distribution in combination with slight imbalances in the transition probabilities (as the probabilities are never *exactly* 1/3):

1. When close to the end of the model's finite horizon and *if the goal cannot be reached* anymore in the remaining number of steps, the model will continue performing the actions with the lowest expected surprisal. These actions, however, will not necessarily bring it closer to the goal. Instead, it will focus on fulfilling its other preferences, i.e. not falling into a hole. This means, depending on its current position, the agent will either try to avoid the hole or, when not in danger of falling into the hole, simply perform the action it perceives as having the lowest expected surprisal. The latter solely depends on the transition probabilities, however small the differences may be (again, the probabilities are never *exactly* 1/3), and can therefore appear quite arbitrary. This effect can be seen in the top right and bottom left corners. In these positions, the agent sometimes performs the 'down' and 'left' actions, respectively, for this reason.

2. Since the positive weight of the reward heavily outweighs the negative weight of falling into a hole in this case, it may happen that the agent's 'desire' to reach the reward drowns out its 'fear' of falling into a hole. This can happen when close to the end of the model's finite horizon and *if the goal can still be reached*. In addition, for this to happen, the imbalance in transition probabilities must be a bit larger, such that the agent believes that one action is more likely than the others to bring it to the goal. This happens in the bottom middle position. Here, the agent may in some cases perform the 'right' action, instead of the optimal 'down'. For reference, this can already happen when the transition probability of going to the goal from the bottom middle position with the 'right' action is 41%. Such a difference is generally no problem, except in edge cases like this one. In fact, under normal circumstances, the transition probabilities have to be much more imbalanced before the method breaks down.

Indeed, reducing the weight of the reward does not reduce the occurrence of the first situation, but does reduce the occurrence of the second situation.

Learning the preferred distributions as in Sajid et al. [58] may be helpful for this. Also, switching to a model with infinite horizon would do away with both situations.

Slippery 4×4 An increase in the size of the environment entails an increase in the depth of the decision tree defined by Eq. (4.14). While the 3×3 environment only requires a depth of 4, the 4×4 environment requires a depth of 6. Since the tree can be pruned during computation to reduce the complexity, the greater depth was no issue for the deterministic case. However, despite pruning, the size of the tree increased a lot more rapidly for the nondeterministic case, since the number of possible paths grew exponentially with the number of positions the agent can slip towards, which made evaluation of these models a lot more time consuming. For this reason, we only show results for a single model trained with $\varepsilon = 0.03$ and MPS length 12.

Figure 4.15 shows the evolution of the bond dimensions for this model. The model starts with all bond dimensions equal to the observation dimension, i.e. 16. After training for 4 epochs using single-site updates, the model performs one epoch of two-site updates, which results in reduced bond dimensions for the first two bonds. During the following epochs of two-site updates, both bond 1 and 2 increase slightly. Most of the bonds remain at 16 dimensions. It is possible that training longer (more epochs) would further increase some of the bond dimensions. However, the agent's desired behavior had already emerged at this point.

Figure 4.16 shows the policy the model followed. Once again, this policy was constructed by counting how often an action was taken at each position. And again, it is very close to the optimal policy (figure 4.12a) with a few slight discrepancies. These discrepancies can be attributed to the same situations described in the previous paragraph for the slippery 3×3 environment. The agent performs the 'up' action in the top left corner when the remaining number of steps is less than 6. This corresponds to the first situation where the goal cannot be reached anymore. In that case, the agent will perform the 'up' action in order to avoid falling into a hole and ignore the goal. Similarly, the agent occasionally performs the 'left' and 'down' actions in the third square in the top row when the remaining number of steps makes it unable to fall into a hole any longer. Finally, the 'up' and 'right' actions in the third square in the bottom row can be explained through the second situation.

4.5 Discussion

4.5.1 Scalability

As it stands, even though the MPS models are rather lightweight (of the order of kB for the T-maze and MB for the Frozen Lake), one of the main concerns regarding this method is computational complexity. Both training and action selection become slower for large action and observation spaces, as well as for long sequences. The largest bottleneck in terms of training is the singular value decomposition that occurs at every bond. This underlines the importance of choosing the right cutoff. The cutoff must be small enough to minimize the number of parameters in the model in order to speed up computation, while being able to solve the task at hand. No other parameter can directly affect the speed of the SVD. The largest bottleneck in terms of action selection is tree depth. The deeper the tree, the more computationally expensive action selection becomes.

Nevertheless, these concerns may simply be teething problems. As a matter of fact, we have already shown a first improvement in the training process by performing single-site updates and only occasionally doing the two-site updates. Single-site updates do not require SVD and resulted in a considerable speedup, in addition to requiring less memory. We note that the benefits of single-site updates are also known from quantum many-body physics [76]. Other possible speedups exist in the algorithms for building and aggregating of the expected surprisal tree. The choice of loss function could have an impact due to the application of active inference. For example, optimizing conditional probabilities directly instead of joint probabilities, may allow the system to converge more rapidly towards a usable model for active inference, since the active inference algorithm requires mostly conditional probabilities. As such, these and other improvements may reduce the required resources in the future and are avenues for future work.

As showcased by the Frozen Lake environment, an important limitation of the current method is the length of the MPS. In the current form, environments with an infinite time horizon, in particular environments where randomness plays an important role, cannot always be solved within the given time limit. A possible solution to this is the use of MPS variants, such as the uniform MPS [68], i.e. an infinite MPS made up of a unit cell of MPS tensors that are infinitely repeated, as demonstrated by Miller et al. [49]. Such an approach comes with the advantage that it is more efficient in the sense that only the tensors comprising the unit cell need to be specified, instead of one MPS tensor per time step, which additionally reduces the chances of overfitting. This, too, is an avenue for future work.

Further, the method in this work takes a rather different approach to

scaling model size with information theoretical requirements when compared to traditional machine learning. For example, in deep learning, the typical modus operandi is to design a model which is large enough to contain all the necessary information, and subsequently prune, i.e. scale down, the model until a balance point is reached between performance and model size, a.k.a. model reduction. In this work, the model generally starts out small and grows depending on how much information must be retained, i.e. the MPS starts with small bond dimensions which are allowed to grow based on a prespecified cutoff value. One could argue that the latter makes use of resources more efficiently.

4.5.2 Data and exploration

In current machine learning literature, the importance of the data is often understated. Much research focuses on creating better models on specific benchmarks, while it is sometimes forgotten that the model always reflects the data, as evidenced by modern challenges such as fairness in AI. While most of the data in this experiment is randomly generated, the T-maze experiments do emphasize the importance of the data. Remark that the engineered data set of size 204 performed better than the randomly generated data set of size 204. By simply constructing a data set in a clever way, the model was able to perform as well as larger data sets (at least for smaller cutoffs).

Clever ways of constructing data sets come up very often in reinforcement learning in the form of "exploration". In brief, exploration is used to generate sequences which are more informative than the sequences that the agent has already seen. For example, sparse reward problems in reinforcement learning are problems which are typically difficult to solve through random movements for the reason that a positive reward signal may be very difficult to find. In that case, exploration algorithms can lead the agent through sequences that would be underexplored by a random agent. Exploration essentially creates a richer data set that the agent can learn from.

Active inference intrinsically performs exploration through the free energy. The free energy takes care of exploration through the information gain term (Eq. (4.11)), which arises through variational Bayesian methods. This means that a true active inference agent will trade off exploration and exploitation while navigating an environment in real time. Although directly minimizing surprisal, as performed in this work, does away with the information gain term, further research is required to know how an agent based on our model would behave in situations that require this kind of exploration. It is important to note that machine learning implementations of active inference, such as deep active inference (i.e. the practice of replacing certain probability

distribution by neural networks), usually suffer from the same issue and do not use this term to generate training sequences, but instead use it at inference time to gather information when the environment is stochastic. The typical procedure is similar to the procedure described in this work: first gathering training data (often through demonstration), then learning a generative model, and finally using the model for inference. However, more relevantly to the initial discussion, it may be beneficial to generate training data through some kind of explorative agent rather than a random agent in order to enrich the data set. Indeed, one could go a step further and devise an online algorithm which incorporates exploration.

4.6 Conclusion and future work

We have shown that by combining tensor networks and active inference, it is possible to plan paths within discrete environments. In this work, we employed matrix product states for the generative model in an active inference agent. This allowed us to minimize surprisal directly instead of variational free energy. Experimentation on the T-maze and Frozen Lake environments confirmed that the MPS-based agent is able to select actions in order to realize its preferred observation, i.e. obtain a reward. Hyperparameters which had an effect on the success rate were the cutoff and data set size in the T-maze environment, and the cutoff and MPS length in the Frozen Lake environment. The effect of the cutoff was reduced greatly when utilizing single-site updates. Results showed that the MPS-based agents were able to perform as well as an optimal agent.

This work allows for several future extensions. First is the extension of the current algorithm to infinite time horizons, which would allow the use of active inference with tensor networks on a broader set of applications. More concretely, this would entail the use of a uniform MPS, i.e. an infinite MPS made up of an infinitely repeated unit cell of MPS tensors. Along the same lines, since the current algorithm was implemented for discrete environments, the application to continuous environments is an interesting prospect. This requires the mapping of continuous variables to the physical space of the MPS, which may be done through a clever feature map. Second are the several possible algorithmic improvements to both the model and action selection, such as the choice of loss function and perhaps a quantum mechanical formulation of the expected surprisal as a replacement to the current tree search formulation. Third is improving the data set and investigating how to reimplement “exploration”.

Acknowledgements

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. This work has received support from the European Union's Horizon 2020 program through Grant No. 863476 (ERC-CoG SEQUAM). Frozen lake assets provided by: `https://franuka.itch.io/rpg-snow-tileset` for elf and stool, Mel Tillery `http://www.cyaneus.com/` for all other assets.

Data availability statement

The data that support the findings of this study are openly available. The code can be found at `https://github.com/SWauthier/aif-mps`. The trained models can be found at `https://cloud.ilabt.imec.be/index.php/s/g4jGBY5EHMft9kF`.

4.7 References

[1] Aizpurua, B., Palmer, S., and Orus, R., "Tensor Networks for Explainable Machine Learning in Cybersecurity." 2024. arXiv:2401.00867 [cs.LG].

[2] Anshu, A., Harrow, A. W., and Soleimanifar, M., "Entanglement spread area law in gapped ground states," *Nature Physics* **18** no. 11, (Sep. 2022) 1362–1366.

[3] Arad, I., Landau, Z., and Vazirani, U., "Improved one-dimensional area law for frustration-free systems," *Physical Review B* **85** no. 19, (May 2012) 195145.

[4] Arad, I., Kitaev, A., Landau, Z., and Vazirani, U., "An area law and sub-exponential algorithm for 1D systems." 2013. arXiv:1301.1162 [quant-ph].

[5] Brandão, F. G. S. L. and Horodecki, M., "An area law for entanglement from exponential decay of correlations," *Nature Physics* **9** no. 11, (Sep. 2013) 721–726.

[6] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym." 2016. arXiv:1606.01540 [cs.LG].

[7] Chen, J., Cheng, S., Xie, H., Wang, L., and Xiang, T., "Equivalence of restricted Boltzmann machines and tensor network states," *Physical Review B* **97** no. 8, (Feb. 2018) 085104.

[8] Cheng, S., Wang, L., Xiang, T., and Zhang, P., "Tree tensor networks for generative modeling," *Physical Review B* **99** no. 15, (Apr. 2019) 155131.

[9] Cheng, S., Wang, L., and Zhang, P., "Supervised learning with projected entangled pair states," *Physical Review B* **103** no. 12, (Mar. 2021) 125117.

[10] Cirac, J. I., Pérez-García, D., Schuch, N., and Verstraete, F., "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems," *Reviews of Modern Physics* **93** no. 4, (Dec. 2021) 045003.

[11] Cohen, N., Sharir, O., and Shashua, A., "On the Expressive Power of Deep Learning: A Tensor Analysis," in *29th Annual Conference on Learning Theory*, Feldman, V., Rakhlin, A., and Shamir, O., eds.,

vol. 49 of *Proceedings of Machine Learning Research*, pp. 698–728. PMLR, Columbia University, New York, New York, USA, 23–26 jun 2016.

[12] Dalzell, A. M. and Brandão, F. G. S. L., "Locally accurate MPS approximations for ground states of one-dimensional gapped local Hamiltonians," *Quantum* **3** (Sep. 2019) 187.

[13] Eckart, C. and Young, G., "The approximation of one matrix by another of lower rank," *Psychometrika* **1** no. 3, (Sep. 1936) 211–218.

[14] Fannes, M., Nachtergaele, B., and Werner, R. F., "Finitely correlated states on quantum spin chains," *Communications in Mathematical Physics* **144** no. 3, (Mar. 1992) 443–490.

[15] Fields, C. and Levin, M., "Metabolic limits on classical information processing by biological cells," *Biosystems* **209** (Nov. 2021) 104513.

[16] Fields, C., Fabrocini, F., Friston, K., Glazebrook, J. F., Hazan, H., Levin, M., and Marcianò, A., "Control Flow in Active Inference Systems—Part I: Classical and Quantum Formulations of Active Inference," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* **9** no. 2, (June 2023) 235–245.

[17] Fields, C., Fabrocini, F., Friston, K., Glazebrook, J. F., Hazan, H., Levin, M., and Marcianò, A., "Control Flow in Active Inference Systems—Part II: Tensor Networks as General Models of Control Flow," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* **9** no. 2, (June 2023) 246–256.

[18] Flatiron Institute, "The Tensor Network,". `https://tensornetwork.org/`. Last accessed 12 February 2023.

[19] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O'Doherty, J., and Pezzulo, G., "Active inference and learning," *Neuroscience & Biobehavioral Reviews* **68** (Sep. 2016) 862–879.

[20] Friston, K., Da Costa, L., Hafner, D., Hesp, C., and Parr, T., "Sophisticated Inference," *Neural Computation* **33** no. 3, (Mar. 2021) 713–763.

[21] Friston, K. J., Lin, M., Frith, C. D., Pezzulo, G., Hobson, J. A., and Ondobaka, S., "Active Inference, Curiosity and Insight," *Neural Computation* **29** no. 10, (Oct. 2017) 2633–2683.

[22] Gillman, E., Rose, D. C., and Garrahan, J. P., "A Tensor Network Approach to Finite Markov Decision Processes." 2020. arXiv:2002.05185 [cond-mat.stat-mech].

[23] Gillman, E., Rose, D. C., and Garrahan, J. P., "Combining Reinforcement Learning and Tensor Networks, with an Application to Dynamical Large Deviations," *Physical Review Letters* **132** no. 19, (May 2024) 197301.

[24] Glasser, I., Pancotti, N., August, M., Rodriguez, I. D., and Cirac, J. I., "Neural-Network Quantum States, String-Bond States, and Chiral Topological States," *Physical Review X* **8** no. 1, (Jan. 2018) 011006.

[25] Glasser, I., Sweke, R., Pancotti, N., Eisert, J., and Cirac, I., "Expressive power of tensor-network factorizations for probabilistic modeling," in *Advances in Neural Information Processing Systems*, Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., eds., vol. 32. Curran Associates, Inc., 2019.

[26] Glasser, I., Pancotti, N., and Cirac, J. I., "From Probabilistic Graphical Models to Generalized Tensor Networks for Supervised Learning," *IEEE Access* **8** (2020) 68169–68182.

[27] Guo, C., Jie, Z., Lu, W., and Poletti, D., "Matrix product operators for sequence-to-sequence learning," *Physical Review E* **98** no. 4, (Oct. 2018) 042114.

[28] Haegeman, J., Osborne, T. J., and Verstraete, F., "Post-matrix product state methods: To tangent space and beyond," *Physical Review B* **88** no. 7, (Aug. 2013) 075133.

[29] Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P., "Unsupervised Generative Modeling Using Matrix Product States," *Physical Review X* **8** no. 3, (July 2018) 031012.

[30] Hastings, M. B., "An area law for one-dimensional quantum systems," *Journal of Statistical Mechanics: Theory and Experiment* **2007** no. 08, (Aug. 2007) P08024–P08024.

[31] Hayden, P., Leung, D. W., and Winter, A., "Aspects of Generic Entanglement," *Communications in Mathematical Physics* **265** no. 1, (Mar. 2006) 95–117.

[32] Heins, C., Millidge, B., Demekas, D., Klein, B., Friston, K., Couzin, I. D., and Tschantz, A., "pymdp: A Python library for active inference

in discrete state spaces," *Journal of Open Source Software* **7** no. 73, (May 2022) 4098.

[33] Hobson, J. and Friston, K., "Waking and dreaming consciousness: Neurobiological and functional considerations," *Progress in Neurobiology* **98** no. 1, (July 2012) 82–98.

[34] Howard, S., "A Tensor Network Implementation of Multi Agent Reinforcement Learning." 2024. `arXiv:2401.03896 [cs.LG]`.

[35] Hur, Y., Hoskins, J. G., Lindsey, M., Stoudenmire, E. M., and Khoo, Y., "Generative modeling via tensor train sketching." 2023. `arXiv:2202.11788 [math.NA]`.

[36] Kerskens, C. M. and López Pérez, D., "Experimental indications of non-classical brain functions," *Journal of Physics Communications* **6** no. 10, (Oct. 2022) 105001.

[37] Khrulkov, V., Novikov, A., and Oseledets, I., "Expressive power of recurrent neural networks," in *International Conference on Learning Representations*. 2018.

[38] Klumper, A., Schadschneider, A., and Zittartz, J., "Equivalence and solution of anisotropic spin-1 models and generalized t-J fermion models in one dimension," *Journal of Physics A: Mathematical and General* **24** no. 16, (Aug. 1991) L955–L959.

[39] Klümper, A., Schadschneider, A., and Zittartz, J., "Matrix Product Ground States for One-Dimensional Spin-1 Quantum Antiferromagnets," *Europhysics Letters (EPL)* **24** no. 4, (Nov. 1993) 293–297.

[40] LeCun, Y., Cortes, C., and Burges, C. J. C., "THE MNIST DATABASE of handwritten digits." 1998. `http://yann.lecun.com/exdb/mnist/`. Last accessed 12 February 2023.

[41] Li, W., Ma, L., Yang, G., and Gan, W.-B., "REM sleep selectively prunes and maintains new synapses in development and learning," *Nature Neuroscience* **20** no. 3, (Jan. 2017) 427–437.

[42] Liu, D., Ran, S.-J., Wittek, P., Peng, C., García, R. B., Su, G., and Lewenstein, M., "Machine learning by unitary tensor network of hierarchical tree structure," *New Journal of Physics* **21** no. 7, (July 2019) 073059.

[43] Liu, J., Li, S., Zhang, J., and Zhang, P., "Tensor networks for unsupervised machine learning," *Physical Review E* **107** no. 1, (Jan. 2023) L012103.

[44] Liu, X.-Y. and Fang, Y. X., "Quantum Tensor Networks for Variational Reinforcement Learning," in *First Workshop on Quantum Tensor Networks in Machine Learning*. 2020.

[45] Lu, Y. and Ran, S.-J., "Many-body control with reinforcement learning and tensor networks," *Nature Machine Intelligence* **5** no. 10, (Oct. 2023) 1058–1059.

[46] Mahajan, A., Samvelyan, M., Mao, L., Makoviychuk, V., Garg, A., Kossaifi, J., Whiteson, S., Zhu, Y., and Anandkumar, A., "Tesseract: Tensorised Actors for Multi-Agent Reinforcement Learning," in *Proceedings of the 38th International Conference on Machine Learning*, Meila, M. and Zhang, T., eds., vol. 139 of *Proceedings of Machine Learning Research*, pp. 7301–7312. PMLR, 18–24 jul 2021.

[47] Mencia Uranga, B. n. and Lamacraft, A., "SchrödingerRNN: Generative modeling of raw audio as a continuously observed quantum state," in *Proceedings of the First Mathematical and Scientific Machine Learning Conference*, Lu, J. and Ward, R., eds., vol. 107 of *Proceedings of Machine Learning Research*, pp. 74–106. PMLR, 20–24 jul 2020.

[48] Metz, F. and Bukov, M., "Self-correcting quantum many-body control using reinforcement learning with tensor networks," *Nature Machine Intelligence* **5** no. 7, (July 2023) 780–791.

[49] Miller, J., Rabusseau, G., and Terilla, J., "Tensor Networks for Probabilistic Sequence Modeling," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, Banerjee, A. and Fukumizu, K., eds., vol. 130 of *Proceedings of Machine Learning Research*, pp. 3079–3087. PMLR, 13–15 apr 2021.

[50] Murg, V., Verstraete, F., Legeza, O., and Noack, R. M., "Simulating strongly correlated quantum systems with tree tensor networks," *Physical Review B* **82** no. 20, (Nov. 2010) 205105.

[51] Ng, R., "Dynamic Programming." 2020. `https://github.com/ritchieng/deep-learning-wizard/commits/master/docs/deep_learning/deep_reinforcement_learning_pytorch/dynamic_programming_frozenlake.ipynb`. Last accessed 19 December 2023.

[52] Orús, R., "Tensor networks for complex quantum systems," *Nature Reviews Physics* **1** no. 9, (Aug. 2019) 538–550.

[53] Oseledets, I. V., "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing* **33** no. 5, (Jan. 2011) 2295–2317.

[54] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, Mar. 2022.

[55] Peng, Y., Chen, Y., Stoudenmire, E. M., and Khoo, Y., "Generative Modeling via Hierarchical Tensor Sketching." 2023. `arXiv:2304.05305 [math.NA]`.

[56] Perez-Garcia, D., Verstraete, F., Wolf, M., and Cirac, J., "Matrix product state representations," *Quantum Information and Computation* **7** no. 5 & 6, (July 2007) 401–430.

[57] Ran, S.-J. and Su, G., "Tensor Networks for Interpretable and Efficient Quantum-Inspired Machine Learning," *Intelligent Computing* **2** (Jan. 2023) 0061.

[58] Sajid, N., Tigas, P., and Friston, K., "Active inference, preference learning and adaptive behaviour," *IOP Conference Series: Materials Science and Engineering* **1261** no. 1, (Oct. 2022) 012020.

[59] Schrödinger, E., *What Is Life? The Physical Aspect of the Living Cell*. Cambridge University Press, Cambridge, 1944.

[60] Schuch, N., Wolf, M. M., Verstraete, F., and Cirac, J. I., "Entropy Scaling and Simulability by Matrix Product States," *Physical Review Letters* **100** no. 3, (Jan. 2008) 030504.

[61] Sharir, O., Shashua, A., and Carleo, G., "Neural tensor contractions and the expressive power of deep neural quantum states," *Physical Review B* **106** no. 20, (Nov. 2022) 205136.

[62] Shi, Y.-Y., Duan, L.-M., and Vidal, G., "Classical simulation of quantum many-body systems with a tree tensor network," *Physical Review A* **74** no. 2, (Aug. 2006) 022320.

[63] Srinivasan, S., Gordon, G., and Boots, B., "Learning Hidden Quantum Markov Models," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, Storkey, A. and Perez-Cruz, F., eds., vol. 84 of *Proceedings of Machine Learning Research*, pp. 1979–1987. PMLR, 09–11 apr 2018.

[64] Stokes, J. and Terilla, J., "Probabilistic Modeling with Matrix Product States," *Entropy* **21** no. 12, (Dec. 2019) 1236.

[65] Stoudenmire, E. and Schwab, D. J., "Supervised Learning with Tensor Networks," in *Advances in Neural Information Processing Systems*, Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., eds., vol. 29. Curran Associates, Inc., 2016.

[66] Sun, Z.-Z., Peng, C., Liu, D., Ran, S.-J., and Su, G., "Generative tensor network classification model for supervised machine learning," *Physical Review B* **101** no. 7, (Feb. 2020) 075135.

[67] Tangpanitanon, J., Mangkang, C., Bhadola, P., Minato, Y., Angelakis, D. G., and Chotibut, T., "Explainable natural language processing with matrix product states," *New Journal of Physics* **24** no. 5, (May 2022) 053032.

[68] Vanderstraeten, L., Haegeman, J., and Verstraete, F., "Tangent-space methods for uniform matrix product states," *SciPost Physics Lecture Notes* (Jan. 2019) 7.

[69] Vanderstraeten, L., Haegeman, J., and Verstraete, F., "Simulating excitation spectra with projected entangled-pair states," *Physical Review B* **99** no. 16, (Apr. 2019) 165121.

[70] Verstraete, F. and Cirac, J. I., "Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions." 2004. `arXiv:cond-mat/0407066 [cond-mat.str-el]`.

[71] Vidal, G., "Entanglement Renormalization," *Physical Review Letters* **99** no. 22, (Nov. 2007) 220405.

[72] Vieijra, T., Vanderstraeten, L., and Verstraete, F., "Generative modeling with projected entangled-pair states." 2022. `arXiv:2202.08177 [quant-ph]`.

[73] Wang, M., Pan, Y., Xu, Z., Yang, X., Li, G., and Cichocki, A., "Tensor Networks Meet Neural Networks: A Survey and Future Perspectives." 2023. `arXiv:2302.09019 [cs.LG]`.

[74] Wauthier, S. T., Vanhecke, B., Verbelen, T., and Dhoedt, B., "Learning Generative Models for Active Inference Using Tensor Networks," in *Active Inference*, Buckley, C. L., Cialfi, D., Lanillos, P., Ramstead, M., Sajid, N., Shimazaki, H., and Verbelen, T., eds., pp. 285–297. Springer Nature Switzerland, 2023.

[75] White, S. R., "Density matrix formulation for quantum renormalization groups," *Physical Review Letters* **69** no. 19, (Nov. 1992) 2863–2866.

[76] White, S. R., "Density matrix renormalization group algorithms with a single center site," *Physical Review B* **72** no. 18, (Nov. 2005) 180403.

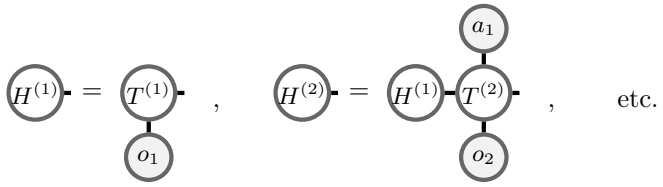
[77] White, S. R. and Huse, D. A., "Numerical renormalization-group study of low-lying eigenstates of the antiferromagnetic S=1 Heisenberg chain," *Physical Review B* **48** no. 6, (Aug. 1993) 3844–3852.

[78] Zauner-Stauber, V., Vanderstraeten, L., Haegeman, J., McCulloch, I. P., and Verstraete, F., "Topological nature of spinons and holons: Elementary excitations from matrix product states with conserved symmetries," *Physical Review B* **97** no. 23, (June 2018) 235155.

4.A Computing probabilities from an MPS

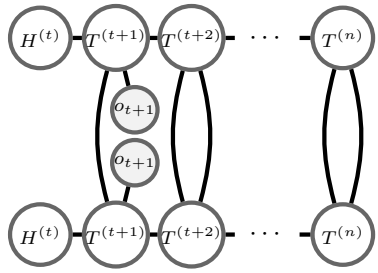
This section explains how to derive the probabilities that are required for the tree in section 4.3.3.

Firstly, note that any action taken by the agent and the resulting observation can be stored as a vector which contains the history and is updated at every time step:



Logically, this vector remains unchanged, since the past cannot be changed. Any probability is therefore calculated using this history vector, e.g.

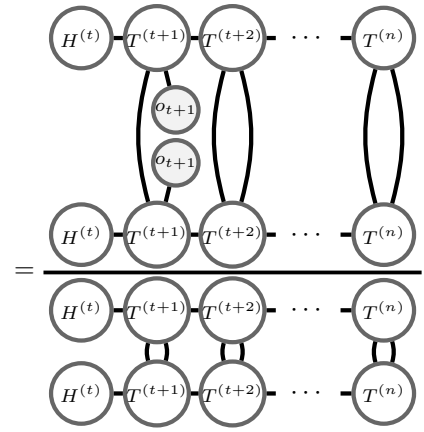
$$P(o_{t+1}, \tilde{o}_{<t+1}, \tilde{a}_{<t+1}) =$$



.

To obtain conditional probabilities, it is sufficient to apply the definition:

$$P(o_{t+1} \,|\, \tilde{o}_{<t+1}, \tilde{a}_{<t+1}) = \frac{P(o_{t+1}, \tilde{o}_{<t+1}, \tilde{a}_{<t+1})}{P(\tilde{o}_{<t+1}, \tilde{a}_{<t+1})}$$



.

— Even a soul submerged in sleep is hard at work and helps make something of the world.

Heraclitus

5

Active inference with uniform matrix product states

The necessity to extend the implementation in the previous chapter to infinite time horizons became evident rather quickly. The inability of the matrix product state formulation to deal with arbitrary time horizons was one of the main drawbacks. At the same time, the lack of access to latent states made computation of the expected surprisal very expensive.

Brainstorming sessions during two short research stays in Ghent and Vienna culminated in the solutions presented in this chapter. The use of uniform matrix product states extends the previous implementation to potentially infinite horizons and allows for an operator formulation of the expected surprisal to accelerate computation.

5.1 Introduction

One of the main drawbacks of the matrix product state (MPS) [7, 17, 8, 18, 6] implementation in Wauthier et al. [28] is the limited time horizon. Due to the fixed length of the MPS, environments with a potentially infinite time horizon can only be modeled up to a certain time point.

Being a class of translation-invariant MPS, uniform MPS (uMPS) [8, 14, 12, 27] offer an elegant solution to this problem. In short, a uMPS is an MPS where the same tensor (or series of tensors) is repeated ad infinitum. As a result, a uMPS should, given large enough bond dimensions, be able to model certain environments with an infinite time horizon. Crucially, the bond dimensions of a uMPS can vary dynamically during training, providing the same benefits as a regular MPS in terms of model size.

Active inference relies on a quantity called expected free energy for action selection [9, 22]. Expanding the expected free energy using the sophisticated inference scheme gives rise to a tree structure and allows planning multiple steps ahead [10]. One of the main drawbacks of the sophisticated formulation with MPS is that the number of leaves of the tree grow exponentially with the time horizon. Despite existing methods to prune the tree, this leads to heavy computational requirements in terms of both compute and memory.

Reformulating the expected surprisal in the language of quantum mechanics may offer a solution to this problem. For one, the surprisal lends itself to a relatively simple reformulation as an operator. For another, in combination with the uMPS formalism, the expected free energy can potentially be computed for infinite time horizons.

This work shows how to integrate uMPS into the active inference framework laid out by Wauthier et al. [28] and presents a novel method for action selection within this framework. Section 5.2 provides a mathematical description of the uMPS formalism and shows how it can be used as a generative model in active inference. Section 5.3 describes action selection in active inference, and shows how an operator formulation of expected surprisal can speed up action selection in active inference. Preliminary results are shown in Section 5.4. Finally, Section 5.5 explains the intended path forward.

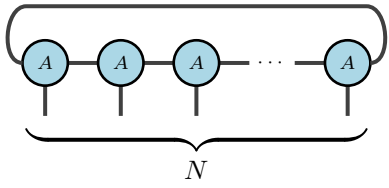
5.2 Extension 1: uniform matrix product states

To understand the use of uniform matrix product states (uMPS) in this work, it is crucial to first explore how this particular form of MPS is derived. The following section provides a formal development of the uMPS framework. Subsequently, its integration with active inference is demonstrated. Finally, the process for training uMPS within the context of active inference is

explained.

5.2.1 Background

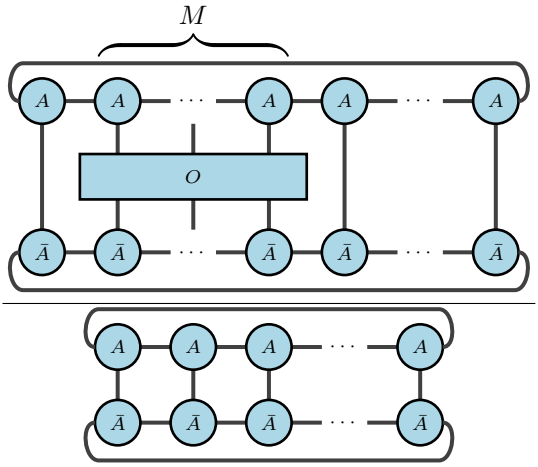
A uniform matrix product state (uMPS) is the thermodynamic limit of an MPS parameterized by a $D \times d \times D$-tensor A repeated N times, where D is the bond dimension and d is the physical dimension [14]. Diagrammatically,

$|\psi_N\rangle =$ 

, (5.1)

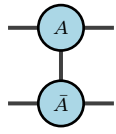
where the bra-ket notation $|\psi_N\rangle$ was used to indicate that the uMPS represents a quantum state.

In this formalism the expected value of a local operator $\hat{O}_M$ acting on M sites is given by

$\frac{\langle\psi_N|\hat{O}_M|\psi_N\rangle}{\langle\psi_N\,|\,\psi_N\rangle} =$ 

, (5.2)

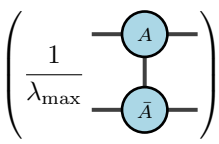
with $\bar{A}$ the complex conjugate of A. By performing an eigendecomposition of the transfer matrix



$= G\Lambda G^{-1}$, (5.3)

one obtains the square matrix G containing eigenvectors g_i and the diagonal matrix Λ containing eigenvalues λ_i. Without loss of generality, let the eigenvalues λ_i be ordered from largest to smallest. Dividing the transfer

matrix by the largest eigenvalue $\lambda_{\max}$ and contracting the resulting matrix N times,

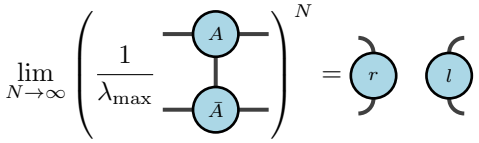


$$= G\left(\frac{\Lambda}{\lambda_{\max}}\right)^N G^{-1}\,, \tag{5.4}$$

ensures that

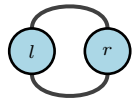
$$\left(\frac{\Lambda}{\lambda_{\max}}\right)^N \xrightarrow{N\to\infty} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}. \tag{5.5}$$

As a result, [27]



$$\lim_{N\to\infty} \left(\frac{1}{\lambda_{\max}}\right)^N = \,, \tag{5.6}$$

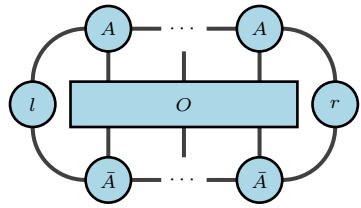
where r and l are the right and left eigenvectors associated with eigenvalue $\lambda_{\max}$, with [27]



$$= 1\,. \tag{5.7}$$

Now, rescaling $A \to \frac{1}{\lambda_{\max}} A$, it is clear that in the thermodynamic limit [27]

$$\lim_{N\to\infty} \frac{\langle\psi_N|\hat{O}_M|\psi_N\rangle}{\langle\psi_N\,|\,\psi_N\rangle} =$$

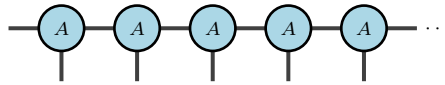


. (5.8)

Thus, it is possible to compute expectation values of local operators using the left and right dominant eigenvectors of the transfer matrix, when appropriately normalized. This only requires placing the left eigenvector l on the left, to represent the left side of the infinite tensor network, and placing the right eigenvector r on the right, to represent the right side.

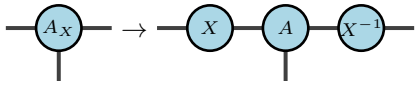
In the remainder of this work, the thermodynamic limit of a uMPS parameterized by tensor A will be written as

$$|\psi(A)\rangle =$$



. (5.9)

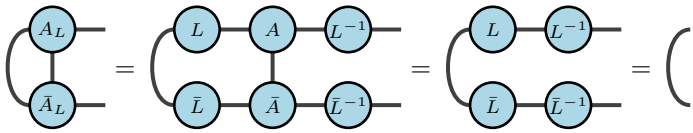
Any arbitrary MPS, or uMPS, possesses a property known as *gauge invariance*. This means that a gauge transformation



(5.10)

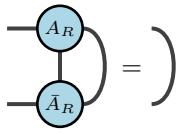
with some transformation matrix X leaves the state invariant [27]. This is a direct consequence of the fact that the tensor A uniquely defines the state, but the state does not uniquely define the tensor A [27]. Indeed, different tensors can define the same state.

This property is useful, because it allows fixing the gauge to obtain desired behavior. For example, it allows placing the uMPS in a gauge where one of the eigenvectors r or l (Eq. (5.6)) can be made trivial. For instance, it is possible to find a gauge transformation with transformation matrix $L = \sqrt{l}$ which places the tensor A in a form where [27]



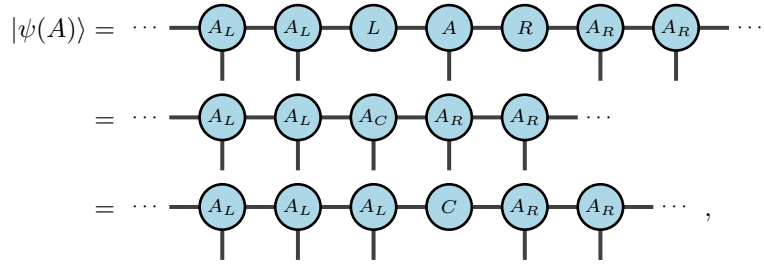
(5.11)

A uMPS placed in this form is called *left orthonormal*. An analogous transformation can bring the uMPS into *right orthonormal* form



(5.12)

In fact, it is possible to combine the left and right orthonormal forms into a form called the *mixed gauge*. By selecting a center tensor, and transforming all the tensors to left into left orthonormal form and all the tensors to the right into right orthonormal form, one can rewrite the uMPS in different forms [27]

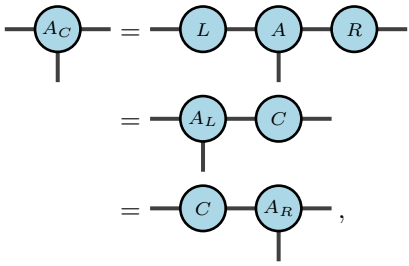


(5.13)

(5.14)

(5.15)

with $C = LR$ and



(5.16)

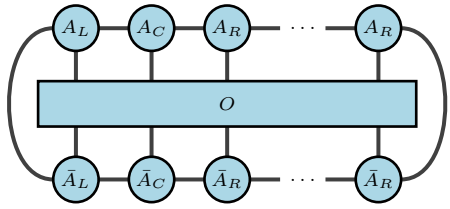
(5.17)

(5.18)

where, since $\langle \psi(A) \,|\, \psi(A) \rangle = 1$, it must be that $||A_C||_F = ||C||_F = 1$. The mixed gauge makes it more practical to compute certain quantities, such as expectation values. Moreover, the center matrix C makes it straightforward to perform a Schmidt decomposition in order to compute the entanglement entropy between two partitions of the system, or to truncate the bonds as to reduce the number of parameters in the model [27].

Assuming a properly normalized uMPS, the expectation value of an operator $\hat{O}$ can be computed through [27]

$\langle \psi(A) | \hat{O} | \psi(A) \rangle =$



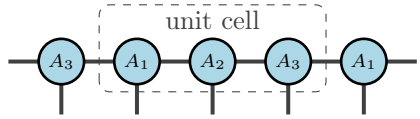
. (5.19)

Evidently, the placement of the center site is irrelevant to the final result and can be chosen per convenience.

While the infinite nature of a uMPS may look daunting, efficient variational techniques have been developed in order to optimize this type of tensor networks [31, 27]. Although infinite system extensions to the DMRG algorithm have been around for a long time [29], more recently, tangent space methods have proven to provide considerable increases in convergence speed and precision. For example, the variational uMPS (VUMPS) algorithm [31] is able to find ground states of quantum systems. Similarly, the time-dependent variational principle (TDVP) [19] for MPS [13] is able to both find ground states and simulate time evolution of quantum systems, depending on whether it is operating in real time or imaginary time.

Importantly, while the summarization in this section is valid for uMPS with a trivial *unit cell* consisting of a single tensor as in Eq. (5.9), it can be generalized to uMPS with a *unit cell* consisting of multiple tensors [27]. For example, one could write down a uMPS with a unit cell of three tensors A_1,

A_2 and A_3:

$$|\psi(A_1, A_2, A_3)\rangle =$$ 

, (5.20)

where A_1 is $D_1 \times d_1 \times D_2$-dimensional, A_2 is $D_2 \times d_2 \times D_3$-dimensional and A_3 is $D_3 \times d_3 \times D_1$-dimensional. All the algorithms in this section can be readily extended to the case where the uMPS contains such a repeating sequence of tensors of finite length [31, 27]. Note that a unit cell of length n, implies that each tensor A_i in the unit cell has an associated transformation matrix L_i, as well as R_i and C_i. For example, the state $|\psi(A_1, A_2, A_3)\rangle$ involves nine matrices: L_1, L_2, L_3, R_1, R_2, R_3, C_1, C_2 and C_3.

5.2.2 Integration with active inference

5.2.2.1 Generative model

An active inference agent is equipped with a generative model, i.e. a model of the world, which represents its beliefs about the world in terms of probability [9]. When appropriately normalized, a uMPS can serve as a generative model, allowing it to represent the agent's beliefs in the context of active inference. In this formalism, the agent's beliefs evolve by contraction with the uMPS's unit cell and its actions and observations at each time step.

Mathematically, a generative model is a joint probability distribution $P(X)$ over some random variables $X = (X_1, X_2, \ldots, X_n)$ [16]. In active inference, the agent's generative model typically consists of a probability distribution over observations o_t, hidden states s_t and actions a_t at different time steps t, $P(\tilde{o}, \tilde{s}, \tilde{a})$ where the tilde-notation indicates a sequence $\tilde{x} = (x_1, x_2, \ldots, x_n)$ over time [9]. When the agent's environment is presumed to be a Markov process, the generative model performs three main functions: (1) it contains the agent's current beliefs about the environment, $P(s_t)$; (2) it models the hidden dynamics of the world, $P(s_{t+1}|s_t, a_t)$; and (3) it determines how observations are generated from hidden states, $P(o_t|s_t)$. Moreover, it conveniently factors as

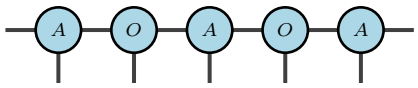
$$P(\tilde{o}, \tilde{s}, \tilde{a}) = P(\tilde{a}) \prod_{t=0}^{N} P(s_t|s_{t-1}, a_{t-1}) P(o_t|s_t)\,.$$

Fundamentally, a generative model can be represented by a quantum state by leveraging the Born rule [15]. Given a quantum state $|\psi\rangle$ and a set of outcomes $\mathcal{X}$, the probability of observing outcome $x \in \mathcal{X}$ can be obtained through

$$P(x) = \frac{|\langle x \,|\, \psi\rangle|^2}{Z}\,, \tag{5.21}$$

where $Z = \sum_{x \in \mathcal{X}} |\langle x \,|\, \psi \rangle|^2$ and $|x\rangle$ is a vector associated with outcome x, such that each distinct $|x\rangle$ is mutually orthogonal. This enables the application of techniques from quantum mechanics to the generative model through the underlying quantum state. Naturally, Eq. (5.21) implies that each outcome must be associated with a vector [26]. The way outcomes are mapped to vectors must be in accordance with certain rules, which will be addressed later in this section.

In this work, the active inference agent's beliefs are encoded by a quantum state $|\psi\rangle$ implemented as a uMPS, where the sites are correlated in *time* as opposed to space. The state is parameterized by a unit cell of two tensors A and O:

$$|\psi(A, O)\rangle = \cdots$$  $$\cdots \,,$$ (5.22)

where A is a $D_1 \times d_a \times D_2$-tensor associated with a d_a-dimensional action space, and O is a $D_2 \times d_o \times D_1$-tensor associated with a d_o-dimensional observation space, and D_1 and D_2 are bond dimensions.

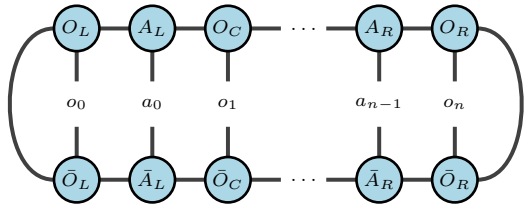
The action and observation spaces are constructed from the actions and observations defined by the agent's environment. Each possible action is mapped to a vector via a feature map $\phi^{(a)}(a_i)$, which takes an action a_i and outputs a vector. These vectors form the basis of the action space. The observation space is obtained in an analogous manner. To lend themselves to the generative interpretation defined by Eq. (5.21), the feature maps must be orthonormal $\sum_{a_i} \bar{\phi}_j^{(a)}(a_i)\phi_k^{(a)}(a_i) = \delta_{jk}$ and normalized $\sum_j |\phi_j^{(a)}(a_i)|^2 = 1$ [26].

When the actions and observations are discrete, it is convenient to use the Kronecker delta function as a feature map. That way, each physical dimension of tensor A corresponds to a specific action, analogously for tensor O. This makes it possible to simply select the desired indices when performing computations. For simplicity, the remainder of this work will assume discrete actions and observations, and a slight abuse of notation will be introduced where the symbols a_t and o_t will be used in diagrams to indicate the indices of tensors A and O associated with these actions and observations.

Importantly, a uMPS has an infinite number of sites. Therefore, computing the probability of a finite sequence of actions and observations implies an infinite number of marginalizations. Fortunately, placing the uMPS in mixed gauge allows computing the probability of any length sequence without

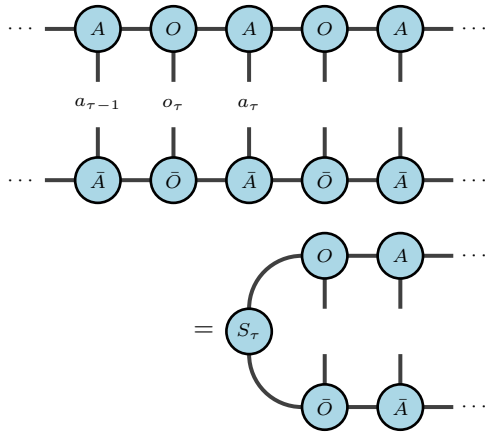
requiring explicit marginalization [27]:

$$P(\tilde{o},\tilde{a}) = \quad , \tag{5.23}$$



where $\tilde{o} = (o_0, o_1, \ldots, o_n)$ and $\tilde{a} = (a_0, a_1, \ldots, a_{n-1})$. Note that it is unnecessary to complete the unit cell at the beginning or the end of the uMPS. Eq. (5.23) illustrates this through the exclusion of action a_n.[1]

The uMPS in Eq. (5.22) fulfills the same main functions as the classical probabilistic model. This can be seen by contracting the network with a sequence of actions and observations up to some time τ,



. (5.24)

Note that Eq. (5.24) only serves as an illustration, since, in practice, a uMPS can never be used this way as it requires an infinite number of contractions. Consider, once again, the three main functions of the generative model in active inference. Firstly, the uMPS contains the agent's current beliefs about the environment through contraction with actions and observation perceived in the past. In Eq. (5.24), the vector S_τ can be interpreted as the agent's current beliefs. Secondly, it contains the dynamics of the world through the tensors O and A. The tensors O and A act on the vector S_τ, which yields a new vector if a new observation and action are provided. Finally, it can determine which observation it should perceive based on its current beliefs through the tensor O. Tensor O acts on S_τ, which yields an observation when future actions and observations are marginalized out.

[1] Not including the action a_n makes sense in a realistic setting, since a sequence would typically start and end with an observation. The design of the model does, however, allow for the action to be included, if desired.

An advantage of this formalism is that it makes marginalizing out variables remarkably easy. On the other hand, a disadvantage is that the uMPS does not allow the use of hidden states in the same way that a probabilistic model does. Indeed, since vectors such as S_τ in Eq. (5.24) are not normalized, they cannot serve as a direct substitute to the hidden states s_t. Two immediate solutions to this problem present themselves. One possibility is to ignore the fact that these vectors are not normalized and utilize them as they would have been in the probabilistic model. A second possibility is to re-derive the existing active inference equations in terms of a generative model which does not contain hidden states. The latter is the approach adopted in this work.

5.2.2.2 Learning from sequences

An active inference agent updates its internal model of the world to minimize its surprisal, $-\log P(\tilde{o})$ [9]. However, this quantity is generally assumed to be intractable, because its computation requires an integral over a non-trivial hidden state space [22]. Therefore, surprisal is typically minimized through variational Bayesian methods [9, 22]. In these methods, surprisal is minimized by minimizing an upper bound called the variational free energy F, [22]

$$-\log P(\tilde{o}) \leq F[Q,\tilde{o}] = \mathrm{E}_{Q(\tilde{s},\tilde{a})}\left[\log Q(\tilde{s},\tilde{a}) - \log P(\tilde{o},\tilde{s},\tilde{a})\right] , \tag{5.25}$$

where $Q(\tilde{s},\tilde{a})$ is an approximation to the posterior probability $P(\tilde{s},\tilde{a}|\tilde{o})$ and is varied to attain the minimum.

In the uMPS formalism, the generative model does not include hidden states. Therefore, the surprisal can be minimized directly. However, there is a catch: while it is possible to *compute* the surprisal of a finite sequence through Eq. (5.23), it is not possible to *sample* such finite sequences. Since the sequences are finite and the model is infinite, the starting point of each sequence within the model is ambiguous, making the KL divergence ill-defined. Instead, it *is* possible to minimize the conditional surprisal $-\log P(\tilde{o}|o_0)$ given the initial observation o_0, since the starting point is provided.

In practice, the quantity that is minimized is $-\log P(\tilde{o},\tilde{a}|o_0)$, where actions $\tilde{a}$ are included. That way, the agent can learn how actions affect the dynamics of the environment. Moreover, the resulting distribution can easily be marginalized to obtain the surprisal as in Eq. (5.25). The loss function for the minimization is

$$\mathcal{L} = \frac{2}{(n-1)|\mathcal{V}|}\sum_{v\in\mathcal{V}}\left(-\log P(v) + \log P(v_0)\right) , \tag{5.26}$$

where $\mathcal{V}$ is a set of sequences $v = (o_0, a_0, o_1, a_1, \ldots, a_{\frac{n-1}{2}-2}, o_{\frac{n-1}{2}-1})$ of uneven length n with $v_0 \coloneqq o_0$. The factor $\frac{2}{n-1}$ is the reciprocal of the number of observations in each sequence and ensures that the loss function is insensitive to the length of the sequence.

In theory, the loss function can be minimized through (stochastic) gradient descent with some step size γ [4],

$$|\psi_{t+1}\rangle = |\psi_t\rangle - \gamma \frac{\partial \mathcal{L}}{\partial \langle\psi|}\,, \tag{5.27}$$

where the gradient is a Wirtinger gradient [30]. By letting $\gamma \to 0$, gradient descent amounts to integrating the gradient flow [24]

$$\frac{\mathrm{d}}{\mathrm{d}t}|\psi\rangle = -\frac{\partial \mathcal{L}}{\partial \langle\psi|}\,. \tag{5.28}$$

The gradient of $\mathcal{L}$ with respect to $\langle\psi|$ is given by

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \langle\psi|} &= \frac{2}{(n-1)|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left(\frac{1}{P(v)} \frac{\partial P(v)}{\partial \langle\psi|} - \frac{1}{P(v_0)} \frac{\partial P(v_0)}{\partial \langle\psi|} \right) \\
&= \frac{2}{(n-1)|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left(\frac{\hat{v}}{\langle\psi|\hat{v}|\psi\rangle} - \frac{\hat{v}_0}{\langle\psi|\hat{v}_0|\psi\rangle} \right) |\psi\rangle \qquad (5.29) \\
&\coloneqq \hat{H}|\psi\rangle\,, \qquad (5.30)
\end{aligned}$$

where $\hat{v}$ is sequence v represented as an operator. Finally,

$$\frac{\mathrm{d}}{\mathrm{d}t}|\psi\rangle = -\hat{H}|\psi\rangle\,. \tag{5.31}$$

This expression is effectively the famous Schrödinger equation [25, 11] with the important caveat that $\hat{H}$ itself depends on the state $|\psi\rangle$. In practice, given that $|\psi\rangle$ is a uMPS, Eq. (5.31) can be integrated with TDVP [19, 13, 27], a specialized algorithm for integrating equations of this form with uMPS.

5.3 Extension 2: operator formulation

Up to this point, a formal description of active inference has not been required. However, to proceed further, it is important to understand some of the intricacies of the action selection process in active inference. This section will first provide the necessary background, followed by an explanation of its implications and how to perform action selection within the uMPS formalism.

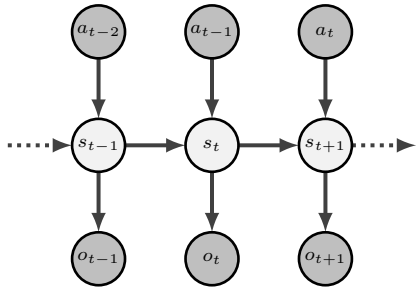


Figure 5.1: Bayesian network depicting the generative model of an active inference agent in discrete time. The network corresponds to a partially observable Markov decision process: the environment is assumed to be a Markov process, but the agent can only partially observe the state of the environment. As a result, the agent contains an internal representation of the environment s_t, which is updated at each time step t based on its sensory observations o_t and the actions a_{t-1} that it performs.

5.3.1 Background

Active inference conceptualizes perception and action as two processes fulfilling the same objective: the minimization of surprisal [9]. Since this objective is often intractable, agents instead minimize a proxy known as variational free energy [9, 22]. In a way, variational free energy directly guides an active inference agent's perception of the world, i.e. how it updates its internal model of the world, yet indirectly guides its actions. Since the agent performs actions in order to ensure variational free energy minimization in the future, but observations in the future are unknown, the quantity that guides its actions is the expected free energy.

The purposes of variational free energy and expected free energy can be understood in the following way. The variational free energy seeks to minimize the discrepancy between what the agent expects to observe and what it actually observes. This is perception: the agent updates its internal model of the environment to fit its sensory observations. In contrast, the expected free energy seeks to minimize the discrepancy between what the agent expects to observe and what it wants to observe. This is action: the agent interacts with its environment to achieve its preferred sensory observations. Thus, the variational free energy is independent of the agent's preferences, while the expected free energy is inherently shaped by them.

Mathematically, an active inference agent is often modeled as a partially observable Markov decision process (POMDP) [1] involving observations o_t, hidden states s_t and actions a_t (see Fig. 5.1) [9]. It minimizes its variational free energy $F[Q, \tilde{o}]$ (Eq. (5.25)), an upper bound on the surprisal, and selects

actions that yield the lowest expected free energy G, [9, 21]

$$G[Q] = \mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s},\tilde{a})} \left[\log Q(\tilde{s},\tilde{a}) - \log P^*(\tilde{o},\tilde{s})\right] , \tag{5.32}$$

where $P^*(\tilde{o},\tilde{s}) = P^*(\tilde{o})P(\tilde{s}|\tilde{o})$ is the agent's preferred distribution, a probability distribution that encodes the observations that the agent wants to realize in the future, and $\tilde{a} = (a_t, a_{t+1}, \ldots, a_{\tau-1})$, $\tilde{s} = (s_{t+1}, s_{t+2}, \ldots, s_\tau)$ and $\tilde{o} = (o_{t+1}, o_{t+2}, \ldots, o_\tau)$ are future actions, states and observations up to some time step τ. Here, F and G are functionals of a variational posterior distribution Q, an approximation to the true posterior distribution P. Note that the expected free energy G includes a distribution $Q(\tilde{a})$ over actions $\tilde{a}$. As such, F is minimized w.r.t. $Q(\tilde{s})$, while G is minimized w.r.t. $Q(\tilde{a})$. Indeed, minimizing F can be equated to updating the agent's understanding of the world, while minimizing G can be equated to updating the agent's understanding of which actions lead to preferred outcomes. In practice, G is usually minimized through the expression [21]

$$G[Q] = \mathrm{D}_{\mathrm{KL}} \left(Q(\tilde{a}) \,\middle|\middle|\, \exp\left\{-\mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s}|\tilde{a})} \left[\log Q(\tilde{s}|\tilde{a}) - \log P^*(\tilde{o},\tilde{s})\right]\right\}\right) , \tag{5.33}$$

which follows directly from Eq. (5.32), such that,

$$Q(\tilde{a}) = \sigma\left(-\mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s}|\tilde{a})} \left[\log Q(\tilde{s}|\tilde{a}) - \log P^*(\tilde{o},\tilde{s})\right]\right) , \tag{5.34}$$

with the softmax function σ. Action selection then proceeds by sampling actions from $Q(\tilde{a})$, or by simply performing the most probable actions.

5.3.2 Expected surprisal

Since the uMPS formulation does not provide access to hidden states, the agent requires a new equation for action selection. This equation assumes that the agent directly minimizes surprisal, $-\log P(o_t)$, instead of variational free energy F (Eq. (5.25)). Therefore, the agent selects actions that yield the lowest expected surprisal $\mathfrak{J}$,

$$\mathfrak{J} = \mathrm{E}_{P(\tilde{o},\tilde{a})} \left[-\log P^*(\tilde{o})\right] , \tag{5.35}$$

as opposed to expected free energy G (Eq. (5.32)). Note that this foregoes the variational approximation step.

For convenience, let the current time step $t = 0$ and redefine $\tilde{a} = (a_1, a_2, \ldots, a_{n-1})$. To find the next action, the agent must find a $P(a_0)$ that minimizes $\mathfrak{J}$,

$$\mathfrak{J} = \mathrm{E}_{P(\tilde{o},\tilde{a},a_0)} \left[-\log P^*(\tilde{o})\right] = \mathrm{E}_{P(a_0)} \left[\mathrm{E}_{P(\tilde{o},\tilde{a}|a_0)} \left[-\log P^*(\tilde{o})\right]\right] . \tag{5.36}$$

Note that the cross entropy of a distribution q relative to the distribution p,

$$\mathrm{E}_p\left[-\log q\right] , \tag{5.37}$$

is minimal when $p = q$. Thus, when interpreting Eq. (5.36) as a cross-entropy, $\mathfrak{J}$ is minimal when

$$P(a_0) \propto \exp\left\{-\mathrm{E}_{P(\tilde{o},\tilde{a}|a_0)}\left[-\log P^*(\tilde{o})\right]\right\} \quad \forall\, a_0 , \tag{5.38}$$

or rather,

$$P(a_0) = \sigma\left(-\mathrm{E}_{P(\tilde{o},\tilde{a}|a_0)}\left[-\log P^*(\tilde{o})\right]\right) \coloneqq \sigma\left(-\mathfrak{J}(a_0)\right) . \tag{5.39}$$

Finally, $\mathfrak{J}(a_0)$ yields the expected surprisal given action a_0 and can be used to sample actions.

Expanding $\mathfrak{J}(a_0)$ with the assumption that $P^*(\tilde{o}) = \prod_\tau P^*(o_\tau)$ leads to the recursive equation

$$\begin{aligned}\mathfrak{J}(o_\tau, a_\tau) = &\underbrace{\mathrm{E}_{P(o_{\tau+1}|o_{<\tau+1},a_{<\tau+1})}\left[-\log P^*(o_{\tau+1})\right]}_{\text{expected surprisal of next action}} \\ &+ \underbrace{\mathrm{E}_{P(o_{\tau+1}|o_{<\tau+1},a_{<\tau+1})P(a_{\tau+1}|o_{\leq\tau+1},a_{<\tau+1})}\left[\mathfrak{J}(o_{\tau+1}, a_{\tau+1})\right]}_{\text{expected surprisal of subsequent actions}} ,\end{aligned} \tag{5.40}$$

where it is imposed that

$$P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1}) = \sigma\left(-\mathfrak{J}(o_{\tau+1}, a_{\tau+1})\right) , \tag{5.41}$$

as in sophisticated active inference [10]. Eq. (5.41) can be understood as enforcing Bellman's principle of optimality, where future action selection must comprise an optimal policy regardless of the current action selection [3]. This form allows the agent to consider current actions assuming that future actions are selected optimally according to their expected surprisal.

The expression in Eq. (5.40) describes a tree structure that exhibits exponential growth over time. In certain environments, the rapid growth of the tree limits the feasibility of calculating the expected surprisal up to only a few future time steps. Details on the computational process are provided in Appendix A. However, the properties of uMPS allow for an alternative approach to expected surprisal computation.

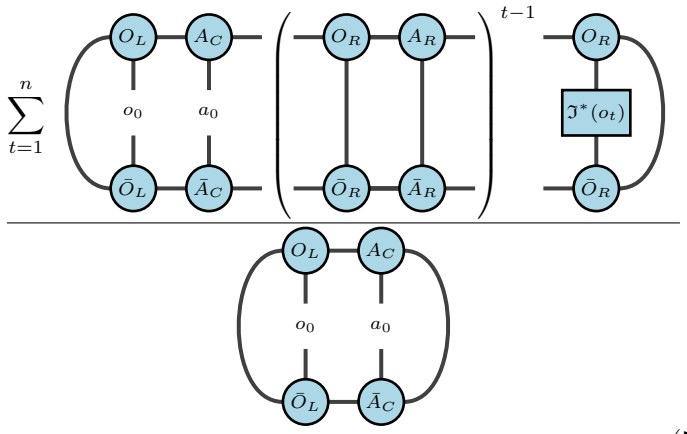
5.3.3 Learning the action distribution

The expansion in the previous section can be avoided by reformulating the expected surprisal using operators. Let $h = (o_{-T}, a_{-T}, \ldots, a_{-1}, o_0, a_0)$ the agent's history of actions and observations from $t = -T$ up to $t = 0$

including action a_0. This history is usually left implicit, but will be made explicit for clarity. Then, the expected surprisal is given by

$$\begin{aligned}\mathfrak{J}(a_0) &= \mathrm{E}_{P(\tilde{o},\tilde{a}|h)}\left[-\log P^*(\tilde{o})\right] \\ &= \sum_{t=1}^{n} \mathrm{E}_{P(\tilde{o},\tilde{a}|h)}\left[-\log P^*(o_t)\right] \\ &= \frac{\sum_{t=1}^{n} \mathrm{E}_{P(\tilde{o},\tilde{a},h)}\left[-\log P^*(o_t)\right]}{P(h)} \\ &= \frac{\langle\psi(A,O)|\left(\hat{h}\sum_{t=1}^{n}\hat{\mathfrak{J}}^*(o_t)\right)|\psi(A,O)\rangle}{\langle\psi(A,O)|\hat{h}|\psi(A,O)\rangle}\,, \end{aligned} \tag{5.42}$$

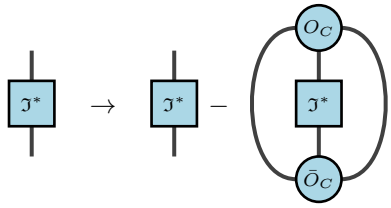
where $\hat{h}$ projects $|\psi(A,O)\rangle$ onto the subspace that encodes the agent's history, and $\hat{\mathfrak{J}}^*(o_t)$ is the surprisal operator for the preferred observation distribution at time t. Diagrammatically,

$\mathfrak{J}(a_o) =$ 

, (5.43)

where the history has been restricted to observation o_0 for brevity.

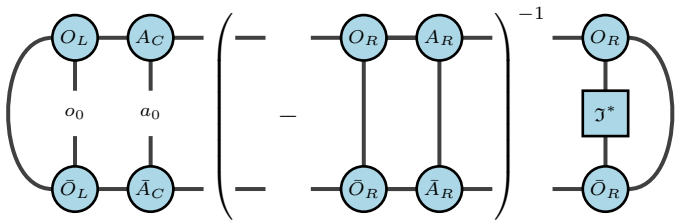
When $n \to \infty$, assuming that $\mathfrak{J}^*(o_t) = \mathfrak{J}^*$ for all o_t, Eq. (5.43) can be simplified by observing that the numerator is a geometric series. Since the largest eigenvalue of the term within brackets is 1, this sum is divergent. Fortunately, it can be made convergent by subtracting the divergent part of the sum, i.e. the expected value of $\hat{\mathfrak{J}}^*$,



. (5.44)

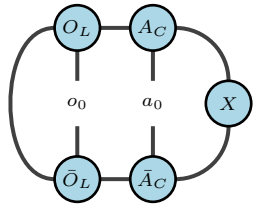
Due to this transformation, the computed value is no longer the expected surprisal, but the difference with the average expected surprisal. Since this value is later inserted into a softmax function (cf. Eq. (5.39)), Eq. (5.44) can simply be regarded as a gauge transformation. Thus, $P(a_0)$ remains unchanged.

Now, the numerator becomes



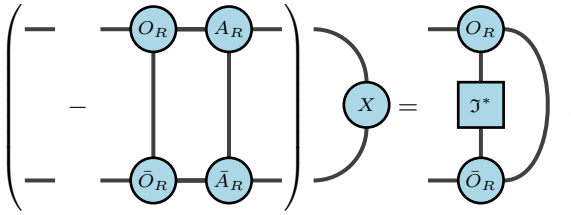
, (5.45)

where the first term within brackets is the unit operator on the bond dimensions. Solving this is relatively straightforward through iterative methods for systems of linear equations, such as the generalized minimal residual (GMRES) [23] method, and yields a solution of the form



, (5.46)

where X satisfies



. (5.47)

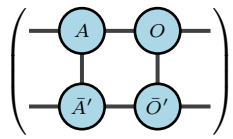
The matrix X needs to be computed only once and can be reused in all subsequent calculations of the expected surprisal, unless the tensors A or O are updated. It is otherwise independent of actions or observations.

Importantly, in this formulation, it is not possible to impose Eq. (5.41). As such, the correctness of the values of the expected surprisal obtained through the above method relies on the encoding of the action distribution $P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1})$ within the generative model. Therefore, in contrast to the method in Section 5.3.2, the method in this section requires

$P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1})$ to be correctly encoded in the model during training. In other words, the agent will have to learn the action distribution $P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1})$.

The action distributions can be learned simultaneously with the environment's dynamics. In other words, learning action distributions does not require a separate training loop. It only requires the model to "see" optimal sequences more often than other sequences, i.e. the data set must contain a larger proportion of optimal sequences. As such, the model must be trained through an online algorithm where the data set gradually fills up with optimal paths as the agent explores and learns. The algorithm proceeds as follows.

Initially, the model lacks knowledge of the environment's dynamics, which prevents the agent from effectively selecting actions that lead towards its preferred observations. To address this, the model is first trained on random sequences to learn the environment's dynamics. Once the model has sufficiently learned, new sequences are generated by allowing the agent to select actions with this model, i.e. by sampling actions from the action distributions obtained using the method described earlier in this section. Next, the model is trained on the newly generated sequences. Once again, once it has sufficiently learned, it is used to generate another set of sequences. This iterative process repeats until the model converges. Note that this process involves two notions of convergence: the first measures how much a model changes while training on a data set, the second measures how much the model changes between data sets (after convergence in the first sense). Convergence of the first type is measured in the gradient norm of the cost function (Eq. (5.26)). Convergence of the second type can be scored using the error $\sqrt{1-|f|}$, where the overlap f between the current model $|\psi(A, O)\rangle$ and the previous model $|\psi(A', O')\rangle$ is equal to the largest eigenvalue of the contraction between the current unit cell and the previous unit cell,

$$f = \lambda_{\max}$$ 

. (5.48)

The model is considered to have converged when the error becomes smaller than some threshold value.

At first, the model will only learn action distributions for observations that lead to the goal in a single time step. This is because, once the dynamics have been learned, the expected surprisal will only be correct for a single time step. During training, the agent will gradually learn action distributions $P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1})$ and be able to correctly compute the expected surprisal at observations that are progressively further away from

the goal. It does this one step at a time, by building on top of the correct expected surprisal computations for observations that are closer to the goal, similar to backward induction.

5.4 Experiments

5.4.1 Summary of setup

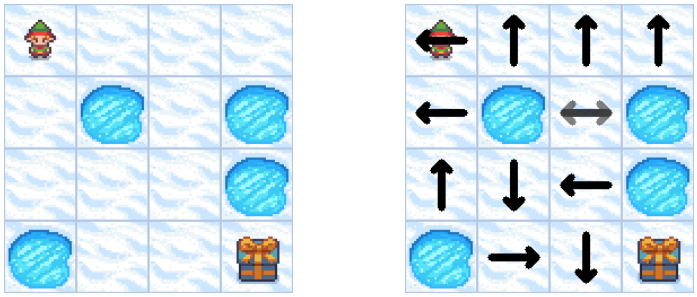
Setup The uMPS used in the experiments is parameterized by a unit cell of two tensors A and O associated with actions and observations respectively, as in Eq. (5.22) It is trained using a set of long sequences of actions and observations. These sequences have length $n = 1 + 2|\tilde{o}|$ and start from the environments' designated starting point. They are sampled according to the algorithm described in Section 5.3.3. The model is trained by minimizing the conditional surprisal (Eq. (5.26)) through a TDVP-based method described in Section 5.2.2.2.

Bond dimensions Once the uMPS has sufficiently converged, its bond dimensions can be adjusted. Bond adjustment is based on the Schmidt decomposition of the two center matrices C_1 and C_2 (cf. Eq. (5.15) and (5.20)). After normalizing all the Schmidt values by dividing by the largest value, the dimensions with the smallest values can be discarded if they are smaller than the precision required of the uMPS. That is, the bond dimension is truncated by discarding dimensions with Schmidt values smaller than some threshold value. Similarly, it is possible to add dimensions with random noise, if the Schmidt values are too large compared to the threshold value.

For increased stability during training, the threshold on the Schmidt values is relaxed. This means that the bond is not truncated unless there are at least 5 Schmidt values smaller than the threshold. Also, expansion is generous, to not have to iterate the process too often. That way, the MPS manifold is less likely to change, which results in a more stable minimization process.

5.4.2 Environment

The environment studied in this work is the Frozen Lake from OpenAI Gym [5] (Fig. 5.2a). Here, the agent must navigate a frozen lake to reach a reward while avoiding the holes in the ice. The frozen lake is represented by a 4×4-grid, where cells consist of frozen patches and holes. The agent starts in the upper left corner and must reach the goal in the lower right corner. The frozen patches are slippery and make it so that the agent has a chance of slipping in the wrong direction.



(a) default environment (b) optimal policy

Figure 5.2: The layout of the default 4×4 Frozen Lake environment [5] and the optimal policy as obtained through dynamic programming [2].

More concretely, the environment provides discrete observations and actions. The observations are the 16 possible locations of the agent, i.e. the 16 cells of the environment. The agent can perform one of four actions: left, down, right, or up; which moves the agent to the adjacent cell in the corresponding direction. An action has no effect, if it leads the agent outside the grid.

The slipperiness of the frozen patches causes the agent to slip in directions perpendicular to the intended direction. For example, if the agent intends to go down, there is a chance that the agent slips left or right instead. In particular, the chance of going in the intended direction is $\frac{1}{3}$, while the chances of slipping in clockwise and anticlockwise direction are $\frac{1}{3}$ each, for a total chance of slipping of $\frac{2}{3}$.

Lastly, in the original Frozen Lake, the agent would be stuck if it were to fall into a hole. In this work, the environment has been adapted to give a small chance of $\frac{1}{5}$ to restart at the starting position after falling into a hole. This is because of the infinite nature of the uMPS. If the agent were to get stuck forever, the model would provide a 100 % probability of being found in a hole, since in that case, at any random moment within an infinite period of time, the agent is infinitely more likely to be located in a hole than on the ice. Thus, there must be at least a small chance for the agent to escape the hole. This mimics how a true game is played as well. The universe generally does not end after a game of chess, so to speak. Therefore, it should always be possible to encode a "restart" of sorts in a natural way.

Dynamic programming [2] can provide insight into what an agent's optimal sequence of actions would be for this environment. In this context, "optimal" refers to Bellman's principle of optimality [3]. Thus, the optimal policy obtained through dynamic programming provides actions that lead to

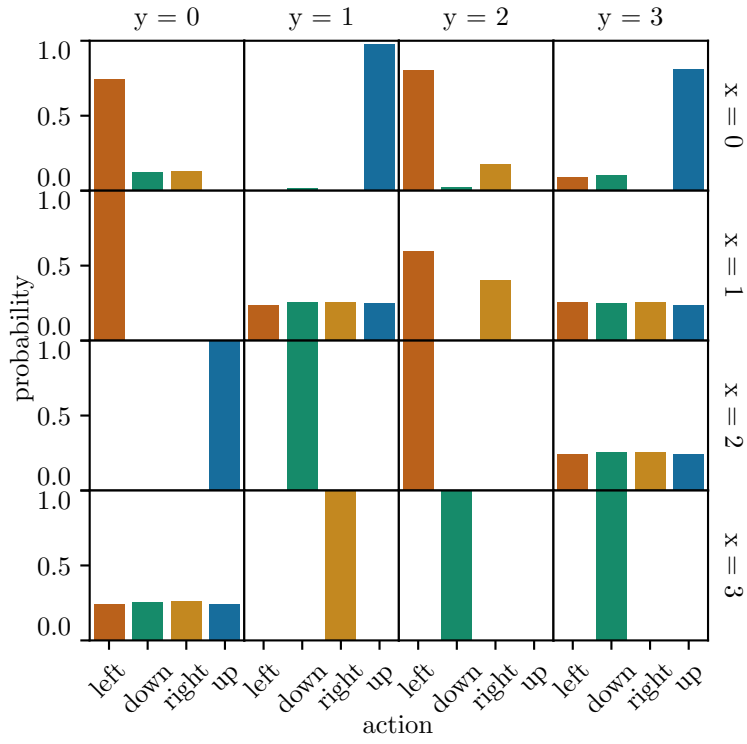


Figure 5.3: Action probability distribution $P(a_0)$ obtained at each cell of the 4×4-grid.

the goal in the shortest amount of time, while avoiding holes. The optimal action for each grid cell is given in Fig. 5.2b. The figure shows that the optimal policy consists of performing actions that lead away from holes in order to minimize the probability of falling. It generally relies on slipping in the correct direction by chance. The policy avoids the first hole by passing along the left edge of the grid. While there are two possible paths around this hole, the path along the top edge passes between two holes, which leads to a higher chance of falling.

5.4.3 Results and discussion

The performance of the uMPS agent can be evaluated through comparison with the dynamic programming agent in the previous section. Since the Frozen Lake environment is Markovian, the dynamic programming agent provides the optimal policy. Although active inference agents do not pro-

vide a policy in the same way dynamic programming does, it is possible to compare the actions obtained through the expected surprisal and action distribution (Eq. (5.39)) for each observation given some preferred observation distribution $P^*(o)$. The preferred observation distribution used in the following experiments is

$$P^* = \sigma(C) \quad \text{with} \quad C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 40 \end{pmatrix}, \tag{5.49}$$

where the indices of the C-matrix coincide with the indices of the grid cells in the environment. Note that in classical treatments of this problem, the holes in the ice typically receive a negative weight to indicate that they are to be avoided. Here, this aspect is learned through planning, since falling into a hole leads to larger expected surprisal.

Fig. 5.3 shows the action distribution obtained by the uMPS agent for each position in the environment. From this, it is clear that an agent that performs only actions with the largest probability corresponds closely to the Bellman optimal agent described above (Fig. 5.2b). The only difference is for position $(0, 2)$, where the agent takes "left" over "up". This discrepancy is due to the small chance to restart after falling into a hole in the adapted version of the Frozen Lake, which the uMPS was trained on, and can most likely be remedied by reducing this chance.

To show how the agent navigates the environment, Fig. 5.4 displays a sequence of observations and corresponding action distributions $P(a_0)$. This particular sequence consists of the best possible course of events, since the agent never slips. The probability of this sequence occurring is very small, but it serves to illustrate how the agent operates.

Fig. 5.5 shows a comparison between the proportion of successes and failures for the uMPS agent and dynamic programming agent as a function of the number of steps performed until termination, where kernel density estimates were added as a visual aid. As expected, the agents produce very similar distributions. However, the failure rate of the uMPS agent seems to be slightly higher. This is clearer when comparing the overall failure rates: 21.4 % for the uMPS and 17.4 % for dynamic programming. This discrepancy can be ascribed to the slight difference in policy in position $(0, 2)$ mentioned earlier.

It is also possible to evaluate the dissimilarity between uMPS models by calculating the error $\sqrt{1 - |f|}$ with f given by Eq. (5.48). To this end, five different models were trained, and the dissimilarity was computed between consecutive models. Interestingly, the average error was 30.4 %, indicating that the uMPS states $|\psi(A, O)\rangle$ were relatively different. On the other

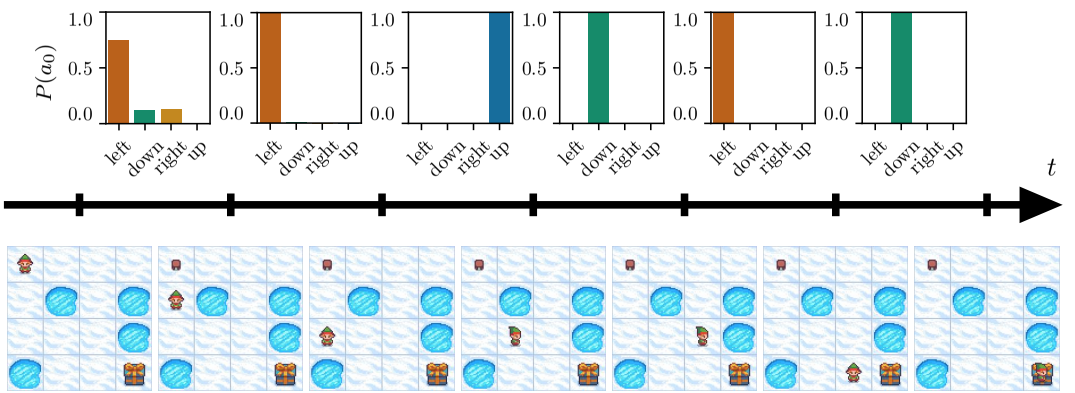


Figure 5.4: An example of a uMPS agent navigating the Frozen Lake environment. In this example, the agent gets very lucky and manages to reach the goal in the minimum required number of steps. The bottom row shows the observation at each time step, while the top row shows the action probability distribution $P(a_0)$ computed after each observation.

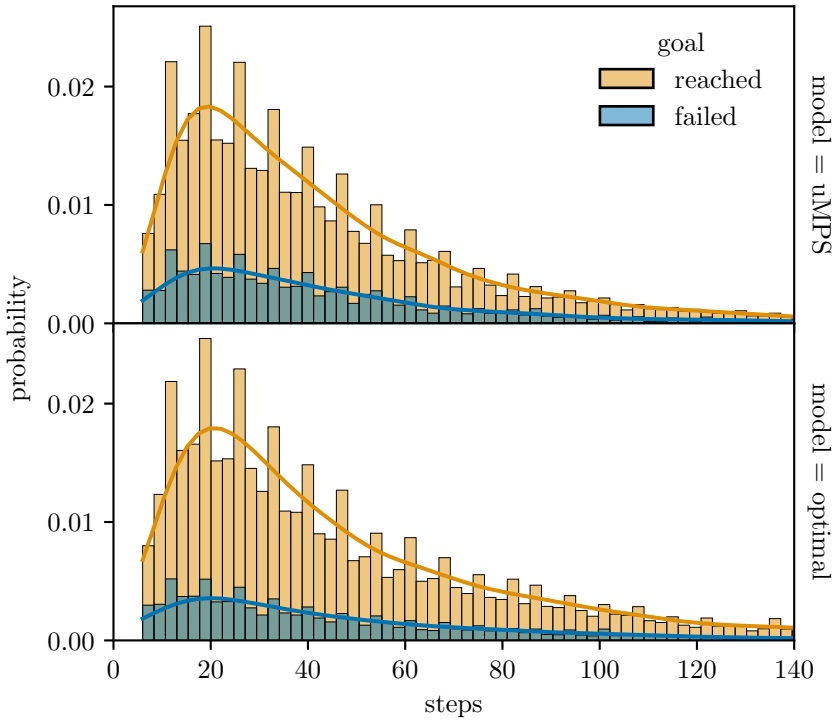
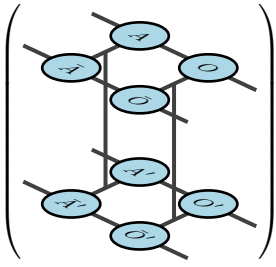


Figure 5.5: Success and failure rates of a uMPS agent compared to an optimal (dynamic programming) agent as a function of number of steps performed in the environment. Smooth curves show the general trend and are obtained through kernel density estimation.

hand, it is also possible to evaluate the difference between the probability distributions represented by the uMPS models by letting

$$f = \lambda_{\max} \left(\right) . \tag{5.50}$$



In this case, the average error is 0.2 %, indicating that the probability distributions are relatively similar. Thus, despite consisting of relatively different states, the generative models provide the same probabilities.

Finally, the bond dimensions obtained are on average 19 for D_1 and 17.5 for D_2. This is comparable to the bond dimensions obtained for the MPS in Wauthier et al. [28], which was 16 for most bonds. Although the bond dimensions in this work are slightly higher, note that a soft threshold was used to adjust the bonds here.

5.5 Conclusion and outlook

This work provides two important extensions to the MPS implementation of active inference. Firstly, it leverages the translation-invariant class of MPS called uniform MPS to enable generative modeling of environment with potentially infinite time horizons. Secondly, it implements the expected free energy from active inference as a physical operator to enable planning with an infinite time horizon using state-of-the-art techniques from the MPS literature. Together, these extensions comprise a method that is able to model an environment from scratch and navigate towards a predefined goal. Moreover, the size of the resulting generative model is automatically adjusted based on the information it holds.

This method presents several avenues for future research. For one, in order to improve the algorithm's rate of convergence, it is possible to generate sequences using a hybrid method which combines the methods from Sections 5.3.2 and 5.3.3. By performing action selection using the tree structure for m time steps and using the uMPS method for time steps beyond m, action distributions can be learned for m time steps at a time instead of a single step. This would constitute a trade-off between memory requirements and training time.

For another, It can be applied and tested on more complicated environments. Environments such as nine men's morris or checkers could be ideal to test scalability. Moreover, this work was completely focused on feature maps in the form of Kronecker delta functions. This type of feature map is generally not applicable to continuous variables. Thus, future research may focus on incorporating continuous observations and actions. For example, this could be accomplished by using transformations, such as Fourier transformations, by using neural networks, or by adapting the structure of the tensor network. In fact, recent work has already shown the efficacy of transformations in the generative learning of continuous data [20].

Acknowledgements

Frozen lake assets provided by: `https://franuka.itch.io/rpg-snow-tileset` for elf and stool, Mel Tillery `http://www.cyaneus.com/` for all

other assets.

5.6 References

[1] Åström, K. J., "Optimal control of Markov processes with incomplete state information," *Journal of Mathematical Analysis and Applications* **10** no. 1, (Feb. 1965) 174–205.

[2] Bellman, R., "The theory of dynamic programming," *Bulletin of the American Mathematical Society* **60** no. 6, (1954) 503–515.

[3] Bellman, R., *Dynamic Programming*, ch. The Structure of Dynamic Programming Processes, pp. 81–115. Princeton University Press, Aug. 2021.

[4] Bottou, L., *Advanced Lectures on Machine Learning*, ch. Stochastic Learning, pp. 146–168. Springer Berlin Heidelberg, 2004.

[5] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym." 2016. `arXiv:1606.01540 [cs.LG]`.

[6] Cirac, J. I., Pérez-García, D., Schuch, N., and Verstraete, F., "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems," *Reviews of Modern Physics* **93** no. 4, (Dec. 2021) 045003.

[7] Fannes, M., Nachtergaele, B., and Werner, R. F., "Valence bond states on quantum spin chains as ground states with spectral gap," *Journal of Physics A: Mathematical and General* **24** no. 4, (Feb. 1991) L185–L189.

[8] Fannes, M., Nachtergaele, B., and Werner, R. F., "Finitely correlated states on quantum spin chains," *Communications in Mathematical Physics* **144** no. 3, (Mar. 1992) 443–490.

[9] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O'Doherty, J., and Pezzulo, G., "Active inference and learning," *Neuroscience & Biobehavioral Reviews* **68** (Sep. 2016) 862–879.

[10] Friston, K., Da Costa, L., Hafner, D., Hesp, C., and Parr, T., "Sophisticated Inference," *Neural Computation* **33** no. 3, (Mar. 2021) 713–763.

[11] Griffiths, D. J. and Schroeter, D. F., *Introduction to Quantum Mechanics*. Cambridge University Press, Aug. 2018.

[12] Haegeman, J. and Verstraete, F., "Diagonalizing Transfer Matrices and Matrix Product Operators: A Medley of Exact and Computational

Methods," *Annual Review of Condensed Matter Physics* **8** no. 1, (Mar. 2017) 355–406.

[13] Haegeman, J., Cirac, J. I., Osborne, T. J., Pižorn, I., Verschelde, H., and Verstraete, F., "Time-Dependent Variational Principle for Quantum Lattices," *Physical Review Letters* **107** no. 7, (Aug. 2011) 070601.

[14] Haegeman, J., Osborne, T. J., and Verstraete, F., "Post-matrix product state methods: To tangent space and beyond," *Physical Review B* **88** no. 7, (Aug. 2013) 075133.

[15] Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P., "Unsupervised Generative Modeling Using Matrix Product States," *Physical Review X* **8** no. 3, (July 2018) 031012.

[16] Jebara, T., *Machine Learning*, ch. Generative Versus Discriminative Learning, pp. 17–60. Springer US, 2004.

[17] Klumper, A., Schadschneider, A., and Zittartz, J., "Equivalence and solution of anisotropic spin-1 models and generalized t-J fermion models in one dimension," *Journal of Physics A: Mathematical and General* **24** no. 16, (Aug. 1991) L955–L959.

[18] Klümper, A., Schadschneider, A., and Zittartz, J., "Matrix Product Ground States for One-Dimensional Spin-1 Quantum Antiferromagnets," *Europhysics Letters (EPL)* **24** no. 4, (Nov. 1993) 293–297.

[19] Kramer, P. and Saraceno, M., eds., *Geometry of the Time-Dependent Variational Principle in Quantum Mechanics*, ch. The time-dependent variational principle (TDVP), pp. 3–14. Springer Berlin Heidelberg, 1981.

[20] Meiburg, A., Chen, J., Miller, J., Tihon, R., Rabusseau, G., and Perdomo-Ortiz, A., "Generative Learning of Continuous Data by Tensor Networks." 2024. `arXiv:2310.20498 [cs.LG]`.

[21] Millidge, B., Tschantz, A., and Buckley, C. L., "Whence the Expected Free Energy?" *Neural Computation* **33** no. 2, (Feb. 2021) 447–482.

[22] Parr, T., Pezzulo, G., and Friston, K. J., *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, Mar. 2022.

[23] Saad, Y. and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing* **7** no. 3, (July 1986) 856–869.

[24] Santambrogio, F., "{Euclidean, metric, and Wasserstein} gradient flows: an overview," *Bulletin of Mathematical Sciences* **7** no. 1, (Mar. 2017) 87–154.

[25] Schrödinger, E., "An Undulatory Theory of the Mechanics of Atoms and Molecules," *Physical Review* **28** no. 6, (Dec. 1926) 1049–1070.

[26] Stoudenmire, E. and Schwab, D. J., "Supervised Learning with Tensor Networks," in *Advances in Neural Information Processing Systems*, Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., eds., vol. 29. Curran Associates, Inc., 2016.

[27] Vanderstraeten, L., Haegeman, J., and Verstraete, F., "Tangent-space methods for uniform matrix product states," *SciPost Physics Lecture Notes* (Jan. 2019) 7.

[28] Wauthier, S. T., Verbelen, T., Dhoedt, B., and Vanhecke, B., "Planning with tensor networks based on active inference," *Machine Learning: Science and Technology* **5** no. 4, (Oct. 2024) 045012.

[29] White, S. R., "Density matrix formulation for quantum renormalization groups," *Physical Review Letters* **69** no. 19, (Nov. 1992) 2863–2866.

[30] Wirtinger, W., "Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen," *Mathematische Annalen* **97** no. 1, (Dec. 1927) 357–375.

[31] Zauner-Stauber, V., Vanderstraeten, L., Fishman, M. T., Verstraete, F., and Haegeman, J., "Variational optimization algorithms for uniform matrix product states," *Physical Review B* **97** no. 4, (Jan. 2018) 045145.

—People who say they sleep like a baby usually don't have one.

Leo J. Burke

6

The big picture and future perspectives

The methods outlined in the previous chapters offer a diverse range of approaches for model reduction. Yet, they also share several common themes. This chapter identifies the similarities and differences between these methods.

As the final chapter of this dissertation, it also marks the conclusion of my PhD journey. Accordingly, it outlines potential directions for future research.

This dissertation investigates various techniques for model reduction in active inference, presenting a framework for creating more efficient internal models in artificial agents. The research was divided into two main parts: the first focused on biologically-inspired algorithms for state space reduction in the ANNs used to model beliefs in deep active inference. The second part introduced physics-inspired models that allowed for dynamic adjustment of model size, offering a novel approach to active inference. The result is a versatile toolkit for reducing the complexity of internal models in active inference agents.

Several research questions arose throughout this work. This chapter serves to connect the various contributions made by the research and highlights unanswered questions that could shape future studies.

6.1 Conclusions

The techniques presented in this dissertation are largely independent of any specific underlying structure. Traditional active inference models require manually defining the different components such as likelihood and state space. This can become restrictive when dealing with more complex environments. Although a method that can learn state spaces has been shown to work on a simple toy model [7], whether it is scalable to complex environments remains an open question. On the other hand, ANNs and TNs have already been used in many more complex problems in the literature.

While deep active inference with ANNs reduces the need for manual configuration, some aspects must still be predefined, such as the size of the state space. Given that the appropriate state space size cannot always be determined *a priori*, the offline (Ch. 2) and online (Ch. 3) sleep methods provide ways to reduce state space dimensionality *post hoc* and on-the-fly, respectively. These methods can also potentially be applied to reduce the size of the layers in the ANNs.

The TN implementations, on the other hand, require minimal manual configuration. The models adjust their size automatically based on how much information must be contained. However, this comes at the cost of direct access to the state space.

Approach to model scaling The approach to model scaling differs fundamentally between the ANN and TN methods. The ANN models start with a large number of parameters that is gradually pruned down. This is to ensure that the initial model is sufficiently expressive to capture the dynamics of the environment before redundant parameters are removed. In contrast, the TN models start with a small number of parameters, which can

increase or decrease based on information requirements. There is no need to ensure initial expressivity, since parameters can be added if necessary.

Online vs. offline Most of the methods presented in previous chapters were online methods, meaning that model reduction occurred during training. Although Chapter 3 showed that online sleep performed slightly worse than offline sleep, online sleep had the advantage of requiring a lot fewer training epochs. In general, this is the primary advantage of online model reduction, which makes it particularly useful in cases where computational resources are limited.

Importantly, online reduction methods may also prove beneficial in an online training setting, where the agent collects a data set by interacting with the environment. That way, the model size can adjust to the information contained in the data set on-the-fly, without requiring expensive retraining. Evidently, this likely also requires model expansion. This will be discussed further on.

The reduced performance of online sleep can be attributed to the constantly changing dimensionality of the state space, as noted in Chapter 3. This is comparable to the issue discussed in Chapter 4 regarding two-site updates, where continually fluctuating bond dimensions can lead to computational issues. In both cases, the recommended solution is to suspend model reduction temporarily, allowing the model to converge before applying further reductions. One may conclude that, in an online setting, model reduction should only occur once the model has sufficiently stabilized.

Note that the TN models can be reduced in both an offline and online way. Both use the same SVD-based algorithm. The only difference lies in the timing: whether reduction occurs periodically during training or only at the end. This timing may lead to differences in the final model, however.

Scalability As mentioned earlier, one of the main reasons for learning from data is the ability to avoid manually defining all components of the active inference framework. This is also a way to make active inference more scalable to more complicated environments, where "more complicated" can mean anything from higher-dimensional observations and actions to more elaborate dynamics. Naturally, model reduction methods should be able to follow suit.

In that respect, the offline sleep method may encounter scalability limitations. While the process that determines the number of informative dimensions only depends on the dimensionality of the state space (and is generally not be chosen to be too large), the model has to be retrained in order to perform the reduction. Retraining can be very computationally

expensive depending on the environment. The online sleep method, on the other hand, remains relatively efficient and scalable. Although it requires two model passes per update, this is generally an acceptable cost.

The scalability of the TN methods is heavily dependent on the bond dimensions. The cost of contracting an MPS scales with $\mathcal{O}(D^3 d)$ with the bond dimensions D and physical dimensions d, which is important for both training and runtime. As environments become more complex, the bond dimensions grow larger. Since matrix product states rely entirely on SVD for reduction, the feasibility of scaling to more complicated environments will depend on the efficiency of SVD computations.

That being said, any type of state space or model reduction will likely not significantly improve action selection, i.e., the tree search. Unfortunately, this relies almost entirely on the sizes of the action and observation space, which can generally not be reduced. The only relevant improvement in this respect is purely computational, in the sense that smaller matrices and tensors lead to faster computations. Even in the ANN framework, where one could make a case that the integration over states in the computation of expected free energy becomes simpler for a lower-dimensional state space, the approximate posterior is chosen in a way that allows analytical evaluation of the integral, so that the benefit of state space reduction becomes entirely computational. Improvements in action selection must come from other developments in the field.

Implications for active inference The methods in this dissertation provide various approaches to reduction of active inference models. These serve to replace the process of Bayesian model reduction in classical active inference. BMR can be seen as a special case of Bayesian model selection, where the models under consideration are those where certain parameters have been removed from the full model. This allows for efficient computation of the reduced models' evidence[1], enabling the models to be scored based on this metric.

To make use of BMR, it is crucial to have a posterior distribution over model parameters. In the models used in this work, the model parameters are effectively replaced by point estimates, eliminating stochasticity. In the absence of a posterior distribution, the reduction algorithms utilize gating parameters or singular values, which act as sufficient statistics of a posterior over model parameters[2]. Under the assumption that the loss function used

[1] It is worth reminding the reader that surprisal is the negative logarithm of the evidence.

[2] However, this reasoning does not hold for the offline sleep method, where singular values do not correspond to specific latent dimensions.

in training closely approximates the reduced model's negative log-evidence, the methods can be considered Bayes-optimal.

6.2 Future research directions

Beyond the numerous possible incremental algorithmic enhancements, there are several significant research questions that remain open for future exploration.

Model expansion A theme touched upon in earlier chapters, but not yet fully explored, is how the models behave when exposed to new data. For instance, one could introduce new types of roads in the Car Racer environment or expand the Frozen Lake environment by adding new rows and columns after the model has been trained on the original setting. In such cases, the model would likely need to allocate additional state (or bond) dimensions to encode the new data. In other words, the model must expand to accommodate the new information.

Conjecturing based on the underlying techniques, it is plausible that all the presented methods can handle model expansion to varying degrees, with only minor modifications required for each approach. Notably, TNs have already demonstrated their ability to expand, as mentioned above. However, the capability of the ANN-based algorithms to manage model expansion remains an open question for future research.

Introducing new data can also raise other potential challenges, such as catastrophic forgetting, especially in non-stationary environments. While this issue is beyond the scope of this dissertation, it may become critical when investigating model expansion. For example, even when new data is introduced and the model is expected to expand, catastrophic forgetting might prevent this by causing the loss of previously learned information.

Consequences of TN formulation The formulation of active inference using tensor networks, in particular, opens up numerous avenues for further research. Given its novelty, little is currently known about this formulation, and much remains to be explored and understood.

The motivation for using tensor networks in active inference models was initially to facilitate efficient scaling of model size. Beyond the model reduction techniques, there are still potential improvements to the existing TN implementation. For example, future implementations will need to handle continuous variables in environments where observations or actions cannot be easily discretized. Additionally, an active inference-based exploration algorithm for data collection would be highly beneficial, as it could result in

better datasets that accelerate learning. In the end, these enhancements do also affect the model size: continuous variables may necessitate more bond dimensions to encode, making reduction algorithms essential. Moreover, exploration strategies affect the data set, which in turn influences the learned model, and thus, the bond dimensions.

Finally, recent developments in both active inference and tensor networks may allow the use of exact Bayesian model reduction. For one, a variational inference framework for supervised learning of TN has been proposed [2]. This approach, referred to as Bayesian tensor networks, uses standard variational inference techniques to learn the parameters of a tensor network. As a result, this method could enable the application of exact Bayesian model reduction.

Another recent advancement is the development of the renormalizing generative model (RGM) [1], which can be interpreted as a multi-scale entanglement renormalization ansatz (MERA) tensor network. Essentially, this model represents a hierarchical tensor network that constructs a world model and enables planning across multiple time scales. This model also supports the use of exact Bayesian model reduction.

Hierarchical tensor networks, like the aforementioned, are likely better at modeling long time correlations than matrix product states. This is because the different layers of the models can encode different time scales, with higher levels encoding longer range correlations and lower levels encoding shorter range correlations. On the other hand, this also brings with it a new hyperparameter: the depth of the model. Whether this model size parameter can be tuned during training is still underexplored in the literature. As such, the recommended path would be to test how well MPS models perform on more complex environments before adding layers to the model.

Robotic deployment Another highly-anticipated path of future work is deployment on robots. Chapter 3 already showed that the online ANN method can effectively reduce the number of latent dimensions in a controlled environment, with minimal losses to accuracy. Therefore, it would be interesting to see whether the methods devised in this dissertation can be run on physical robots, such as a TurtleBot [5].

One extension to the online sleep method is model reduction in all the layers of the ANN. Li and Ji [3] have shown promising results applying L_0-ARM-based reduction to all the layers in fully connected and convolutional networks, instead of simply reducing the state space. Reducing the full ANN models is essential to robotic deployment.

Model training for robots is typically performed offline, on servers. Later, the model is uploaded to the robot. This is because training is faster on

external resources, such as NVIDIA A100 GPUs. However, to obtain self-learning adaptive robots, it is essential that the robot can learn on-device. Model reduction is key to keeping the models smaller and manageable.

The application of TN models in robots is still far off. For one, the models in their current form are unable to handle continuous input variables, yet many signals a robot obtains from the environment are continuous. This is likely to change soon, however, since recent work [4] has provided effective methods to solve this. For another, the TN models presented in this dissertation have not yet been applied to more complex environments. It would be highly instructive to investigate how these models perform in more complex environments before venturing on to robotics.

Quantum computing Finally, the TN framework provides a potential gateway to machine learning with quantum computers. Matrix product states and many other TN architectures can be mapped to quantum circuits, making them easily deployable on quantum hardware [6]. Although matrix product states are efficient on classical hardware, they can still provide valuable insight into the world of quantum machine learning.

6.3 References

[1] Friston, K., Heins, C., Verbelen, T., Costa, L. D., Salvatori, T., Markovic, D., Tschantz, A., Koudahl, M., Buckley, C., and Parr, T., "From pixels to planning: scale-free active inference." 2024. arXiv:2407.20292 [cs.LG].

[2] Konstantinidis, K., Xu, Y. L., Zhao, Q., and Mandic, D. P., "Variational Bayesian Tensor Networks with Structured Posteriors," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 29, pp. 3638–3642. IEEE, May 2022.

[3] Li, Y. and Ji, S., *L_0-ARM: Network Sparsification via Stochastic Binary Optimization*, pp. 432–448. Springer International Publishing, 2020.

[4] Meiburg, A., Chen, J., Miller, J., Tihon, R., Rabusseau, G., and Perdomo-Ortiz, A., "Generative Learning of Continuous Data by Tensor Networks." 2024. arXiv:2310.20498 [cs.LG].

[5] Open Source Robotics Foundation, "TurtleBot," https://www.turtlebot.com/, 2024.

[6] Rieser, H.-M., Köster, F., and Raulf, A. P., "Tensor networks for quantum machine learning," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **479** no. 2275, (July 2023) .

[7] Smith, R., Schwartenbeck, P., Parr, T., and Friston, K. J., "An Active Inference Approach to Modeling Structure Learning: Concept Learning as an Example Case," *Frontiers in Computational Neuroscience* **14** (May 2020) 41.

— The only time I have problems is when I sleep.

Tupac Shakur

A

Computational details, or how to build a tree for sophisticated active inference

The tree defined by the sophisticated active inference scheme can be computationally very expensive to compute. There are multiple ways to approach this problem. This chapter shows some of the solutions that were explored during this PhD.

A.1 Introduction

An active inference agent's beliefs about the environment are encoded in a generative model $P(\tilde{o}, \tilde{s}, \tilde{a})$ in terms of observations $\tilde{o} = (o_0, o_1, \ldots, o_t)$, hidden states $\tilde{s} = (s_0, s_1, \ldots, s_t)$ and actions $\tilde{a} = (a_0, a_1, \ldots, a_t)$. An agent changes its beliefs about the environment and select actions to maximize the evidence $P(\tilde{o})$. Put differently, active inference agents minimize their surprisal $-\log P(\tilde{o})$. In theory, beliefs are updated through Bayesian inference. However, since this computation is often intractable, inference is performed through variational Bayesian methods instead. This entails minimizing an upper bound to the surprisal called the variational free energy [4],

$$-\log P(\tilde{o}) \leq F[Q, \tilde{o}] = -\mathrm{E}_{Q(\tilde{s},\tilde{a})}\left[\log Q(\tilde{s}, \tilde{a}) - \log P(\tilde{o}, \tilde{s}, \tilde{a})\right] , \tag{A.1}$$

which yields an approximation to the posterior $P(\tilde{s}, \tilde{a}|\tilde{o})$ denoted by $Q(\tilde{s}, \tilde{a})$. At the same time, agents select actions based on the expected free energy in the future [3],

$$G[Q] = \mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s},\tilde{a})}\left[\log Q(\tilde{s}, \tilde{a}) - \log P^*(\tilde{o}, \tilde{s})\right] , \tag{A.2}$$

where the sequences now refer to observations, states and actions that will occur in the future, and $P^*(\tilde{o}, \tilde{s})$ is a distribution that encodes the agents preferences, i.e. its goals. Actions are sampled from the distribution $Q(\tilde{a})$ which minimizes the expected free energy. It can be shown that this distribution is given by [3]

$$Q(\tilde{a}) = \sigma\left(-\mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s}|\tilde{a})}\left[\log Q(\tilde{s}|\tilde{a}) - \log P^*(\tilde{o}, \tilde{s})\right]\right) , \tag{A.3}$$

where σ is a softmax function.

The expected free energy itself is a lower bound on the expected surprisal [3],

$$G[Q] = \underbrace{\mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s},\tilde{a})}\left[-\log P^*(\tilde{o})\right]}_{\text{expected surprisal}} - \underbrace{\mathrm{E}_{P(\tilde{o}|\tilde{s})Q(\tilde{s},\tilde{a})}\left[\log P(\tilde{s}|\tilde{o}) - \log Q(\tilde{s}, \tilde{a})\right]}_{\text{expected information gain}} . \tag{A.4}$$

For the sake of clarity, the remainder of this chapter will be based on the expected surprisal with marginalized out hidden states,

$$G[Q] = \mathrm{E}_{P(\tilde{o},\tilde{a})}\left[-\log P^*(\tilde{o})\right] . \tag{A.5}$$

This will simplify equations. Note that all equations and algorithms in the following sections can easily be extended for Eq. (A.2).

Usually, one only wants to sample the next action. In that case, the expected surprisal (Eq. (A.5)) is minimized by

$$Q(a_0) = \sigma\left(-\mathrm{E}_{P(\tilde{o},\tilde{a}|a_0)}\left[-\log P^*(\tilde{o}, \tilde{s})\right]\right) . \tag{A.6}$$

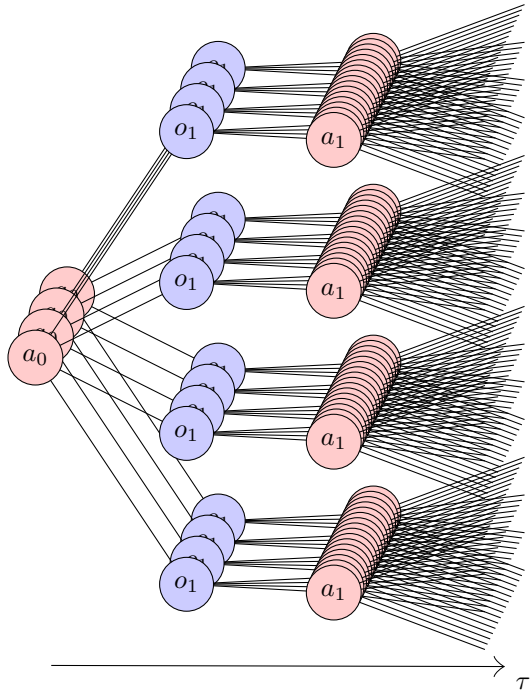


Figure A.1: Example of the set of rooted trees defined by Eq. (A.7) for an environment with 4 actions and 4 observations. The number of leaf nodes increases exponentially: each action spawns 4 observations, and, in turn, each observation spawns 4 actions.

This form is rather difficult to work with. For this reason, the equation is usually expanded over time. That way, Eq. (A.6) can be expanded into a recursive form:

$$G(o_\tau, a_\tau) = \underbrace{\mathrm{E}_{P(o_{\tau+1}|o_{<\tau+1}, a_{<\tau+1})}\left[-\log P^*(o_{\tau+1})\right]}_{\text{expected surprisal of next action}} + \underbrace{\mathrm{E}_{P(o_{\tau+1}|o_{<\tau+1}, a_{<\tau+1})P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1})}\left[G(o_{\tau+1}, a_{\tau+1})\right]}_{\text{expected surprisal of subsequent actions}}, \tag{A.7}$$

where it is chosen that

$$P(a_{\tau+1}|o_{\leq\tau+1}, a_{<\tau+1}) = \sigma\left(-G(o_{\tau+1}, a_{\tau+1})\right) . \tag{A.8}$$

This is often referred to as sophisticated inference [2]. Eq. (A.8) makes the equations self-consistency, i.e. it ensures that subsequent actions are treated according to the same rules as the action currently under consideration.

The recursive form in Eq. (A.7) defines a rooted tree with alternating observation and action nodes for each possible action (see Fig. A.1). The number of leaf nodes grows exponentially with the depth of the tree. This can make it a challenge to create efficient exact algorithms. The following sections suggest increasingly efficient methods.

A.2 Naive for-loops

The naive method to perform the calculation in Eq. (A.7) is to use nested for-loops. Implementing this is relatively easy, however, execution is very inefficient. It requires each node of the tree to be visited separately. Since the number of nodes in the tree grows with $\mathcal{O}(|A|^\tau|O|^\tau)$, where A and O are the sets containing all possible actions and observations and $|A|$ and $|O|$ are the cardinality of these sets, the time complexity does too.

Algorithm A.1 shows a recursive depth-first method. It computes the surprisal at the leaf nodes first and subsequently aggregates the results. In fact, this is the only possible way of traversing the tree, since Eq. (A.7) requires knowledge of the expected surprisal of future time steps in order to compute the expected surprisal of the current time step.

The algorithm proceeds by first specifying an initial observation o_0 and action a_0, and afterwards going through alternating for-loops over subsequent observations and actions. Performing this computation for each possible

Algorithm A.1: A naive algorithm using for-loops.

$\tilde{o} \leftarrow \{o_0\}$
$\tilde{a} \leftarrow \{a_0\}$
`NaiveForLoops`$(\tilde{o}, \tilde{a}, h; P, P^*)$
for $o \in O$ **do** `// for-loop over observations`
  if $h > 1$ **then**
    for $a \in A$ **do** `// for-loop over actions`
      $\tilde{o}' \leftarrow \{o\} + \tilde{o}$
      $\tilde{a}' \leftarrow \{a\} + \tilde{a}$
      $G_{o,a} \leftarrow$ `NaiveForLoops`$(\tilde{o}', \tilde{a}', h-1; P, P^*)$
    $Q_{o,a} \leftarrow \sigma(G_{o,a})$
    $\bar{G}_o \leftarrow \sum_a Q_{o,a} G_{o,a}$
  else
    $\bar{G}_o \leftarrow 0$
  $F_o \leftarrow P(o|\tilde{o}, \tilde{a})(-\log P^*(o) + \bar{G}_o)$
$F \leftarrow \sum_o F_o$
return F

initial action, yields an array of expected surprisal per action $G(a_0)$.

A.3 Vectorization

It is possible to improve on the naive implementation in the previous section through vectorization. This section presents an algorithm that vectorizes the levels of the tree. This method does not require each leaf node to be visited individually and speeds up computation considerably. However, it comes with its own downsides in terms of space complexity.

Vectorization entails constructing a large array with a path in each row, and subsequently working backwards from future time steps to the current time step. Algorithm A.2 shows how this works. In this algorithm, V contains an array of all possible combinations of actions and observations of length $2h$ and is procedurally expanded according to the desired depth, or time horizon, h. Subsequent steps are similar to the previous algorithm.

While at first glance, vectorization speeds up computation considerably, it comes with its own set of problems. Specifically, V can become very large: at time horizon h, there are $|A|^h|O|^h$ possible paths. Thus, the algorithm requires an array with $2h|A|^h|O|^h$ elements at the last time step. With $h = 6$, $|A| = 4$ and $|O| = 16$, such as in the OpenAI Gym environment Frozen Lake [1], assuming 32 bit integers, this equates to approximately 2200 GB of memory. Nevertheless, in the ideal case without memory constraints and perfectly parallelized operations, the algorithm's running time increases with $\mathcal{O}(h)$. Note that this is a lower limit. In reality, there are a lot of technical limitations that prevent this limit from being reached.

Algorithm A.2: A vectorized algorithm using an array of all possible combinations of actions and observations of length $2h$.

$V \leftarrow \{o_0\} \times A$ `// initial array of possible paths`

`Vectorization(`$V, h; P, P^*$`)`

if *h > 1* **then**
  $V' \leftarrow V \times O \times A$ `// expand paths by one time step`
  $G_{V,O,A} \leftarrow$ `Vectorization(`$V', h-1; P, P^*$`)`
  $Q_{V,O,A} \leftarrow \sigma(-G_{V,O,A})$
  $\bar{G}_{V,O} \leftarrow \sum_a Q_{V,O,a} G_{V,O,a}$
else
  $\bar{G}_{V,O} \leftarrow 0$
$F_V \leftarrow \sum_o P(o|V)(-\log P^*(o) + \bar{G}_{V,o})$
return F_V

A.4 Pruning

The algorithm in the previous section has a relatively serious problem regarding memory requirements. Although this issue will always remain due to the nature of the expected free energy, it is possible to alleviate it. Admittedly, the previous algorithm overlooks the fact that not all combinations of actions and observations are viable under the generative model P. Pruning the paths on the tree that are unlikely according to the model reduces memory requirements.

Algorithm A.3 shows how to prune paths on-the-fly through a simple masking method. Evidently, the operations in lines 2–5 and 13–17 can be vectorized, but have been expanded for clarity. The algorithm requires defining a threshold parameter γ on the transition distribution $P(o|V)$, e.g. $\gamma = 0.01$. This threshold is used to create a mask $M_{V,o}$, which is to be applied to the set of paths $V \times O$. In short, this mask defines which observations should be included and which should not. The remaining set of

Algorithm A.3: A vectorized algorithm using a pruned array of possible combinations of actions and observations of length $2h$.

$V \leftarrow \{o_0\} \times A$
`Vectorization(`$V, h; P, P^*$`)`
1 **if** $h > 1$ **then**
2 $\quad M_{v,o} \leftarrow P(o|V) > \gamma$ `// create mask`
3 $\quad W \leftarrow \{\}$
4 $\quad$ **for** $(v, o) \in V \times O$ **do** `// apply mask`
5 $\quad\quad$ **if** $M_{v,o}$ *is true* **then**
6 $\quad\quad\quad W \leftarrow W + \{(v, o)\}$
7 $\quad V' \leftarrow W \times A$
8 $\quad G'_{W,A} \leftarrow$ `Vectorization(`$V', h-1; P, P^*$`)`
9 $\quad$ **for** $(v, o) \in V \times O$ **do** `// undo mask`
10 $\quad\quad$ **if** $(v, o) \in W$ **then**
11 $\quad\quad\quad G_{v,o,A} \leftarrow G'_{v,o,A}$
12 $\quad\quad$ **else**
13 $\quad\quad\quad G_{v,o,A} \leftarrow \max G'_{W,A}$
14 $\quad Q_{V,O,A} \leftarrow \sigma(-G_{V,O,A})$
15 $\quad \bar{G}_{V,O} \leftarrow \sum_a Q_{V,O,a} G_{V,O,a}$
16 **else**
17 $\quad \bar{G}_{V,O} \leftarrow 0$
18 $F_V \leftarrow \sum_o P(o|V)(-\log P^*(o) + \bar{G}_{v,o})$
19 **return** F_V

observations is then used to continue the tree.

After obtaining the expected surprisal of subsequent actions, it is necessary to undo the mask. Indeed, since the action that will be selected is the one with the smallest expected surprisal, unlikely paths should yield a large expected surprisal. Therefore, one should not forget about the masked paths. Instead, the expected surprisal for these paths is set to the largest expected surprisal found among the viable paths. This ensures that the expected surprisal for these paths is large and automatically set to a reasonable value, i.e. it avoids additional hyperparameters.

This method allows a relatively significant reduction in memory. In the Frozen Lake environment, depending on the model's accuracy and the value of γ, it essentially reduces $|O|$ from 16 to 3. In other words, it reduces the memory requirement of V at $h = 6$ from 2200 GB to approximately 0.1 GB.

A.5 Runtime

A final improvement that can be made relates to runtime, during which the expected free energy must be computed multiple times in quick succession for different time steps. Although the agent plans several steps ahead, it only performs one action at a time. Therefore, when the model does not change at runtime, e.g. when *not* performing online learning, the probabilities $P(o_{\tau+1}|o_{<\tau+1}, a_{<\tau+1})$ also remain unchanged. As a result, these probabilities can be reused for the next time step. The workflow is described in Procedure Runtime.

Procedure Runtime for sophisticated active inference.

```
initialize the tree using the initial observation
repeat
    obtain a new action through action selection with model P
    obtain a new observation from the environment
    if the new action-observation pair is in the tree then
        remove the branches that have become useless
        compute the last layer of the tree
    else
        reinitialize the tree using the new observation
until end of episode
```

A.6 References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym." 2016. `arXiv:1606.01540 [cs.LG]`.

[2] Friston, K., Da Costa, L., Hafner, D., Hesp, C., and Parr, T., "Sophisticated Inference," *Neural Computation* **33** no. 3, (Mar. 2021) 713–763.

[3] Millidge, B., Tschantz, A., and Buckley, C. L., "Whence the Expected Free Energy?" *Neural Computation* **33** no. 2, (Feb. 2021) 447–482.

[4] Parr, T. and Friston, K. J., "Generalised free energy and active inference," *Biological Cybernetics* **113** no. 5–6, (Sep. 2019) 495–513.

— Without enough sleep, we all become tall two-year-olds.

JoJo Jensen

B
Essential algorithms

Some of the algorithms used throughout this dissertation were not explicitly written out. The main reason is that most chapters consist of published papers and the required algorithms can be found in the literature. In order to make this dissertation more self-contained, the relevant algorithms are included in this appendix.

B.1 Two-site updates

The two-site update algorithm used in Chapter 4 is largely based on the density matrix renormalization group (DMRG), a variational method for finding the lowest energy state of 1-dimensional quantum spin system. In the language of matrix product states (MPS), DMRG becomes conceptually relatively simple. The main idea is to update the state by performing local updates, sweeping back and forth across the system until a minimum is reached. The difference with DMRG is in the updates: while DMRG solves an eigenvalue problem each time, the proposed algorithm performs stochastic gradient descent (SGD) based on a loss function. Similar two-site update algorithms have been described by Stoudenmire and Schwab [2] and Han et al. [1].

B.1.1 Updating tensors

The algorithm at hand was described in the appendix of one of our earlier papers. The following excerpt is Appendix 2 of Wauthier et al. [3] and follows the notation of Eq. (4.4) and (4.5).

Learning Generative Models for Active Inference Using Tensor Networks — Appendix 2

S. T. Wauthier • B. Vanhecke • T. Verbelen • B. Dhoedt
Published in Proceedings of the International Workshop on Active Inference, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), 2022.

The loss function must be chosen in such a way that the model captures the probability distribution of the data [1]. A straightforward method for estimating the parameters of a probability distribution is maximum likelihood estimation. In machine learning terms, this means we will optimize the parameters of the model with respect to the negative log-likelihood (NLL):

$$\mathcal{L} = -\frac{1}{|D|} \sum_{x \in D} \log P(x), \tag{B.1}$$

where D denotes the data set. Through NLL minimization, the generative model becomes more similar to the probability distribution of the data.

Training proceeds as depicted in Figure B.1. Firstly, tensor $T^{(i)}$ and $T^{(i+1)}$ are contracted to form the tensor $B^{(i,i+1)}$. The update to $B^{(i,i+1)}$ is

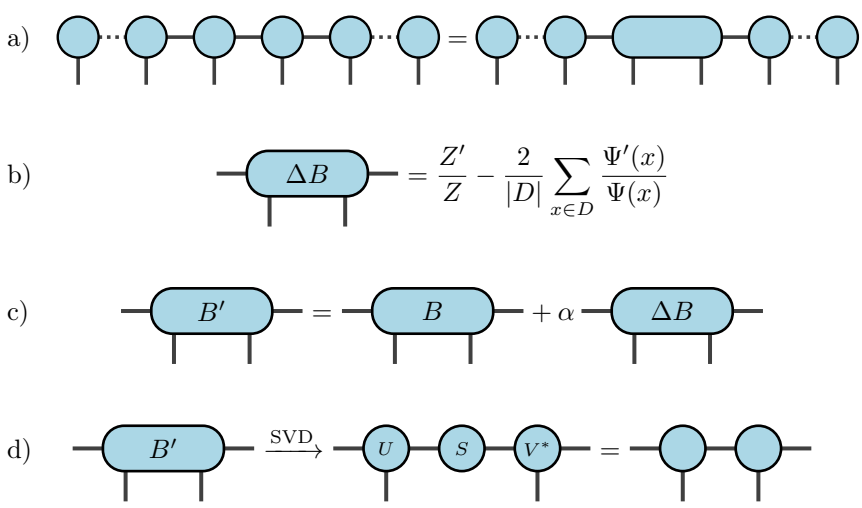


Figure B.1: Training scheme for an MPS. a) Contraction of two adjacent tensors. b) Computing the update to the contracted tensor. c) Updating the contracted tensor. d) Decomposition of the contracted tensor using SVD.

then computed using the loss function:

$$\Delta B^{(i,i+1)} = \frac{\partial \mathcal{L}}{\partial B^{(i,i+1)}_{\alpha_{i-1}\beta_i\beta_{i+1}\alpha_{i+1}}} = \frac{Z'}{Z} - \frac{2}{|D|} \sum_{x \in D} \frac{\Psi'(x)}{\Psi(x)}, \tag{B.2}$$

where $Z' = 2 \sum_{x \in D} \Psi'(x)\Psi(x)$ and Ψ' is the derivative of Ψ with respect to $B^{(i,i+1)}$. Subsequently, the elements of $B^{(i,i+1)}$ are adjusted by adding $\Delta B^{(i,i+1)}$ multiplied by the learning rate. Finally, the newly computed $B'^{(i,i+1)}$ is decomposed into two tensors again. This decomposition is typically done through singular value decomposition (SVD), where the singular value matrix is then contracted with either the left or the right tensor, such that we are left with two tensors.

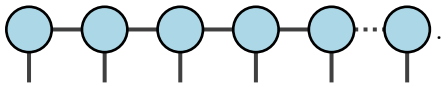
By starting this scheme at the leftmost tensor and iteratively moving one tensor to the right, the algorithm can update the entire MPS. Indeed, it is possible to update one tensor $T^{(i)}_{\alpha_{i-1}\beta_i\alpha_i}$ at a time, however, the current method allows the dimensions of the indices α_i (graphically, the edges connecting $T^{(i)}$ nodes), the so-called bond dimensions, to vary during training. This is made possible by truncated SVD, which truncates dimensions with singular values that fall beneath some manually specified threshold. Truncating dimensions with small singular values can be interpreted as truncating less informative dimensions. As a result, truncated SVD ensures that the

model remains as small as possible, while containing the most information. Moreover, the size of the model will vary depending on how much information it must learn.

B.1.2 Sweeping

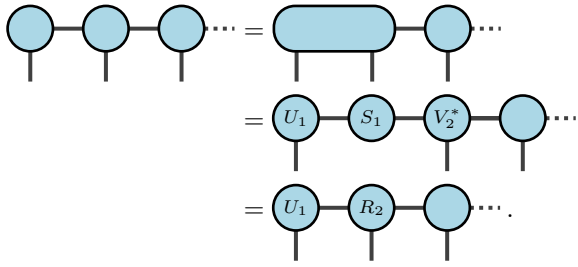
The procedure above omits an important detail. In order to simplify the equations used to update $B^{(i,i+1)}$, as well as, perform a mathematically precise compression of Ψ, i.e., perform truncated SVD with minimal truncation error, it is important that the MPS be placed in a specific gauge, called the canonical gauge.

Note that SVD provides two unitary matrices: U and V^*. Using this, the MPS can be placed in a form where all the tensors, except the rightmost tensor, are unitary. To see this, take an arbitrary MPS,

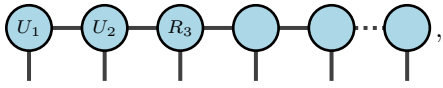


(B.3)

Start by contracting tensors $T^{(1)}$ and $T^{(2)}$ to form $B^{(1,2)}$. Then, perform SVD on $B^{(1,2)} = U_1 S_1 V_2^*$ and absorb S_1 into V_2^*, $R_2 = S_1 V_2^*$,

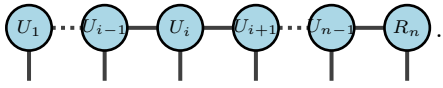


Now, repeat this for the newly formed R_2 and $T^{(3)}$,



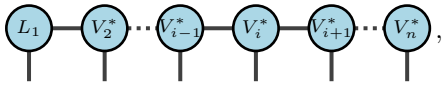
(B.4)

and continue until the last tensor is reached,



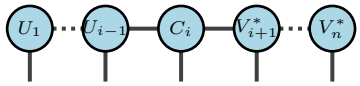
(B.5)

This is the left-canonical gauge. The right-canonical gauge can be obtained in an analogous way through $L_i = U_i S_i$,



(B.6)

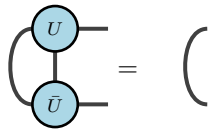
where now all the tensors, except the leftmost tensor, are unitary. Finally, it is possible to combine the two gauges into a mixed canonical gauge,



(B.7)

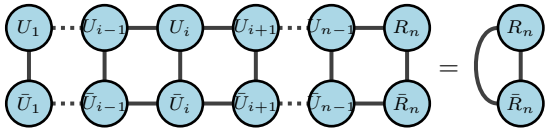
where the center tensor C_i is obtained by placing tensors to the left of site i in the left-canonical gauge, and to the right of site i in the right-canonical gauge. Alternatively, one can use QR decomposition on individual tensors to obtain the same results.

To understand the usefulness of canonical gauges, observe that



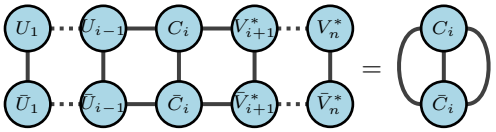
(B.8)

Using this, computation of the norm Z becomes much simpler. In left-canonical gauge,

$Z =$ 

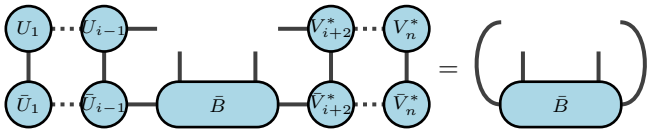
(B.9)

or in mixed canonical gauge,

$Z =$ 

(B.10)

Similarly, for the derivative Z' defined above,

$Z' =$ 

(B.11)

The moving of the mixed canonical center tensor C_i is generally what is referred to as "sweeping". Evidently, this can be performed during the update algorithm described in the previous section (Fig. B.1d) and does not require any extra steps. The MPS will initially be placed in left- or right-canonical form, after which, the update algorithm will automatically keep it in canonical form through the final SVD step.

B.2 References

[1] Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P., "Unsupervised Generative Modeling Using Matrix Product States," *Physical Review X* **8** no. 3, (July 2018) 031012.

[2] Stoudenmire, E. and Schwab, D. J., "Supervised Learning with Tensor Networks," in *Advances in Neural Information Processing Systems*, Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., eds., vol. 29. Curran Associates, Inc., 2016.

[3] Wauthier, S. T., Vanhecke, B., Verbelen, T., and Dhoedt, B., "Learning Generative Models for Active Inference Using Tensor Networks," in *Active Inference*, Buckley, C. L., Cialfi, D., Lanillos, P., Ramstead, M., Sajid, N., Shimazaki, H., and Verbelen, T., eds., pp. 285–297. Springer Nature Switzerland, 2023.