MACHINE LEARNING
Science and Technology

PURPOSE-LED PUBLISHING™

PAPER • **OPEN ACCESS**

Planning with tensor networks based on active inference

To cite this article: Samuel T Wauthier *et al* 2024 *Mach. Learn.: Sci. Technol.* **5** 045012

View the article online for updates and enhancements.

You may also like

- Hand-waving and interpretive dance: an introductory course on tensor networks
Jacob C Bridgeman and Christopher T Chubb
- Learning relevant features of data with multi-scale tensor networks
E Miles Stoudenmire
- Modeling sequences with quantum states: a look under the hood
Tai-Danae Bradley, E M Stoudenmire and John Terilla

This content was downloaded from IP address 178.117.34.188 on 10/10/2024 at 11:27

IOP Publishing Mach. Learn.: Sci. Technol. 5 (2024) 045012 https://doi.org/10.1088/2632-2153/ad7571

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
6 March 2024

REVISED
10 July 2024

ACCEPTED FOR PUBLICATION
29 August 2024

PUBLISHED
10 October 2024

Original Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

Planning with tensor networks based on active inference

Samuel T Wauthier[1,*], **Tim Verbelen**[2], **Bart Dhoedt**[1] **and Bram Vanhecke**[3,*]

[1] IDLab, Department of Information Technology at Ghent University—imec,Technologiepark-Zwijnaarde 126, B-9052 Ghent, Belgium
[2] VERSES AI Research Lab,Los Angeles, CA 90016, United States of America
[3] Faculty of Physics and Faculty of Mathematics, Quantum Optics, Quantum Nanophysics and Quantum Information, University of Vienna, Boltzmanngasse 5, Vienna 1090, Austria
[*] Authors to whom any correspondence should be addressed.

E-mail: samuel.wauthier@ugent.be and bram.andre.roland.vanhecke@univie.ac.at

Keywords: active inference, generative modeling, planning, tensor networks, matrix product state

Abstract

Tensor networks (TNs) 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 TN-based agent acts Bayes optimally as expected under active inference.

1. Introduction

Tensor networks (TNs) were originally developed to represent quantum states in many-body quantum physics [10, 14, 50, 52, 71], 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 TNs in a supervised manner to perform image classification on the Modified National Institute of Standards and Technology (MNIST) handwritten digits data set [40]. Furthermore, different types of TNs 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 TNs (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 (MPOs) for probabilistic modeling, while Stokes and Terilla [64] investigated the use of MPS for the same purpose. Similarly, Glasser *et al* [25] investigated how graphical models can map to generalized TNs 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 TNs with actor-critic techniques. Finally, although this work is situated largely within AI planning, it differentiates itself from earlier work

© 2024 The Author(s). Published by IOP Publishing Ltd

through probabilistic modeling of action-observation sequences with action selection based on active inference, as opposed to learning some RL policy.

The use of TNs in ML 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. TNs 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 [2–4, 30, 60] and they have also been shown to efficiently capture low energy physics [28, 68, 78]. 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 TN description in that case. Such states could also be useful to ML, 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 [26, 37] and interpretability [1, 57, 67] of TNs, leading to a relatively good understanding compared to neural networks. In fact, often TNs are used to characterize neural networks [7, 11, 24, 37, 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], TN-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 [20]. 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.

TNs 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 [21, 33, 41]. Lastly, it has been proposed that cells require quantum coherence for information processing [17, 59], 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 [15]. 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 TN in certain systems [15, 16].

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 ML using TNs. In particular, we provide the first account of AI planning using TNs by making use of the ability of MPS to model sequences and the action selection algorithm provided by active inference. From a ML 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 TN. 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 2, we introduce the basic concepts that constitute this work. These include MPS (section 2.1), generative modeling with TNs (section 2.2), and active inference (section 2.3). In section 3, we present our methods. We demonstrate the results of the experiments in section 4 and provide a discussion on scalability, and data and exploration in section 5. We conclude this work with section 6 by summarizing our findings and presenting possible paths for future work.

2. Background

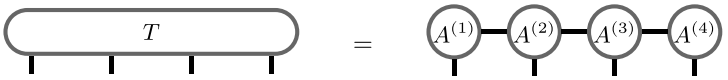
2.1. MPS

At its core, a TN 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 MPS [10, 14, 38, 39], TTN [52, 62], which are TNs without loops in their graphical representation and can therefore be treated with similar ease as MPS, and PEPS [10, 52, 70], which are TNs designed for two dimensional quantum systems and hence their graphical representation is also manifestly two dimensional (figure 1). In particular, the loops in the PEPS make them significantly more challenging to work with.

Arguably the simplest form of TN is the MPS, also known as 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:

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

or graphically:



(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 equation (2) to hold. For an MPS to be able to exactly represent an arbitrary tensor with Nd-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].

2.2. Generative modeling with TNs

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:

$$P(X = x) = \frac{|\Psi(x)|^2}{Z} \ , \tag{3}$$

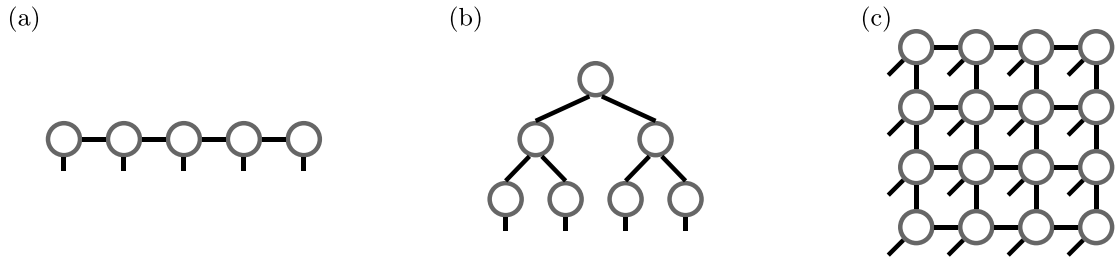


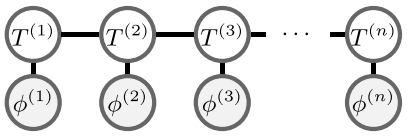
Figure 1. Popular types of tensor networks: (a) a matrix product state (MPS), a 1D array of tensors, (b) a tree tensor network (TTN), a set of tensors arranged in a tree structure, and (c) a projected entangled pair state (PEPS), a 2D array of tensors.

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:

$$\Psi(x) = \sum_{\alpha_1,\ldots,\alpha_{n-1},\beta_1,\ldots,\beta_n} T^{(1)}_{\beta_1\alpha_1} T^{(2)}_{\alpha_1\beta_2\alpha_2} T^{(3)}_{\alpha_2\beta_3\alpha_3} \cdots T^{(n)}_{\alpha_{n-1}\beta_n} \phi^{(1)}_{\beta_1}(x_1)\, \phi^{(2)}_{\beta_2}(x_2)\, \phi^{(3)}_{\beta_3}(x_3) \cdots \phi^{(n)}_{\beta_n}(x_n) \quad . \tag{4}$$

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

$\Psi(x) =$  . (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 TN, 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 three-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 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 TN 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* 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 (see [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 TN starts out with a low number of parameters and grows based on how many dimensions are needed.

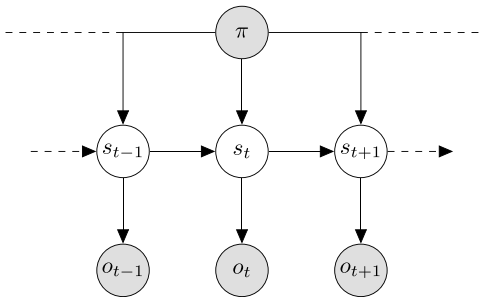


Figure 2. Bayesian network corresponding to the generative model $P(\tilde{o},\tilde{s},\pi)$ in discrete time active inference. The model is defined in terms of sequences of observations o_t and hidden states s_t evolving over time, and a policy $\pi(t) = a_t$ with a_t the selected action at time t.

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

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, $a = \pi(s)$ [20]. 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]

$$\mathfrak{I}(\tilde{o}) = -\log P(\tilde{o}) \quad , \tag{6}$$

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) [20]. In that case, the generative model $P(\tilde{o},\tilde{s},\pi)$ factorizes as in figure 2,

$$P(\tilde{o},\tilde{s},\pi) = P(\pi)\,P(s_0)\,P(o_0|s_0)\prod_{t=1}^{\tau} P(s_t|s_{t-1},\pi)\,P(o_t|s_t) \quad , \tag{7}$$

and the agent instead minimizes its variational free energy

$$F[Q,\tilde{o}] = E_{Q(\tilde{s},\pi)}\left[\log Q(\tilde{s},\pi) - \log P(\tilde{o},\tilde{s},\pi)\right] \tag{8}$$

$$= \underbrace{\mathrm{D}_{\mathrm{KL}}\left(Q(\tilde{s},\pi)\,||\,P(\tilde{s},\pi\,|\,\tilde{o})\right)}_{\text{divergence}} \underbrace{-\log P(\tilde{o})}_{\text{surprisal}} \quad , \tag{9}$$

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 (equation (9)) is an upper bound on the surprisal (equation (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]

$$G(\pi) = \mathrm{E}_{\hat{Q}(\tilde{o},\tilde{s}\,|\,\pi)}\left[\log Q(\tilde{s}\,|\,\pi) - \log P(\tilde{o},\tilde{s}\,|\,\pi,C)\right] \tag{10}$$

$$\approx \underbrace{-\mathrm{E}_{\hat{Q}(\tilde{o}\,|\,\pi)}\left[\mathrm{D}_{\mathrm{KL}}\left(\hat{Q}(\tilde{s}\,|\,\tilde{o},\pi)\,||\,Q(\tilde{s}\,|\,\pi)\right)\right]}_{\text{information gain}} \underbrace{-\mathrm{E}_{\hat{Q}(\tilde{o}\,|\,\pi)}\left[\log P(\tilde{o}\,|\,C)\right]}_{\text{expected surprisal}} \quad , \tag{11}$$

where we have used $\hat{Q}(\tilde{o},\tilde{s}\,|\,\pi) := P(\tilde{o}\,|\,\tilde{s})Q(\tilde{s}\,|\,\pi)$ 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.

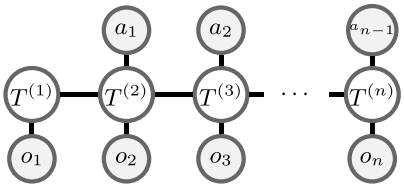
3. Methods

3.1. Adapting the MPS for action selection

When employing an TN 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 equation (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 x_k to refer to $\phi_{\mathrm{x}}(x_k)$, the network becomes

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



, (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.

3.2. Updating the MPS

Both update methods mentioned in section 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.

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 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.

3.3. Active inference with an MPS

Using an MPS, allows the agent to minimize surprisal (equation (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

$$\mathfrak{J}(\pi) = -\mathrm{E}_{P(\tilde{o}\,|\,\pi)}\left[\log P(\tilde{o}\,|\,C)\right] \quad . \tag{13}$$

In practice, computing the expected surprisal for each possible policy is computationally expensive. Using the sophisticated inference [19] 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:

$$\begin{aligned}\mathfrak{J}(o_t, a_t) = &\underbrace{-\mathrm{E}_{P(o_{t+1}\,|\,\tilde{o}_{<t+1},\tilde{a}_{<t+1})}\left[\log P(o_{t+1}\,|\,C)\right]}_{\text{expected surprisal of next action}} \\ &+\underbrace{\mathrm{E}_{P(a_{t+1}\,|\,o_{t+1})P(o_{t+1}\,|\,\tilde{o}_{<t+1},\tilde{a}_{<t+1})}\left[\mathfrak{J}(o_{t+1}, a_{t+1})\right]}_{\text{expected surprisal of subsequent actions}} \quad ,\end{aligned} \tag{14}$$

$$P(a_t\,|\,o_t) = \sigma\left[-\mathfrak{J}(o_t, a_t)\right] \quad . \tag{15}$$

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

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* [20]. 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.

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.

4.1. 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 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

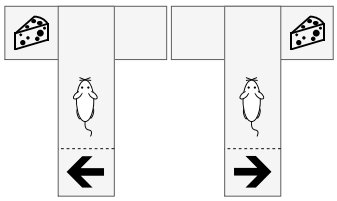


Figure 3. Possible instances of the T-maze environment as presented by Friston *et al* [20]. The agent (mouse) starts off in the center. The reward (cheese) is located in either the left branch (left instance) or right branch (right instance). The cue reveals the location of the reward: a left-pointing arrow indicates the reward is on the left, a right-pointing arrow indicates the reward is on the right.

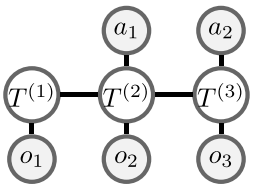
the correct context with probability 1 when the agent is in the cue location. Lastly, the possible actions that the agent can take include: center, right, left, and cue, corresponding to which location the agent wants to go.

The above implementation was slightly modified to better reflect the environment designed by Friston *et al* [20]. 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.

4.1.1. Model

To accommodate the T-maze environment, the model (equation (12)) was limited in time as follows:

$$\Psi(\tilde{a},\tilde{o}) = \quad , \tag{16}$$



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.

4.1.2. 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.

4.1.2.1. 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.

4.1.2.2. 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 10 000, 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 10 000 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 50 000 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 10 000, this translates to 50 000, 25 000, 5000 and 500 epochs, respectively.

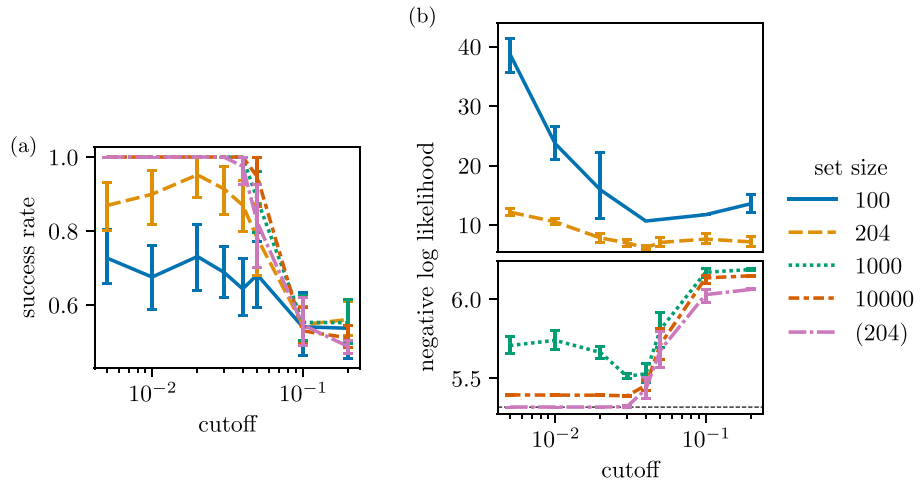


Figure 4. (a) Success rate as a function of cutoff and data set size. Each point reflects the average across 20 models of the success rate over 100 runs. (b) Average negative log likelihood over all possible sequences as a function of cutoff and data set size. Each point reflects the average across 20 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 4.1.2.

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.

4.1.3. Results

To evaluate the performance of each model, we measured the success rate over 100 runs using the action selection scheme presented in section 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 [20]:

$$P\left(\text{modality} \mid C_{\text{modality}}\right) = \sigma\left(C_{\text{modality}}\right), \quad \text{modality} \in \{\text{position}, \text{reward}, \text{context}\} \tag{17}$$

$$C_{\text{position}} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}, \quad C_{\text{reward}} = \begin{bmatrix} 0 & 3 & -3 \end{bmatrix}, \quad C_{\text{context}} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \tag{18}$$

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 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 (equation (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 4.1.2, demonstrate the belief shifts and action selection under the surprisal minimization scheme in section 3.3 and investigate performance when using single-site updates.

4.1.3.1. Performance

We first show the performance when the model is trained with only two-site updates and without single-site updates. Figure 4(a) 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

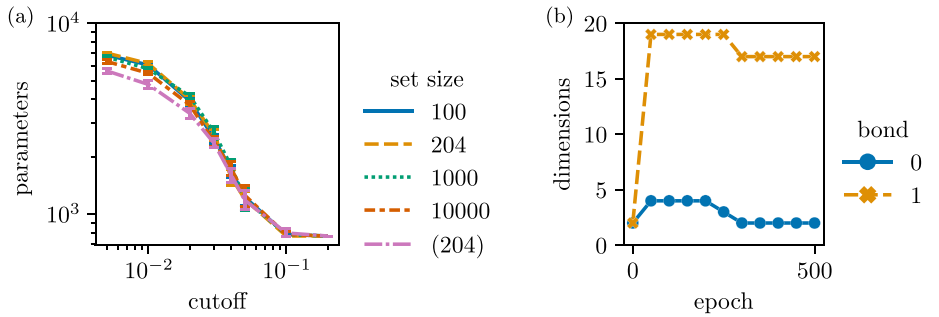


Figure 5. (a) Average number of parameters as a function of cutoff and data set size. Each point reflects the average across 10 models. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.1.2. (b) Evolution of bond dimensions during training 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.

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(b) 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 5(a).

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 5(b) 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.

4.1.3.2. Beliefs

When zooming in on a single well-performing model, one expects to see some shifts in the probability distributions provided by the TN 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 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

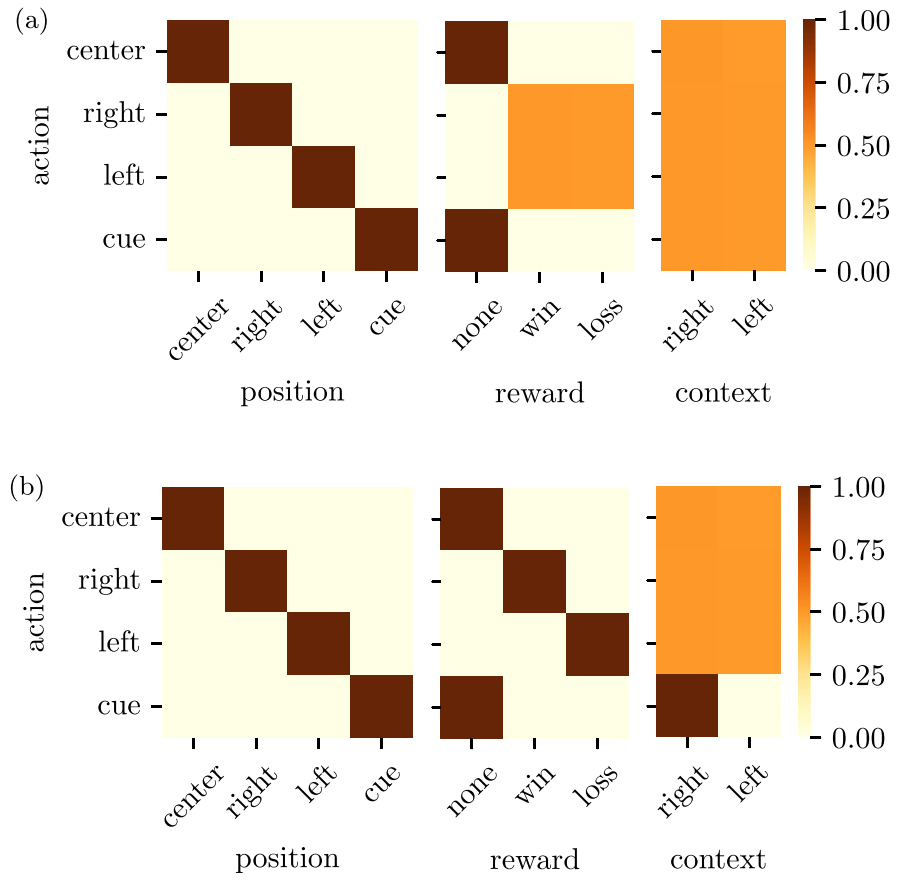


Figure 6. Belief shift (a) before and (b) after having seen the cue 'right'. Computed for a model with $\varepsilon = 0.01$ trained on the engineered data set.

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 (equation (14)) can provide some insight on how the agent behaves given the preferences in equation (18) and how it reaches the reward. Figure 7(a) 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 7(b) and (c), respectively. In these cases, it is clear the agent will 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.

4.1.3.3. Single-site updates

Figure 8(a) 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(a).

Figure 8(b) 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 5(b)). 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.

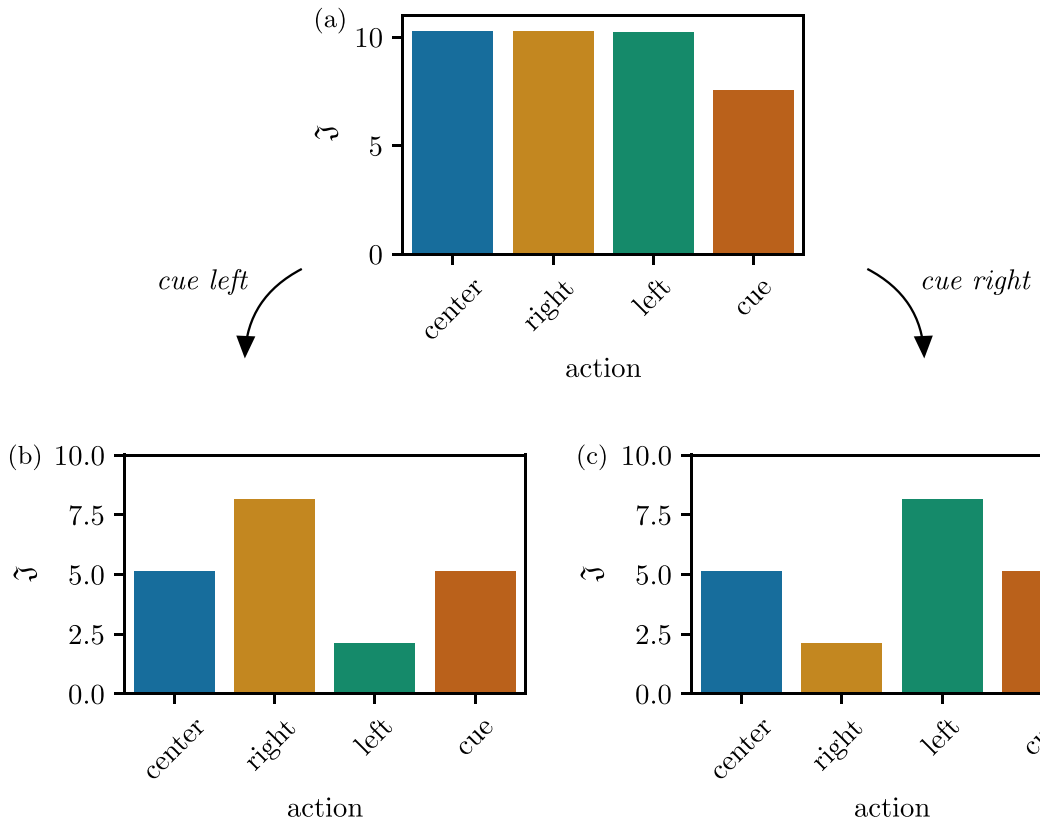


Figure 7. Expected surprisal: (a) for the first action, (b) for the second action after having seen the cue 'left', and (c) for the second action after having seen the cue 'right'. Computed for a model with $\varepsilon = 0.01$ trained on the engineered data set.

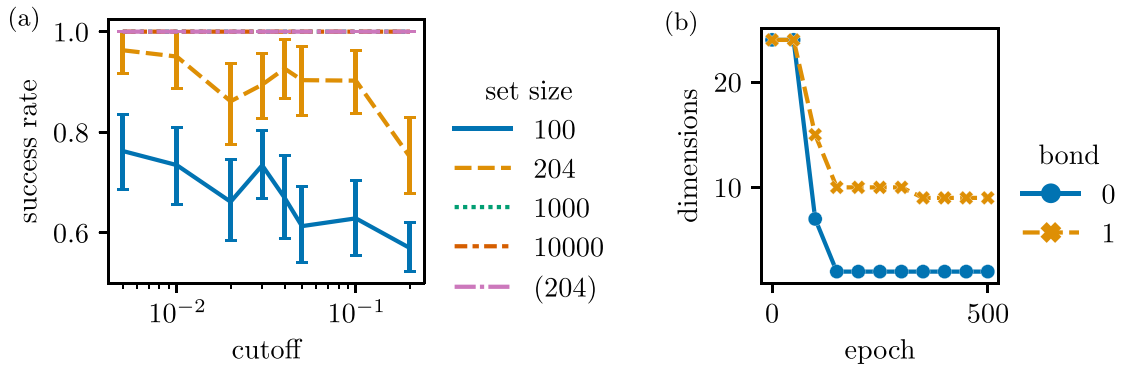
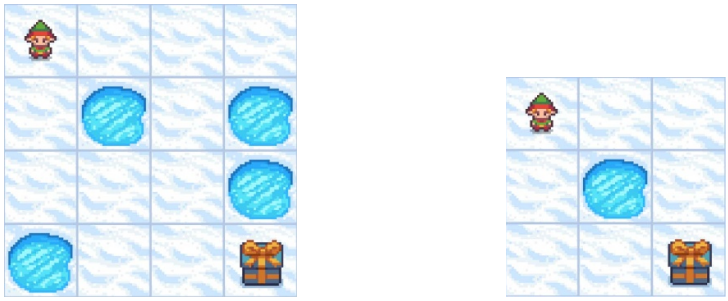


Figure 8. (a) Success rate obtained using single-site updates as a function of cutoff and data set size. Each point reflects the average across 20 models of the success rate over 100 runs. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.1.2. (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.

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 9(a)). 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.



(a) default 4×4 (b) custom 3×3

Figure 9. The Frozen Lake environment [6]. Reproduced with permission from [79]. [© 2021 M. Tillery. All rights reserved.].

It is possible to design custom environments, such as the 3×3-grid in figure 9(b), 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.

4.2.1. Model

For the Frozen Lake, the model (equation (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.

4.2.2. 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 100 096 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.

4.2.2.1. Cutoff

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

4.2.2.2. 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 1000 192 (another multiple of 256), such that it would contain more samples which reached the goal.

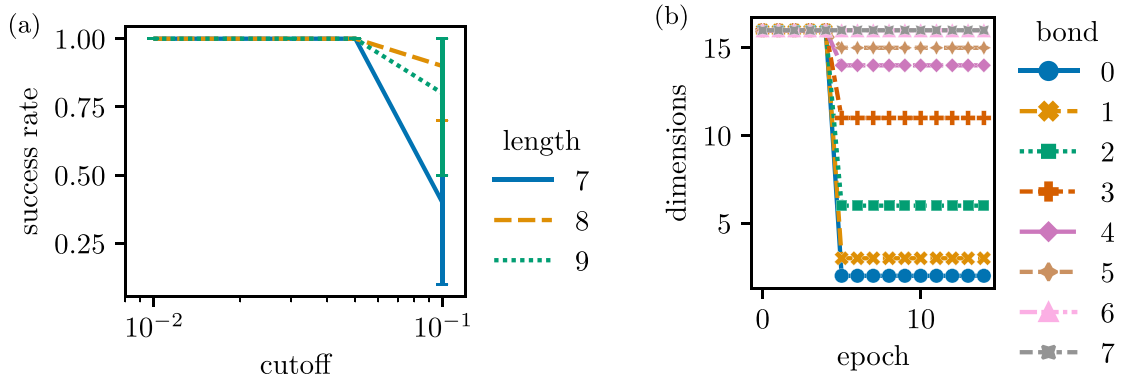


Figure 10. (a) Success rate for the non-slippery 4×4 environment as a function of cutoff and MPS length. Each point reflects the model average across three models. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.2.2. (b) Evolution of bond dimensions during training for a model with $\varepsilon = 0.03$ and MPS length 9 on the non-slippery 4×4 environment.

4.2.3. Results

To evaluate the performance of each model, we measured the success rate using the action selection scheme presented in section 3.3. Here too, the preferred distribution $P(o_t \,|\, C)$ remained constant for every t, such that for the 3×3 environment (cf figure 9(b))

$$P(o \,|\, C) = \sigma(C) \tag{19}$$

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

and for the 4×4 environment (cf figure 9(a))

$$P(o \,|\, C) = \sigma(C) \tag{21}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -5 & 0 & -5 \\ 0 & 0 & 0 & -5 \\ -5 & 0 & 0 & 20 \end{bmatrix} \tag{22}$$

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 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.

4.2.3.1. 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.

4.2.3.2. Non-slippery 4×4

Figure 10(a) 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. Once

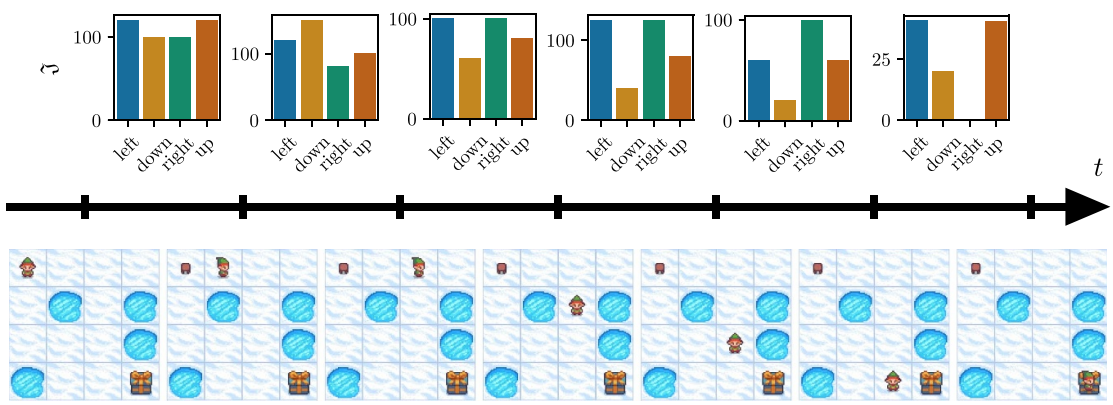
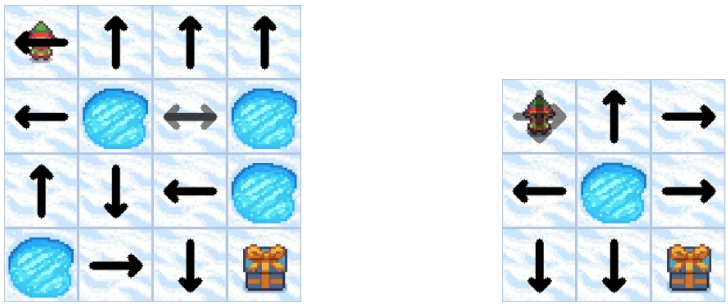


Figure 11. An example of an agent navigating the Frozen Lake environment. The agent's model was trained using $\varepsilon = 0.03$ and an MPS length of 9. The bottom row shows the observation at each time step, while the top row shows the expected surprisal per action computed after each observation. Reproduced with permission from [79]. [© 2021 M. Tillery. All rights reserved.].



(a) default 4×4 (b) custom 3×3

Figure 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. Reproduced with permission from [79]. [© 2021 M. Tillery. All rights reserved.].

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 10(b) 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 11 shows an example of how an agent can act within the non-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.

4.2.3.3. 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 12(b) 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 13(a) 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

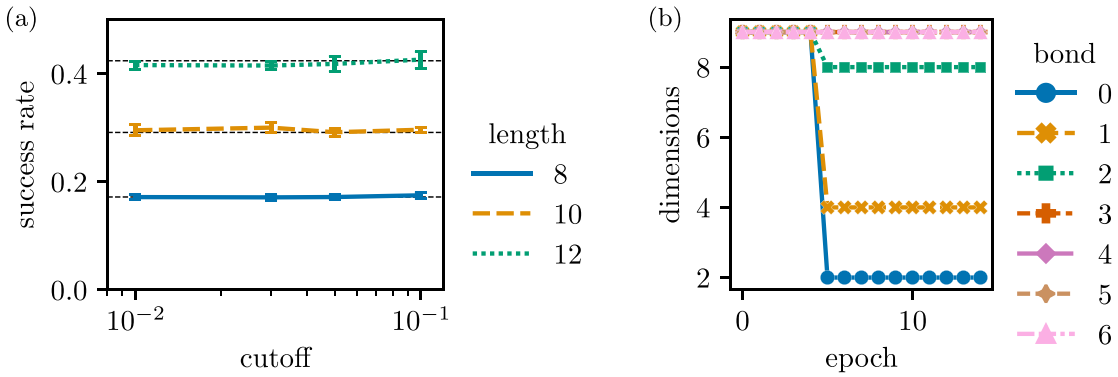


Figure 13. (a) Success rate obtained for the slippery 3 × 3 environment as a function of cutoff and MPS length. Each point reflects the average across five models of the success rate across 1000 runs. Error bars display a bootstrapped 95% confidence interval. Details on hyperparameters can be found in section 4.2.2. Horizontal lines indicate success rates for agents running the optimal policies (figure 12(b)) 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 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). Reproduced with permission from [79]. [© 2021 M. Tillery. All rights reserved.].

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 13(b) 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 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 12(b)), 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

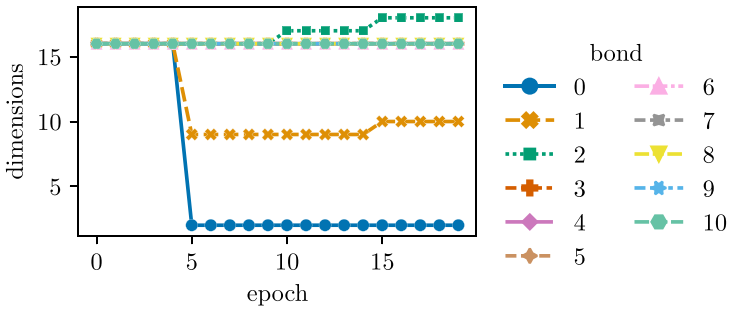


Figure 15. Evolution of bond dimensions during training for a model with $\varepsilon = 0.03$ and MPS length 12 on the slippery 4×4 Frozen Lake environment.

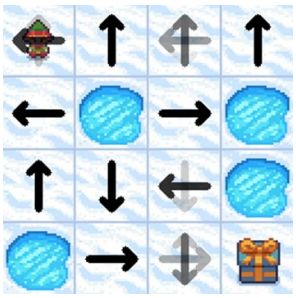


Figure 16. Policy obtained for the slippery 4×4 Frozen Lake environment for a model with $\varepsilon = 0.03$ and MPS length 12, by counting how often an action was taken at each position. The transparency of the arrow indicates the frequency of the action (less transparency is more frequent). Reproduced with permission from [79]. [© 2021 M. Tillery. All rights reserved.].

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.

4.2.3.4. Slippery 4×4

An increase in the size of the environment entails an increase in the depth of the decision tree defined by equation (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 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 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 12(a)) 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.

5. Discussion

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 SVD 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 [69], 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 ML. 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.

5.2. Data and exploration

In current ML 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 RL 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 RL 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 (equation (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 ML 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.

6. Conclusion and future work

We have shown that by combining TNs and active inference, it is possible to plan paths within discrete environments. In this work, we employed MPS 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 TNs 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'.

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.

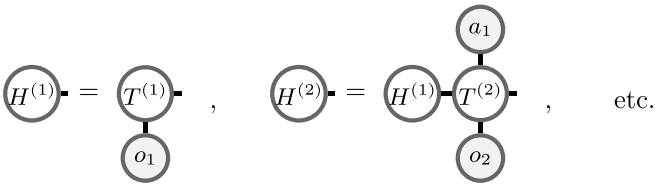
Acknowledgments

This research received funding from the Flemish Government under the 'Onderzoeksprogramma Artificiéle 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 www.cyaneus.com/ for all other assets.

Appendix. Computing probabilities from an MPS

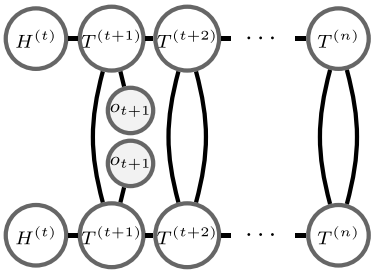
This section explains how to derive the probabilities that are required for the tree in section 3.3.

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.

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

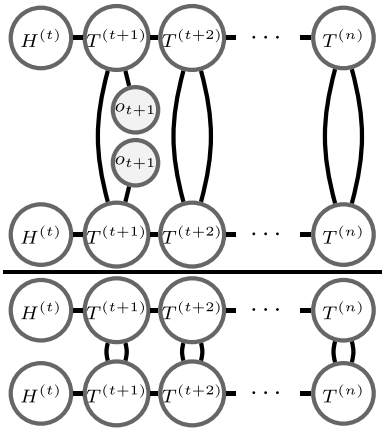


.

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

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

$=$



.

ORCID iD

Samuel T Wauthier https://orcid.org/0000-0002-1967-2195

References

[1] Aizpurua B, Palmer S and Orus R 2024 Tensor networks for explainable machine learning in cybersecurity (arXiv:2401.00867)
[2] Anshu A, Harrow A W and Soleimanifar M 2022 Entanglement spread area law in gapped ground states *Nat. Phys.* **18** 1362–6
[3] Arad I, Kitaev A, Landau Z, Vazirani U 2013 An area law and sub-exponential algorithm for 1D systems (arXiv:1301.1162)
[4] Arad I, Landau Z and Vazirani U 2012 Improved one-dimensional area law for frustration-free systems *Phys. Rev.* B **85** 195145
[5] Brandão F G S L and Horodecki M 2013 An area law for entanglement from exponential decay of correlations *Nat. Phys.* **9** 721–6
[6] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J and Zaremba W 2016 Openai gym (arXiv:1606.01540)
[7] Chen J, Cheng S, Xie H, Wang L and Xiang T 2018 Equivalence of restricted Boltzmann machines and tensor network states *Phys. Rev.* B **97** 085104

[8] Cheng S, Wang L, Xiang T and Zhang P 2019 Tree tensor networks for generative modeling *Phys. Rev.* B **99** 155131
[9] Cheng S, Wang L and Zhang P 2021 Supervised learning with projected entangled pair states *Phys. Rev.* B **103** 125117
[10] Cirac J I, Pérez-García D, Schuch N and Verstraete F 2021 Matrix product states and projected entangled pair states: concepts, symmetries, theorems *Rev. Mod. Phys.* **93** 045003
[11] Cohen N, Sharir O and Shashua A 2016 On the expressive power of deep learning: a tensor analysis *29th Annual Conf. on Learning Theory* vol 49, ed V Feldman, A Rakhlin and O Shamir (PMLR, Columbia University) pp 698–728
[12] Dalzell A M and Brandão F G S L 2019 Locally accurate MPS approximations for ground states of one-dimensional gapped local Hamiltonians *Quantum* **3** 187
[13] Eckart C and Young G 1936 The approximation of one matrix by another of lower rank *Psychometrika* **1** 211–8
[14] Fannes M, Nachtergaele B and Werner R F 1992 Finitely correlated states on quantum spin chains *Commun. Math. Phys.* **144** 443–90
[15] Fields C, Fabrocini F, Friston K, Glazebrook J F, Hazan H, Levin M and Marcianò A 2023 Control flow in active inference systems—part I: classical and quantum formulations of active inference *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **9** 235–45
[16] Fields C, Fabrocini F, Friston K, Glazebrook J F, Hazan H, Levin M and Marcianò A 2023 Control flow in active inference systems—part II: tensor networks as general models of control flow *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **9** 246–56
[17] Fields C and Levin M 2021 Metabolic limits on classical information processing by biological cells *Biosystems* **209** 104513
[18] Flatorion Institute The Tensor Network (available at: https://tensornetwork.org/) (Accessed 12 February 2023)
[19] Friston K, Da Costa L, Hafner D, Hesp C and Parr T 2021 Sophisticated inference *Neural Comput.* **33** 713–63
[20] Friston K, FitzGerald T, Rigoli F, Schwartenbeck P, O'Doherty J and Pezzulo G 2016 Active inference and learning *Neurosci. Biobehav. Rev.* **68** 862–79
[21] Friston K J, Lin M, Frith C D, Pezzulo G, Hobson J A and Ondobaka S 2017 Active inference, curiosity and insight *Neural Comput.* **29** 2633–83
[22] Gillman E, Rose D C, Garrahan J P 2020 A tensor network approach to finite markov decision processes (arXiv:2002.05185)
[23] Gillman E, Rose D C and Garrahan J P 2024 Combining reinforcement learning and tensor networks, with an application to dynamical large deviations *Phys. Rev. Lett.* **132** 197301
[24] Glasser I, Pancotti N, August M, Rodriguez I D and Cirac J I 2018 Neural-network quantum states, string-bond states and chiral topological states *Phys. Rev.* X **8** 011006
[25] Glasser I, Pancotti N and Cirac J I 2020 From probabilistic graphical models to generalized tensor networks for supervised learning *IEEE Access* **8** 68169–82
[26] Glasser I, Sweke R, Pancotti N, Eisert J and Cirac I 2019 Expressive power of tensor-network factorizations for probabilistic modeling *Advances in Neural Information Processing Systems* vol 32, ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché Buc, E Fox and R Garnett (Curran Associates, Inc)
[27] Guo C, Jie Z, Lu W and Poletti D 2018 Matrix product operators for sequence-to-sequence learning *Phys. Rev.* E **98** 042114
[28] Haegeman J, Osborne T J and Verstraete F 2013 Post-matrix product state methods: to tangent space and beyond *Phys. Rev.* B **88** 075133
[29] Han Z Y, Wang J, Fan H, Wang L and Zhang P 2018 Unsupervised generative modeling using matrix product states *Phys. Rev.* X **8** 031012
[30] Hastings M B 2007 An area law for one-dimensional quantum systems *J. Stat. Mech.* P08024
[31] Hayden P, Leung D W and Winter A 2006 Aspects of generic entanglement *Commun. Math. Phys.* **265** 95–117
[32] Heins C, Millidge B, Demekas D, Klein B, Friston K, Couzin I D and Tschantz A 2022 pymdp: a python library for active inference in discrete state spaces *J. Open Source Softw.* **7** 4098
[33] Hobson J and Friston K 2012 Waking and dreaming consciousness: neurobiological and functional considerations *Prog. Neurobiol.* **98** 82–98
[34] Howard S 2024 A tensor network implementation of multi agent reinforcement learning (arXiv:2401.03896)
[35] Hur Y, Hoskins J G, Lindsey M, Stoudenmire E M and Khoo Y 2023 Generative modeling via tensor train sketching (arXiv:2202.11788)
[36] Kerskens C M and Pérez D L 2022 Experimental indications of non-classical brain functions *J. Phys. Commun.* **6** 105001
[37] Khrulkov V, Novikov A and Oseledets I 2018 Expressive power of recurrent neural networks *Int. Conf. on Learning Representations*
[38] Klumper A, Schadschneider A and Zittartz J 1991 Equivalence and solution of anisotropic spin-1 models and generalized t-j fermion models in one dimension *J. Phys. A: Math. Gen.* **24** L955
[39] Klümper A, Schadschneider A and Zittartz J 1993 Matrix product ground states for one-dimensional spin-1 quantum antiferromagnets *Europhys. Lett.* **24** 293
[40] LeCun Y, Cortes C and Burges C J C 1998 *The mnist database of handwritten digits* (available at: http://yann.lecun.com/exdb/mnist/) (Accessed 12 February 2023)
[41] Li W, Ma L, Yang G and Gan W B 2017 REM sleep selectively prunes and maintains new synapses in development and learning *Nat. Neurosci.* **20** 427–37
[42] Liu D, Ran S J, Wittek P, Peng C, García R B, Su G and Lewenstein M 2019 Machine learning by unitary tensor network of hierarchical tree structure *New J. Phys.* **21** 073059
[43] Liu J, Li S, Zhang J and Zhang P 2023 Tensor networks for unsupervised machine learning *Phys. Rev.* E **107** L012103
[44] Liu X Y and Fang Y X 2020 Quantum tensor networks for variational reinforcement learning *First Workshop on Quantum Tensor Networks in Machine Learning*
[45] Lu Y and Ran S J 2023 Many-body control with reinforcement learning and tensor networks *Nat. Mach. Intell.* **5** 1058–9
[46] Mahajan A, Samvelyan M, Mao L, Makoviychuk V, Garg A, Kossaifi J, Whiteson S, Zhu Y and Anandkumar A 2021 Tesseract: tensorised actors for multi-agent reinforcement learning *Proc. 38th Int. Conf. on Machine Learning* vol 139, ed M Meila and T Zhang (PMLR) pp 7301–12
[47] Mencia Uranga B N and Lamacraft A 2020 Schrödingerrnn: generative modeling of raw audio as a continuously observed quantum state *Proc. First Mathematical and Scientific Machine Learning Conf.* vol 107, ed J Lu and R Ward (PMLR) pp 74–106
[48] Metz F and Bukov M 2023 Self-correcting quantum many-body control using reinforcement learning with tensor networks *Nat. Mach. Intell.* **5** 780–91
[49] Miller J, Rabusseau G and Terilla J 2021 Tensor networks for probabilistic sequence modeling *Proc. of The 24th Int. Conf. on Artificial Intelligence and Statistics* vol 130, ed A Banerjee and K Fukumizu (PMLR) pp 3079–87
[50] Murg V, Verstraete F, Legeza O and Noack R M 2010 Simulating strongly correlated quantum systems with tree tensor networks *Phys. Rev.* B **82** 205105

[51] Ng R 2020 Dynamic programming (available at: https://github.com/ritchieng/deep-learning-wizard/commits/master/docs/deep_learning/deep_reinforcement_learning_pytorch/dynamic_programming_frozenlake.ipynb) (Accessed 19 December 2023)
[52] Orús R 2019 Tensor networks for complex quantum systems *Nat. Rev. Phys.* **1** 538–50
[53] Oseledets I V 2011 Tensor-train decomposition *SIAM J. Sci. Comput.* **33** 2295–317
[54] Parr T, Pezzulo G and Friston K J 2022 *Active Inference: The Free Energy Principle in Mind, Brain and Behavior* (The MIT Press)
[55] Peng Y, Chen Y, Stoudenmire E M and Khoo Y 2023 Generative modeling via hierarchical tensor sketching (arXiv:2304.05305)
[56] Perez-Garcia D, Verstraete F, Wolf M M and Cirac J I 2007 Matrix product state representations *Quantum Inf. Comput.* **7** 401–30
[57] Ran S J and Su G 2023 Tensor networks for interpretable and efficient quantum-inspired machine learning *Intell. Comput.* **2** 0061
[58] Sajid N, Tigas P and Friston K 2022 Active inference, preference learning and adaptive behaviour *IOP Conf. Ser.: Mater. Sci. Eng.* **1261** 012020
[59] Schrödinger E 1944 *What is Life? the Physical Aspects of the Living Cell With Mind and Matter and Autobiographical Sketches* (Cambridge University Press)
[60] Schuch N, Wolf M M, Verstraete F and Cirac J I 2008 Entropy scaling and simulability by matrix product states *Phys. Rev. Lett.* **100** 030504
[61] Sharir O, Shashua A and Carleo G 2022 Neural tensor contractions and the expressive power of deep neural quantum states *Phys. Rev.* B **106** 205136
[62] Shi Y Y, Duan L M and Vidal G 2006 Classical simulation of quantum many-body systems with a tree tensor network *Phys. Rev.* A **74** 022320
[63] Srinivasan S, Gordon G and Boots B 2018 Learning hidden quantum markov models *Proc. 21st Int. Conf. on Artificial Intelligence and Statistics* vol 84, ed A Storkey and F Perez-Cruz (PMLR) pp 1979–87
[64] Stokes J and Terilla J 2019 Probabilistic modeling with matrix product states *Entropy* **21** 1236
[65] Stoudenmire E and Schwab D J 2016 Supervised learning with tensor networks *Advances in Neural Information Processing Systems* vol 29, ed D Lee, M Sugiyama, U Luxburg, I Guyon and R Garnett (Curran Associates, Inc)
[66] Sun Z Z, Peng C, Liu D, Ran S J and Su G 2020 Generative tensor network classification model for supervised machine learning *Phys. Rev.* B **101** 075135
[67] Tangpanitanon J, Mangkang C, Bhadola P, Minato Y, Angelakis D G and Chotibut T 2022 Explainable natural language processing with matrix product states *New J. Phys.* **24** 053032
[68] Vanderstraeten L, Haegeman J and Verstraete F 2019 Simulating excitation spectra with projected entangled-pair states *Phys. Rev.* B **99** 165121
[69] Vanderstraeten L, Haegeman J and Verstraete F 2019 Tangent-space methods for uniform matrix product states *SciPost Phys. Lect. Notes* 7
[70] Verstraete F and Cirac J I 2004 Renormalization algorithms for quantum-many body systems in two and higher dimensions (arXiv:cond-mat/0407066)
[71] Vidal G 2007 Entanglement renormalization *Phys. Rev. Lett.* **99** 220405
[72] Vieijra T, Vanderstraeten L and Verstraete F 2022 Generative modeling with projected entangled-pair states (arXiv:2202.08177)
[73] Wang M, Pan Y, Xu Z, Yang X, Li G and Cichocki A 2023 Tensor networks meet neural networks: a survey and future perspectives (arXiv:2302.09019)
[74] Wauthier S T, Vanhecke B, Verbelen T and Dhoedt B 2023 Learning generative models for active inference using tensor networks *Active Inference*, ed C L Buckley, D Cialfi, P Lanillos, M Ramstead, N Sajid, H Shimazaki and T Verbelen (Springer) pp 285–97
[75] White S R 1992 Density matrix formulation for quantum renormalization groups *Phys. Rev. Lett.* **69** 2863–6
[76] White S R 2005 Density matrix renormalization group algorithms with a single center site *Phys. Rev.* B **72** 180403
[77] White S R and Huse D A 1993 Numerical renormalization-group study of low-lying eigenstates of the antiferromagnetic s = 1 Heisenberg chain *Phys. Rev.* B **48** 3844–52
[78] Zauner-Stauber V, Vanderstraeten L, Haegeman J, McCulloch I P and Verstraete F 2018 Topological nature of spinons and holons: elementary excitations from matrix product states with conserved symmetries *Phys. Rev.* B **97** 235155
[79] Mel Tillery (available at: www.cyaneus.com/)