Information Fusion 100 (2023) 101908

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

Context-aware deep learning with dynamically assembled weight matrices

David Vander Mijnsbrugge *, Femke Ongenae, Sofie Van Hoecke

IDLab, Ghent University - imec, 126 Technologiepark, 9000 Ghent, Belgium

ARTICLE INFO

Keywords:
Context
Deep learning
Singular value decomposition
Dynamic neural networks

ABSTRACT

Deep neural networks are static by nature, meaning they use a single set of parameters to process each data sample. However, for more complex and larger systems, for which samples can be obtained under a large variety of circumstances, a need for more dynamic networks that adapt to these variations seems apparent. To this end, the concept of context information and its integration in deep learning is considered. Contrary to current practices, that treat all modalities as equally informative to the decision process, contextual and feature information is considered to be vastly different in nature. A set of definitions and subsequent arguments are given in order to provide the necessary clarification regarding the interpretation of context with the purpose of using all information in a maximally efficient way. From this interpretation, a problem statement of context aware deep learning is constructed and consequently linked to its multiple model and transfer learning solutions. These solutions, however, are in-efficient since samples are spread across models. Based on the existence of this multiple model solution, a new approach, which integrates contextual information directly into a single context-dependent model, is proposed. This single model uses weight matrices that are dynamically assembled based on the contextual information to process each data sample individually. This allows for the single model to consume and learn from all samples. The corresponding training routine is constructed and evaluated on multiple benchmark problems. We start with an artificially generated problem on which the methods' ability to model multiple linear classification problems concurrently is confirmed. Next, both a time series forecasting and image classification dataset are used for evaluation. Evaluations of our proposed method are done and compared to standard context aware implementations based on concatenation and gating. These standard methods implement context information by adding additional parameters in order to try modeling all interactions between the context information and the samples. However, our proposed approach integrates the contextual information directly into the network weights, allowing parameter efficient modeling of dynamic contextual behavior. In both cases the proposed solution outperforms its standard counterparts with significant margins in both evaluation metrics and parameter efficiency. Specifically, a mean absolute error improvement of eleven standard deviations and an eight percent increase in classification accuracy, for the forecasting and image classification problems respectively, is observed, showing the potential of our approach.

1. Introduction

Deep learning system are being implemented more and more throughout industry and for a wide variety of applications [1–3]. However, one of the main drawbacks of deep learning is its static nature, which leaves these systems mostly unable to adapt to new environments. This proves to be a problem, especially in these real-world use cases where the problems are more dynamic and complex. For example, in an industrial environment many factors such as temperature, humidity, etc., may influence operations of equipment and consequent measurement values. One way to deal with this problem is defining contexts in which different behaviors are observed and taking this information into account. To this end a general characterization of context is given in Definition 1.1.

Definition 1.1. **Context** is any information that can be used to characterize the situation of different entities, such as persons, places or objects. [4]

Context is also referred to as metadata or 'data about data'. In this sense, the actual data refers to the entities from Definition 1.1 and the metadata, the data describing the environments of these entities, to context. It is this distinction between what to consider as context and what as data that, when translating into a more precise machine learning problem, can often result in disarray. Traditionally, the contextual information is considered as a feature, omitting any semantic differences and assuming everything is a similar type of information. In order to frame the problem of context-awareness in the setting of

* Corresponding author.
E-mail address: david.vandermijnsbrugge@ugent.be (D.V. Mijnsbrugge).

https://doi.org/10.1016/j.inffus.2023.101908
Received 12 May 2022; Received in revised form 8 June 2023; Accepted 2 July 2023
Available online 7 July 2023

deep learning, different aspects from both literature and conceptual are considered to hone in on a more usable definition.

A first consideration is the presence of feature correlations in machine learning. More specifically, correlations between features or in the input data are considered to be detrimental. In many cases, correlated features do not improve model performance. Moreover correlations are often used to do feature selection [5,6]. More generally, this can be considered to be a special case of Occam's razor [7], which states that the simplest model that explains all observations made, is preferable. So a minimal set of features that lead to sufficient performance is preferred. As such, endlessly adding context as features is destined to decrease performance and thus an undesirable way of dealing with the additional context information.

From a more practical perspective, one could discriminate input and context via data type or dimension. Data, especially in deep learning, has a certain dimensionality and before conversion to numerical vectors, this dimensionality can be used to distinguish between data sources. For example, geolocated images contain two different dimensional and structural types of data. The image has dimensions of pixels, while the geolocation is expressed in latitude and longitude angles [8]. In this case, we can distinguish data, i.e. the image, and context, i.e. the geolocation, based on that difference in dimensionality and structure.

Lastly, a concept borrowed from database theory is considered. In database theory information completeness corresponds to the fact that the database contains all the information to return a given query [9,10]. When considering a machine learning problem we can think of a query as a task and the database as the data set. Based on the previous example this translates into the following. Given the task of finding out what is described in a provided image, all the information necessary to answer this question is captured in the image itself. Any additional external information seemingly, as previously mentioned, will only make more confused, complex models. This notion leads to the description of a somewhat informal concept, "information (over-)completeness" for learning, given by Definition 1.2, which represents whether or not the input information is complete for a given task.

Definition 1.2. Data is said to be **information (over-)complete** with respect to a specific task when the information required for the solution to this task is a subset of the information contained in the data.

The mentioned degradation of model performance seems to be in disagreement with how we perceive completing such tasks. Since knowing the location of an image gives a large amount of information about what to expect in the image and consequently allows more efficient processing of the image. For example, one could expect to only see polar bears and no penguins when analyzing pictures taken the North Pole. Coupling back to the original definition of context, the notion that descriptive information gives bias to the task related decision making process about the entities has been added. Generalizing this, leads to a different formulation for context which is more restrictive and more importantly relates to the to be performed task, given in Definition 1.3.

Definition 1.3. Context is any information that describes a priori knowledge about the **decision making process** regarding the data.

It is with this definition and with the notion of information completeness in mind, that this work will continue in order to design a novel approach of introducing external contextual information into deep learning architectures.

1.1. Problem statement

In this section, the problem statement for context-aware deep learning is formulated more in-depth. More specifically a set of definitions and arguments is presented to obtain a mathematical problem description corresponding with the formulation of context from Definition 1.3. In order to start from a solid basis, the idea of a learning setting is adopted from statistical learning [11], see Definition 1.4.

Definition 1.4. A learning setting $\mathcal{T} = (\mathcal{H}, \mathcal{Z}, \mathcal{L})$ is defined by the following:

- A hypothesis class or function class $\mathcal{H}$;
- Samples $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ consisting of input data; $\mathcal{X}$ and targets $\mathcal{Y}$.
- A loss function $\mathcal{L} : \mathcal{H} \times \mathcal{Z} \to \mathcal{R}$.

In the particular case of deep learning, neural network architectures as defined by Definition 1.5, are used for the hypothesis class $\mathcal{H}$.

Definition 1.5. A neural network architecture $\mathcal{F}_{\mathcal{V},\mathcal{E},\mathcal{A}} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ consists of a set of nodes $\mathcal{V}$, edges $\mathcal{E}$ and activation functions $\mathcal{A}$.

Based on Definitions 1.4 and 1.5, a description for a supervised deep learning task is constructed, as given in Definition 1.6.

Definition 1.6. A **supervised deep learning problem**, given a learning setting $\mathcal{T}$, consists of finding the element(s) in the hypothesis class $\mathcal{F}_{\mathcal{V},\mathcal{E},\mathcal{A}}$ that minimize the loss function $\mathcal{L}$ over $\mathcal{Z}$. This corresponds to assigning a set of weights $\mathcal{W}$ that maps to each element of $\mathcal{E}$ and for which the following objective is met:

$$\mathcal{W} = \min_{\mathcal{E}} \mathcal{L}(\mathcal{F}_{\mathcal{V},\mathcal{E},\mathcal{A}}, \mathcal{Z}) \tag{1}$$

The resulting architecture with a specific set of weights is denoted as $\mathcal{F}_{\mathcal{V},\mathcal{W},\mathcal{A}}$.

Considering the above introduction of a learning setting, the decision making process for this task is represented by the network architecture and the optimal decision making strategy by the corresponding set of learned weights. Looking back, Definition 1.3 relates the context to the task to be performed, corresponding to the minimization of the loss function. Definition 1.7 reflects this context dependency by integrating Definition 1.3 into the formulation of deep learning setting.

Definition 1.7. Given a set of data samples $\mathcal{Z}$ for which a task $\mathcal{T}$ is defined, the **context** $\mathcal{C}$ is any amount of information that describes any a priori knowledge about a sample that relates to the decision making process about this sample. The corresponding dataset can now be described as follows.

$$\{(z_i, c_i) \in (\mathcal{Z}, \mathcal{C})\} \tag{2}$$

Finally, the learning setting definition is extended to include this context formulation, which is represented in Definition 1.8.

Definition 1.8. A context-aware learning setting $\mathcal{T}_{\mathcal{C}} = (\mathcal{H}, \mathcal{Z}, \mathcal{L}, \mathcal{C})$ is defined by the following:

- A hypothesis class or function class $\mathcal{H}$;
- A set of samples $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ consisting of input data $\mathcal{X}$ and targets $\mathcal{Y}$;
- A set of contexts $\mathcal{C}$;
- A loss function $\mathcal{L}_{\mathcal{C}} : \mathcal{H} \times \mathcal{Z} \times \mathcal{C} \to \mathcal{R}$.

As before, this learning setting can be used to summarize the context dependent problem statement, described by Definition 1.9, reflecting the properties of context from Definition 1.3.

Definition 1.9. A **context dependent supervised deep learn- ing problem**, given a context-aware learning setting $\mathcal{T}_{\mathcal{C}}$, consists of finding the element(s) in the hypothesis class $\mathcal{F}_{\mathcal{V},\mathcal{E},\mathcal{A}}$ that minimize the loss function $\mathcal{L}_{\mathcal{C}}$ over the set of samples $\mathcal{Z}$ given their contexts in $\mathcal{C}$. This corresponds to assigning a set of weights $\mathcal{W}^{\mathcal{C}}$ to each element of $\mathcal{E}$ for which this objective is met:

$$\mathcal{W}^{\mathcal{C}} = \min_{\mathbb{R}^{\#\mathcal{E}}} \mathcal{L}_{\mathcal{C}}(\mathcal{F}_{\mathcal{V},\mathcal{E},\mathcal{A}}, \mathcal{Z}, \mathcal{C}) \tag{3}$$

An architecture with a specific sets of weight is denoted as $\mathcal{F}_{\mathcal{V},\mathcal{W}^{\mathcal{C}},\mathcal{A}}$

Consider all the samples for each distinct context in C separately with a joint loss function $\mathcal{L}$. When the architecture information is omitted, the solution, being the set of optimal weights per context, can be written as in Eq. (4). This simple solution is called the multiple model solution since for each context, there is a specific set of weights.

$$\mathcal{W}^C = \{\mathbf{W}_{c_i} | c_i \in C\} \tag{4}$$

A multiple model solution is highly inefficient due to the lack of information sharing between contexts. All separate sets of weights have been acquired by only training for the data given for a certain context. Many methods exist to deal with this shortcoming, of which the most well-known and current state-of-the-art, is transfer learning [12,13]. The classic approach to transfer learning in deep learning is done by shared weights. Concretely, some layers are shared between neural networks with similar architectures for each context. We adopt the following notation.

$$\mathcal{F}_{\mathcal{V},\mathcal{W},\mathcal{A}} = \mathcal{F}^{\mathcal{I}}_{\mathcal{V},\mathcal{W},\mathcal{A}} \cup \mathcal{F}^{\mathcal{E}\setminus\mathcal{I}}_{\mathcal{V},\mathcal{W}^C,\mathcal{A}} \tag{5}$$

$\mathcal{F}_{\mathcal{V},\mathcal{W},\mathcal{A}}$ is an element of the hypothesis class $\mathcal{H}$ consisting of the networks with identical weights on the subset of edges $\mathcal{I} \subset \mathcal{E}$ and context specific weights on its complement. Concretely, this implementation only calculates a multiple model solution for a smaller subset of edges that contains specific decision making capacity, while the shared edges learn context independent behavior. The resulting weight matrix solution can be written as follows.

$$\mathcal{W}^C = \mathcal{W}^0 \cup \{\mathcal{W}^{c_i} | c_i \in C\} \tag{6}$$

Another implementation of transfer learning results not from sharing a subset of edges, but from a single architecture where all weights are shared. The single architecture is trained on a large dataset leading to a pre-trained network. Following pre-training, a fine-tuning step can be performed to adapt the network to have the desired properties in a new context. For each new context a fine-tuning step is now done where the pre-trained network is tuned to the specific context often with a smaller learning rate, leading to the following notation for this solution.

$$\mathcal{F}_{\mathcal{V},\mathcal{W}^C,\mathcal{A}} = \mathcal{F}_{\mathcal{V},\mathcal{W}^0,\mathcal{A}} + \{\mathcal{F}_{\mathcal{V},\mathcal{W}^{c_i},\mathcal{A}} | c_i \in C\} \tag{7}$$

The solution takes the following form in just weight matrix notation. Since all weights are shared across all architectures for each context, a summation is used to connect all the weights.

$$\mathcal{W}^C = \mathcal{W}^0 + \{\mathcal{W}^{c_i} | c_i \in C\} \tag{8}$$

2. Related work

As stated in the introduction, the presented problem statement has overlap with a couple domains depending on the interpretation given to the context and availability of representations. For example, for multi-task learning each specific task can be considered to be a new context and would require explicit task representations [14]. In a domain adaptation setting, context corresponds to the description of the domain. The common denominator is the adaptation of model performance to new circumstances or context, which is currently most readily associated with transfer learning [11–13]. Furthermore, mixture of experts models, which can also alleviate some of these adaptations issues, have recently been implemented in deep learning as switch transformers [15]. These transformers use sparse routing to connect multiple separate models to do sample dependent predictions, which in the light of context aware prediction, corresponds to routing samples between each multiple model solution and are part of the recently developed research domain of dynamic neural networks [16]. As part of this domain and in the context of convolutional networks, a version of a dynamically composed kernels has been presented [17]. However, as with natural language processing and image recognition, context is mostly inferred or explicitly determined from the data. E.g. The context for a certain word or pixel is its neighborhood or known annotations within the sentence or image are used as context. This is what in this work is referred to as internal context and does not correspond to the definition for external context information from Section 1. To the best of our knowledge no method such as presented in this work, being the dynamic assembly of general weight matrices based on this external information, has been proposed. This integration of external information is commonly done using the concatenation of context features and data features [17–19]. The link with these works will be more thoroughly discussed in the next section.

3. Solution formulation

In this section, our novel solution formulation for the context dependent deep learning problem from Definition 1.9 is presented. This solution conceptually assumes the existence of a multiple model solution as given in Eq. (4). The meaning of existence in this setting is as follows. There is a set of weight matrices $\mathbf{W}_{\mathbf{c_i}}$ that are solutions of the context dependent problem. More concretely these weights are considered to be unknown and the main quantity of interest. In other words, the existence of these weight matrices represents an idealized situation where a representative model can be learned per context. These discrete sets of contexts can always be constructed, where in the worst case scenario a separate context per sample is needed. Quite straightforwardly, it can be seen that in this case, the assumption that a model can be learned from this single sample does not hold. The following method will address this shortcoming by sharing information across a set of basis matrices and learning coefficients from explicit context representation for these basis components. Without loss of generality, the formulation is done for a single weight matrix, since the procedure applies to each weight matrix in any architecture separately. Furthermore, a discrete set of contexts C of size K is used which generalizes to any situation with the worst case being the one where K is equal to the number of samples.

$$C = \{c_i | i = 1 \dots K\}$$

The assumed multiple model solution implies a distinct set of weights $\mathbf{W}_{c_i}$ for each context c_i. Denoting a single weight matrix for a given context, can be done by assembling this set of weights in a vector and using the dot-product for vectors with a one-hot encoded vector of the context indicator $\vec{c}_i$, as given by Eq. (9). This is visually represented in Fig. 1(a).

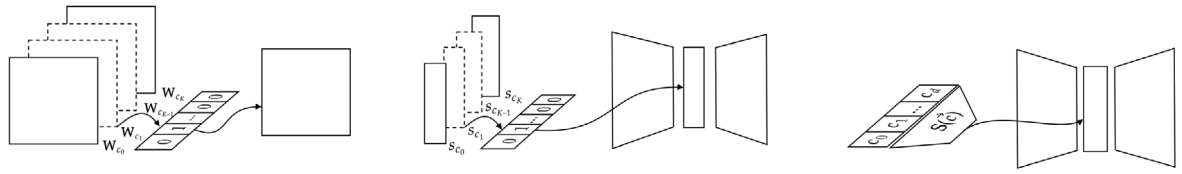
$$\begin{aligned} &\mathbf{W}_{c_i} = [\mathbf{W}_{c_0}, \dots, \mathbf{W}_{c_K}] \cdot \vec{c}_i = \bar{\mathbf{W}} \cdot \vec{c}_i \\ &\vec{c}_i = [0, \dots, \underbrace{1}_{i\text{th}}, \dots, 0] \end{aligned} \tag{9}$$

The use of this context indicator hints to the lack of information sharing for this multiple model solution since the gradients for each sample will only be propagated towards the selected weights. This can be seen from the chain rule during backpropagation. The gradients with respect to the stacked tensor $\bar{\mathbf{W}}$ are given by Eq. (10). The resulting gradient only contributes to the weights of context c_i. Furthermore, the tensor $\bar{\mathbf{W}}$ which stores all the weights is of order $\mathcal{O}(M \times N \times K)$.

$$\begin{aligned} \frac{\partial L}{\partial \bar{\mathbf{W}}} &= \frac{\partial L}{\partial \mathbf{W}_{\vec{c}_i}} \cdot \frac{\partial \mathbf{W}_{\vec{c}_i}}{\partial \bar{\mathbf{W}}} \\ &= [0, \dots, \underbrace{\frac{\partial L}{\partial \mathbf{W}_{\vec{c}_i}}}_{i\text{th}}, \dots, 0] \end{aligned} \tag{10}$$

The problem statement in Definition 1.9 makes use of a dataset described by Definition 1.7, each context in C has a corresponding description represented by an embedding $\vec{c}$. This work proposes making the network weights a general function of this embedding instead of doing weight selection with an indicator variable. Concretely, the following transformation is proposed which drops the context index i and uses the embedding $\vec{c}$ to generate the network weights.

$$\mathbf{W}(\vec{c}_i) \rightarrow \mathbf{W}(\vec{c})$$



(a) Stacked weight selection using context indicator.

(b) Stacked singular value selection using context indicator.

(c) Singular value generation from context input.

Fig. 1. Representations of weight selection using context indicators (a, b) and dynamic generation of weights (c). The trapezoids in both b and c represent the **U** and **V** matrices from the singular value decomposition.

In order to generate these weights from the explicit context representation, which is assumed to be of dimension D, at minimum a linear relationship for each weight as in Eq. (11) is needed. The resulting representation for the dynamic weights is of order $\mathcal{O}(M \times N \times D)$. This cubic scaling is similar to the stacked context representation where instead of K discrete contexts the dimensions of the context dependence are now contained in the $D \times 1$ learned weights per weight in $\mathbf{W}(\vec{c})$.

$$\mathbf{W}_{ij}(\vec{c}) \sim \vec{a}_{ij} \cdot \vec{c} \sim \mathcal{O}(D) \quad \forall ij \in [0, \dots, N \times M] \tag{11}$$

This cubic scaling is highly unwanted and as such, before applying the proposed transformation to context embeddings, each weight matrix is expressed as a linear combination of basis matrices as per Eq. (12). The summation index j indicates the rank R of the basis decomposition.

$$\mathbf{W} = \sum_{j=1}^{R} s_j \cdot \mathbf{E}_j \tag{12}$$

This decomposed representation allows us to only express the basis coefficients as functions of the context indicator $\vec{c}_i$ given by Eq. (13). This means that now our original assumption that each weight matrix per context is reduced to each set of optimal coefficients per context.

$$\mathbf{W}(\vec{c}_i) = \sum_j s_j(\vec{c}_i) \cdot \mathbf{E}_j \quad s_j(\vec{c}_i) = [s_{j,c_0}, \dots, s_{j,c_K}] \cdot \vec{c}_i \tag{13}$$

This new representation expresses the context dependent weights using $\mathcal{O}(R \times [N \times M + K])$ parameters. By using the Kronecker product for the basis matrices, given in Eq. (14), the amount of parameters is reduced back to quadratic, $\mathcal{O}(R \times [N + M + K])$, albeit with different dimensions.

$$\mathbf{E}_j = \vec{u}_j \otimes \vec{v}_j \tag{14}$$

The result is a singular value decomposed representation [20] of the weight matrix where the singular values are selected using a one-hot encoded context indicator $\vec{c}_i$, as given by Eq. (15) and visually represented in Fig. 1(b).

$$\mathbf{W}_{c_i} = \sum_j s_j(\vec{c}_i) \cdot [\vec{u}_j \otimes \vec{v}_j] = \mathbf{U} \cdot \mathbf{S}(\vec{c}_i) \cdot \mathbf{V}^T \tag{15}$$

However, the same restriction on the information sharing across contexts is still present since, by making use of the indicator vector $\vec{c}_i$, the gradients are propagated to the singular values in a per-context fashion. In order to illustrate this, the conceptual update equations for the singular value decomposition are given by the following set of equations where k is the training iteration. The dependencies of each update increment dX^k are given, however, calculation of these quantities are left out and can be found in [20].

$$\mathbf{U}^{k+1} = \mathbf{U}^k + d\mathbf{U}^k(\mathbf{U}^k, \partial\mathbf{U}^k) \tag{16}$$

$$\mathbf{S}^{k+1} = \mathbf{S}^k + d\mathbf{S}^k(\{\mathbf{X}^k\}, \{\partial\mathbf{X}^k\}) \tag{17}$$

$$\mathbf{V}^{k+1} = V^k + d\mathbf{V}^k(\mathbf{V}^k, \partial\mathbf{V}^k) \quad \{\mathbf{X}^k\} = \{\mathbf{U}^k, \mathbf{S}^k, \mathbf{V}^k\} \tag{18}$$

$$\partial\mathbf{X}^k = \frac{\partial\mathcal{L}}{\partial\mathbf{X}^k}$$

These equations show that the training of the basis matrices represented by **U** and **V** does not depend on the context since only the partial gradients $\partial\mathbf{U}$ and $\partial\mathbf{V}$ are needed arguments for the calculation of the increments. However, the singular values have been made function of $\vec{c}_i$ and as such $d\mathbf{S}^k$ does contain the context indicator via the partial gradient for the singular values $\partial\mathbf{S}$. So by the same reasoning as with the full matrices the gradient for the singular values only applies to the c_i^{th} context.

$$d\mathbf{S}^k = [0, \dots, \underbrace{d\mathbf{S}^k_{c_i}}_{i\text{th}}, \dots, 0]$$

However, since the dynamic behavior now is only present in the singular values, contrary to the full matrix, when applying the same linear relationship from the context embedding to each of the basis coefficients, the resulting representation is of order $\mathcal{O}(D \times R)$.

$$\mathbf{S}(\vec{c}_i) \rightarrow \mathbf{S}(\vec{c}) \quad s_i(\vec{c}) \sim \vec{a}_i \cdot \vec{c} \sim \mathcal{O}(D) \quad \forall i \in [0, \dots, R] \tag{19}$$

Consequently the full dynamic network can be composed using the basis coefficients represented by Eq. (20) using $\mathcal{O}(R \times [N + M + D])$ parameters.

$$\mathbf{W}(\vec{c}) = \sum_j s_j(\vec{c}) \cdot \mathbf{E}_j = \mathbf{U} \cdot \mathbf{S}(\vec{c}) \cdot \mathbf{V}^T \tag{20}$$

Each input sample now generates coefficients $\mathbf{S}(\vec{c})$ based on its corresponding context $\vec{c}$. These coefficients are used to assemble a linear combination of shared basis matrices, rather than selection from a disjoint set of coefficients by the indicator $\vec{c}_i$. This formulation improves sample efficiency since each sample now contributes to learning $\mathbf{S}(\vec{c})$ due to the shared dimensions of the context embeddings $\vec{c}$. This is visually represented in Fig. 1(c) and can also be seen from the chain rule.

$$\frac{\partial\mathcal{L}}{\partial a_i} = \frac{\partial\mathcal{L}}{\partial s_i}\frac{\partial s_i}{\partial a_i} \sim \vec{c} \tag{21}$$

The gradient for the parameters of the linear relationship does not contain a selection using the one-hot encoded indicator variable, but rather the context vector with shared embedding dimensions. This implies that each sample which generates a gradient for the coefficients $d\mathbf{S}(\vec{c})$ contributes towards the learning of the context dependent representation $\mathbf{S}(\vec{c})$. The question remains how to construct $\mathbf{S}(\vec{c})$ and the corresponding gradients. In principle one could use a different neural network $\mathcal{F}^C$ to learn the coefficients from the context data. In the following subsections, two different formulations for $\mathbf{S}(\vec{c})$ and their gradients are derived and discussed.

3.1. Embedding decomposition

The coefficients in $\mathbf{S}(\vec{c})$ represent the contribution of a basis kernel $\mathbf{E}_i$ towards the weight matrix. This corresponds to how much the resulting basis vector $\mathbf{E}_i \cdot \vec{x}$ will contribute towards the linear activation

$\vec{h}$. For an arbitrary layer in the architecture and its linear activation this can be expressed as follows:

$$\vec{h} = \mathbf{W} \cdot \vec{x} = \sum_j s_j \mathbf{E}_j \cdot \vec{x} = \sum_j s_j \vec{h}_j \tag{22}$$

Definition 3.1 describes a theoretical result which allows an approximate decomposition of an embedding into a context-free and dependent part [21].

Definition 3.1. The log-linear model of a conditional probability $p(x|c)$ can be approximately decomposed into two parts, a context sensitive and context free part. Both are connected by the $\psi(\cdot,\cdot)$ function that indicates the amount of context-dependence.

$$\vec{h} = \psi(\vec{c},\vec{x})\vec{h}' + [1 - \psi(\vec{c},\vec{x})]\vec{h}_0 \tag{23}$$

$$\psi(\vec{x},\vec{c}) \in [0,1]$$

Using Eq. (23), these terms can be rearranged to an expression that is more logical for our purposes, given in Eq. (25).

$$\vec{h} = \vec{h}_c + \vec{h}_0 \tag{24}$$

$$\vec{h}_c = \psi(\vec{c},\vec{x})(\vec{h}' - \vec{h}_0) \tag{25}$$

Substituting Eq. (25) for each basis component of the latent vector $\vec{h}$ (Eq. (26)) and grouping the terms (Eq. (27)) gives an expression for the context-dependent coefficients, given in Eq. (28).

$$\vec{h} = \sum_j s_j \vec{h}_{c,j} \tag{26}$$

$$\vec{h} = \sum_j s_j \psi_j(\vec{c},\vec{x})\vec{h}_j \tag{27}$$

$$s_j(\vec{c}) = \psi_j(\vec{c},\vec{x}) s_j \tag{28}$$

Finally, the $\vec{x}$ dependence is dropped, leading to the expression for the context-dependent kernel, given in Eq. (29).

$$\mathbf{W}(\vec{c}) = \sum_j \psi_j(\vec{c}) \cdot s_j \mathbf{E}_j \tag{29}$$

The additional index for each basis kernel corresponds to a unique set of functions.

$$\{\psi_i | i \in 1 \dots R\}$$

In order to reduce this complexity, a single ψ function with the index being moved to the argument, is considered. The scalar argument is constructed using the context vector $\vec{c}$ and a trainable vector $\vec{\lambda}_i$, resulting in the following expression:

$$\mathbf{W}(\vec{c}) = \sum_j \psi(\vec{\lambda}_i \cdot \vec{c}) \cdot s_j \mathbf{E}_j \tag{30}$$

In matrix form, using the Kronecker basis kernels, this leads to the following expression for $\mathbf{S}(\vec{c})$, with use of short-hand notation where the ψ function is computed element-wise and the dot product is passed outside the diagonal matrix:

$$\mathbf{S}(\vec{c}) = \mathbf{PS} \tag{31}$$

$$\mathbf{P} = \psi\left(\begin{bmatrix} \vec{\lambda}_0 & & & \\ & \vec{\lambda}_1 & & \\ & & \ddots & \\ & & & \vec{\lambda}_R \end{bmatrix} \cdot \vec{c}\right)$$

Given the interpretation of the singular values as the contributions of their corresponding basis kernels $\mathbf{E}_j$, as seen in Eq. (29), each $\psi_j(\vec{c})$ now represents an activation of the scaled kernel $s_j\mathbf{E}_j$ given the context. Since $\psi_j(\vec{c})$ is a value between zero and one, this represent maximally irrelevant and important scaled kernels respectively.

3.2. Linear approximation

Another formulation, besides the embedding decomposition, is obtained by linear approximation of the general context dependent weight functions. By using a Taylor expansion to first order in the context variable $\vec{c}$, the following formulation is obtained:

$$\mathbf{W}_{nm}(\vec{c}) = \mathbf{W}_{nm}(\vec{c}_0) + \frac{\partial \mathbf{W}_{nm}}{\partial \vec{c}}|_{\vec{c}_0} \cdot (\vec{c} - \vec{c}_0) \tag{32}$$

Substituting Eq. (12) for each of the matrix elements yields the following representations, given in Eqs. (33) and (34).

$$\mathbf{W}(\vec{c}) = \sum_i [s_i(\vec{c}_0) + \frac{\partial s_i(\vec{c})}{\partial \vec{c}}|_{\vec{c}_0} \cdot (\vec{c} - \vec{c}_0)]\, \mathbf{E}_i \tag{33}$$

$$= \sum_i [s_i(\vec{c}_0) + \vec{\lambda}_i \cdot (\vec{c} - \vec{c}_0)]\, \mathbf{E}_i \tag{34}$$

$$\vec{\lambda}_i = \frac{\partial s_i}{\partial \vec{c}}$$

The vector $\vec{\lambda}_i$ represent the first order change of the singular value with respect to the context $\vec{c}$ and is made a trainable parameter. The dependence on the static part of the context $\vec{c}_0$ can be omitted by reshaping the equations as follows and defining the context independent coefficients s_i:

$$s_i(\vec{c}_0) + \vec{\lambda}_i \cdot (\vec{c} - \vec{c}_0) \tag{35}$$

$$= s_i(\vec{c}_0) - \vec{\lambda}_i \cdot \vec{c}_0 + \vec{\lambda}_i \cdot \vec{c} \tag{36}$$

$$= s_i + \vec{\lambda}_i \cdot \vec{c} \tag{37}$$

These can be put in matrix form in similar fashion as the multiplicative singular value representation. The same short-hand notation as in Eq. (31) is used where the dot product for the $\vec{\lambda}_i$ vectors is taken outside the diagonal matrix for brevity.

$$\mathbf{W}(\vec{c}) = \mathbf{U}[\mathbf{S} + \mathbf{\Lambda}]\mathbf{V}^T \tag{38}$$

$$\mathbf{\Lambda} = \begin{bmatrix} \vec{\lambda}_0 & & & \\ & \vec{\lambda}_1 & & \\ & & \ddots & \\ & & & \vec{\lambda}_R \end{bmatrix} \cdot \vec{c}$$

The corresponding interpretation of these equations is that each weight matrix is composed of a context sensitive part and a context-free part, where both parts are expressed as a singular value decomposition. For the context sensitive part the singular values are inner products between the context embedding and the $\vec{\lambda}_i$ vectors. Considering the correspondence of these vectors to the gradient of the singular value with respect to the context representation, a single $\vec{\lambda}_i$ vector can be associated with direction in context space for which the corresponding $\mathbf{E}_i$ will change in importance.

3.3. Discussion

In this subsection, a comparison between both the formulations of the presented method and other context-aware methods is made. Both formulations are referred to as the additive and multiplicative implementation and are represented by Eqs. (39) and (40) respectively.

$$\mathbf{W}(\vec{c}) = \mathbf{U}[\mathbf{S} + \mathbf{\Lambda}]\mathbf{V}^T \tag{39}$$

$$\mathbf{W}(\vec{c}) = \mathbf{U}[\mathbf{PS}]\mathbf{V}^T \tag{40}$$

Since both formulations contain diagonal matrices, the formulations can be equated for an arbitrary diagonal element as follows.

$$\psi(\vec{\lambda}_i \cdot \vec{c}) s_i = s_i + \vec{\lambda}_i \cdot \vec{c} \tag{41}$$

$$\Rightarrow \psi(\vec{\lambda}_i \cdot \vec{c}) = I + \frac{\vec{\lambda}_i}{s_i} \cdot \vec{c} \tag{42}$$

The additive formulation is a linearized version of the multiplicative approach, as expected, when the context functions ψ is equal to the

linear approximation as in Eq. (42). Furthermore, when no context is present, both models reduce to a static singular value representation neural network. In the additive case this corresponds to only using the general model part of the singular values and in the multiplicative case the average value determined by the zero value of the ψ function. In the following subsections, a series of comparisons to current state-of-the-art methods for implementing external context information is made.

3.3.1. Transfer learning

Consider the transfer learning solution mentioned in Eq. (8). Here all the samples not in the chosen context constitute the general model and the samples in context are used for fine tuning. At the end of training with learning rates α_i for the general model and β_i for fine tuning, the weight matrix $\mathbf{W}_{c_i}$ in context c_i can be written as follows:

$$\mathcal{W}^C = \sum_{i}^{c_i \notin C} \alpha_i \nabla \mathbf{W}_i + \sum_{j}^{c_j \in C} \beta_j \nabla \mathbf{W}_j \tag{43}$$

This representation separates information between samples during training of the general model and fine-tuning. An equivalent formulation for the additive decomposition can be constructed. The additive solution, using a generalized notation where $\Delta\mathbf{S}_i$ and $\Delta\mathbf{\Lambda}_i$ represent the changes in $\mathbf{S}$ and $\mathbf{\Lambda}$ respectively, can be written as follows.

$$\mathcal{W}^C = \mathbf{U}\mathbf{S}\mathbf{V}^T + \mathbf{U}[\mathbf{\Lambda} \cdot C]\mathbf{V}^T \tag{44}$$

$$= \mathbf{U}[\sum_{i}^{c_i \in C} \Delta\mathbf{S}_i + \Delta\mathbf{\Lambda}_i \cdot \vec{c}_i]\mathbf{V}^T \tag{45}$$

When comparing Eqs. (43) and (45), in which the training of the basis matrix components has been omitted for brevity, one can see both contain a static and context dependent part. However, the additive formulation allows both parts to learn from all available samples, while transfer learning separates samples per context. Furthermore, the transfer learning solution does not scale well with larger amounts of contexts and cannot be applied when no discrete contexts are defined. Our proposed approach alleviates this problem by using the explicit context representation, which can be any representation from a continuous spectrum of contexts.

3.3.2. Concatenation

A regular concatenation implementation, which puts the contextual and input features in one vector, as in Eq. (46), is often used in cases where context representations are available. In this case, the standard weight matrix, denoted as $\mathbf{W}^*$, can be constructed from four block matrices representing the interaction between $\vec{x}, \vec{c}$ and their respective linear activations, given in Eq. (47).

$$\vec{x}^* = [\vec{x}, \vec{c}] \tag{46}$$

$$\mathbf{W}^* = \left(\begin{array}{c|c} \mathbf{W}_{xx} & \mathbf{W}_{cx} \\ \hline \mathbf{W}_{xc} & \mathbf{W}_{cc} \end{array}\right) \tag{47}$$

The following representation for the linear activation, which consists of the data and context linear activation, $\vec{h}_x$ and $\vec{h}_c$ respectively, is obtained. This representation clearly reflects the purpose of each block component of $\mathbf{W}^*$.

$$\vec{h}^* = \mathbf{W}^* \cdot \vec{x}^* = [\vec{h}_x, \vec{h}_c] \tag{48}$$

$$= [\mathbf{W}_{xx} \cdot \vec{x} + \mathbf{W}_{xc} \cdot \vec{c}, \mathbf{W}_{cc} \cdot \vec{c} + \mathbf{W}_{cx} \cdot \vec{x}] \tag{49}$$

Similarly, for the additive formulation, the data linear activation, $\vec{h}_x$ can be constructed, leading to the following representations. The derivation of Eq. (52) from Eq. (51) is given in Appendix.

$$\vec{h}_x = \mathbf{U}[\mathbf{S} + \mathbf{\Lambda} \cdot \vec{c}]\mathbf{V}^T \cdot \vec{x} \tag{50}$$

$$= \mathbf{U}\mathbf{S}\mathbf{V}^T \cdot \vec{x} + \mathbf{U}[\mathbf{\Lambda} \cdot \vec{c}]\mathbf{V}^T \cdot \vec{x} \tag{51}$$

$$= \mathbf{U}\mathbf{S}\mathbf{V}^T \cdot \vec{x} + \mathbf{U}[\mathbf{V}^T \cdot \vec{x}]\mathbf{\Lambda} \cdot \vec{c} \tag{52}$$

When comparing Eq. (49) with Eqs. (52) and (51) the following correspondence is obtained.

$$\mathbf{W}_{xx} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{53}$$

$$\mathbf{W}_{xc} = \mathbf{U}[\mathbf{V}^T \cdot \vec{x}]\mathbf{\Lambda} \tag{54}$$

The same transformation of the linear activation, h_x, using singular value decomposed weight matrix representations, is present. The learning of $\vec{h}_c$, can now be decoupled in a different neural network. This way of incorporating context reduces the dimensionality of the process since the additive representation is considerably smaller. The dimensions of the latent representations are taken to be the same as the respective input dimensions, written as n_x and n_c, for simplicity of the argument. These bounds can be directly extended to non-square matrices. The resulting concatenated representation has dimension $(n_x + n_c)^2$, while the additive representation only requires $(n_c + 2n_x + 1) \cdot R$ parameters. From this, a parameter efficiency upper bound for the decomposition rank can be found:

$$R < n_c + \frac{n_x^2 - 1}{2n_x + n_c + 1}$$

3.3.3. Gating

Whereas the additive implementation resembles concatenation, the multiplicative formulation resembles a context gating mechanism [22–24]. This mechanism weighs a latent vector element-wise based on a sigmoidal activation function, which can be written as follows:

$$\vec{h} = \vec{\sigma}(\vec{c}) \odot \vec{z} \tag{55}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Substituting Eq. (55) into Eq. (22) for the decomposition of the linear activation due to the kernel decomposition gives the following:

$$\vec{h}^* = \vec{\sigma}(\vec{c}) \odot \sum_i s_i \vec{h}_i \tag{56}$$

$$= \vec{\sigma}(\vec{c}) \odot \sum_i s_i [\mathbf{E}_i \cdot \vec{x}] \tag{57}$$

$$= \sum_i s_i \beta_i \vec{\sigma}_i(\vec{c}) \odot \vec{u}_i \tag{58}$$

$$\beta_i = \vec{v}_i \cdot \vec{x}$$

$$\vec{\sigma}_i(\vec{c}) = \frac{\vec{\sigma}(\vec{c})}{R}$$

Considering the multiplicative formulation from Eq. (29), a similar representation is obtained.

$$\vec{h}^* = \sum_i \psi_i(\vec{c}) s_i \beta_i \vec{u}_i \tag{59}$$

The context gating is shifted from an equally divided, element-wise application of the gated context on the basis kernels to a scalar application on the singular values. In other words, the gating is moved from the individual latent features to groups of features represented by the $\mathbf{U}$ components of the basis kernels $\mathbf{E}_i$. As such the expressivity of the multiplicative approach is moved to the training of the $\vec{\lambda}_i$ vectors. Furthermore, this comparison suggests setting the ψ function to be the sigmoid function.

$$\psi(\vec{\lambda}_i \cdot \vec{c}) = \frac{1}{1 + e^{-\vec{\lambda}_i \cdot \vec{c}}}$$

Similarly to the comparison of the additive decomposed implementation and context concatenation, a parameter efficiency evaluation can be done. Regular gating requires $n_c \cdot n_x$ parameters, corresponding to $\mathbf{W}_{cx}$, to create the latent context embedding which is of the same size as the input data embedding. A single feed-forward layer for the input data requires n_x^2 parameters, corresponding to $\mathbf{W}_{xx}$. This gives a total of $(n_c + n_x) \cdot n_x$ parameters, compared to the multiplicative decomposed version which requires $(n_c + 2n_x + 1) \cdot R$ parameters. The following upper bound is obtained.

$$R < \frac{n_c + n_x}{n_c + 2n_x + 1} n_x \tag{60}$$

This upper bound is smaller than the additive one thanks to lack of the $\mathbf{W}_{xc}$ and $\mathbf{W}_{cc}$ components.

3.4. Gradients

Now that the construction of $\mathbf{S}(\vec{c})$ is done, we present the gradient calculation and corresponding update procedure for the context dependent weights. The shared basis matrices are context independent and are trained according to the procedure for static singular value decomposed weight matrices [20]. On the other hand, the update equation for the singular values presented in Section 3 is now fully written out. The general context-dependent singular values are denoted as $\mathbf{S}_*$ containing singular values $\{s_i^*|_{i=1\dots R}\}$ and can be evaluated using the chain rule.

$$\frac{\partial \mathcal{L}}{\partial s_i^*} = \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i}{\partial s_i^*} \tag{61}$$

This results in the following expressions for the additive and multiplicative representations, given by Eqs. (62) and (63), respectively.

$$\frac{\partial \mathcal{L}}{\partial s_i^*} = \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i^* - \vec{\lambda}_i \cdot \vec{c}}{\partial s_i^*} = \frac{\partial \mathcal{L}}{\partial s_i} \tag{62}$$

$$\frac{\partial \mathcal{L}}{\partial s_i^*} = \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i \psi_i(\vec{c})^{-1}}{\partial s_i} = \frac{\partial \mathcal{L}}{\partial s_i} \psi_i(\vec{c})^{-1} \tag{63}$$

Since this is the only change necessary these expression can be readily inserted in the calculation of the gradient step for the singular values using Eq. (64) [20]. Here $\nabla_{\mathbf{W}_*^k}\mathcal{L}$ contains the derivatives $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{S}^*}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$.

$$\begin{aligned}
&\mathbf{S}_*^{k+1} = [\mathbf{I} + \Psi_{\mathbf{U}^k}^T] \cdot \mathbf{S}_*^k \cdot [\mathbf{I} + \Psi_{\mathbf{V}^k}] - \alpha \Delta \mathbf{S}_*^k \\
&\Psi_{\mathbf{X}} = (\mathbf{X})^T \chi_v(\mathbf{X}) \mathbf{X} \\
&\Delta \mathbf{S}_*^k = (\mathbf{U}^{k+1})^T \nabla_{\mathbf{W}_*^k} \mathcal{L} \mathbf{V}^{k+1}
\end{aligned} \tag{64}$$

For the additive case, the resulting expression can be reduced to a function of the decomposed update equation and an additional term correcting for the context dependent kernel change.

$$\begin{aligned}
&\mathbf{S}^{k+1} = \mathbf{S}^k + \Gamma_{\mathbf{S}_*^k} - \alpha[\Delta \mathbf{S}_*^k - \frac{\partial \mathcal{L}}{\partial \mathbf{\Lambda}} \cdot \vec{c}] \\
&\Gamma_{\mathbf{S}_*^k} = \Psi_{\mathbf{U}^k}^T [\mathbf{S}_*^k + \mathbf{S}_*^k \Psi_{\mathbf{V}^k}] + \mathbf{S}_*^k \Psi_{\mathbf{V}^k}
\end{aligned} \tag{65}$$

For the multiplicative case this reduction cannot be done and the final expression reads as follows:

$$\begin{aligned}
&\mathbf{S}^{k+1} = (\mathbf{P}^{k+1})^{-1} [\mathbf{S}^k + \Gamma_{\mathbf{S}_*^k} - \alpha \Delta \mathbf{S}_*^k] \\
&\Gamma_{\mathbf{S}_*^k} = \Psi_{\mathbf{U}^k}^T [\mathbf{S}_*^k + \mathbf{S}_*^k \Psi_{\mathbf{V}^k}] + \mathbf{S}_*^k \Psi_{\mathbf{V}^k}
\end{aligned} \tag{66}$$

Since these updates are values of the diagonal, the inverse and its multiplication, present in Eq. (66), are element-wise. As such they introduce no significant computational increase. Finally, regular batch stochastic gradient descent can be thought of as reducing the partial gradients w.r.t. the optimized variable over the batch dimension before application.

$$\nabla \theta = \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{L}_i}{\partial \theta} \tag{67}$$

Here B is the size of the batch dimension. Since calculation of the gradients for the singular values now is dependent on every sample as seen in both Eqs. (65) & (66), this reduction is applied to the updated singular values after calculation.

$$\bar{\mathbf{S}}^{k+1} = \frac{1}{B} \sum_{i=1}^{B} [\mathbf{S}^{k+1}]_i \tag{68}$$

For the additive case this is equivalent to averaging the singular values updates, while the context dependence is fully contained in the $\mathbf{\Lambda}$ terms. In the multiplicative case, an extra term inversely proportional to $\mathbf{P}$, that corrects for the consequent multiplicative term, is present in the gradients. The expression for $\bar{\mathbf{S}}^{k+1}$ can now be used to replace the update step in the static singular value decomposed procedure [20].

Table 1
Accuracies and corresponding standard deviations of a zero hidden layer architecture over ten runs at different values of α.

α	$\vec{x}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	$\mathbf{S} + \mathbf{\Lambda}$	**PS**
0.0	$96.5_{\pm 0.1}$	$96.4_{\pm 0.1}$	$92.5_{\pm 0.1}$	$96.3_{\pm 0.1}$	$96.6_{\pm 0.2}$
0.5	$75.4_{\pm 0.1}$	$75.8_{\pm 0.1}$	$89.1_{\pm 0.1}$	$94.4_{\pm 0.4}$	$95.5_{\pm 0.5}$
1.0	$67.2_{\pm 0.1}$	$67.1_{\pm 0.1}$	$81.2_{\pm 0.2}$	$94.5_{\pm 0.8}$	$94.8_{\pm 0.6}$

Table 2
Accuracies and corresponding standard deviations of a single hidden layer architecture over ten runs at different values of α.

α	$\vec{x}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	$\mathbf{S} + \mathbf{\Lambda}$	**PS**
0.0	$96.0_{\pm 0.1}$	$95.4_{\pm 0.1}$	$94.4_{\pm 0.1}$	$94.1_{\pm 0.2}$	$94.8_{\pm 0.2}$
0.5	$79.0_{\pm 0.1}$	$95.0_{\pm 0.3}$	$93.7_{\pm 0.1}$	$95.2_{\pm 0.2}$	$95.5_{\pm 0.1}$
1.0	$65.4_{\pm 0.1}$	$94.0_{\pm 0.3}$	$93.3_{\pm 0.2}$	$93.9_{\pm 0.1}$	$93.4_{\pm 0.2}$

4. Evaluation

In the following section, experiments are performed to evaluate the proposed method on a variety of datasets, starting with artificial data for validation of the methods' properties and leading up to real-world data. All experiments were performed on one Tesla V100-SXM3-32 GB GPU and eight Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70 GHz CPU cores. All implementations are done using Tensorflow 2.4.0. The corresponding code is available on GitHub.[1]

4.1. Artificial data

To get an understanding of the properties of both implementations, confirm convergence and evaluate performance, artificial data is generated. The data consists of a discrete set of contexts which are clustered around K centers using an implementation of the sklearn library.[2] Per cluster, a linear classification problem with different coefficients is created, given by Eq. (69). The coefficients are determined by a constant a_0 and a context dependent coefficient which is different per context cluster a_i. The amount of context dependence is regulated by a parameter α which is varied from zero to one.

$$\begin{aligned}
&y_i|c_i = [(1-\alpha)\vec{a}_0 + \alpha \mathbf{A} \cdot \vec{c}_i] \cdot \vec{x}_i + \epsilon < 0 \\
&\mathbf{A} = [\vec{a}_1, \dots, \vec{a}_K]
\end{aligned} \tag{69}$$

This generated problem has an optimal multiple model solution, consisting of a sigmoidal output neural network without any hidden layers per context. When considering the joint problem, the dynamic weight algorithm is expected to converge without any hidden layers. However, a regular feed-forward network should not be able to model the context dependence of the weights.

The performances of a zero and single layer feed-forward architecture are evaluated for a data set of eight-thousand generated samples with K equal to four. The data is split eighty/twenty for training and testing. For each network the regular implementation ($\vec{x}$) with and without addition of the context vector is evaluated. Both the gating ($\vec{c} \odot \vec{x}$) and concatenation ($[\vec{x}, \vec{c}]$) mechanism are used to include the context in this case. These methods are compared to both the additive ($\mathbf{S} + \mathbf{\Lambda}$) and multiplicative (**PS**) dynamic network implementation. The decomposition ranks used are the upper bound rank for the hidden layer and rank K, corresponding to the number of clusters, for the output layer. Results are given in Tables 1 and 2. The tables portray mean values over ten runs on the same data. Furthermore Table 3 contains the amount of parameters for each model.

[1] https://github.com/predict-idlab/dynamic-kernels.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html.

Table 3
Parameter comparison for different architectures. H_0 and H_1 indicate the zero and single hidden layer architectures respectively.

α	$\vec{x}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	$\mathbf{S+\Lambda}$	**PS**
H_0	66	130	66	270	274
H_1	1122	2148	1122	1382	1386

A couple of observations can be made from these results. First, as expected, the decomposed layers are capable of modeling the dynamic problem without any hidden layers. This allows more efficient, simpler models, which is confirmed by the parameter counts in Table 3. Even tough the amount of parameters for the decomposed approach is larger for the output layer, it still is much smaller than adding a hidden layer. Second, the context independent and Kronecker decomposed kernel assumptions do not necessarily restrict performance on individually dependent weights. Lastly, since the context embeddings are numerical vectors, while the weight dependence is strictly based on the context class, we would expect optimal performance by the multiplicative kernels because of the sigmoid activation of the individual kernels. These have the capacity of classifying the context vectors and changing weights accordingly. A small increase in performance is observed between the linear approximation and the embedding decomposition, but it seems that most of the behavior can be captured in a linear approximation of the context coefficients. This effect is also and more strongly observed for the zero hidden layer gating and concatenation implementations. Lastly, a observation with respect to the standard deviations is made. All the deviations are insignificant with respect to the differences in performance except for a small increase for the zero layer dynamic networks. This is attributed to the fact that a small amount of basis matrices is used and as a result there are some small deviations on the final accuracy over different training runs.

4.2. Indian state-wise covid data

The following evaluation dataset was obtained from Kaggle[3] and consists of confirmed covid cases in India from May 2020 until September 2021. We complemented this dataset with population statistics of the states obtained from the Indian central bank.[4]

As problem statement, a one day forecast of the covid incidence is considered. A couple of pre-processing steps are done. First, the raw incidence count is log transformed as per Eq. (70), as covid data often exhibits exponential behavior. Besides the log transform, the added one keeps the transformed numbers strictly positive.

$$y = \log(x + 1) \tag{70}$$

Second, the transformed values are shifted to represent daily difference instead of incidence. And lastly, a smoothing window of five days is applied to reduce noise on the time series. The resulting processed data per state is given in Fig. 2.

A time-based data split is performed where the first 60% of the time series of each state is used as training and the remainder as test data. Corresponding to the multiple model solution, a baseline is created from both ARIMA (ARM) and a feed-forward neural models (FFN) per state. The neural models are trained on windows of 14 days of data with standard mean squared error loss. Models with 1, 3 and 5 hidden layers, represented by the subscript in their names, each with 64 nodes are trained. Testing is done by sequentially predicting with a 14 day sliding window and concatenating the results. The ARIMA models are fit and evaluated in two different ways. For the first model, ARM_F, all data before the testing point is used as train data and a one day forecast is done. This step is repeated for all test data points and are aggregated to obtain a full test prediction, similar to the predictions by the neural networks. The second ARIMA model, ARM_P, uses all train and test data during fitting and an in sample prediction on just the test data is done. The resulting baseline performances, using the mean absolute error as evaluation metric, are given in Table 4.

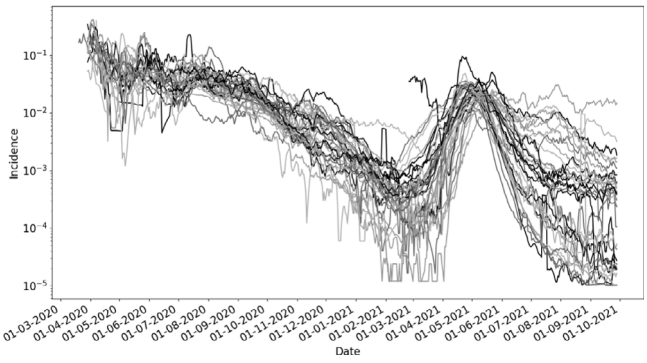


Fig. 2. Pre-processed covid incidence per Indian state. State names are not annotated since data is for illustrative purposes.

Table 4
Test mean absolute error and standard deviations of the baseline models for the Indian Covid Dataset.

FFN_1	FFN_3	FFN_5	ARM_F	ARM_P
$25.1_{\pm 0.1}$	$26.9_{\pm 0.1}$	$28.1_{\pm 0.1}$	$80.7_{\pm 0.0}$	$18.8_{\pm 0.0}$

Table 5
Mean absolute errors and standard deviations on these errors for different evaluated general architectures on the Indian Covid Dataset. In class best performance is indicated in bold and the overall best performance in gray.

Model	$\vec{x}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	$\mathbf{S+\Lambda}$	**PS**
FFN_1	26.2 ± 7.1	59.6 ± 12.9	19.9 ± 1.6	19.5 ± 1.5	$\mathbf{19.2 \pm 0.9}$
FFN_3	24.5 ± 2.0	62.4 ± 11.8	20.2 ± 2.5	$\mathbf{19.1 \pm 0.5}$	19.7 ± 1.5
FFN_5	24.3 ± 2.4	53.4 ± 4.8	21.9 ± 3.2	$\mathbf{20.1 \pm 1.9}$	22.9 ± 1.7
$LSTM_1$	20.5 ± 0.2	52.0 ± 6.5	19.3 ± 0.4	18.7 ± 0.3	$\mathbf{17.7 \pm 0.9}$
$LSTM_3$	19.3 ± 0.2	53.4 ± 4.8	18.6 ± 0.2	18.6 ± 0.7	$\mathbf{17.6 \pm 0.6}$
$LSTM_5$	18.3 ± 0.1	44.2 ± 8.5	18.7 ± 0.4	19.0 ± 0.1	$\mathbf{17.2 \pm 0.4}$

The ARM_P model has the best performance by far, since it uses in sample prediction. The ARM_F model, however, performs significantly worse than the baseline neural models. Both have a zero standard deviation since ARIMA is deterministic. The neural baselines, $FFN_1 \rightarrow FFN_5$, exhibit a decreasing performance with model complexity. Since they are trained per state, each model only gets a fraction of the available data, which is likely not enough to train the larger models. Finally, we note that the standard deviations for these models are very low which is expected when they are trained and evaluated on a subset of data with very similar properties. On this case that is the Covid data from the same Indian state.

Furthermore, general models are tested and compared to these baselines. Both a set of feed-forward and LSTM architectures are evaluated. For the LSTM network a single bi-directional LSTM layer with 64 units is used, followed by 1, 3 and 5 feed-forward layers. The networks are denoted by their names, FFN or $LSTM$, and a subscript number that represents the amount of hidden layers, each with 64 neurons. Results, which are average mean absolute errors over 10 training runs with different initialization and are accompanied by their corresponding standard deviations, for all implementations are given in Table 5.

As a first observation, for both the feed-forward and LSTM regular architectures, the same trend is recognized. Namely, performance increases with increasing model complexity and is significantly better

[3] https://www.kaggle.com/oddasparagus11/covid19-statewise-time-series-datatill-sep29.

[4] https://m.rbi.org.in/scripts/AnnualPublications.aspx?head=Handbook+of+Statistics+on+Indian+States.

than the baseline models as seen in Table 4, attributed to the larger amount of data available. This also holds for the feature and context concatenation implementation, however, these models perform much worse than any other implementation. This is postulated to be due to the fact that a static vector is added to the windowed dynamic time series data. This illustrates one of the drawbacks mentioned in the introduction with respect to data dimensionality. The gating models do not suffer from this drawback and as such perform better than the regular implementations. However, this implementation decreases in performance for the feed-forward network with increasing model depth. This suggests that implementing context in this manner leads to more confused models at higher depths, while for the LSTM implementations the gain with respect to the regular network is marginal and not even present at model depth five. Finally for all model depths and for both the regular as well as the LSTM networks, the dynamic models consistently outperform the standard implementations. The multiplicative kernel LSTM architectures show significant, based on the standard deviations, performance increases with respect to all other implementations. The best performing architecture, being the $LSTM_5$ model, has a mean absolute error decrease of 1.1 which corresponds to 11 standard deviations of the best performing regular implementation.

As a final remark it seems that both the data structure and Occam's razor argument hold for this problem. Despite the task of predicting Covid cases not explicitly giving an idea about the information completeness of the time series data.

4.3. YFCCM100-GEO100

As prefaced in the introduction, the classification of images using geolocation information is a context aware problem, and therefore this is the third evaluation problem considered. The dataset used is called YFCCM100 - GEO100, which is a subset of the popular dataset YFCCM100M [25], presented by Tang et al. 2015 [18]. Due to the inavailability of the mentioned contextual data mentioned in their work, a slightly different dataset is created by scraping all the image links that are still available on Flickr for their hashtags. The images have corresponding sets of tags, which were filtered to contain none of the class labels directly to avoid data leakage. Furthermore, all the images without tags were omitted. Following this, for each image, the list of tags was embedded using standard pre-trained GloVe embeddings [26] and a smooth inverse frequency approach [27] as given in Eq. (71). This final embedding is taken to be the context of the image. The images themselves are pre-processed using normal standardization over the channels, as is custom with convectional networks for image classification.

$$v_s = \frac{1}{|s|} \sum_{v_w \in S} \frac{a}{a + p(w)} v_w \tag{71}$$

The model architectures consist of a ResNet50V2 [28] with pre-trained ImageNet weights as feature extractor, a set of M, taken from 1 to 5, hidden layers (H_M) with 'elu' activation and a final softmax activated layer for classification. Five image based models are trained for evaluation. The first model only uses images as input. The second and third model use both images and contextual tags. These apply concatenation or gating in the hidden layers respectively. The fourth and fifth model implement the proposed additive and multiplicative methods with solely the images as input and tags as context. Furthermore, to get an understanding of the inherent classification information present in the context, a feed-forward model with only the tags as input was also trained. All models were trained with the following optimization parameters.

Epochs	Decays	L_r	ν	Optimizer
50	4×0.5	10^{-6}	10^{-6}	ADAM

Table 6
Accuracies for different sizes of the Glove embeddings. The best performing architecture per embedding size is indicated in bold and the overall best performance in gray.

	$\vec{x}$	$\vec{c}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	**S + Λ**	**PS**
.5	46.6	15.4	47.4	51.6	53.3	**53.6**
1	46.8	21.9	47.2	50.8	**53.7**	53.3
2	46.9	30.9	47.5	50.1	53.7	**54.1**
3	46.5	35.9	47.9	49.9	**55.0**	53.8

Table 7
Accuracies and training times per epoch for the decomposed architectures at different ranks with a GloVe embedding size of 300. The highest values at the best performing rank are indicated in bold.

R^{-1}_{upper}	0.25	0.5	0.75	1	1.25
S + Λ	52.1	54.9	**55.9**	55.0	54.1
PS	52.2	53.1	**54.9**	53.7	53.6
s/epoch	58	92	153	241	358

Three stages of evaluations were done to study the performance of the previously mentioned models. Firstly, the GloVe embedding dimensions were varied to see the impact of the contextual information. Secondly, after fixing the GloVe dimension, different ranks of the weight matrices' decomposition were evaluated and lastly, at fixed GloVe embedding dimension and rank, different model depths were tested.

4.3.1. Glove dimensionality

First the dependence of performance on the expressivity of the GloVe embeddings at different dimensions was tested. For the rank of each model the upper bounds from Sections 3.3.2 and 3.3.3 were used to compare models with the same amount of trainable parameters. Test accuracies for each individual model and GloVe embeddings of sizes 50, 100, 200 and 300, which are expressed in units $10e2$ for brevity, are given in Table 6.

Two expected results for the basic context and image models are observed. First, with increasing embedding size, the classification potential of the context only model increases significantly. Second, the image only model performance stays consistent within some range due to training variance. The combined models, both the concatenated and gated implementations, outperform the image only model, with the gated model being the better of the two. However, the increased accuracy declines with increasing GloVe dimensions, which indicates, as suggested in the introduction, that the added information from the contextual tags confuses the model more than it is capable of introducing useful information. This is in contrast to both decomposed implementations which outperform all other models with significant margins and also increase in performance with increasing GloVe dimension, especially for the additive implementation. The best performance is achieved for the 300 dimensional GloVe embedding with a testing accuracy of 55% which is a 10% relative increase with respect to the best performing regular implementation at this dimension.

4.3.2. Rank dependence

The previous subsection determined performance for different GloVe embedding sizes at the upper bound rank. In the following, a comparison for different ranks in function of the respective upper bounds is made. The GloVe embedding dimension is kept fixed at 300. The results and training times per epoch are given in Table 7.

Table 7, at first sight, contains a rather unusual trend. For the smallest rank, at a quarter of the upper bound rank, both the decomposed models perform sub-optimally at this GloVe dimension. This is to be expected since the small rank gives less flexibility to learn the dynamic nature of the classification problem. However, when increasing the rank, one would expect the performance to keep increasing. This is not the case, rather the performance peaks at rank three quarters of the

Table 8
Accuracies for different model depths. Best same depth performance architectures are highlighted in bold and best overall architecture in gray.

	$\vec{x}$	$\vec{c}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	**S + Λ**	**PS**
H_1	46.8	35.9	47.9	49.9	**55.9**	54.9
H_3	53.1	39.2	55.5	51.4	**58.3**	57.2
H_5	54.0	38.8	**54.9**	49.2	54.1	53.7

Table 9
Trainable parameter comparison of image models for different model depths. Expressed in factor of a million.

	$\vec{x}$	$[\vec{x}, \vec{c}]$	$\vec{c} \odot \vec{x}$	**S + Λ**	**PS**
H_1	4.3	5.6	4.9	4.3	3.7
H_3	8.5	11.1	9.8	8.3	7.4
H_5	12.7	16.6	14.5	12.5	11.0

upper bound and starts decreasing for larger ranks. This decrease can be attributed to the same confusion phenomenon present with the concatenation and gating models related to apparent over-parametrization. The training times do keep increasing with larger ranks, attributed to the matrix inverse and multiplications needed for back-propagation, which also serve as the main drawback of the proposed solution.

4.3.3. Model depth dependence

With both fixed context dimensionality and rank, 300 and $0.75 \times R_{upper}$ respectively, a comparison of different model depths is done. Tables 8 and 9 present the test accuracies and amount of parameters for depths one, three and five respectively.

The image only model increases in performance with each increase in depth, reaching an overall accuracy of 54% for five layers. When looking at the context only model, maximal performance is reached at three layers depth. This behavior seemingly carries over to the combined image and context models, which also all maximize performance at depth three. The best model for this depth is the additive decomposed approach which reaches 58.3% accuracy, a performance increase of 8% with respect to the image only model and a 5.5% increase with respect to the best standard implementation model, being the depth three concatenated model, which peaks at 55.5% accuracy. Furthermore the multiplicative approach also outperforms all the classical approaches at this depth, but with a smaller increase in accuracy. At a depth of five layers the concatenated classical implementation is the best performing architecture, however all accuracies except for the image only model are lower than at three layers depth. This implies that five layers is too deep for the context dependent models. Furthermore, both the additive and multiplicative approaches reduce the amounts of parameters even with respect to the image only model as can be seen from Table 9. This reduction is greater when the optimal performance is at smaller ranks with respect to the rank upper bounds.

5. Conclusion

This work proposes a novel methodology for the dynamic construction of parameter shared neural networks through the use of explicit context representations. These context representations are assumed to be informative representations about the decision making process, given a specific task. This definition is obtained starting from a general definition of contextual information and guided by the concept of information completeness and Occam's razor.

The method starts by decomposing the weight matrices in the neural network architecture into a linear combination of basis components corresponding to the singular value decomposition of these matrices. By contracting the context-specific kernels based on the informative context representation for each sample, a dynamic neural architecture is created. This contraction is done by taking the basis coefficients, or conversely the singular values, to be direct function of the context representations. Both an additive and multiplicative version of the kernel composition is derived corresponding to the linear approximation of the context dependent weights and an EDF decomposition of the context dependent latent vector respectively. By benchmarking on an artificial dataset, we show the capacity of this method to model individual context dependent weights in a parameter efficient way. A side note to this performance is that the context dependence is modeled by a hyper-parameter α and performance improvements are only noticeable for problem statements that have an effective high α or large amount of context dependence. Consequently, the model is also evaluated on both a real-world time series and image dataset. Performance comparisons are done with standard latent feature concatenation, context gating and both the general and multiple model solutions to the context-aware problem statement. The proposed method, both the additive and multiplicative version, outperforms all standard implementations both in parameter efficiency as in performance, with a trade-off in training time. Since both benchmark cases, Indian Covid Data and YCCM-GEO100, represent aspects of information completeness and differing data structures, we hypothesise that the assumptions made on the context dependent problem formulation are sufficiently broad to include many problems both in benchmarks as in complex use-cases. The dynamic nature of this approach promises to improve performance on highly context specific and dependent learning problems.

6. Future work

The main concepts introduced in this work are the context representations $\vec{c}$ and the decomposition coefficients $\mathbf{S}(\vec{c})$ generated by these context representations. The context dependent coefficients lead to the construction of dynamic neural networks using the singular value decomposition. In order for these to perform the context representations should represent the complex context situation accurately and the given problem should contain significant context dependence. In this light a couple of future research directions are planned.

- Implementation of the proposed approach in complex use-cases where context-dependence is of large importance.
- Different methods to construct context representations should be investigated.
- Additional investigation to more complex architectures and/or loss functions to increase information usage of both context and input data.

CRediT authorship contribution statement

David Vander Mijnsbrugge: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Writing – original draft, Visualization. **Femke Ongenae:** Project administration, Funding acquisition, Supervision, Writing – review & editing. **Sofie Van Hoecke:** Project administration, Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data will be shared on the code GitHub as referenced in the manuscript.

Acknowledgments

This research received funding from the Flemish Government (AI Research Program), Belgium.

Appendix. Proof of Eq. (52)

Starting with the decomposition of the latent vector in the context and static part.

$$\vec{h} = \mathbf{U}[\mathbf{S} + \mathbf{\Lambda} \cdot \vec{c}]\mathbf{V}^T \cdot \vec{x} \tag{A.1}$$

$$= \underbrace{\mathbf{USV}^T}_{\text{I}} \cdot \vec{x} + \underbrace{\mathbf{U}[\mathbf{\Lambda} \cdot \vec{c}]\mathbf{V}^T}_{\text{II}} \cdot \vec{x} \tag{A.2}$$

Focusing on the second equation (II) and writing out the full matrix multiplication summation.

$$\sum_{ij} [\mathbf{\Lambda} \cdot \vec{c}]_i \vec{u}^i v_j^i x_j \tag{A.3}$$

$$\sum_{ijk} \lambda_k^i c_k \vec{u}^i v_j^i x_j \tag{A.4}$$

Since c_k, x_j, λ_k^i and v_j^i are scalars the summations can be switched in order. In matrix notation and with addition of (I) the full latent vector is written as follows.

$$\vec{h} = \mathbf{USV}^T \cdot \vec{x} + \mathbf{U}[\mathbf{V}^T \cdot \vec{x}]\mathbf{\Lambda} \cdot \vec{c} \tag{A.5}$$

References

[1] F. Xu, G. Guo, F. Zhu, X. Tan, L. Fan, Protein deep profile and model predictions for identifying the causal genes of male infertility based on deep learning, Inf. Fusion 75 (2021) 70–89, http://dx.doi.org/10.1016/j.inffus.2021.04.012, URL https://www.sciencedirect.com/science/article/pii/S1566253521000853.

[2] L. Li, F. Zhu, H. Sun, Y. Hu, Y. Yang, D. Jin, Multi-source information fusion and deep-learning-based characteristics measurement for exploring the effects of peer engagement on stock price synchronicity, Inf. Fusion 69 (2021) 1–21, http://dx.doi.org/10.1016/j.inffus.2020.11.006, URL https://www.sciencedirect.com/science/article/pii/S1566253520304127.

[3] A. Belhadi, Y. Djenouri, G. Srivastava, D. Djenouri, J.C.-W. Lin, G. Fortino, Deep learning for pedestrian collective behavior analysis in smart cities: A model of group trajectory outlier detection, Inf. Fusion 65 (2021) 13–20, http://dx.doi.org/10.1016/j.inffus.2020.08.003, URL https://www.sciencedirect.com/science/article/pii/S1566253520303316.

[4] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, P. Steggles, Towards a better understanding of context and context-awareness, in: H.-W. Gellersen (Ed.), Handheld and Ubiquitous Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 304–307.

[5] K. Akande, T. Owolabi, S. Olatunji, Investigating the effect of correlation-based feature selection on the performance of support vector machines in reservoir characterization, J. Natl. Gas Sci. Eng. 22 (2015) http://dx.doi.org/10.1016/j.jngse.2015.01.007.

[6] M.A. Hall, Correlation-Based Feature Selection for Machine Learning (Ph.D. thesis), University of Waikato Hamilton, 1999.

[7] P. Gibbs, S. Hiroshi, What is Occam's razor, 1996.

[8] K. Ayush, B. Uzkent, C. Meng, K. Tanmay, M. Burke, D. Lobell, S. Ermon, Geography-aware self-supervised learning, in: 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2020, pp. 10161–10170.

[9] W. Fan, F. Geerts, Relative information completeness, 35 (4) 2010. http://dx.doi.org/10.1145/1862919.1862924.

[10] F. Naumann, J.-C. Freytag, U. Leser, Completeness of integrated information sources, Inf. Syst. 29 (7) (2004) 583–615.

[11] T. Galanti, L. Wolf, T. Hazan, A theoretical framework for deep transfer learning, Information and Inference: A Journal of the IMA 5 (2) (2016) 159–209.

[12] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2009) 1345–1359.

[13] L. Torrey, J. Shavlik, Transfer learning, in: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, IGI global, 2010, pp. 242–264.

[14] E.V. Bonilla, F.V. Agakov, C.K. Williams, Kernel multi-task learning using task-specific features, in: Artificial Intelligence and Statistics, PMLR, 2007, pp. 43–50.

[15] W. Fedus, B. Zoph, N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021, arXiv preprint arXiv:2101.03961.

[16] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, Y. Wang, Dynamic neural networks: A survey, IEEE Trans. Pattern Anal. Mach. Intell. (2021).

[17] Y. Li, Y. Chen, M. Dai, D. Chen, Y. Yu, L. Yuan, Z. Liu, M. Chen, N. Vasconcelos, Revisiting dynamic convolution via matrix decomposition, 2021, arXiv preprint arXiv:2103.08756.

[18] K. Tang, M. Paluri, L. Fei-Fei, R. Fergus, L. Bourdev, Improving image classification with location context, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1008–1016.

[19] J.C.D. Terry, H.E. Roy, T.A. August, Thinking like a naturalist: Enhancing computer vision of citizen science images by harnessing contextual data, Methods Ecol. Evol. 11 (2) (2020) 303–315.

[20] D.V. Mijnsbrugge, F. Ongenae, S. Van Hoecke, Parameter efficient neural networks with singular value decomposed kernels, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–11, http://dx.doi.org/10.1109/TNNLS.2021.3130756.

[21] Y. Zeng, Context aware machine learning, 2019, arXiv preprint arXiv:1901.03415.

[22] N.Y. Masse, G.D. Grant, D.J. Freedman, Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization, Proc. Natl. Acad. Sci. 115 (44) (2018) E10467–E10475.

[23] A. Miech, I. Laptev, J. Sivic, Learnable pooling with context gating for video classification, 2017, arXiv preprint arXiv:1706.06905.

[24] H. Wen, S. You, Y. Fu, Cross-modal context-gated convolution for multi-modal sentiment analysis, Pattern Recognit. Lett. 146 (2021) 252–259.

[25] B. Thomee, D.A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, L.-J. Li, YFCC100M: The new data in multimedia research, Commun. ACM 59 (2) (2016) 64–73.

[26] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1532–1543.

[27] S. Arora, Y. Liang, T. Ma, A simple but tough-to-beat baseline for sentence embeddings, in: International Conference on Learning Representations, 2017.

[28] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.