Learning Representations for Discrete Sensor Networks using Tensor Decompositions

Stephan Baier[1], Denis Krompass[2] and Volker Tresp[3]

***Abstract*— With the rising number of sensing devices installed in today's and future sensor networks, there is an increasing demand for machine learning solutions performing tasks like automatic behavior detection and decision making. In particular, to classify the state of the complete sensor network, machine learning models are needed, which are capable of fusing the information from multiple sensors. In this paper we examine the use of tensor models to describe the relationship between multiple discrete sensor outputs and attendant class labels describing the overall system state. Tensor decompositions can be considered as a form of representation learning and they have been used for a variety of tasks, e.g. knowledge graph modeling and EEG data analysis. We propose a new approach for multiclass classification using tensor decompositions. As the dimensions of the tensors used in the multi-sensor classification are much higher than in traditional tasks, not all standard decomposition approaches are applicable due to scaling problems. In our experiments on real data, we show that the PARAFAC and Tensor Train decompositions work well for discrete multi-sensor fusion tasks and outperform other state-of-the-art machine learning algorithms.**

I. INTRODUCTION

Due to the growing number of sensing devices in many application areas, the study and the modeling of large sensor networks have received increasing attention. To make sense of the collected information, novel machine learning models are needed, which are able to detect and predict a system's behavior. These models need to fuse and filter relevant information from a variety of sensing devices to obtain high quality classification results [1][2].

Representation learning, also referred to embedding learning, is a key concept in many modern machine learning applications including natural language processing, computer vision and relational learning [3], [4], [5]. The goal is to learn a vector representation of fixed length for every entity, where similar entities get similar representations. Those expressive representations are then the basis for tasks like classification, clustering or outlier-detection.

Tensor decompositions, which are a generalization of matrix factorizations, can be considered as a form of representation learning. They have been used successfully for modeling large semantic knowledge graphs [4]. Another application area of tensor decompositions is the analysis of factors in multi-channel time series like EEG data [6].

[1] Stephan Baier, Ludwig Maximilian University of Munich, Oettingenstr. 67, 80538 Munich stephan.baier@campus.lmu.de

[2]Denis Krompass, Siemens AG, Otto-Hahn-Ring 6, 81739 Munich denis.krompass@siemens.com

[3]Volker Tresp, Siemens AG, Otto-Hahn-Ring 6, 81739 Munich and Ludwig Maximilian University of Munich, Oettingenstr. 67, 80538 Munich volker.tresp@siemens.com

In this paper we propose a new modeling approach for sensor networks with discrete sensor measurements. The models learn a representation for each possible value of each sensor and fuse these representations for classifying the current behavioral state of the whole sensor network.

To perform multiclass classification tasks, the data is mapped to multiple class tensors which are decomposed using standard factorization techniques. The approximations of the class tensors are then combined by a softmax function. To enforce representations that are suitable for the classification task, the tensor decompositions are directly learned end-to-end, using multinomial classification labels of the training data. The number of free parameters is reduced by sharing the representations between the decompositions of the class tensors.

In typical machine learning applications where tensor models are used, the number of dimensions is relatively low, typically three or four [7], [4]. Our proposed method uses tensors with a rather large number of dimensions: More specifically, the number of dimensions is as large as the number of sensors in the network. However, the tensor is extremely sparse as only entries describing settings from the training data are non-zero. Many tensor decomposition approaches do not scale to large dimensions. Exceptions are the standard PARAFAC decomposition [8] and the recently proposed Tensor Train decomposition [9], which are thus both applicable to high-dimensional domains, e.g. to data fusion in large systems.

The novel contributions of this paper are, first, the modeling of discrete sensor inputs using a high dimensional tensor representation as opposed to a single feature vector, and, second, a novel approach for doing multiclass classification using tensor factorization techniques. In our experiments, we use board configurations from the game connect-4 to simulate a sensor network with discrete sensor values. The task is to classify the most probable outcome of the game given a certain board configuration. The problem is transferable to classifying the system state of a sensor network with 42 sensors and 3^{42} possible combinations of sensor states.

The paper is organized as follows. In the next section we give an overview of related work. Section III summarizes the most important tensor decompositions. In Section IV we show how representation learning using tensor decompositions can be used for multi-sensor fusion and propose a framework for solving multiclass classification using tensor decompositions. In Section V an empirical study shows the effectiveness of the proposed method. We conclude our work in Section VI.

II. Related Work

Representation learning is used in many areas of machine learning. In particular, representation learning using matrix factorizations has become popular due to its success in the Netflix challenge [10]. In natural language processing, word embeddings are used for automatic machine translation, knowledge extraction, clustering, etc. [3], [11], [12]. Large knowledge bases like Yago or DBPedia have been modeled using latent embedding models like the tensor decomposition RESCAL [4] or Google's Knowledge Vault model [12][13]. Representation learning is also used in state-of-the-art computer vision applications like Facebook's DeepFace [5] for similarity search in face images. Tensor decompositions are a great way to learn representations of entities, which are involved in relationships of higher orders. An overview of tensor decompositions can be found in [7] and [14]. Latent representations obtained by tensor decompositions are also used for factor analysis in data mining [6], [8].

The Tensor Train decomposition [9] is a generalization of the well known Tucker2 decomposition [15], [7]. It found applications in physics and quantum chemistry for function approximation [16], [17]. An application of Tensor Trains in machine learning can be found in [18] where the memory footprint of fully connected layers in a deep convolutional neural network are significantly reduced by using a low rank factorization. A similar approach has been taken in [19] using the PARAFAC decomposition [8] for compressing weight tensors in convolutional neural networks.

III. Tensor Decompositions

Tensor decompositions are a generalization of low rank matrix factorizations to higher order tensors. There are multiple ways of decomposing a higher order tensor into low rank tensors. In this section we present the basic variants Canonical Decomposition and Tucker Decomposition as well as the relatively new Tensor Train Decomposition, which solves the scalability problem of the full Tucker Decomposition.

In the following, scalars are represented by small letters x, vectors are represented by bold small letters $\mathbf{x}$ and matrices by capital letters X. Cursive capital letters $\mathcal{X}$ denote higher order tensors. Furthermore, elements of a vector are represented by indexing with a scalar $\mathbf{x}(i)$. Elements of a matrix are denoted by indexing with a tuple of length two $X(i_1, i_2)$ and elements of a higher order tensor are denoted by indexing with a tuple of length $\tilde{d} \in \mathbb{N}$ as $\mathcal{X}(i_1, ..., i_{\tilde{d}})$ accordingly.

A. Canonical Decomposition

A straight forward generalization of the matrix factorization to higher dimensions is the canonical decomposition also called PARAFAC. It decomposes a tensor $\mathcal{X} \in \mathbb{R}^{n_1, n_2, ..., n_{\tilde{d}}}$ into matrices $A_d \in \mathbb{R}^{n_d \times \tilde{r}}$ for $d \in \{1, ..., \tilde{d}\}$ and a core vector $\mathbf{g} \in \mathbb{R}^{\tilde{r}}$ such that

$$\mathcal{X}(i_1, i_2, ..., i_{\tilde{d}}) = \sum_{r=1}^{\tilde{r}} \mathbf{g}(r) \cdot A_1(i_1, r) \cdot A_2(i_2, r) ... \cdot A_{\tilde{d}}(i_{\tilde{d}}, r). \tag{1}$$

The rows of the matrices A_1 to $A_{\tilde{d}}$ contain the representations for all entities in all dimensions. The length of the representation vectors is $\tilde{r}$, which is equal for all dimensions. The vector $\mathbf{g}$ can be interpreted as a weight vector fusing the representations.

B. Tucker Decomposition

In the Tucker decomposition all interactions between the latent dimensions are modeled in a core tensor $\mathcal{G} \in \mathbb{R}^{\tilde{r}_1, ..., \tilde{r}_{\tilde{d}}}$. The Tucker decomposition is defined as

$$\begin{aligned}\mathcal{X}(i_1, i_2, ..., i_{\tilde{d}}) = \sum_{r_1}^{\tilde{r}_1} ... \sum_{r_{\tilde{d}}}^{\tilde{r}_{\tilde{d}}} \mathcal{G}(r_1, ..., r_{\tilde{d}}) \cdot A_1(i_1, r_1) \\ \cdot A_2(i_2, r_2) ... \cdot A_{\tilde{d}}(i_{\tilde{d}}, r_{\tilde{d}}).\end{aligned} \tag{2}$$

The matrices A_1 to $A_{\tilde{d}}$ contain representations for the indexed entities similar to the PARAFAC decomposition. However, the fusion of the multiple representations is more powerful as each possible combination of factors is weighted differently by an element of the core tensor $\mathcal{G}$. Note that PARAFAC is a special case of the Tucker decomposition, where the core tensor is diagonal. For higher order tensors the core tensor in the Tucker decomposition quickly explodes, as the number of elements in the core tensor grows exponentially with the number of dimensions.

C. Tensor Train Decomposition

Due to the limitations on scalability of the Tucker decomposition, the Tensor Train decomposition for higher order tensors has been proposed. Tensor Trains consist of multiple low rank tensors in a row, each with a fixed order of three, such that the number of elements does not grow exponentially with the dimensions. The tensor train decomposition is defined as

$$\begin{aligned}\mathcal{X}(i_1, i_2, ..., i_{\tilde{d}}) = \sum_{r_0=1}^{\tilde{r}_0} ... \sum_{r_{\tilde{d}}=1}^{\tilde{r}_{\tilde{d}}} \mathcal{A}_1(r_0, i_1, r_1) \cdot \mathcal{A}_2(r_1, i_2, r_2) \\ ... \cdot \mathcal{A}_{\tilde{d}}(r_{\tilde{d}-1}, i_{\tilde{d}}, r_{\tilde{d}}),\end{aligned} \tag{3}$$

where $\mathcal{A}_d \in \mathbb{R}^{\tilde{r}_{d-1}, n_d, \tilde{r}_d}$ for $d \in \{1, ..., \tilde{d}\}$ and $\tilde{r}_0 = \tilde{r}_{\tilde{d}} = 1$ so that $\mathcal{A}_1$ and $\mathcal{A}_{\tilde{d}}$ are in fact matrices as the third dimension has only length one and $\mathcal{A}_2, ..., \mathcal{A}_{\tilde{d}-1}$ are tensors of order 3. To reduce the number of hyperparameters, the length of the representations $\tilde{r}_1$ to $\tilde{r}_{\tilde{d}-1}$ are assumed equal in our paper.

If the factorized tensor $\mathcal{X}$ has only three dimensions, this decomposition is similar to the earlier proposed Tucker-2 model [7]. Note that in the Tensor Train decomposition the representations for the dimensions are not vectors but matrices. Thus every entity is represented by a latent matrix, which controls the interaction with the neighboring entities. Only entities of the first and last dimensions are represented by vectors as in the PARAFAC and the Tucker decompositions mentioned before.

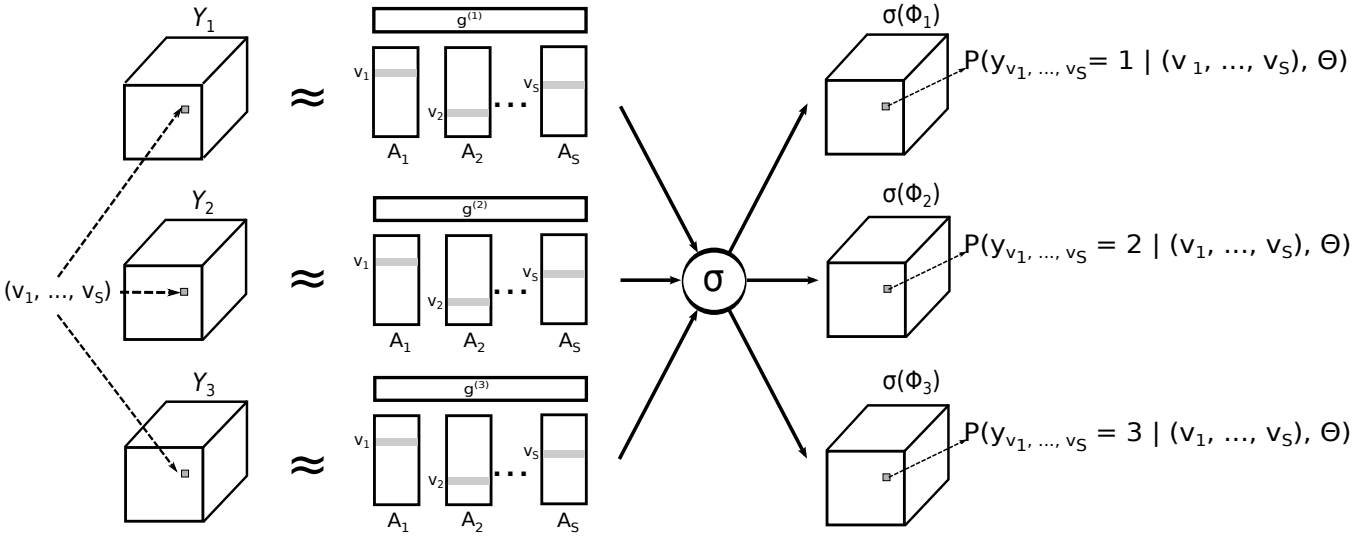

Fig. 1. Multinomial classification model using PARAFAC decomposition. The tensors are plotted three dimensional for illustrative reasons, however in reality their dimensionality is determined by the number of sensors in the network. The three class tensors $\mathcal{Y}_c$ are decomposed into the embedding matrices A_1 to A_S. σ denotes the softmax function such that the output can be interpreted as probabilities for each class. Θ denotes all parameters of the decomposition.

D. Low Rank Approximation

Tensors can be approximated by a reconstruction based on a factorization with limited rank. If the tensor is sparse, i.e. most entries are zero, only the non-zero elements have to be stored. The tensor can still be decomposed using either sparse linear algebra or batch training where only a small amount of zero entries is sampled for each batch [4], [20], [21]. Using low ranks, the decompositions are much smaller in the number of dimension than the full tensor and can thus be stored more easily in memory. Also, single entities of the approximate tensor can be computed, without having to compute the full approximate tensor. This way one can deal with very large tensors as long as their entries are sparse. In this paper we make use of this property.

IV. Tensor Models for Sensor Fusion

In this chapter we formulate the problem of discrete multi-sensor fusion for multiclass classification as a tensor decomposition problem and show how the model can be trained using a maximum likelihood approach. Figure 1 shows the model schematically when using a PARAFAC decomposition and Figure 2 when using Tensor Train decomposition.

A. Discrete Multi Sensor Fusion

We consider the scenario of multiple sensors where $S \in \mathbb{N}$ is the amount of sensors in a sensor network. The i-th sensor for $i \in \{1, ..., S\}$ measures one out of $F_i \in \mathbb{N}$ discrete sensor values. Additionally we consider a class label y, which describes the state of the whole system, e.g. *normal*, *incident*, *shut down*. The training data consists of tuples $(v_1, ..., v_S)$ with $v_i \in \{1, ..., F_i\}$, which describe the discrete measurements of all sensors at a certain point in time, and the attendant class labels $y_{v_1,...,v_S}$ for the respective system state.

All training examples belonging to the same class are mapped to a sparse tensor $\mathcal{Y}_c \in \mathbb{R}^{F_1,...,F_S}$ for $c \in \{1, ..., C\}$ where $C \in \mathbb{N}$ describes the number of classes. The tensors $\mathcal{Y}_c$ are filled as

$$\mathcal{Y}_c(v_1, ..., v_S) = \Pi(y_{v_1,...,v_S} = c), \tag{4}$$

where the indicator function $\Pi(x)$ is one if the statement x is true and zero otherwise. Thus, in the class tensors all positions indexed by the training tuples are set to one, other positions are set to zero. The dimensions of the class tensors $\mathcal{Y}_c$ with order S represent the multiple sensors in the network. The elements along each dimension describe the discrete values of the respective sensor. This way every sensor can take a different set of possible values, which is necessary for modeling various types of sensors within the same sensor network.

B. Multinomial Logistic Regression

We assume a categorical distribution for the class label y. The class label $y_{v_1,...,v_S}$ which is associated with the sensor measurements described by the indexes $(v_1, ..., v_s)$ is modeled as

$$y_{v_1,...,v_S} \sim Cat\big([\sigma(\Phi_1(v_1, ..., v_S)), ..., \sigma(\Phi_C(v_1, ..., v_S))]\big), \tag{5}$$

where Cat stands for the categorical distribution, parameterized by a vector of probabilities, Φ_c denotes a low rank tensor approximation of $\mathcal{Y}_c$ and σ is the softmax function, which is defined as

$$\sigma(\Phi_c(v_1, ..., v_S)) = \frac{e^{\Phi_c(v_1,...,v_S)}}{\sum_k^C e^{\Phi_k(v_1,...,v_S)}}. \tag{6}$$

The softmax function normalizes the values across the low rank approximations such that $\sum_{c=1}^{C} \sigma(\Phi_c(v_1, ..., v_S)) = 1$.

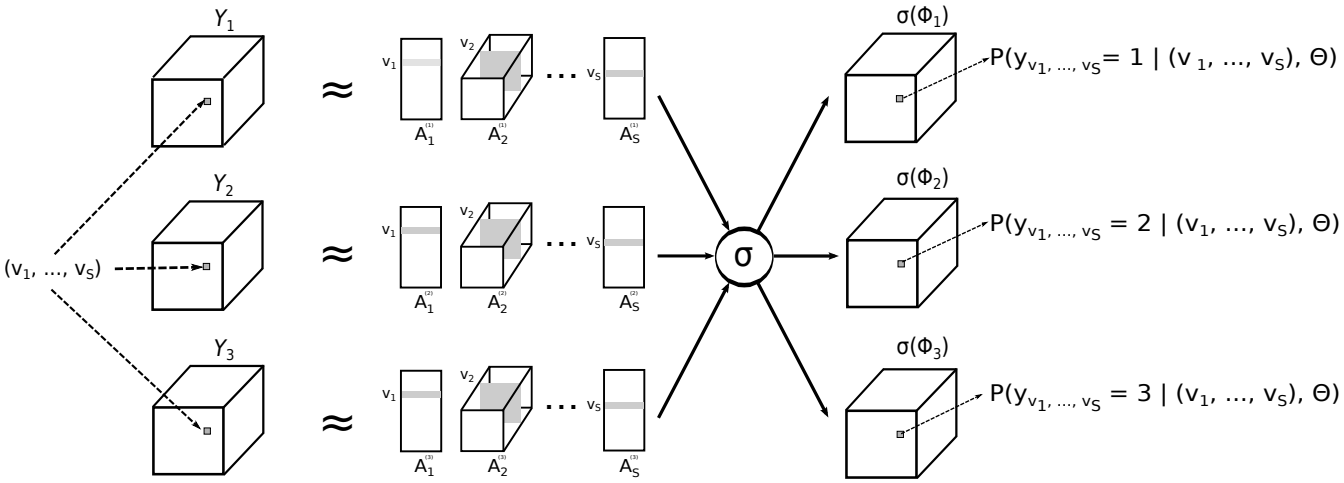

Fig. 2. Multinomial classification model with three classes using Tensor Train decomposition. The order of the tensors is S, but drawn three dimensional. The three rows correspond to the number of classes. The three class tensors $\mathcal{Y}_c$ are decomposed into the core tensors A_1 to A_S. σ denotes the softmax function such that the output can be interpreted as probabilities for each class. Θ denotes all parameters of the decomposition.

This makes the values interpretable as probabilities for the categorical distribution, i.e.

$$\sigma(\Phi_c(v_1, ..., v_S)) = P(y_{v_1,...,v_S} = c \mid (v_1, ..., v_S);\ \Theta), \quad (7)$$

where Θ are all parameters of the decomposition. The categorical cross-entropy loss function, which corresponds to the negative log-likelihood for the categorical distribution, is defined as

$$l = -\sum_{i=1}^{N}\sum_{c=1}^{C} \Pi(y_{v_1^i,...,v_S^i} = c)\ \log\ \sigma(\Phi_c(v_1^i, ..., v_S^i)), \quad (8)$$

where $i \in \{1, ..., N\}$ denotes the i-th training example. The negative log-likelihood can be minimized using stochastic gradient descent. Note, that the iteration only goes over the entries in the training set and not over all positions in the tensors, i.e. the sparsity of the tensors is exploited. All tensor decompositions are learned jointly end-to-end. Thus, optimal representations for the multinomial classification task are learned. It would also be possible to use a separate cost function for the decomposition of each class tensor and normalize the outcomes only for the predictions in order to obtain probabilities. Efficient closed form solutions exist for all common tensor factorizations, when using a squared error cost function. However, the assumption of a Gaussian distribution, implied by a squared error, for each tensor element is not very natural for the classification task we are considering, as opposed to the categorical distribution we are using. It also has been found for other representation learning methods, that learning the decompositions directly end-to-end for the respective task yields a significant performance gain [20], [22], [23].

During training the weights of the tensor decompositions are learned in a way that they are optimal for the multi-class classification. When predicting a class label for a new sensor setting $(v_1, ..., v_S)$, which is not included in the training set, the representations for that tuple are indexed and the approximations Φ_1 to Φ_C are computed. Applying the softmax function one obtains the class probabilities for that new sensor setting.

C. Learning Representations

The approximation Φ_c to the class tensors $\mathcal{Y}_c$ can be obtained by any of the tensor decompositions stated in equation 1 to 3 using low ranks for the decompositions. To reduce the number of parameters, representations can be shared between the decompositions of the multiple class tensors. Using the PARAFAC model one can leave only the core vector $\mathbf{g}$ independent for each class and share all representation matrices A_1 to A_S. The same is possible for the Tucker model where all representations can be shared except for the core tensor $\mathcal{G}$. For the Tensor Train model this sharing is not possible, as there is no clear distinction between embedding and fusion components in the factorization. Note that for Tensor Trains, there is an ordering in the dimensions, which interact with each other as next neighbors. As the dimensions in $\mathcal{Y}_c$ represent the sensors in the sensor network, it makes sense to use a neighboring order of the sensors.

By the factorization approach, latent representations are learned for all possible values of all sensors. The multiplicative fusion of these representations enables the modeling of complex sensor interactions. By using low rank decompositions the model learns to generalize, such that predictions can also be performed for new positions in the tensor, which have not been included in the training data.

V. Experiments

In this section we evaluate our method for the task of multi-sensor classification and compare the results against state-of-the-art machine learning algorithms.

A. Dataset

We perform our experiments on a dataset from the game connect-4 from [24] which is transferable to a multi-sensor classification scenario. Consider a game board of size 6×7 where sensors report the occupancy at every position of

TABLE I

ACCURACY AND STANDARD DEVIATION FOR THE CONNECT-4 DATASET

Method	Accuracy
Random Baseline	0.3339 ± 0.0038
Decision Tree	0.7575 ± 0.0036
Logistic Regression	0.7573 ± 0.0021
PCA Logistic Regression	0.7572 ± 0.0022
Support Vector Machine	0.7720 ± 0.0020
Tensor-Fusion with PARAFAC	0.8182 ± 0.0036
Tensor-Fusion with Tensor Train ordered	0.8311 ± 0.0032
Tensor-Fusion with Tensor Train unordered	0.7763 ± 0.0039

the board. The possible values are *player1* if a token of player one is placed, *player2* if a token of player two is placed and *blank* if the field is not occupied. This setting leads to 3^{42} possible board configurations. However, not all configurations appear in practice, as the game ends as soon as one player has four tokens in a row and the game board can only be filled column-wise from bottom to top. For each setting in the game there is a class label which is the game theoretical outcome of the game for the first player. The class label can be either *win*, *loss* or *draw*.

When transferring the game setting to an industrial sensor network it would refer to a network with 42 sensors each reporting discrete values like temperature *low*, *medium* or *high* or any discrete scale. The outcome of the game would relate to the system's overall state which is going to be classified, e.g. *normal*, *incident* or *shut down*.

The whole dataset consists of 67557 game states, where none of the player has won yet and the next move is not forced.[1] We split the dataset into 70 percent training and 30 percent test data. The goal is to classify the outcomes of the game given a specific setting of the board, by using a classification model that has been trained on the training set. Within the training data we hold out 10 percent as validation set to tune the hyperparameters of the models which are described in the next section. The splits are randomly repeated 10 times. We report the mean classification accuracies of all models along with their standard deviations.

B. Methods

We compare the classification results of our models against state-of-the-art machine learning algorithms. The baseline models are decision tree, logistic regression and support vector machine with a linear kernel. For these methods the board setting was encoded in one long feature vector. A standard factorization approach to this kind of data is to factorize the matrix consisting of the feature vectors of the various training examples. Therefore, we performed as a fourth baseline model a Principal Component Analysis (PCA) on that matrix and built a logistic regression model on the data projected to the top principal components. The logistic regression models were regularized using L2-regularization. The amount of regularization and the penalty term for the support vector machine were tuned on the validation set.

[1]https://archive.ics.uci.edu/ml/machine-learning-databases/connect-4/connect-4.names

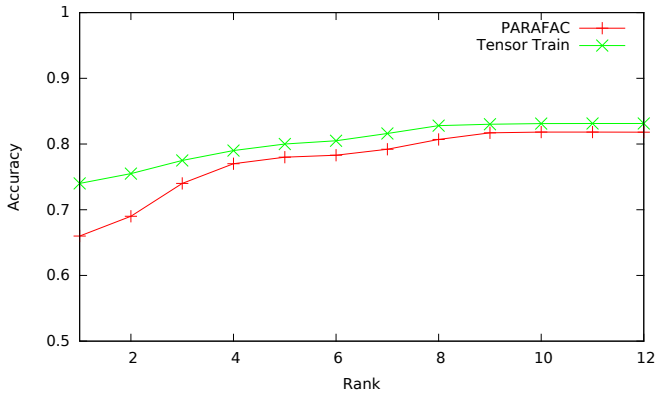

Fig. 3. Classification accuracy in dependency of the embedding rank.

For our method, which we refer to as *tensor-fusion* in the following, we used the PARAFAC and the Tensor Train decomposition. As the dataset has 42 dimensions, the full Tucker decomposition is not tractable. For PARAFAC we do the parameter sharing of the embedding matrices as proposed in section IV-C. For the Tensor Train model we once use a random ordering of the dimensions respectively the sensors, and once a row-wise order. Other space filling curves like z-ordering and Hilbert curve could also be useful for obtaining a one dimensional ordering of the sensors, however we found the row-wise order to perform best for this dataset. The learning rate of stochastic gradient descent and the initialization parameters of the tensor fusion model were tuned on the validation set. For the baseline models the implementations from the python package scikit-learn [25] where used. For the implementation of the tensor models we used the python package theano [26]. We initialized the weights of the PARAFAC decomposition uniformly between 0.9 and 1.1 to yield numerical stable results. The core tensors of the Tensor Train model where initialized with a stack of identity matrices where Gaussian noise was added to each element of the matrix. The noise was parameterized with a standard deviation of 0.01 and mean zero. The entries of the vector embeddings in the first and the last dimension of the Tensor Train where initialized with a uniform distribution between -0.05 and 0.05.

C. Results

Table V-A summarizes the results of the experiments. It shows that our tensor-fusion models clearly outperform the baseline algorithms. The tensor-fusion model with Tensor Train decomposition and row-wise ordered dimensions reaches the highest accuracy with 0.8311, followed by the tensor-fusion model using PARAFAC decomposition with an accuracy of 0.8182. The support vector machine, which is the best baseline approach, reaches 0.7720. Our method used with Tensor Train with a random order of the dimensions performs about as good as the support vector machine (0.7763), however this is much worse than the model with row-wise orders. This might be due to the fact, that the similarities between neighboring dimensions are explicitly modeled in

the Tensor Train decomposition and this might be useful for the game connect-4 as the goal for every player is to build sequences of 4 connected fields. For all decompositions, a rank of 10 was found to be sufficient. Higher ranks did not improve the classification results. Figure 3 shows the results of the factorization models using different ranks. With the rank of 10 it was also not necessary to introduce additional parameter regularization. All baseline models performed best with a one hot encoding of the categorical input features. Due to the multiplicative interactions of latent representations, the tensor-fusion models might be able to model more complex interactions between sensors, if compared to linear models. The PCA for the logistic regression model did not improve the results. With a high number of principal components (top 80%) it performs as good as the regular logistic regression; with fewer principal components the performance decreased.

VI. Conclusion

We have shown how tensor models can be used for multinomial classification in sensor networks with discrete sensor values. Using tensor decompositions, latent representations for the outputs of all sensors are learned. By representing the data in a high dimensional tensor the tensor decompositions become applicable. To obtain a categorical distribution over the class labels describing the overall system state, multiple class tensors are built and decomposed. Representations between the decompositions can be shared for some decompositions. With our classification and training method, any tensor decomposition can be used for multiclass classification. However, not all decompositions scale to the multi-sensor setting. Our method may also be interesting for other domains, where classification tasks are performed given many categorical input variables. Our experimental results show the effectiveness of our method in comparison to state-of-the-art machine learning algorithms, indicating the great potential of tensor models for multi-sensor classification. An important observation from the experiments is that the order of the dimensions plays a critical role when using Tensor Train decomposition. With a proper ordering, the Tensor Train model yields excellent classification results. The PARAFAC model turns out to be a good sensor fusion model due to its ability to share representations across classes and due to its good classification performance. In future work an extension to continues inputs could be realized by explicitly learning mapping functions such as radial basis functions. Also an extension to distributed architectures is conceivable by using distributed tensor factorization techniques.

References

[1] M. Abu Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1996–2018, 2014.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 809–816.

[5] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[6] M. Mørup, L. K. Hansen, C. S. Herrmann, J. Parnas, and S. M. Arnfred, "Parallel factor analysis as an exploratory tool for wavelet transformed event-related eeg," *NeuroImage*, vol. 29, no. 3, pp. 938–947, 2006.

[7] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[8] R. A. Harshman, "Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis," 1970.

[9] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[12] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 601–610.

[13] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction," *arXiv preprint arXiv:1503.00759*, 2015.

[14] A. Cichocki, "Tensor networks for big data analytics and large-scale optimization problems," *arXiv preprint arXiv:1407.3124*, 2014.

[15] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[16] S. V. Dolgov, B. N. Khoromskij, I. V. Oseledets, and D. V. Savostyanov, "Computation of extreme eigenvalues in higher dimensions using block tensor train format," *Computer Physics Communications*, vol. 185, no. 4, pp. 1207–1216, 2014.

[17] T. Rohwedder and A. Uschmajew, "On local convergence of alternating schemes for optimization of convex problems in the tensor train format," *SIAM Journal on Numerical Analysis*, vol. 51, no. 2, pp. 1134–1162, 2013.

[18] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.

[19] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[20] M. Nickel and V. Tresp, "Logistic tensor factorization for multi-relational data," *arXiv preprint arXiv:1306.2084*, 2013.

[21] D. Krompaß, S. Baier, and V. Tresp, "Type-constrained representation learning in knowledge graphs," in *The Semantic Web-ISWC 2015*. Springer, 2015, pp. 640–655.

[22] C. Esteban, D. Schmidt, D. Krompaß, and V. Tresp, "Predicting sequences of clinical events by using a personalized temporal latent embedding model," in *Healthcare Informatics (ICHI), 2015 International Conference on*. IEEE, 2015, pp. 130–139.

[23] C. Esteban, V. Tresp, Y. Yang, S. Baier, and D. Krompaß, "Predicting the co-evolution of event and knowledge graphs," *arXiv preprint arXiv:1512.06900*, 2015.

[24] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688