Tensor Decomposition-Based Training Method for High-Order Hidden Markov Models

Matej Cibul'a , Radek Mařík

Department of Telecommunication Engineering,
Faculty of Electrical Engineering,
Czech Technical University in Prague,
166 27 Prague, Czech Republic
matej.cibula@fel.cvut.cz
marikr@fel.cvut.cz

Abstract: Hidden Markov models (HMMs) are one of the most widely used unsupervised-learning algorithms for modeling discrete sequential data. Traditionally, most of the applications of HMMs have utilized only models of order 1 because higher-order models are computationally hard to train. We reformulate HMMs using tensor decomposition to efficiently build higher-order models with the use of stochastic gradient descent. Based on this, we propose a new modified version of a training algorithm for HMMs, especially suitable for high-order HMMs. Further, we show its capabilities and convergence on synthetic data.

1 Introduction

Hidden Markov models (HMMs) are a popular member of the group of unsupervised-learning algorithms. HMMs have been extensively used in a variety of fields, including bioinformatics[1], speech recognition[2], image processing [3], etc. Nevertheless, most of the works on the topic of HMMs considered only models of an order 1.

Hidden Markov Models of orders higher than 1 have been used very scarcely even though it has been demonstrated that higher-order HMMs provide substantial improvements in the areas of bioinformatics [4], speech recognition [5]) or image processing [6]. Higher-order HMMs can be replaced with a model of an order 1 with with the same amount of free parameters but with considerably higher amount of hidden states [7]. However, increased amount of hidden states obstructs their interpretability. The complexity of parameter estimation of higher-order models has been the main reason for their scarce use.

Traditionally, HMMs are trained using the Baum-Welch algorithm [8] which itself is a particular case of the Expectation-Maximization algorithm. The algorithm requires non-trivial modifications to be applicable to higher-order HMMs and still be able to converge consistently to optima [9]. Another approach is to reformulate higher-order HMM into an HMM of an order 1 [10], [11]. Furthermore, the number of HMM parameters that need to be estimated grows exponentially with the model's order.

Another approach to estimate parameters of an HMM utilizes gradient methods [12]. Traditionally, deterministic gradient descent has been used for training purposes because of the added advantage of online training [13]. Contrary to other training methods, training algorithms utilizing gradient descent can be provided data sequentially, one-by-one, and the complete dataset does not need to be available during the whole training process. With the improvement of stochastic gradient descent (SGD) techniques [14], those have been applied to HMM parameter estimation as well [15]. The SGD approach provides better convergence guarantees and is preferable to its deterministic counterpart.

The least commonly used approach involves Markov Chain Monte Carlo (MCMC) sampling [16]. Its main advantage is in providing a complex distribution of HMM parameters instead of only computing their point estimates. In practice, other techniques are preferred for their lower computational time.

Lastly, the spectral algorithm is the newest one for training HMMs [17]. As opposed to the previously discussed methods, this algorithm optimizes observation-operator representation of HMMs instead of directly modifying HMM parameters. The most significant advantage of this approach is its provable convergence to global optima (under mild assumptions on the model parameters). Another factor is the fact that this is the only non-iterative training algorithm. However, this learning method has only been proposed for HMMs of order 1.

Higher-order models involve the use of tensors instead of matrices. When it comes to use of tensors in practice in general, we often want to find a tensor of fixed rank that approximates the original one. Such approximations have generally advantageous properties when it comes to noise reduction in the original data. From a certain point of view, low-rank tensor approximation is similar to a dimensionality reduction using singular value decomposition for matrices.

The algorithm that we propose combines the modern stochastic gradient descent optimization with tensor decomposition methods. We utilize the Canonical Polyadic

(CP) decomposition [18], hence, we refer to the algorithm as the CP-Decomposition-Stochastic Gradient Descent (CPD-SGD) algorithm.

In the next section, we introduce a notation and provide basic definitions and background to our work. Following, in Section 3, we propose a new method and point out its most essential ideas. We experimentally evaluate our algorithm and discuss the results in Section 4.

1.1 Related Work

The idea of using the stochastic gradient descent for HMM optimization was explored before, for example, in [15], or [19]. However, it has been only applied to HMMs of an order 1, while this work focuses on higher-order models. Furthermore, both works used the traditional representation of an HMM instead of the observation-operator representation utilized in this paper.

The application of tensor decomposition methods to HMMs has been partially explored in the works [9] and [20]. In [20], authors combine the Tensor-Train decomposition [21] with the spectral algorithm for HMMs [17], as opposed to our work where we combine CP-decomposition and SGD. Additionally, the authors do not consider higher-order models, leaving any possible extension in this direction unmentioned.

In [9], the author combine tensor decomposition with Markov chain Monte Carlo methods. However, their model is considerably more complicated and, again, considers only the standard HMM representation. On the other hand, it can straightforwardly provide confidence intervals for estimated parameters instead of just a point estimate.

1.2 Our Contribution

We propose a new method of training hidden Markov models. It is based on recent advancements in the fields of tensor decomposition and stochastic gradient methods for optimization. The most important advantage of the approach is its applicability to the training of HMMs of higher orders. The applicability has been achieved by introducing multiple tricks into the original gradient descent algorithm [12] for the training of HMMs.

Firstly, we introduce a permutation decomposition for transition-probability tensor, which provides possibly even an exponential improvement in the memory required to store those probabilities. We achieve this by clever reuse of vectors representing individual states.

This work focuses solely on the CP-decomposition, although there have been proposed multiple other decompositions [22], such as Tucker decomposition [23], and tensor train (TT) decomposition [21].

Secondly, we modify the operator representation of the HMM (similar to [10]) to allow for the optimization of higher-order models. We use this representation during the stochastic gradient descent instead of the "raw" model that has been commonly used before.

Thirdly, we modify the computation of the observation probabilities given a fixed state by the application of Bayes's theorem and use of stationary distribution of the Markov chains [24]. This modification provides performance improvements, especially in cases where the number of possible different observations is far higher than the number of hidden states.

2 Preliminaries

2.1 Notation

In this work, we follow the generally used notation, based on [25] in a case of tensors, and [8], [17] for Hidden Markov Models. Bold capital letters ($\mathbf{A}$) denote matrices, bold lowercase letters ($\mathbf{a}$) denote vectors and normal lowercase letters (a) denote scalars. Tensors are denoted by uppercase calligraphy letters, such as $\mathscr{A}$. In ambiguous cases, we add an arrow over vectors, for example, $\overrightarrow{\pi}$.

2.2 Tensor Decomposition

Tensors are multidimensional arrays. They can be considered to be a generalization of vectors and matrices into higher dimensions, where vector is a 1st-order tensor and matrix is a 2nd-order tensor, respectively.

The *Kronecker product* of $\mathbf{A} \in \mathbb{R}^{k\times l}$ and $\mathbf{B} \in \mathbb{R}^{m\times n}$ is denoted as $\mathbf{A}\otimes\mathbf{B}$. It applies that $\mathbf{A}\otimes\mathbf{B} \in \mathbb{R}^{km\times ln}$ and it is defined as

$$\mathbf{A}\otimes\mathbf{B} = \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,l}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,l}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1}\mathbf{B} & a_{k,2}\mathbf{B} & \cdots & a_{k,l}\mathbf{B} \end{pmatrix}.$$

The *Khatri-Rao product* is the column-wise Kronecker product. Given $\mathbf{A} \in \mathbb{R}^{k\times m}$ and $\mathbf{B} \in \mathbb{R}^{l\times m}$, their Khatri-Rao product is denoted $\mathbf{A} \bigodot \mathbf{B}$, $\mathbf{A}\odot\mathbf{B} \in \mathbb{R}^{kl\times m}$, and defined as

$$\mathbf{A}\odot\mathbf{B} = [\mathbf{a}_1\otimes\mathbf{b}_1 \quad \ldots \quad \mathbf{a}_m\otimes\mathbf{b}_m].$$

For decomposition algorithms, the operation of *tensor matricization* is commonly used [26]. Tensor matricization, often referred to as reshaping or flattening, is a process of reordering elements of a higher-order (>2) tensor into a matrix. All elements of a tensor must be reused; hence, the amount of memory required to store a tensor is the same as to store its matricized form. The standard forms of matricized tensors are *mode-n unfoldings*.

Let $\mathscr{X} \in \mathbb{R}^{I_1\times\cdots\times I_k}$. If an element in the tensor has indices $(i_1,\ldots,i_k)$, then it will have indices (i_n, j) in its mode-n unfolding, where j is defined as

$$j = 1 + \sum_{\substack{l=1 \\ l\neq n}}^{k} (i_l - 1) J_l, \quad \text{where} \quad J_l = \prod_{\substack{m=1 \\ m\neq n}}^{l-1} I_m.$$

We denote a mode-n unfolding of a tensor $\mathscr{X}$ as $\mathbb{X}_n$.

The *tensor rank* of $\mathscr{X}$, denoted $\texttt{rank}(\mathscr{X})$, is defined as

$$\texttt{rank}(\mathscr{X}) = \min\left\{ r \in \mathbb{N} \,\middle|\, \left(\exists \mathbf{a}_1^1 \dots \mathbf{a}_k^r\right)\left(\mathscr{X} = \sum_{i=1}^{r} \mathbf{a}_1^i \circ \cdots \circ \mathbf{a}_k^i\right)\right\},$$

where the symbol $\circ$ represents the vector outer product.

The above definition of the tensor rank gives rise to the *tensor rank decomposition*. Given a tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ of a rank q, it can be decomposed as

$$\mathscr{X} = \sum_{i=1}^{q} \lambda_i \mathbf{a}_1^i \circ \cdots \circ \mathbf{a}_k^i.$$

Here, $\lambda_i \in \mathbb{R}$ are normalizing coefficients, ensuring that all components are of the same scale while at the same time making the convergence during computation quicker.

Trivially, each tensor has a finite rank. However, estimating the tensor rank is an NP-hard problem [27]. In applications, we often want to find a tensor of fixed rank that approximates the original one.

The rank-$\hat{q}$ *canonical polyadic (CP) decomposition* of a tensor $\mathscr{X}$ of an order k is a problem of finding such vectors $\mathbf{a}_i^j$ that

$$\hat{\mathscr{X}} = \sum_{i=1}^{\hat{q}} \lambda_i \mathbf{a}_1^i \circ \cdots \circ \mathbf{a}_k^i,$$

and $\hat{\mathscr{X}}$ minimizes

$$||\mathscr{X} - \hat{\mathscr{X}}||_{\mathscr{F}}.$$

2.3 Hidden Markov Models

The Hidden Markov Model (HMM) defines a probability distribution over sequences from a finite alphabet. Additionally, it associates each sequence of observations $o_1 \dots o_t$ with a sequence of hidden states of the same length $s_1 \dots s_t$. There is a finite amount of different states. We denote the set of all possible hidden states as $[n] = \{1, \dots, n\}$, and the set of all possible observations, a.k.a. the finite alphabet of sequences, as $[m] = \{1, \dots, m\}$.

It is always assumed that the HMM satisfies 2 conditional independence properties. Firstly, one states that hidden state s_t depends only on a finite amount of previous states, typically only 1, and nothing else. Secondly, one requires that the observation o_t at a time t is only dependent on the hidden state s_t at that given moment t and no other element of the sequence itself or other hidden states.

Formally, the HMM of the order $r-1$ can be described by 3 matrices and/or tensors. Let $\mathscr{T} \in \mathbb{R}^{n \times \cdots \times n}$ be a tensor of state transition probabilities where

$$[\mathscr{T}]_{i_1,\dots,i_r} = \Pr\left[s_t = i_r | s_{t-1} = i_{r-1}, \dots, s_{t-r+1} = i_1\right].$$

Similarly, let $\mathbf{O} \in \mathbb{R}^{n \times m}$ be a matrix of observation probabilities with

$$[\mathbf{O}]_{i,j} = \Pr\left[o_t = j | s_t = i\right].$$

Lastly, let $\pi \in \mathbb{R}^{n \times \cdots \times n}$ be of order 1 less than the tensor $\mathscr{T}$. It defines an initial hidden state distribution by

$$[\pi]_{i_1,\dots,i_{r-1}} = \Pr\left[s_{r-1} = i_{r-1}, \dots, s_1 = i_1\right].$$

In the most common case in the literature, the hidden state at the time t is assumed to only depend on its immediate predecessor. This means that the transition between states can be modeled as a Markov chain of order 1. Consequently, the previously defined parameters are reduced to: $\mathbf{T} \in \mathbb{R}^{n \times n}$, $\mathbf{O} \in \mathbb{R}^{n \times m}$ and $\overrightarrow{\pi} \in \mathbb{R}^n$.

Authors in [28], [29] showed that such HMM of the order 1 can be represented by the means of observation operators. Those operators are a sufficient and computationally efficient way to evaluate observation-sequence probabilities. The probability of a particular sequence $o_1, \dots, o_t$ is given by

$$\Pr\left[o_1, \dots, o_t\right] = \overrightarrow{1}_m \mathbf{A}_{o_t} \dots \mathbf{A}_{o_1} \overrightarrow{\pi},$$

where the observation operators $\mathbf{A}_i$ are defined [17] as

$$\mathbf{A}_i = \mathbf{T}\text{diag}\left([\mathbf{O}]_{1,i}, \dots, [\mathbf{O}]_{n,i}\right).$$

The standard strategy to estimate the HMM parameters is to maximize the provided samples' likelihood. As the hidden states are not directly observable, algorithms often resort to heuristics or other similar methods without any convergence guarantees.

3 CPD-SGD Algorithm

3.1 Permutation Decomposition

We assume that we have a HMM of an order $r-1$ described by parameters $\mathscr{T}, \mathbf{O}, \pi$. The transition-probability tensor $\mathscr{T}$ has a finite tensor rank, let's denote it as q_1. Hence, it has a CP-decomposition

$$\mathscr{T} = \sum_{i=1}^{q_1} \lambda_i \mathbf{t}_1^i \circ \cdots \circ \mathbf{t}_r^i.$$

Further, as we can incorporate the normalizing coefficients λ_i into the individual components, we will omit them in the following, writing only

$$\mathscr{T} = \sum_{i=1}^{q_1} \mathbf{t}_1^i \circ \cdots \circ \mathbf{t}_r^i.$$

This omission is based on two facts. Firstly, all of the $\mathbf{t}_j^i$ vectors will be inferred during the training of the model. Secondly, we do not need to constrain decomposition components into a certain form, as we do not work with them directly and the tensor approximation is invariant to the way they are incorporated into decomposition components. Hence, omitting of λ_i coefficients results in a lower number of parameters to be estimated.

Every hidden state is now represented by r vectors of length q_1. Each one of them is for one position relative to the current state, a.k.a. the first one defines the influence of the given state on the state observed $r-1$ steps in the future from it. The next to last vector defines the influence of the given state on the following one, and the last vector defines the behavior of the current state based on history. For a fixed hidden state i and for $j \in \{1,\dots,r\}$, those vectors have a form

$$\left[\left[\mathbf{t}_j^1\right]_i \left[\mathbf{t}_j^2\right]_i \cdots \left[\mathbf{t}_j^{q_1}\right]_i\right]^{\mathrm{T}}.$$

For the purposes of interpretability, we would like to represent each state by only one vector. Trivial solution would be to just concatenate them. Another option is to replace all of those r vectors with just one. If we did this, our model would be able to recognize previous states, however, it would not be able to distinguish their combinations (order in time). For example, state 1 followed by state 2 would be represented the same way as state 2 followed by state 1.

Based on the thoughts from the previous paragraph, we propose a solution in the following form: we represent each hidden states by a vector of a fixed length, and depending on their relative position in time we permute those vectors. More precisely, we shift their elements by one for each time-step into the past, eg. a vector $[1 \quad 2 \quad 3]^{\mathrm{T}}$ turns into $[2 \quad 3 \quad 1]^{\mathrm{T}}$ and then into $[3 \quad 1 \quad 2]^{\mathrm{T}}$.

Such a replacement vector may need to have higher dimension, worst-case it needs to be of length rq_1, although we will be interested only in a low-rank approximation of the tensor $\mathscr{T}$. Hence, we introduce a new parameter for our algorithm, q_2, which denotes the length of those vectors (we will refer to them as state vectors). The q_2 acts as a hyperparameter that needs to be set at the beginning of an experiment. Its choice is beyond this publication, however, it should be considerably higher than q_1. With those modifications, we can rewrite the decomposition as

$$\mathscr{T} = \sum_{i=1}^{q_2} \mathbf{t}^{i+r-1} \circ \cdots \circ \mathbf{t}^{i+1} \circ \mathbf{t}^i,$$

where $\mathbf{t}^k = \mathbf{t}^j$ if $j = k \mod q_2$.

We apply the same trick to the initial-state-probability tensor π and the observation-probability matrix $\mathbf{O}$,

$$\mathbf{O} = \sum_{i=1}^{q_2} \mathbf{t}^{i+r} \circ \mathbf{u}^i,$$

where $\mathbf{u}^i$ represent observations.

3.2 Matricized Representation

The observation-operator representation of HMM is advantageous during training, as it involves only matrix-vector products. Unfortunately, this representation only works for HMMs of order 1, whose transition probabilities can be represented as a matrix. To alleviate this problem, we introduce matricized tensor using the representation from the previous subsection.

Plainly, instead of using transition probability tensor $\mathscr{T} \in \mathbb{R}^{n\times\cdots\times n}$, we use its matricized form $\mathbf{T} \in \mathbb{R}^{n^{r-1}\times n}$.

Using the previous notation, let a matrix $\mathbf{H} \in \mathbb{R}^{n\times q_2}$ be defined as

$$\mathbf{H} = \begin{bmatrix}\mathbf{t}^1 & \dots & \mathbf{t}^{q_2}\end{bmatrix},$$

where $\mathbf{t}^i$ are individual columns of the matrix.

This matrix represents behaviour of states at the time of current observation. To get the matrix for a time step -1, we multiply it by a permutation matrix $\mathbf{P} \in \mathbb{R}^{q_2\times q_2}$,

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

Furthermore, we define a matrix $\mathbf{S} \in \mathbb{R}^{n^{r-1}\times q_2}$ as

$$\mathbf{S} = \mathbf{H}\mathbf{P}^{r-1} \odot \cdots \odot \mathbf{H}\mathbf{P}^{\mathbf{2}} \odot \mathbf{H}\mathbf{P}.$$

It is obvious, that we get the mode-r unfolding of the tensor $\mathscr{T}$ by multiplying it with $\mathbf{H}^{\mathrm{T}}$,

$$\mathbf{T} = \mathbf{S}\mathbf{H}^{\mathrm{T}}.$$

For the direct replacement in the operator representation, a square matrix is necessary. We do this by imputing zeros into matrix $\mathbf{T}$, creating a sparse matrix. This is a consequence of past hidden states being already given, hence, reducing the degree of freedom of the system. By exploiting the structure created by Khatri-Rao product, each row in the imputed matrix needs to be padded to the left. The i-th row is padded by $m(i\%r)$ to the left, creating a matrix $\mathbf{T}_0$.

3.3 Unconstrained Optimization

To ensure that probabilities will not be negative and that they will add up to 1, we would need to solve a constrained optimization problem. To avoid this complication, we instead apply a row-wise softmax function to $\mathbf{T}_0$. For a general vector $\mathbf{x}$, the softmax function is defined as

$$\left[\mathrm{softmax}\left(\mathbf{x}\right)\right]_i = \frac{e^{x_i}}{\sum_{j=0}^{m^r} e^{x_j}}.$$

Thanks to this, we can optimize this problem as an unconstrained one.

The same trick can be applied to the matrix $\mathbf{O}$. The estimation of $\mathbf{O}$, however, requires the application of the softmax function to each row, where rows are typically of considerably higher dimension than columns. In those cases where there are tens of thousands or even more different possible observations, this computation would be prohibitively computationally expensive. Instead, we would like to estimate only columns. A solution to this problem is sketched in the following subsection.

3.4 Probability Replacement

To compute the probabilities $[\mathbf{O}]_{i,j} = \Pr[o_t = j | s_t = i]$, we would need to use our decomposed representation and compute a whole row $[\mathbf{O}]_i$. However, this row contains m elements, usually far more than the amount of hidden states n. Instead, we use the Bayes' theorem to compute the following formula

$$\Pr[o_t = j | s_t = i] = \frac{\Pr[s_t = i | o_t = j] \Pr[o_t = j]}{\Pr[s_t = i]}.$$

Here, the $\Pr[o_t = j]$ is independent of hidden states and cannot be optimized, which means we can remove it from the expression during training as it does not provide any valuable gradients.

The estimation of $\Pr[s_t = i | o_t = j]$ requires the estimation of m probabilities, one for each hidden state, which is computationally more feasible than previous approach. We will denote this modified matrix as $\mathbf{O}_{\mathrm{T}}$.

Lastly, the expression $\Pr[s_t = i]$ is from a stationary distribution of the Markov chain represented by the hidden-state transitions $\mathbf{T}_0$. It can be calculated by solving a linear system of equations which is a completely differentiable process. We denote its solution as $\mathbf{z}$,

$$[\mathbf{z}]_i = \Pr[s_t = i].$$

Furthermore, the estimation of $\mathbf{z}$ needs to be done only once during each batch which further improves performance.

3.5 The CPD-SGD algorithm

The pseudocode of the complete algorithm is presented in Algorithm 1. We formulate the algorithm using the automatic differentiation framework. Hence, the exact update rules for individual parameters are determined based on the chosen optimizer.

The optimizer can be chosen based on the concrete task or other requirements. It can be an SGD with or without adaptive learning rate. We choose the later option in this work. Another option is to utilize one of the momentum-based methods, such as Adam [30]. In any case, the update of the optimizer parameters is done within the body of the function `update_params_based_on()` in the Algorithm 1.

The computation is organized into epochs that repeat until the model has converged. We assume that the $\overrightarrow{\pi}$ is a uniform distribution, hence we replaced it in the algorithm with the vector of ones. The algorithm evaluates the log-likelihood of the given sequence or sequences during each epoch and computes their respective gradients. Hence, the algorithm can be used in both online and batch settings. Afterward, the parameters are updated based on the chosen optimizer. The use of log-likelihood is necessary as otherwise, tiny probabilities may cause computational errors/underflows.

Algorithm 1: The CPD-SGD Algorithm

Result: $\mathbf{H}, \mathbf{O}_{\mathrm{T}}$
initialization
ln_prob $\leftarrow 0$
while not_converged() **do**
  $\mathbf{S} \leftarrow \mathbf{HP^{r-1}} \odot \cdots \odot \mathbf{HP^2} \odot \mathbf{HP}$
  $\mathbf{T} \leftarrow \mathrm{softmax}(\mathbf{SH}^{\mathrm{T}})$
  $\mathbf{z} \leftarrow \mathrm{solve}(\mathbf{T})$
  $\mathbf{T} \leftarrow \mathbf{T}\mathrm{diag}(\mathbf{z})$
  $\mathbf{T}_0 \leftarrow$ add_zeros_and_shift($\mathbf{T}$)
  $\mathbf{O}_{\mathrm{T}} \leftarrow$ eval_cols($x_k \ldots x_1$)
  forall *sequences* $x_t \ldots x_1$ *in current batch* **do**
    ln_prob $\leftarrow$
    ln_prob $- \ln\left(\overrightarrow{1}^{\mathrm{T}} \ldots \mathbf{T}_0 \mathrm{diag}([\mathbf{O}_{\mathrm{T}}]_{\bullet, x_1}) \overrightarrow{1}\right)$
  end
  update_params_based_on(ln_prob)
  ln_prob $\leftarrow 0$
end

4 Experiments

We experimentally evaluated the proposed CPD-SGD training algorithm that was presented in Section 3. To do this, we generated a synthetic dataset. The synthetic dataset was generated to resemble natural language properties and to emphasize the advantages of higher-order HMMs. To evaluate and compare trained models, we compute the likelihood of the test sequences for each model. The focus of this work is to provide a proof of concept of the proposed algorithm, hence, further experiments and applications to disjoint datasets are relegated to the future work.

4.1 Synthetic Dataset

We generate the synthetic dataset as an output from an HMM model of an order 5 with 20 hidden states. The amount of possible distinct observations is equal to 10 000. This model was created in such a way as to be close to but not completely deterministic. The generated sequences are all of length 50, as this is long enough to alleviate the influence of the initial-states distribution and short enough not to cause an arithmetic underflow. The initial state probabilities of the model were chosen as a stationary distribution of its underlying Markov chain.

4.2 Convergence

To evaluate the convergence of the CPD-SGD algorithm, we divide the dataset into two disjoint sets: training data and test data. While training data is used to estimate parameters of the HMM, test data is used to estimate the model's fit.

In the first experiment, we hypothesize an HMM model of an order 3, represented by state vectors of length 50.

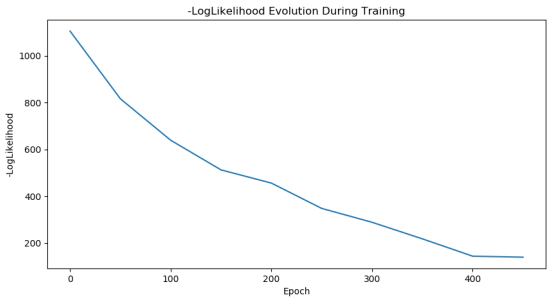

Figure 1: A continuous convergence of the CPD-SGD algorithm.

The number of hidden states was chosen to be 20. Figure 1 shows the evolution of the model's fit after each epoch (an epoch consisted of 100 sequences). It can be seen that the parameters of the model are being optimized.

We follow by assessing the convergence of our proposed algorithm in comparison to the Baum-Welch algorithm [8] for models of order 1. We use a set of sequences to estimate parameters of HMMs, and afterward, we compute each model's fit using a different set of test sequences. We repeat this process 500 times, each time with differently reorganized sequences into train and test sets. Furthermore, we always use 10 different initializations for the Baum-Welch algorithm, train all of them, and choose the model with the best fit.

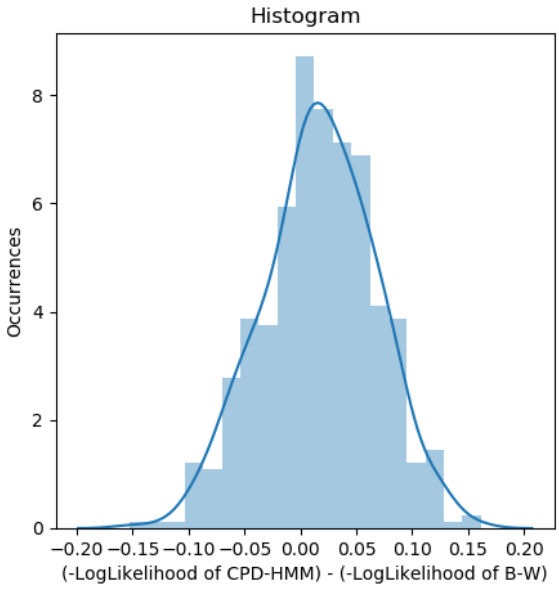

Figure 2: A distribution of differences between log-likelihoods of models trained by Baum-Welch and CPD-SGD algorithms.

The distribution of differences between fits of both of those algorithms can be seen in Figure 2. As apparent from the figure, our algorithm attains results comparable to traditional approaches with low differences in absolute and relative likelihood differences.

4.3 Model Comparisons

In the final experiment, we compare the expressive power of the higher-order models. We train models of orders from 1 up to 5, using the same train set. After they converge, we compute their fit on the test sequences. All models were trained using the CPD-SGD algorithm, with the number of hidden states equal to 20, and with state vectors of length 50. They were randomly initialized.

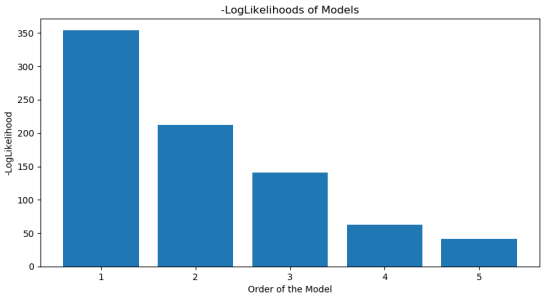

Figure 3: Maximal attained log-likelihoods of models of given order.

Figure 3 shows individual fits of those models. As expected, the higher-order models perform better.

5 Conclusion

In this work, we build upon one of the well-known unsupervised learning methods - hidden Markov models. We explored the possibilities to improve the training of HMMs of higher orders. We followed up on previous works in the area and proposed a new algorithm for HMM training, denoted as the CPD-SGD algorithm.

Our algorithm is based on multiple ideas that stem from - among others - an emerging area of tensor decomposition methods. We utilize permutation decomposition which assigns only one vector to each state while it reuses all of its values for each time step. Further, we used matricized representation of the probability tensor to extend the operator representation of HMMs to higher-order models. By applying the softmax function, we avoided the problematic constrained optimization. Finally, we reformulated computation of observation probabilities, making them more efficient to compute in case of models with observation space far larger than state space.

We experimentally showed that higher-order HMMs could be efficiently represented in memory as decomposed tensors while being suitable for accelerated training using stochastic gradient descent optimization.

The focus of this paper has been on constructing an algorithm for training higher-order HMMs in an efficient

way. We have experimentally verified that our method works and is able to train an HMM model on a synthetic dataset accurately.

In future work, we plan to apply this method to real-world problems in natural language processing, which requires models with large state spaces and observation spaces.

References

[1] Byung-Jun Yoon. Hidden markov models and their applications in biological sequence analysis. *Current Genomics*, 10:402 – 415, 2009.

[2] M. Gales and S. Young. The application of hidden markov models in speech recognition. *Found. Trends Signal Process.*, 1:195–304, 2007.

[3] J. Bobulski and Lukasz Adrjanowicz. Two-dimensional hidden markov models for pattern recognition. In *ICAISC*, 2013.

[4] M. Seifert, K. Abou-El-Ardat, Betty Friedrich, B. Klink, and A. Deutsch. Autoregressive higher-order hidden markov models: Exploiting local chromosomal dependencies in the analysis of tumor expression profiles. *PLoS ONE*, 9, 2014.

[5] Lee-Min Lee and Jia-Chien Lee. A study on high-order hidden markov models and applications to speech recognition. In *IEA/AIE*, 2006.

[6] L. Benyoussef, C. Carincotte, and S. Derrode. Extension of higher-order hmc modeling with application to image segmentation. *Digit. Signal Process.*, 18:849–860, 2008.

[7] W. Zucchini and I. Macdonald. Hidden markov models for time series: An introduction using r. 2009.

[8] L. Rabiner. A tutorial on hidden markov models and selected applications. *Proceedings of the IEEE*, 1989.

[9] Abhra Sarkar and D. Dunson. Bayesian nonparametric higher order hidden markov models. *arXiv: Methodology*, 2018.

[10] C. Xiong, Di Yang, and L. Zhang. A high-order hidden markov model and its applications for dynamic car ownership analysis. *Transp. Sci.*, 52:1365–1375, 2018.

[11] U. Hadar and H. Messer. High-order hidden markov models - estimation and implementation. *2009 IEEE/SP 15th Workshop on Statistical Signal Processing*, pages 249–252, 2009.

[12] P. Bagos, T. Liakopoulos, and S. Hamodrakas. Faster gradient descent training of hidden markov models, using individual learning rate adaptation. In *ICGI*, 2004.

[13] P. Baldi and Y. Chauvin. Smooth on-line learning algorithms for hidden markov models. *Neural Computation*, 6:307–318, 1994.

[14] Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, 2012.

[15] Ahmed Saaudi, Yan Tong, and Csilla Farkas. Probabilistic graphical model on detecting insiders: Modeling with sgd-hmm. In *ICISSP*, 2019.

[16] Md Pavel Mahmud and Alexander Schliep. Fast mcmc sampling for hidden markov models to determine copy number variations. *BMC Bioinformatics*, 12:428 – 428, 2011.

[17] Daniel J. Hsu, S. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *J. Comput. Syst. Sci.*, 78:1460–1480, 2009.

[18] R. Harshman. Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-model factor analysis. 1970.

[19] Yian Ma, Y. Ma, N. Foti, and E. Fox. Stochastic gradient mcmc methods for hidden markov models. In *ICML*, 2017.

[20] Maxim A. Kuznetsov and I. Oseledets. Tensor train spectral method for learning of hidden markov models (hmm). *Computational Methods in Applied Mathematics*, 19:93 – 99, 2019.

[21] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[22] Majid Janzamin, Rong Ge, Jean Kossaifi, and Anima Anandkumar. Spectral learning on matrices and tensors. *Found. Trends Mach. Learn.*, 12:393–536, 2019.

[23] LR Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279—311, September 1966.

[24] P. Brémaud. Non-homogeneous markov chains. 2020.

[25] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.

[26] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *ArXiv*, abs/1711.10781, 2017.

[27] Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *J. ACM*, 60(6), November 2013.

[28] Eyal Even-Dar, S. Kakade, and Y. Mansour. The value of observation for monitoring dynamic systems. In *IJCAI*, 2007.

[29] H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12:1371–1398, 2000.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.