Online Learning in Tensor Space

Yuan Cao **Sanjeev Khudanpur**
Center for Language & Speech Processing and Human Language Technology Center of Excellence
The Johns Hopkins University
Baltimore, MD, USA, 21218
{yuan.cao, khudanpur}@jhu.edu

Abstract

We propose an online learning algorithm based on tensor-space models. A tensor-space model represents data in a compact way, and via rank-1 approximation the weight tensor can be made highly structured, resulting in a significantly smaller number of free parameters to be estimated than in comparable vector-space models. This regularizes the model complexity and makes the tensor model highly effective in situations where a large feature set is defined but very limited resources are available for training. We apply with the proposed algorithm to a parsing task, and show that even with very little training data the learning algorithm based on a tensor model performs well, and gives significantly better results than standard learning algorithms based on traditional vector-space models.

1 Introduction

Many NLP applications use models that try to incorporate a large number of linguistic features so that as much human knowledge of language can be brought to bear on the (prediction) task as possible. This also makes training the model parameters a challenging problem, since the amount of labeled training data is usually small compared to the size of feature sets: the feature weights cannot be estimated reliably.

Most traditional models are linear models, in the sense that both the features of the data and model parameters are represented as vectors in a vector space. Many learning algorithms applied to NLP problems, such as the Perceptron (Collins, 2002), MIRA (Crammer et al., 2006; McDonald et al., 2005; Chiang et al., 2008), PRO (Hopkins and May, 2011), RAMPION (Gimpel and Smith, 2012) etc., are based on vector-space models. Such models require learning individual feature weights directly, so that the number of parameters to be estimated is identical to the size of the feature set. When millions of features are used but the amount of labeled data is limited, it can be difficult to precisely estimate each feature weight.

In this paper, we shift the model from vector-space to tensor-space. Data can be represented in a compact and structured way using tensors as containers. Tensor representations have been applied to computer vision problems (Hazan et al., 2005; Shashua and Hazan, 2005) and information retrieval (Cai et al., 2006a) a long time ago. More recently, it has also been applied to parsing (Cohen and Collins, 2012; Cohen and Satta, 2013) and semantic analysis (Van de Cruys et al., 2013). A linear tensor model represents both features and weights in tensor-space, hence the weight tensor can be factorized and approximated by a linear sum of rank-1 tensors. This low-rank approximation imposes structural constraints on the feature weights and can be regarded as a form of regularization. With this representation, we no longer need to estimate individual feature weights directly but only a small number of "bases" instead. This property makes the the tensor model very effective when training a large number of feature weights in a low-resource environment. On the other hand, tensor models have many more degrees of "design freedom" than vector space models. While this makes them very flexible, it also creates much difficulty in designing an optimal tensor structure for a given training set.

We give detailed description of the tensor space

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 666–675,
Baltimore, Maryland, USA, June 23-25 2014. ©2014 Association for Computational Linguistics

model in Section 2. Several issues that come with the tensor model construction are addressed in Section 3. A tensor weight learning algorithm is then proposed in 4. Finally we give our experimental results on a parsing task and analysis in Section 5.

2 Tensor Space Representation

Most of the learning algorithms for NLP problems are based on vector space models, which represent data as vectors $\boldsymbol{\phi} \in \mathbb{R}^n$, and try to learn feature weight vectors $\boldsymbol{w} \in \mathbb{R}^n$ such that a linear model $y = \boldsymbol{w} \cdot \boldsymbol{\phi}$ is able to discriminate between, say, good and bad hypotheses. While this is a natural way of representing data, it is not the only choice. Below, we reformulate the model from vector to tensor space.

2.1 Tensor Space Model

A tensor is a multidimensional array, and is a generalization of commonly used algebraic objects such as vectors and matrices. Specifically, a vector is a 1^{st} order tensor, a matrix is a 2^{nd} order tensor, and data organized as a rectangular cuboid is a 3^{rd} order tensor etc. In general, a D^{th} order tensor is represented as $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$, and an entry in $\mathcal{T}$ is denoted by $\mathcal{T}_{i_1,i_2,\ldots,i_D}$. Different dimensions of a tensor $1, 2, \ldots, D$ are named modes of the tensor.

Using a D^{th} order tensor as container, we can assign each feature of the task a D-dimensional index in the tensor and represent the data as tensors. Of course, shifting from a vector to a tensor representation entails several additional degrees of freedom, e.g., the order D of the tensor and the sizes $\{n_d\}_{d=1}^{D}$ of the modes, which must be addressed when selecting a tensor model. This will be done in Section 3.

2.2 Tensor Decomposition

Just as a matrix can be decomposed as a linear combination of several rank-1 matrices via SVD, tensors also admit decompositions[1] into linear combinations of "rank-1" tensors. A D^{th} order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$ is rank-1 if it can be written as the outer product of D vectors, i.e.

$$\mathcal{A} = \mathbf{a}^1 \otimes \mathbf{a}^2 \otimes, \ldots, \otimes \mathbf{a}^D,$$

where $\mathbf{a}^i \in \mathbb{R}^{n_d}, 1 \leq d \leq D$. A D^{th} order tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$ can be factorized into a sum of component rank-1 tensors as

$$\mathcal{T} = \sum_{r=1}^{R} \mathcal{A}_r = \sum_{r=1}^{R} \mathbf{a}_r^1 \otimes \mathbf{a}_r^2 \otimes, \ldots, \otimes \mathbf{a}_r^D$$

where R, called the rank of the tensor, is the minimum number of rank-1 tensors whose sum equals $\mathcal{T}$. Via decomposition, one may approximate a tensor by the sum of H major rank-1 tensors with $H \leq R$.

2.3 Linear Tensor Model

In tensor space, a linear model may be written (ignoring a bias term) as

$$f(\boldsymbol{W}) = \boldsymbol{W} \circ \boldsymbol{\Phi},$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$ is the feature tensor, $\boldsymbol{W}$ is the corresponding weight tensor, and $\circ$ denotes the Hadamard product. If $\boldsymbol{W}$ is further decomposed as the sum of H major component rank-1 tensors, i.e. $\boldsymbol{W} \approx \sum_{h=1}^{H} \boldsymbol{w}_h^1 \otimes \boldsymbol{w}_h^2 \otimes, \ldots, \otimes \boldsymbol{w}_h^D$, then

$$\begin{aligned} & f(\boldsymbol{w}_1^1, \ldots, \boldsymbol{w}_1^D, \ldots, \boldsymbol{w}_h^1, \ldots, \boldsymbol{w}_h^D) \\ &= \sum_{h=1}^{H} \boldsymbol{\Phi} \times_1 \boldsymbol{w}_h^1 \times_2 \boldsymbol{w}_h^2 \ldots \times_D \boldsymbol{w}_h^D, \quad (1) \end{aligned}$$

where $\times_l$ is the l-mode product operator between a D^{th} order tensor $\mathcal{T}$ and a vector $\mathbf{a}$ of dimension n_d, yielding a $(D-1)^{\text{th}}$ order tensor such that

$$\begin{aligned} & (\mathcal{T} \times_l \mathbf{a})_{i_1,\ldots,i_{l-1},i_{l+1},\ldots,i_D} \\ &= \sum_{i_l=1}^{n_d} \mathcal{T}_{i_1,\ldots,i_{l-1},i_l,i_{l+1},\ldots,i_D} \cdot a_{i_l}. \end{aligned}$$

The linear tensor model is illustrated in Figure 1.

2.4 Why Learning in Tensor Space?

So what is the advantage of learning with a tensor model instead of a vector model? Consider the case where we have defined 1,000,000 features for our task. A vector space linear model requires estimating 1,000,000 free parameters. However if we use a 2^{nd} order tensor model, organize the features into a 1000×1000 matrix $\boldsymbol{\Phi}$, and use just

[1] The form of tensor decomposition defined here is named as CANDECOMP/PARAFAC(CP) decomposition (Kolda and Bader, 2009). Another popular form of tensor decomposition is called Tucker decomposition, which decomposes a tensor into a core tensor multiplied by a matrix along each mode. We focus only on the CP decomposition in this paper.

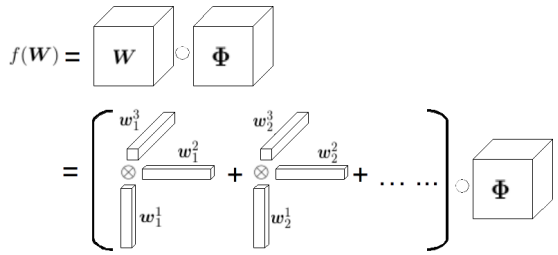


Figure 1: A 3^{rd} order linear tensor model. The feature weight tensor $\boldsymbol{W}$ can be decomposed as the sum of a sequence of rank-1 component tensors.

one rank-1 matrix to approximate the weight tensor, then the linear model becomes

$$f(\boldsymbol{w}_1, \boldsymbol{w}_2) = \boldsymbol{w}_1^T \boldsymbol{\Phi} \boldsymbol{w}_2,$$

where $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^{1000}$. That is to say, now we only need to estimate 2000 parameters!

In general, if V features are defined for a learning problem, and we (i) organize the feature set as a tensor $\boldsymbol{\Phi} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$ and (ii) use H component rank-1 tensors to approximate the corresponding target weight tensor. Then the total number of parameters to be learned for this tensor model is $H \sum_{d=1}^{D} n_d$, which is usually much smaller than $V = \prod_{d=1}^{D} n_d$ for a traditional vector space model. Therefore we expect the tensor model to be more effective in a low-resource training environment.

Specifically, a vector space model assumes each feature weight to be a "free" parameter, and estimating them reliably could therefore be hard when training data are not sufficient or the feature set is huge. By contrast, a linear tensor model only needs to learn $H \sum_{d=1}^{D} n_d$ "bases" of the m feature weights instead of individual weights directly. The weight corresponding to the feature $\Phi_{i_1, i_2, \ldots, i_D}$ in the tensor model is expressed as

$$w_{i_1, i_2, \ldots, i_D} = \sum_{h=1}^{H} w_{h,i_1}^1 w_{h,i_2}^2 \ldots w_{h,i_D}^D, \quad (2)$$

where w_{h,i_j}^j is the i_j^{th} element in the vector $\boldsymbol{w}_h^j$.

In other words, a true feature weight is now approximated by a set of bases. This reminds us of the well-known low-rank matrix approximation of images via SVD, and we are applying similar techniques to approximate target feature weights, which is made possible only after we shift from vector to tensor space models.

This approximation can be treated as a form of model regularization, since the weight tensor is represented in a constrained form and made highly structured via the rank-1 tensor approximation. Of course, as we reduce the model complexity, e.g. by choosing a smaller and smaller H, the model's expressive ability is weakened at the same time. We will elaborate on this point in Section 3.1.

3 Tensor Model Construction

To apply a tensor model, we first need to convert the feature vector into a tensor $\boldsymbol{\Phi}$. Once the structure of $\boldsymbol{\Phi}$ is determined, the structure of $\boldsymbol{W}$ is fixed as well. As mentioned in Section 2.1, a tensor model has many more degrees of "design freedom" than a vector model, which makes the problem of finding a good tensor structure a non-trivial one.

3.1 Tensor Order

The order of a tensor affects the model in two ways: the expressiveness of the model and the number of parameters to be estimated. We assume $H = 1$ in the analysis below, noting that one can always add as many rank-1 component tensors as needed to approximate a tensor with arbitrary precision.

Obviously, the 1^{st} order tensor (vector) model is the most expressive, since it is structureless and any arbitrary set of numbers can always be represented exactly as a vector. The 2^{nd} order rank-1 tensor (rank-1 matrix) is less expressive because not every set of numbers can be organized into a rank-1 matrix. In general, a D^{th} order rank-1 tensor is more expressive than a $(D+1)^{\text{th}}$ order rank-1 tensor, as a lower-order tensor imposes less structural constraints on the set of numbers it can express. We formally state this fact as follows:

Theorem 1. *A set of real numbers that can be represented by a $(D+1)^{\text{th}}$ order tensor $\mathcal{Q}$ can also be represented by a D^{th} order tensor $\mathcal{P}$, provided $\mathcal{P}$ and $\mathcal{Q}$ have the same volume. But the reverse is not true.*

Proof. See appendix. □

On the other hand, tensor order also affects the number of parameters to be trained. Assuming that a D^{th} order has equal size on each mode (we will elaborate on this point in Section 3.2) and the volume (number of entries) of the tensor is fixed as V, then the total number of parameters

of the model is $DV^{\frac{1}{D}}$. This is a convex function of D, and the minimum[2] is reached at either $D^* = \lfloor \ln V \rfloor$ or $D^* = \lceil \ln V \rceil$.

Therefore, as D increases from 1 to D^*, we lose more and more of the expressive power of the model but reduce the number of parameters to be trained. However it would be a bad idea to choose a D beyond D^*. The optimal tensor order depends on the nature of the actual problem, and we tune this hyper-parameter on a held-out set.

3.2 Mode Size

The size n_d of each tensor mode, $d = 1, \ldots, D$, determines the structure of feature weights a tensor model can precisely represent, as well as the number of parameters to estimate (we also assume $H = 1$ in the analysis below). For example, if the tensor order is 2 and the volume V is 12, then we can either choose $n_1 = 3, n_2 = 4$ or $n_1 = 2, n_2 = 6$. For $n_1 = 3, n_2 = 4$, the numbers that can be precisely represented are divided into 3 groups, each having 4 numbers, that are scaled versions of one another. Similarly for $n_1 = 2, n_2 = 6$, the numbers can be divided into 2 groups with different scales. Obviously, the two possible choices of (n_1, n_2) also lead to different numbers of free parameters (7 vs. 8).

Given D and V, there are many possible combinations of $n_d, d = 1, \ldots, D$, and the optimal combination should indeed be determined by the structure of target features weights. However it is hard to know the structure of target feature weights before learning, and it would be impractical to try every possible combination of mode sizes, therefore we choose the criterion of determining the mode sizes as minimization of the total number of parameters, namely we solve the problem:

$$\min_{n_1,\ldots,n_D} \sum_{d=1}^{D} n_d \quad s.t \quad \prod_{d=1}^{D} n_d = V$$

The optimal solution is reached when $n_1 = n_2 = \ldots = n_D = V^{\frac{1}{D}}$. Of course it is not guaranteed that $V^{\frac{1}{D}}$ is an integer, therefore we choose $n_d = \lfloor V^{\frac{1}{D}} \rfloor$ or $\lceil V^{\frac{1}{D}} \rceil, d = 1, \ldots, D$ such that $\prod_{d=1}^{D} n_d \geq V$ and $\left[\prod_{d=1}^{D} n_d\right] - V$ is minimized. The $\left[\prod_{d=1}^{D} n_d\right] - V$ extra entries of the tensor correspond to no features and are used just for padding. Since for each n_d there are only two possible values to choose, we can simply enumerate all the possible 2^D (which is usually a small number) combinations of values and pick the one that matches the conditions given above. This way $n_1, \ldots, n_D$ are fully determined.

Here we are only following the principle of minimizing the parameter number. While this strategy might work well with small amount of training data, it is not guaranteed to be the best strategy in all cases, especially when more data is available we might want to increase the number of parameters, making the model more complex so that the data can be more precisely modeled. Ideally the mode size needs to be adaptive to the amount of training data as well as the property of target weights. A theoretically guaranteed optimal approach to determining the mode sizes remains an open problem, and will be explored in our future work.

3.3 Number of Rank-1 Tensors

The impact of using $H > 1$ rank-1 tensors is obvious: a larger H increases the model complexity and makes the model more expressive, since we are able to approximate target weight tensor with smaller error. As a trade-off, the number of parameters and training complexity will be increased. To find out the optimal value of H for a given problem, we tune this hyper-parameter too on a held-out set.

3.4 Vector to Tensor Mapping

Finally, we need to find a way to map the original feature vector to a tensor, i.e. to associate each feature with an index in the tensor. Assuming the tensor volume V is the same as the number of features, then there are in all $V!$ ways of mapping, which is an intractable number of possibilities even for modest sized feature sets, making it impractical to carry out a brute force search. However while we are doing the mapping, we hope to arrange the features in a way such that the corresponding target weight tensor has approximately a low-rank structure, this way it can be well approximated by very few component rank-1 tensors.

Unfortunately we have no knowledge about the target weights in advance, since that is what we need to learn after all. As a way out, we first run a simple vector-model based learning algorithm (say the Perceptron) on the training data and estimate a weight vector, which serves as a "surro-

[2]The optimal integer solution can be determined simply by comparing the two function values.

gate" weight vector. We then use this surrogate vector to guide the design of the mapping. Ideally we hope to find a permutation of the surrogate weights to map to a tensor in such a way that the tensor has a rank as low as possible. However matrix rank minimization is in general a hard problem (Fazel, 2002). Therefore, we follow an approximate algorithm given in Figure 2a, whose main idea is illustrated via an example in Figure 2b.

Basically, what the algorithm does is to divide the surrogate weights into hierarchical groups such that groups on the same level are approximately proportional to each other. Using these groups as units we are able to "fill" the tensor in a hierarchical way. The resulting tensor will have an approximate low-rank structure, provided that the sorted feature weights have roughly group-wise proportional relations.

For comparison, we also experimented a trivial solution which maps each entry of the feature tensor to the tensor just in sequential order, namely ϕ_0 is mapped to $\Phi_{0,0,\ldots,0}$, ϕ_1 is mapped to $\Phi_{0,0,\ldots,1}$ etc. This of course ignores correlation between features since the original feature order in the vector could be totally meaningless, and this strategy is not expected to be a good solution for vector to tensor mapping.

4 Online Learning Algorithm

We now turn to the problem of learning the feature weight tensor. Here we propose an online learning algorithm similar to MIRA but modified to accommodate tensor models.

Let the model be $f(\boldsymbol{T}) = \boldsymbol{T} \circ \boldsymbol{\Phi}(x, y)$, where $\boldsymbol{T} = \sum_{h=1}^{H} \boldsymbol{w}_h^1 \otimes \boldsymbol{w}_h^2 \otimes, \ldots, \otimes \boldsymbol{w}_h^D$ is the weight tensor, $\boldsymbol{\Phi}(x, y)$ is the feature tensor for an input-output pair (x, y). Training samples $(x_i, y_i), i = 1, \ldots, m$, where x_i is the input and y_i is the reference or oracle hypothesis, are fed to the weight learning algorithm in sequential order. A prediction z_t is made by the model T_t at time t from a set of candidates $\mathcal{Z}(x_t)$, and the model updates the weight tensor by solving the following problem:

$$\min_{\boldsymbol{T} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}} \frac{1}{2}\|\boldsymbol{T} - \boldsymbol{T}_t\|^2 + C\xi \quad (3)$$
$$s.t.$$
$$\mathscr{L}_t \leq \xi, \xi \geq 0$$

where $\boldsymbol{T}$ is a decomposed weight tensor and

$$\mathscr{L}_t = \boldsymbol{T} \circ \boldsymbol{\Phi}(x_t, z_t) - \boldsymbol{T} \circ \boldsymbol{\Phi}(x_t, y_t) + \rho(y_t, z_t)$$

Input:

Tensor order D, tensor volume V, mode size $n_d, d = 1, \ldots, D$, surrogate weight vector $\boldsymbol{v}$

Let

$\boldsymbol{v}^+ = [v_1^+, \ldots, v_p^+]$ be the non-negative part of $\boldsymbol{v}$

$\boldsymbol{v}^- = [v_1^-, \ldots, v_q^-]$ be the negative part of $\boldsymbol{v}$

Algorithm:

$\tilde{\boldsymbol{v}}^+$ = *sort*($\boldsymbol{v}^+$) in descending order

$\tilde{\boldsymbol{v}}^-$ = *sort*($\boldsymbol{v}^-$) in ascending order

$u = V/n_D$

$e = p - mod(p, u)$, $f = q - mod(q, u)$

Construct vector

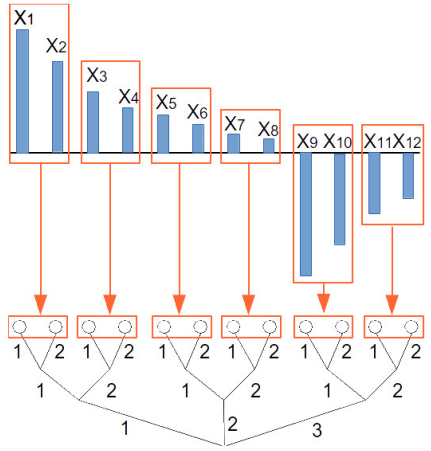
$\mathbf{X} = [\tilde{v}_1^+, \ldots, \tilde{v}_e^+, \tilde{v}_1^-, \ldots, \tilde{v}_f^-,$
$\tilde{v}_{e+1}^+, \ldots, \tilde{v}_p^+, \tilde{v}_{f+1}^-, \ldots, \tilde{v}_q^-]$

Map $X_a, a = 1, \ldots, p + q$ to the tensor entry $\mathcal{T}_{i_1,\ldots,i_D}$, such that

$$a = \sum_{d=1}^{D} (i_d - 1) l_{d-1} + 1$$

where $l_d = l_{d-1} n_d$, and $l_0 = 1$

(a) Mapping a surrogate weight vector to a tensor



(b) Illustration of the algorithm

Figure 2: Algorithm for mapping a surrogate weight vector X to a tensor. (2a) provides the algorithm; (2b) illustrates it by mapping a vector of length $V = 12$ to a $(n_1, n_2, n_3) = (2, 2, 3)$ tensor. The bars X_i represent the surrogate weights — after separately sorting the positive and negative parts — and the labels along a path of the tree correspond to the tensor-index of the weight represented by the leaf resulting from the mapping.

is the structured hinge loss.

This problem setting follows the same "passive-aggressive" strategy as in the original MIRA. To optimize the vectors $\boldsymbol{w}_h^d, h = 1, \ldots, H, d = 1, \ldots, D$, we use a similar iterative strategy as proposed in (Cai et al., 2006b). Basically, the idea is that instead of optimizing $\boldsymbol{w}_h^d$ all together, we optimize $\boldsymbol{w}_1^1, \boldsymbol{w}_1^2, \ldots, \boldsymbol{w}_H^D$ in turn. While we are updating one vector, the rest are fixed. For the problem setting given above, each of the sub-problems that need to be solved is convex, and according to (Cai et al., 2006b) the objective function value will decrease after each individual weight update and eventually this procedure will converge.

We now give this procedure in more detail. Denote the weight vector of the d^{th} mode of the h^{th} tensor at time t as $\boldsymbol{w}_{h,t}^d$. We will update the vectors in turn in the following order: $\boldsymbol{w}_{1,t}^1, \ldots, \boldsymbol{w}_{1,t}^D, \boldsymbol{w}_{2,t}^1, \ldots, \boldsymbol{w}_{2,t}^D, \ldots, \boldsymbol{w}_{H,t}^1, \ldots, \boldsymbol{w}_{H,t}^D$. Once a vector has been updated, it is fixed for future updates.

By way of notation, define

$$\begin{aligned}
&\boldsymbol{W}_{h,t}^d \\
&= \boldsymbol{w}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{w}_{h,t+1}^{d-1} \otimes \boldsymbol{w}_{h,t}^d \otimes, \ldots, \otimes \boldsymbol{w}_{h,t}^D \\
&\quad \text{(and let } \boldsymbol{W}_{h,t}^{D+1} \triangleq \boldsymbol{w}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{w}_{h,t+1}^D\text{)}, \\
&\widehat{\boldsymbol{W}}_{h,t}^d \\
&= \boldsymbol{w}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{w}_{h,t+1}^{d-1} \otimes \boldsymbol{w}^d \otimes, \ldots, \otimes \boldsymbol{w}_{h,t}^D \\
&\quad \text{(where } \boldsymbol{w}^d \in \mathbb{R}^{n_d}\text{)},
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{T}_{h,t}^d &= \sum_{h'=1}^{h-1} \boldsymbol{W}_{h',t}^{D+1} + \boldsymbol{W}_{h,t}^d + \sum_{h'=h+1}^{H} \boldsymbol{W}_{h',t}^1 \quad (4) \\
\widehat{\boldsymbol{T}}_{h,t}^d &= \sum_{h'=1}^{h-1} \boldsymbol{W}_{h',t}^{D+1} + \widehat{\boldsymbol{W}}_{h,t}^d + \sum_{h'=h+1}^{H} \boldsymbol{W}_{h',t}^1
\end{aligned}$$

$$\begin{aligned}
&\boldsymbol{\phi}_{h,t}^d(x, y) \\
&= \boldsymbol{\Phi}(x, y) \times_2 \boldsymbol{w}_{h,t+1}^2 \ldots \times_{d-1} \boldsymbol{w}_{h,t+1}^{d-1} \times_{d+1} \\
&\quad \boldsymbol{w}_{h,t}^{d+1} \ldots \times_D \boldsymbol{w}_{h,t}^D \qquad (5)
\end{aligned}$$

In order to update from $\boldsymbol{w}_{h,t}^d$ to get $\boldsymbol{w}_{h,t+1}^d$, the sub-problem to solve is:

$$\begin{aligned}
&\min_{\boldsymbol{w}^d \in \mathbb{R}^{n_d}} \frac{1}{2} \|\widehat{\boldsymbol{T}}_{h,t}^d - \boldsymbol{T}_{h,t}^d\|^2 + C\xi \\
&= \min_{\boldsymbol{w}^d \in \mathbb{R}^{n_d}} \frac{1}{2} \|\widehat{\boldsymbol{W}}_{h,t}^d - \boldsymbol{W}_{h,t}^d\|^2 + C\xi \\
&= \min_{\boldsymbol{w}^d \in \mathbb{R}^{n_d}} \frac{1}{2} \beta_{h,t+1}^1 \ldots \beta_{h,t+1}^{d-1} \beta_{h,t}^{d+1} \ldots \beta_{h,t}^D \\
&\qquad \|\boldsymbol{w}^d - \boldsymbol{w}_{h,t}^d\|^2 + C\xi \\
&s.t. \qquad \mathscr{L}_{h,t}^d \le \xi, \xi \ge 0.
\end{aligned}$$

where

$$\begin{aligned}
\beta_{h,t}^d &= \|\boldsymbol{w}_{h,t}^d\|^2 \\
\mathscr{L}_{h,t}^d &= \widehat{\boldsymbol{T}}_{h,t}^d \circ \boldsymbol{\Phi}(x_t, z_t) - \widehat{\boldsymbol{T}}_{h,t}^d \circ \boldsymbol{\Phi}(x_t, y_t) \\
&\quad + \rho(y_t, z_t) \\
&= \boldsymbol{w}^d \cdot \left(\boldsymbol{\phi}_{h,t}^d(x_t, z_t) - \boldsymbol{\phi}_{h,t}^d(x_t, y_t) \right) \\
&\quad - \left(\sum_{h'=1}^{h-1} \boldsymbol{W}_{h',t}^{D+1} + \sum_{h'=h+1}^{H} \boldsymbol{W}_{h',t}^1 \right) \circ \\
&\quad (\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, z_t)) \\
&\quad + \rho(y_t, z_t)
\end{aligned}$$

Letting

$$\Delta \boldsymbol{\phi}_{h,t}^d \triangleq \boldsymbol{\phi}_{h,t}^d(x_t, y_t) - \boldsymbol{\phi}_{h,t}^d(x_t, z_t)$$

and

$$\begin{aligned}
s_{h,t}^d \triangleq &\left(\sum_{h'=1}^{h-1} \boldsymbol{W}_{h',t}^{D+1} + \sum_{h'=h+1}^{H} \boldsymbol{W}_{h',t}^1 \right) \circ \\
&(\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, z_t))
\end{aligned}$$

we may compactly write

$$\mathscr{L}_{h,t}^d = \rho(y_t, z_t) - s_{h,t}^d - \boldsymbol{w}^d \cdot \Delta \boldsymbol{\phi}_{h,t}^d.$$

This convex optimization problem is just like the original MIRA and may be solved in a similar way. The updating strategy for $\boldsymbol{w}_{h,t}^d$ is derived as

$$\begin{aligned}
&\boldsymbol{w}_{h,t+1}^d = \boldsymbol{w}_{h,t}^d + \tau \Delta \boldsymbol{\phi}_{h,t}^d \\
&\tau = \qquad (6) \\
&\min \left\{ C, \frac{\rho(y_t, z_t) - \boldsymbol{T}_{h,t}^d \circ (\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, z_t))}{\|\Delta \boldsymbol{\phi}_{h,t}^d\|^2} \right\}
\end{aligned}$$

The initial vectors $\boldsymbol{w}_{h,1}^i$ cannot be made all zero, since otherwise the l-mode product in Equation (5) would yield all zero $\boldsymbol{\phi}_{h,t}^d(x, y)$ and the model would never get a chance to be updated. Therefore, we initialize the entries of $\boldsymbol{w}_{h,1}^i$ uniformly such that the Frobenius-norm of the weight tensor $\boldsymbol{W}$ is unity.

We call the algorithm above "Tensor-MIRA" and abbreviate it as T-MIRA.

5 Experiments

In this section we shows empirical results of the training algorithm on a parsing task. We used the Charniak parser (Charniak et al., 2005) for our experiment, and we used the proposed algorithm to train the reranking feature weights. For comparison, we also investigated training the reranker with Perceptron and MIRA.

5.1 Experimental Settings

To simulate a low-resource training environment, our training sets were selected from sections 2-9 of the Penn WSJ treebank, section 24 was used as the held-out set and section 23 as the evaluation set. We applied the default settings of the parser. There are around $V = 1.33$ million features in all defined for reranking, and the n-best size for reranking is set to 50. We selected the parse with the highest f-score from the 50-best list as the oracle.

We would like to observe from the experiments how the amount of training data as well as different settings of the tensor degrees of freedom affects the algorithm performance. Therefore we tried all combinations of the following experimental parameters:

Parameters	Settings
Training data (m)	Sec. 2, 2-3, 2-5, 2-9
Tensor order (D)	2, 3, 4
# rank-1 tensors (H)	1, 2, 3
Vec. to tensor mapping	approximate, sequential

Here "approximate" and "sequential" means using, respectively, the algorithm given in Figure 2 and the sequential mapping mentioned in Section 3.4. According to the strategy given in 3.2, once the tensor order and number of features are fixed, the sizes of modes and total number of parameters to estimate are fixed as well, as shown in the tables below:

D	Size of modes	Number of parameters
2	1155×1155	2310
3	$110 \times 110 \times 111$	331
4	$34 \times 34 \times 34 \times 34$	136

5.2 Results and Analysis

The f-scores of the held-out and evaluation set given by T-MIRA as well as the Perceptron and MIRA baseline are given in Table 1. From the results, we have the following observations:

1. When very few labeled data are available for training (compared with the number of features), T-MIRA performs much better than the vector-based models MIRA and Perceptron. However as the amount of training data increases, the advantage of T-MIRA fades away, and vector-based models catch up. This is because the weight tensors learned by T-MIRA are highly structured, which significantly reduces model/training complexity and makes the learning process very effective in a low-resource environment, but as the amount of data increases, the more complex and expressive vector-based models adapt to the data better, whereas further improvements from the tensor model is impeded by its structural constraints, making it insensitive to the increase of training data.

2. To further contrast the behavior of T-MIRA, MIRA and Perceptron, we plot the f-scores on both the training and held-out sets given by these algorithms after each training epoch in Figure 3. The plots are for the experimental setting with mapping=surrogate, # rank-1 tensors=2, tensor order=2, training data=sections 2-3. It is clearly seen that both MIRA and Perceptron do much better than T-MIRA on the training set. Nevertheless, with a huge number of parameters to fit a limited amount of data, they tend to over-fit and give much worse results on the held-out set than T-MIRA does.

   As an aside, observe that MIRA consistently outperformed Perceptron, as expected.

3. Properties of linear tensor model: The heuristic vector-to-tensor mapping strategy given by Figure 2 gives consistently better results than the sequential mapping strategy, as expected.

   To make further comparison of the two strategies, in Figure 4 we plot the 20 largest singular values of the matrices which the surrogate weights (given by the Perceptron after running for 1 epoch) are mapped to by both strategies (from the experiment with training data sections 2-5). From the contrast between the largest and the 2^{nd}-largest singular values, it can be seen that the matrix generated

by the first strategy approximates a low-rank structure much better than the second strategy. Therefore, the performance of T-MIRA is influenced significantly by the way features are mapped to the tensor. If the corresponding target weight tensor has internal structure that makes it approximately low-rank, the learning procedure becomes more effective.

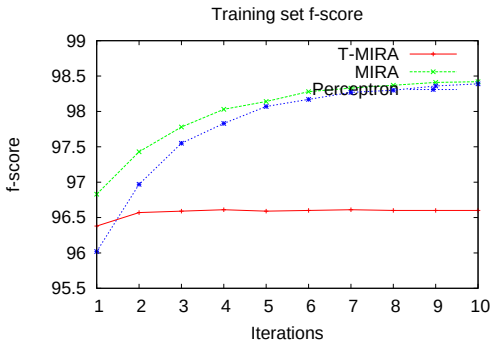
The best results are consistently given by 2^{nd} order tensor models, and the differences between the 3^{rd} and 4^{th} order tensors are not significant. As discussed in Section 3.1, although 3^{rd} and 4^{th} order tensors have less parameters, the benefit of reduced training complexity does not compensate for the loss of expressiveness. A 2^{nd} order tensor has already reduced the number of parameters from the original 1.33 million to only 2310, and it does not help to further reduce the number of parameters using higher order tensors.

4. As the amount of training data increases, there is a trend that the best results come from models with more rank-1 component tensors. Adding more rank-1 tensors increases the model's complexity and ability of expression, making the model more adaptive to larger data sets.

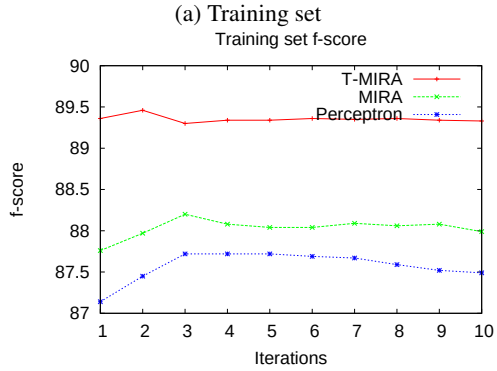
6 Conclusion and Future Work

In this paper, we reformulated the traditional linear vector-space models as tensor-space models, and proposed an online learning algorithm named Tensor-MIRA. A tensor-space model is a compact representation of data, and via rank-1 tensor approximation, the weight tensor can be made highly structured hence the number of parameters to be trained is significantly reduced. This can be regarded as a form of model regularization.Therefore, compared with the traditional vector-space models, learning in the tensor space is very effective when a large feature set is defined, but only small amount of training data is available. Our experimental results corroborated this argument.

As mentioned in Section 3.2, one interesting problem that merits further investigation is how to determine optimal mode sizes. The challenge of applying a tensor model comes from finding a proper tensor structure for a given problem, and



(a) Training set



(b) Held-out set

Figure 3: f-scores given by three algorithms on training and held-out set (see text for the setting).

the key to solving this problem is to find a balance between the model complexity (indicated by the order and sizes of modes) and the number of parameters. Developing a theoretically guaranteed approach of finding the optimal structure for a given task will make the tensor model not only perform well in low-resource environments, but adaptive to larger data sets.

7 Acknowledgements

This work was partially supported by IBM via DARPA/BOLT contract number HR0011-12-C-0015 and by the National Science Foundation via award number IIS-0963898.

References

Deng Cai , Xiaofei He , and Jiawei Han. 2006. Tensor Space Model for Document Analysis *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR)*, 625–626.

Deng Cai, Xiaofei He, and Jiawei Han. 2006. Learning with Tensor Representation *Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign.*

Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	**89.43**	89.16	89.22	89.16	89.21	89.24	89.27	89.14	89.24	**89.21**	88.90	88.89	89.13	88.88	88.88	89.15	88.87	88.99
Evaluation score	**89.83**									89.69								
MIRA	88.57																	
Percep	88.23																	

(a) Training data: Section 2 only

Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.26	89.06	89.12	**89.33**	89.11	89.19	89.18	89.14	89.15	89.2	89.01	88.82	**89.24**	88.94	88.95	89.19	88.91	88.98
Evaluation score	**90.02**									89.82								
MIRA	89.00																	
Percep	88.59																	

(b) Training data: Section 2-3

Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.40	89.44	89.17	**89.5**	89.37	89.18	89.47	89.32	89.18	89.23	89.03	88.93	**89.24**	88.98	88.94	89.16	89.01	88.85
Evaluation score	**89.96**									89.78								
MIRA	89.49																	
Percep	89.10																	

(c) Training data: Section 2-5

Mapping	Approximate									Sequential								
Rank-1 tensors	2			3			4			2			3			4		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.43	89.23	89.06	89.37	89.23	89.1	**89.44**	89.22	89.06	89.21	88.92	88.94	89.23	88.94	88.93	**89.23**	88.95	88.93
Evaluation score	**89.95**									89.84								
MIRA	89.95																	
Percep	89.77																	

(d) Training data: Section 2-9

Table 1: Parsing f-scores. Tables (a) to (d) correspond to training data with increasing size. The upper-part of each table shows the T-MIRA results with different settings, the lower-part shows the MIRA and Perceptron baselines. The evaluation scores come from the settings indicated by the best held-out scores. The best results on the held-out and evaluation data are marked in bold.

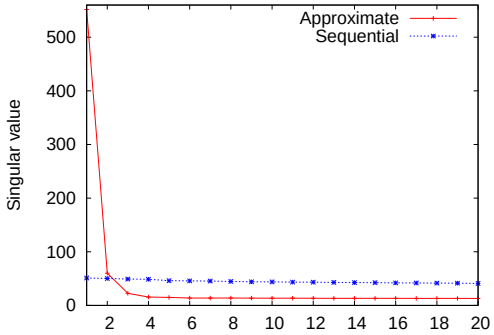


Figure 4: The top 20 singular values of the surrogate weight matrices given by two mapping algorithms.

Eugene Charniak, and Mark Johnson 2005. Coarse-to-fine n-Best Parsing and MaxEnt Discriminative Reranking *Proceedings of the 43th Annual Meeting on Association for Computational Linguistics(ACL)* 173–180.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online Large-Margin Training of Syntactic and Structural Translation Features *Proceedings of Empirical Methods in Natural Language Processing(EMNLP)*, 224–233.

Shay Cohen and Michael Collins. 2012. Tensor Decomposition for Fast Parsing with Latent-Variable PCFGs *Proceedings of Advances in Neural Information Processing Systems(NIPS)*.

Shay Cohen and Giorgio Satta. 2013. Approximate PCFG Parsing Using Tensor Decomposition *Proceedings of NAACL-HLT*, 487–496.

Michael Collins. 2002. Discriminative training methods for hidden Markov Models: Theory and Experiments with Perceptron. Algorithms *Proceedings of Empirical Methods in Natural Language Processing(EMNLP)*, 10:1–8.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Schwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms *Journal of Machine Learning Research(JMLR)*, 7:551–585.

Maryam Fazel. 2002. Matrix Rank Minimization with Applications *PhD thesis, Stanford University*.

Kevin Gimpel, and Noah A. Smith 2012. Structured Ramp Loss Minimization for Machine Translation *Proceedings of North American Chapter of the Association for Computational Linguistics(NAACL)*, 221-231.

Tamir Hazan, Simon Polak, and Amnon Shashua 2005. Sparse Image Coding using a 3D Non-negative Tensor Factorization *Proceedings of the International Conference on Computer Vision (ICCV)*.

Mark Hopkins and Jonathan May. 2011. Tuning as Reranking *Proceedings of Empirical Methods in Natural Language Processing(EMNLP)*, 1352-1362.

Tamara Kolda and Brett Bader. 2009. Tensor Decompositions and Applications *SIAM Review*, 51:455-550.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-Margin Training of Dependency Parsers *Proceedings of the 43rd Annual Meeting of the ACL*, 91–98.

Amnon Shashua, and Tamir Hazan. 2005. Non-Negative Tensor Factorization with Applications to Statistics and Computer Vision *Proceedings of the International Conference on Machine Learning (ICML)*.

Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. 2013. A Tensor-based Factorization Model of Semantic Compositionality *Proceedings of NAACL-HLT*, 1142–1151.

A Proof of Theorem 1

Proof. For $D = 1$, it is obvious that if a set of real numbers $\{x_1, \ldots, x_n\}$ can be represented by a rank-1 matrix, it can always be represented by a vector, but the reverse is not true.

For $D > 1$, if $\{x_1, \ldots, x_n\}$ can be represented by $\mathcal{P} = \mathbf{p}_1 \otimes \mathbf{p}_2 \otimes \ldots \otimes \mathbf{p}_D$, namely $x_i = \mathcal{P}_{i_1,\ldots,i_D} = \prod_{d=1}^{D} p^d_{i_d}$, then for any component vector in mode d,

$$[p^d_1, p^d_2, \ldots, p^d_{n_d}] = [s^d_1 p^d_1, s^d_2 p^d_1, \ldots, s^d_{n^p_d} p^d_1]$$

where n^p_d is the size of mode d of $\mathcal{P}$, s^d_j is a constant and $s^d_j = \frac{p_{i1,\ldots,i_{d-1},j,i_{d+1},\ldots,i_D}}{p_{i1,\ldots,i_{d-1},1,i_{d+1},\ldots,i_D}}$ Therefore

$$x_i = \mathcal{P}_{i_1,\ldots,i_D} = x_{1,\ldots,1} \prod_{d=1}^{D} s^d_{i_d} \tag{7}$$

and this representation is unique for a given D(up to the ordering of $\mathbf{p}_j$ and s^d_j in $\mathbf{p}_j$, which simply assigns $\{x_1, \ldots, x_n\}$ with different indices in the tensor), due to the pairwise proportional constraint imposed by $x_i/x_j, i, j = 1, \ldots, n$.

If x_i can also be represented by $\mathcal{Q}$, then $x_i = \mathcal{Q}_{i_1,\ldots,i_{D+1}} = x_{1,\ldots,1} \prod_{d=1}^{D+1} t^d_{i_d}$, where t^d_j has a similar definition as s^d_j. Then it must be the case that

$$\begin{aligned} &\exists d_1, d_2 \in \{1, \ldots, D+1\}, d \in \{1, \ldots, D\}, d_1 \neq d_2 \\ &s.t. \\ &t^{d_1}_{i_{d_1}} t^{d_2}_{i_{d_2}} = s^d_{i_d}, \\ &t^{d_a}_{i_{d_a}} = s^{d_b}_{i_{d_b}}, \quad d_a \neq d_1, d_2, \quad d_b \neq d \end{aligned} \tag{8}$$

since otherwise $\{x_1, \ldots, x_n\}$ would be represented by a different set of factors than those given in Equation (7).

Therefore, in order for tensor $\mathcal{Q}$ to represent the same set of real numbers that $\mathcal{P}$ represents, there needs to exist a vector $[s^d_1, \ldots, s^d_{n_d}]$ that can be represented by a rank-1 matrix as indicated by Equation (8), which is in general not guaranteed.

On the other hand, if $\{x_1, \ldots, x_n\}$ can be represented by $\mathcal{Q}$, namely

$$x_i = \mathcal{Q}_{i_1,\ldots,i_{D+1}} = \prod_{d=1}^{D+1} q^d_{i_d}$$

then we can just pick $d_1 \in \{1, \ldots, D\}, d_2 = d_1 + 1$ and let

$$\mathbf{q}' = [q^{d_1}_1 q^{d_2}_1, q^{d_1}_1 q^{d_2}_2, \ldots, q^{d_1}_{n^q_{d_2}} q^{d_2}_{n^q_{d_1}}]$$

and

$$\mathcal{Q}' = \mathbf{q}_1 \otimes \ldots \otimes \mathbf{q}_{d_1-1} \otimes \mathbf{q}' \otimes \mathbf{q}_{d_2+1} \otimes \ldots \otimes \mathbf{q}_{D+1}$$

Hence $\{x_1, \ldots, x_n\}$ can also be represented by a D^{th} order tensor $\mathcal{Q}'$. □