Semi-Orthogonal Low-Rank Matrix Factorization for Deep Neural Networks

Daniel Povey[1,2], *Gaofeng Cheng*[3], *Yiming Wang*[1], *Ke Li*[1],
Hainan Xu[1], *Mahsa Yarmohamadi*[1], *Sanjeev Khudanpur*[1,2]

[1]Center for Language and Speech Processing,
[2]Human Language Technology Center of Excellence,
Johns Hopkins University, Baltimore, MD, USA
[3]University of Chinese Academy of Sciences, Beijing, China

{dpovey,gfcheng.cn}@gmail.com, {yiming.wang,kli26,hxu31,mahsa,khudanpur}@jhu.edu

Abstract

Time Delay Neural Networks (TDNNs), also known as one-dimensional Convolutional Neural Networks (1-d CNNs), are an efficient and well-performing neural network architecture for speech recognition. We introduce a factored form of TDNNs (TDNN-F) which is structurally the same as a TDNN whose layers have been compressed via SVD, but is trained from a random start with one of the two factors of each matrix constrained to be semi-orthogonal. This gives substantial improvements over TDNNs and performs about as well as TDNN-LSTM hybrids.

Index Terms: speech recognition, acoustic modeling, deep neural networks

1. Introduction

The most popular method to reduce the number of parameters of already-trained neural networks is to use Singular Value Decomposition (SVD) to factorize each learned weight matrix as a product of two much smaller factors, discarding the smaller singular values [1, 2, 3]. In those cases the network parameters are fine-tuned after this. An obvious idea is to train neural networks with this same structure– effectively, with linear bottlenecks– from a random start; but when this has been tried in the past, the stability of training has been a problem [3].

In this paper we show that we can obtain good results by training such networks from a random start with one of the two factors constrained to be semi-orthogonal[1]. The basic observation is that we do not lose any modeling power by forcing one of the two factors in $\mathbf{M} = \mathbf{AB}$ to be semi-orthogonal; this is easy to show using the existence of the SVD.

We use low-rank factorized layers in TDNN acoustic models trained with lattice-free maximum mutual information (LF-MMI) criterion [4]. Furthermore, we use an idea– skip connections– that is inspired by the "dense LSTM" of [5]. This is somewhat related to the shortcut connections of residual learning [6] and highway connections [7, 8], but consists of appending/concatenating the output of previous layers rather than summing them.

The rest of this paper is organized as follows. Section 2 describes how we enforce the semi-orthonormal constraint on some parameter matrices. Section 3 discusses our use of skip connections. The experimental setup is explained in Section 4. Section 5 presents the experiments and results. Finally, the conclusions are presented in Section 6.

[1]"Semi-orthogonal" is a generalization to non-square matrices of 'orthogonal', i.e. $\mathbf{M}$ is semi-orthogonal if $\mathbf{MM}^T = \mathbf{I}$ or $\mathbf{M}^T\mathbf{M} = \mathbf{I}$.

2. Training with semi-orthogonal constraint

2.1. Basic case

The way we enforce a parameter matrix to be semi-orthogonal is: after every few (specifically, every 4) time-steps of SGD, we apply an efficient update that brings it closer to being a semi-orthogonal matrix.

Let us suppose the parameter matrix $\mathbf{M}$ that we are constraining has fewer rows than columns (otherwise, we would do the same procedure on its transpose). Define $\mathbf{P} \equiv \mathbf{MM}^T$. The condition that we are trying to enforce is: $\mathbf{P} = \mathbf{I}$. Define $\mathbf{Q} \equiv \mathbf{P} - \mathbf{I}$. We can formulate this as minimizing a function $f = \mathrm{tr}\left(\mathbf{QQ}^T\right)$, i.e. the sum of squared elements of $\mathbf{Q}$. The following will use a convention where the derivative of a scalar w.r.t. a matrix is not transposed w.r.t. that matrix. Then $\partial f / \partial \mathbf{Q} = 2\mathbf{Q}$, $\partial f / \partial \mathbf{P} = 2\mathbf{Q}$, and $\partial f / \partial \mathbf{M} = 4\mathbf{QM}$. Our update is what we would get if we did a single iteration of SGD with a learning rate $\nu = 1/8$. Specifically, we do $\mathbf{M} \leftarrow \mathbf{M} - 4\nu\mathbf{QM}$, which expands to:

$$\mathbf{M} \leftarrow \mathbf{M} - \frac{1}{2}\left(\mathbf{MM}^T - \mathbf{I}\right)\mathbf{M}. \tag{1}$$

The choice of $\nu = 1/8$ is special: it gives the method quadratic convergence. The analysis is beyond the scope of this paper but it is easy to verify in the scalar case, or experimentally.

The update of (1) can diverge if $\mathbf{M}$ is too far from being orthonormal to start with, but this does not happen in practice if we use Glorot-style initialization [9] where the standard deviation of the random initial elements of $\mathbf{M}$ is the inverse of square root of number of columns.

It would also be possible to apply the constraint by explicitly adding this penalty term to the objective function.

2.2. Scaled case

Suppose we want $\mathbf{M}$ to be a scaled version of a semi-orthogonal matrix, i.e. some specified constant α times a semi-orthogonal matrix. (This is not directly used in our experiments but helps to derive the "floating" case, below, which is used). By substituting $\frac{1}{\alpha}\mathbf{M}$ into Equation (1) and simplifying, we get this:

$$\mathbf{M} \leftarrow \mathbf{M} - \frac{1}{2\alpha^2}\left(\mathbf{MM}^T - \alpha^2\mathbf{I}\right)\mathbf{M} \tag{2}$$

2.3. Floating case

An option that we have found useful is to enforce $\mathbf{M}$ to be a semi-orthogonal matrix times α, for *any* α. We refer to this as a "floating" semi-orthogonal constraint. The reason why this is useful is that it allows us to control how fast the parameters of the various layers change in a more consistent way. We already use l_2 parameter regularization to help control how fast the parameters change. (Because we use batchnorm and because the ReLU nonlinearity is scale invariant, l_2 does not have a true regularization effect when applied to the hidden layers; but it reduces the scale of the parameter matrix which makes it learn faster). By allowing the scale of the semi-orthogonal-constrained parameter matrices to "float", we can meaningfully apply l_2 regularization to the constrained layers as well, which allows for more consistent control of how fast the various types of parameters change.

To apply the "floating" semi-orthogonal constraint, let $\mathbf{P} \equiv \mathbf{M}\mathbf{M}^T$, and we compute the scale α as follows:

$$\alpha = \sqrt{\operatorname{tr}(\mathbf{P}\mathbf{P}^T)/\operatorname{tr}(\mathbf{P})}. \tag{3}$$

We then apply the scaled update of Equation (2). This formula for α is the one that ensures that the change in $\mathbf{M}$ is orthogonal to $\mathbf{M}$ itself (viewing $\mathbf{M}$ is a vector of concatenated rows).

We also tried an alternative method where we set α to the square root of the average diagonal element of $\mathbf{P}$. This alternative method would be easier to formulate as an added term on the objective function, but we found that it tended to decrease the magnitude of $\mathbf{M}$ each time it was applied, so we favor the method described above.

3. Factorized model topologies

3.1. Basic factorization

The most direct application of our idea is to take an existing topology and factorize its matrices into products of two smaller pieces. Suppose we have a typical TDNN topology with a hidden layer dimension of 700. A typical parameter matrix would be of dimension 700 by 2100, where the number of columns (2100) corresponds to 3 frame offsets of the previous layer spliced together. You can view this as a different way to formulate a 1-d CNN, i.e. a CNN with a 3x1 kernel and 700 filters.

The basic factorization idea would be to factorize the 700 by 2100 matrix into two successive matrices as $\mathbf{M} = \mathbf{A}\mathbf{B}$, with a smaller "interior" dimension of, say, 250: i.e. with $\mathbf{A}$ of size 700×250 and $\mathbf{B}$ of size 250×2100, with $\mathbf{B}$ constrained to be semi-orthogonal. The number 250 in this example is what we will call the linear bottleneck dimension, and 700 is the hidden layer dimension.

This difference between a normal TDNN layer and a factorized TDNN layer is shown in Figures 1 and 2.

3.2. Tuning the dimensions

In practice, after tuning this setup for optimal performance on Switchboard-sized data (300 hours), we ended up using larger matrix sizes, with a hidden-layer dimension[2] of 1280 or 1536, a linear bottleneck dimension of 256, and more hidden layers than before (11, instead of 9 previously)– even when counting each matrix-factorized-into-two as a single layer. The number of parameters in our TDNN-F systems ends up being roughly the same as the baseline.

[2]We have recently started using dimensions based on powers of 2 rather than 10

3.3. Factorizing the convolution

The setup described above uses a constrained[3] 3x1 convolution followed by 1x1 convolution (or equivalently, a TDNN layer splicing 3 frames followed immediately by a feedforward layer). We have found that better results can be obtained by using a constrained 2x1 convolution followed by a 2x1 convolution. We refer to this as "factorizing the convolution".

3.4. 3-stage splicing

Something that we have found to be even better than the "factorized convolution" above, is to have a layer with a constrained 2x1 convolution to dimension 256, followed by another constrained 2x1 convolution to dimension 256, followed by a 2x1 convolution back to the hidden-layer dimension (e.g. 1280). The dimension now goes from, for example, $1280 \rightarrow 256 \rightarrow 256 \rightarrow 1280$, within one layer. The effective temporal context of this setup is of course wider than the TDNN baseline, due to the extra 2x1 convolution.

3.5. Dropout

A feature of some of the systems presented here is a particular form of dropout. This improved these results slightly (around 0.2% or 0.3% absolute). The dropout masks are shared *across time*, so if, for instance, a dimension is zeroed on a particular frame it will be zeroed on all frames of that sequence. In addition, instead of using a zero-one dropout mask we use a continuous dropout scale. Let α be a user-specified value that determines the "strength" with which dropout is applied (this is intended to be analogous to the dropout probability). Then the random scale is chosen from the uniform distribution on the interval $[1 - 2\alpha, 1 + 2\alpha]$. Similar to [10], we use a dropout schedule where α rises from 0 at the start of training to 0.5 halfway through, and 0 at the end. This dropout is applied after the ReLU and batchnorm. Our dropout method, and other variants of it that we tried, did not seem to help in regular (non-factorized) TDNNs.

3.6. Factorizing the final layer

All the TDNN-F (i.e. factorized TDNN) experiments reported here factorize the parameters of the final layer in addition to those of the hidden layers, using a linear bottleneck of 256. We found that even with very small datasets in which factorizing the TDNN layers was not helpful, factorizing the final layer was helpful.

3.7. Skip connections

A further feature that we have found to be helpful for this type of model, inspired by the "dense LSTM" of [5], is skip connections. In the following, by "layer" we mean a possibly-factorized layer that may contain a product of matrices. The basic idea is that some layers receive as input, not just the output of the previous layer but also selected other prior layers which are appended to the previous one. In our currently favored topologies, approximately every other layer receives skip connections in addition to the output of the previous layer– it receives up to 3 other (recent but non-consecutive) layers' outputs appended to the previous layer's output.

In fact what we have said above is a simplification because the skip connections connect the interiors of the layers. In the

[3]By "constrained" in this paper we mean, with a semi-orthogonal constraint on the parameter matrix

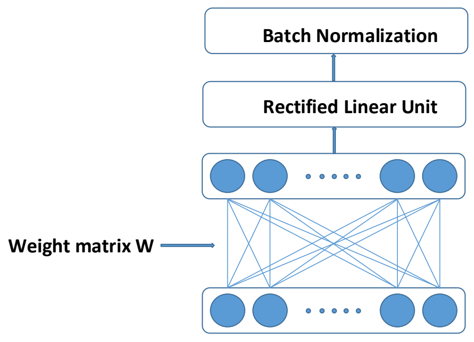

Figure 1: *One standard feed-forward layer in our system.*

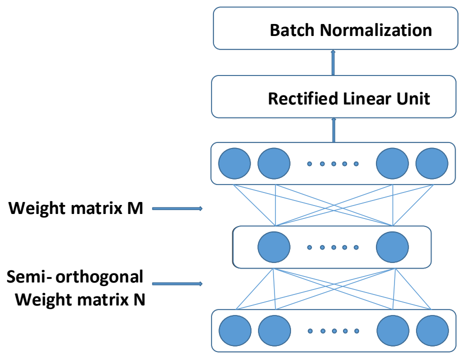

Figure 2: *Factorized layer with semi-orthogonal constraint*

factorized TDNN there are several choices of how to do this – e.g. do we connect small dimension to large, or large to small, or small to small, via the skip connections. What seemed to work the best is the "small to large" option, where we provide as additional input the linear bottleneck from the skip-connected layers (of dimension 256, for example), and append this just before projecting back up to the hidden dimension. In the "factorized convolution" setup, for example, instead of the dimension of the second matrix being $1280 \times (256 * 2)$, it would be $1280 \times (256 * 5)$, if the layer receives 3 skip connections.

The improvement from adding these skip connections is of the order of 0.2% or 0.3% absolute, on average over various Switchboard and Fisher +Switchboard experiments.

4. Experimental setup

All our experiments are conducted using the Kaldi speech recognition toolkit [11] using neural nets trained with lattice-free MMI [4]. We present experiments with models trained on 300 hours of Switchboard data; on Fisher+Switchboard data combined together (2000 hours); and on Swahili and Tagalog data from the MATERIAL program (80 hours each). All setups use 3-fold speed perturbation [12]. We use i-vectors for adaptation.

4.1. Switchboard and Fisher+Switchboard setups

We present experiments 2000 hours Fisher + Switchboard large vocabulary continuous speech recognition (LVCSR) task and 300 hours Switchboard (SWBD) LVCSR task. The number of decision tree leaves is 6078 for Switchboard and 6149 for

Table 1: *WER for TDNN models on Switchboard LVCSR task.*

Acoustic Model	Size	Eval2000		RT03	Time [4](s)
		SWBD	Total		
Baseline TDNN (625)	19M	9.5	14.3	17.5	90
+ l2 regularization		9.1	14.0	16.9	96
Baseline TDNN (1536)	80M	9.4	14.6	17.2	211
+ l2 regularization		9.0	13.9	16.6	210
Factorized TDNN (1536-256)	20M	9.7	14.4	17.4	154
+ l2 regularization		9.1	13.9	17.0	155
++ semi-orthogonal		9.2	**13.7**	**16.0**	147

Fisher+Switchboard. We report results on full HUB5'00 evaluation set (also known as Eval2000) and its "switchboard" subset, which are indicated in the table by 'Total' and 'SWBD'. We also report results on the RT03 test set (LDC2007S10). Results are shown with a 4-gram back-off language model unless otherwise stated.

Results are shown with a 4-gram backoff language model estimated from Switchboard and Fisher transcripts (even for the acoustic models trained only on Switchboard).

4.2. MATERIAL setup

We include ASR experiments on data from two low-resource languages: Swahili and Tagalog. These datasets are provided as part of IARPA's MATERIAL (Machine Translation for English Retrieval of Information in Any Language) program [13]. Each language has about 80 hours of training data, 20 hours of dev data (this data is segmented and is from the same recording conditions as the training data), and 10 hours of analysis data (unsegemented and with domain mismatch). The unsegmented analysis data is first segmented based on the output of the 1st-pass decoding, and then we do a 2nd-pass decoding with the new segments. The ASR system is the same for both passes. The first-pass decoding uses a 3-gram backoff LM which we rescore with an RNNLM [14, 15], with explicit modeling for silence probabilities [16].

5. Experiments

Due to time constraints, we will not be showing a detailed investigation of all the modeling choices discussed in Section 3. Section 5.1 will focus on just factorizing the matrices of a TDNN, the basic factorization of Section 3.1. Then in Section 5.2 we will show a comparison between our previous recipes based on TDNNs [17], BLSTMs, and TDNN-LSTM hybrids with dropout [18, 10], and our TDNN-F recipe with all the additional features discussed above, including l_2 regularization, "floating" semi-orthogonal constraint (Sec. 2.3), 3-stage convolution per layer (Sec. 3.4), dropout (Sec. 3.5) and skip connections (Sec. 3.7).

5.1. Factorizing a TDNN system

Table 1 reports some experiments in which the only change we make is to factorize the parameter matrix with an orthonormal constraint on the first factor. This is with layer dimensions of 625 and 1536, and with and without l_2 regularization. When we factorize the 1536-dimension system with a linear bottleneck dimension of 256, we show results with and without the semi-orthogonal constraint on the first factor. Our baseline TDNN results are surprisingly insensitive to the layer dimension. We get the best results with the factorized layer including the semi-orthogonal constraint.

[4]Time consumed by the model to train on around 1.5 million frames.

Table 2: *Comparing model types: Switchboard (+ Fisher), WERs and decoding real time factor (RTF)*

Switchboard	Params	Eval2000		RT03	RTF
		SWBD	Total		
TDNN	19M	9.5	14.3	17.5	0.36
BLSTM	41M	9.2	13.7	16.0	1.77
TDNN-LSTM	40M	9.0	13.5	15.6	1.26
TDNN-F	23M	**8.7**	**12.7**	**15.1**	0.45
Fisher+Switchboard	Params	Eval2000		RT03	RTF
		SWBD	Total		
TDNN	19M	8.9	13.3	12.5	0.39
BLSTM	41M	8.4	**12.0**	**11.2**	1.78
TDNN-LSTM	40M	**8.2**	**12.0**	**11.2**	1.19
TDNN-F*	20M	8.8	12.4	11.5	0.43

*Older version without dropout, 3-stage splicing or "floating" semi-orthogonal constraint

Table 3: *Comparing model types: MATERIAL results, WERs*

Acoustic Model	Params	Swahili		Tagalog	
		Dev	Analysis	Dev	Analysis
TDNN-LSTM	11M	39.1	52.7	47.0	58.8
+ RNNLM		37.3	50.6	44.7	57.1
TDNN-F	17M	**37.6**	**50.3**	**45.6**	**57.0**
+ RNNLM		**35.7**	**48.5**	**42.9**	**55.5**

In our setup training converged fine without the constraint, although the constraint did improve WERs.

5.2. Comparing model types

In this section we compare our previous architectures based on TDNN, TDNN+LSTM and BLSTM, with our new TDNN-F architecture with l_2 regularization, 3-stage convolution, skip connections and dropout.

Table 2 shows results on Switchboard and Fisher+Switchbard, and Table 3 shows results on the MATERIAL setup. In all cases our TDNN-F results are the best, except for the Fisher+Switchboard setup. Due to time constraints, the Fisher+Switchboard experiment used an older topology missing the 3-stage splicing, dropout and "floating" semi-orthogonal constraint; likely the model we used there was also too small. In any case, it is clear that TDNN-F models give very competitive results.

We won't specify the fine details of the TDNN-F model specifications used, but we use linear bottlenecks of 256, 11 layers, 3-fold frame sub-sampling after the first few layers, and hidden-layer dimensions varying from 1024 to 1536.

6. Conclusions

We have described an effective way to train networks with parameter matrices represented as the product of two or more smaller matrices, with all but one of the factors constrained to be semi-orthogonal. By applying this idea to our TDNN systems giving a factorized TDNN (TDNN-F), and applying several other improvements such as skip connections and a dropout mask that is shared across time, we get results with a TDNN-F model that are often better than our previous TDNN-LSTM and BLSTM results, while being much faster to decode.

Regardless of whether we continue to use this particular architecture, we believe the trick of factorizing matrices with a semi-orthogonal constraint is potentially valuable in many circumstances, and particularly for the final layer. We will continue to experiment with this factorization trick using other network topologies.

7. Acknowledgements

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via contract # FA8650-17-C-9115. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

8. References

[1] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Interspeech*, 2013, pp. 2365–2369.

[2] R. Prabhavalkar, O. Alsharif, A. Bruguier, and I. McGraw, "On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition," *CoRR*, vol. abs/1603.08042, 2016. [Online]. Available: http://arxiv.org/abs/1603.08042

[3] G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, and S. Vitaladevuni, "Model compression applied to small-footprint keyword spotting." in *INTERSPEECH*, 2016, pp. 1878–1882.

[4] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi." in *Interspeech*, 2016, pp. 2751–2755.

[5] K. J. Han, S. Hahm, B.-H. Kim, J. Kim, and I. Lane, "Deep learning-based telephony speech recognition in the wild," in *Proc. Interspeech*, 2017, pp. 1323–1327.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] Y. Zhang, G. Chen, D. Yu, K. Yaco, S. Khudanpur, and J. Glass, "Highway long short-term memory rnns for distant speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5755–5759.

[8] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[9] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[10] G. Cheng, V. Peddinti, D. Povey, V. Manohar, S. Khudanpur, and Y. Yan, "An exploration of dropout with lstms," in *Proceedings of Interspeech*, 2017.

[11] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[12] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[13] "IARPA MATERIAL website," https://www.iarpa.gov/index.php/research-programs/material.

[14] H. Xu, T. Chen, D. Gao, Y. Wang, K. Li, N. Goel, Y. Carmiel, D. Povey, and S. Khudanpur, "A pruned rnnlm lattice-rescoring algorithm for aumatic speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018.

[15] H. Xu, K. Li, Y. Wang, J. Wang, S. Kang, X. Chen, D. Povey, and S. Khudanpur, “Neural network language modeling with letter-based features and importance sampling,” in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018.

[16] G. Chen, H. Xu, M. Wu, D. Povey, and S. Khudanpur, “Pronunciation and silence probability modeling for asr,” in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[17] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” in *Proceedings of Interspeech*. ISCA.

[18] V. Peddinti, Y. Wang, D. Povey, and S. Khudanpur, “Low latency acoustic modeling using temporal convolution and lstms,” *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 373–377, March 2018.