LOW-RANK MATRIX FACTORIZATION OF LSTM AS EFFECTIVE MODEL COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale Long Short-Term Memory (LSTM) cells are often the building blocks of many state-of-the-art algorithms for tasks in Natural Language Processing (NLP). However, LSTMs are known to be computationally inefficient because the memory capacity of the models depends on the number of parameters, and the inherent recurrence that models the temporal dependency is not parallelizable. In this paper, we propose simple, but effective, low-rank matrix factorization (MF) algorithms to compress network parameters and significantly speed up LSTMs with almost no loss of performance (and sometimes even gain). To show the effectiveness of our method across different tasks, we examine two settings: 1) compressing core LSTM layers in Language Models, 2) compressing biLSTM layers of ELMo (Peters et al., 2018a) and evaluate in three downstream NLP tasks (Sentiment Analysis, Textual Entailment, and Question Answering). The latter is particularly interesting as embeddings from large pre-trained biLSTM Language Models are often used as contextual word representations. Finally, we discover that matrix factorization performs better in general, additive recurrence is often more important than multiplicative recurrence, and we identify an interesting correlations between matrix norms and compression performance.

1 INTRODUCTION

Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997; Gers et al., 2000) have become the core of many models for tasks that require temporal dependency. They have particularly shown great improvements in many different NLP tasks, such as Language Modeling (Sundermeyer et al., 2012; Mikolov, 2012), Semantic Role Labeling (He et al., 2017), Named Entity Recognition (Lee et al., 2017), Machine Translation (Bahdanau et al., 2014), and Question Answering (Seo et al., 2016). Recently, a bidirectional LSTM has been used to train deep contextualized Embeddings from Language Models (ELMo) (Peters et al., 2018a), and has become a main component of state-of-the-art models in many downstream NLP tasks.

However, there is an obvious drawback of scalability that accompanies these excellent performances, not only in training time but also during inference time. This shortcoming can be attributed to two factors: the temporal dependency in the computational graph, and the large number of parameters for each weight matrix. The former problem is an intrinsic nature of RNNs that arises while modeling temporal dependency, and the latter is often deemed necessary to achieve better generalizability of the model (Hochreiter & Schmidhuber, 1997; Gers et al., 2000). On the other hand, despite such belief that the LSTM memory capacity is proportional to model size, several recent results have empirically proven the contrary, claiming that LSTMs are indeed over-parameterized (Denil et al., 2013; James Bradbury & Socher, 2017; Merity et al., 2018; Melis et al., 2018; Levy et al., 2018).

Naturally, such results motivate us to search for the most effective compression method for LSTMs in terms of performance, time, and practicality, to cope with the aforementioned issue of scalability. There have been many solutions proposed to compress such large, over-parameterized neural networks including parameter pruning and sharing (Gong et al., 2014; Huang et al., 2018), low-rank Matrix Factorization (MF) (Jaderberg et al., 2014), and knowledge distillation (Hinton et al., 2015). However, most of these approaches have been applied to Feed-forward Neural Networks and Convolutional Neural Networks (CNNs), while only a small attention has been given to compressing LSTM architectures (Lu et al., 2016; Belletti et al., 2018), and even less in NLP tasks. Notably, See et al. (2016) applied parameter pruning to standard Seq2Seq (Sutskever et al., 2014) architecture in Neural Machine Translation, which uses LSTMs for both encoder and decoder. Furthermore, in language modeling, Grachev et al. (2017) uses Tensor-Train Decomposition (Oseledets, 2011), Liu et al. (2018) uses binarization techniques, and Kuchaiev & Ginsburg (2017) uses an architectural change to approximate low-rank factorization.

All of the above mentioned works require some form of training or retraining step. For instance, Kuchaiev & Ginsburg (2017) requires to be trained completely from scratch, as well as distillation based compression techniques (Hinton et al., 2015). In addition, pruning techniques (See et al., 2016) often accompany selective retraining steps to achieve optimal performance. However, in scenarios involving large pre-trained models, e.g. ELMo (Peters et al., 2018a), retraining can be very expensive in terms of time and resources. Moreover, compression methods are normally applied to large and over-parameterized networks, but this is not necessarily the case in our paper. We consider strongly tuned and regularized state-of-the-art models in their respective tasks, which often already have very compact representations. These circumstances make the compression much more challenging, but more realistic and practically useful.

In this work, we advocate low-rank matrix factorization as an effective post-processing compression method for LSTMs which achieve good performance with guaranteed minimum algorithmic speed compared to other existing techniques. We summarize our contributions as the following:

- We thoroughly explore the limits of several different compression methods (matrix factorization and pruning), including fine-tuning after compression, in Language Modeling, Sentiment Analysis, Textual Entailment, and Question Answering.
- We consistently achieve an average of 1.5x (50% faster) speedup inference time while losing $\sim$1 point in evaluation metric across all datasets by compressing additive and/or multiplicative recurrences in the LSTM gates.
- In PTB, by further fine-tuning very compressed models ($\sim$98%) obtained with both matrix factorization and pruning, we can achieve $\sim$2x (200% faster) speedup inference time while even slightly improving the performance of the uncompressed baseline.
- We discover that matrix factorization performs better in general, additive recurrence is often more important than multiplicative recurrence, and we identify clear and interesting correlations between matrix norms and compression performance.

2 Methodology

Long-Short Term Memory (LSTMs) networks are parameterized with two large matrices, $\mathbf{W}_i$ and $\mathbf{W}_h$, which adds the four gates to standard RNNs. Once the parameters are learned, they became static matrices during inference time. Hence, $\mathbf{W}_i$ and $\mathbf{W}_h$ can be compressed using Matrix Factorization to speed up running time, save memory, and possibly improve performance of LSTMs. In this section, we define the basic LSTM structure and introduce Matrix Factorization (MF), namely Semi Non-Negative Factorization (NMF) and Singular Value Decomposition (SVD). Lastly, we show how to apply Low-Rank Matrix Factorization to LSTMs parameters, $\mathbf{W}_i$ and $\mathbf{W}_h$.

2.1 Long-Short Term Memory Network

LSTM is an extended variation of RNN with the aim to capture long-term dependencies in the input and to avoid the exploding/vanishing gradient problems (Hochreiter & Schmidhuber, 1997). It includes input, output, and forget gates along with an explicit memory cell. The gating layers control the information flow within the network, and decide which information to keep, discard, or update in the memory. The memory cells learn the salient information through time. The input gate decides what to keep from current input, and the forget gate removes less important information from the previous memory. Finally, the output hidden state is extrapolated using the output gate and memory cell. The following recurrent equations show the LSTM dynamics.

$$\begin{pmatrix}\mathbf{i}_t\\ \mathbf{f}_t\\ \mathbf{o}_t\\ \hat{\mathbf{c}}_t\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}\begin{pmatrix}\mathbf{W}_i & \mathbf{W}_h\end{pmatrix}\begin{pmatrix}\mathbf{x}_t\\ \mathbf{h}_{t-1}\end{pmatrix}, \qquad \mathbf{W}_i = \begin{pmatrix}\mathbf{W}_i^i\\ \mathbf{W}_i^f\\ \mathbf{W}_i^o\\ \mathbf{W}_i^c\end{pmatrix}, \mathbf{W}_h = \begin{pmatrix}\mathbf{W}_h^i\\ \mathbf{W}_h^f\\ \mathbf{W}_h^o\\ \mathbf{W}_h^c\end{pmatrix} \tag{1}$$

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \hat{\mathbf{c}}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \\ \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \end{aligned} \tag{2}$$

where $\mathbf{x}_t \in \mathbb{R}^{n_{inp}}$, and $\mathbf{h}_t \in \mathbb{R}^{n_{dim}}$ at time t. Here, $\sigma()$ and $\odot$ denote the sigmoid function and element-wise multiplication operator, respectively. The model parameters can be summarized in a compact form with: $\Theta = [\mathbf{W}_i, \mathbf{W}_h]$, where $\mathbf{W}_i \in \mathbb{R}^{4*n_{inp} \times 4*n_{dim}}$ which is the input matrix, and $\mathbf{W}_h \in \mathbb{R}^{4*n_{dim} \times 4*n_{dim}}$ which is the recurrent matrix. Note that we often refer $\mathbf{W}_i$ as additive recurrence and $\mathbf{W}_h$ as multiplicative recurrence, following terminology of Levy et al. (2018).

2.1.1 Low-Rank Matrix Factorization

In this section, we present an overview of Low-Rank Matrix Factorization and its application in LSTM. The rank of a matrix $\mathbf{W}_{m \times n}$ is defined as the number of linearly independent rows or columns in $\mathbf{W}$. Rank of a matrix could be computed either by finding the number of nonzero singular values of $\mathbf{W}$, i.e., $\|\sigma(\mathbf{W})\|_0$, or the smallest number r such that there exists a full-rank matrix $\mathbf{U}_{m \times r}$ and $\mathbf{V}_{r \times n}$, in which $\mathbf{W} = \mathbf{UV}$ (Fazel, 2002). The rank minimization problem $\|\sigma(\mathbf{W})\|_0$ is NP-hard, and a well known convex surrogate function for this problem is the nuclear norm $\|\mathbf{W}\|_{nuc} = \Sigma_{i=1}^{r} \sigma_i$. Since computing the singular values for large scale data is expensive, we aim to find $\mathbf{U}$ and $\mathbf{V}$ to calculate the low-rank representation, in which a exists (Lu et al., 2016).

The matrix $\mathbf{W}$ requires mn parameters and mn flops, while $\mathbf{U}$ and $\mathbf{V}$ require $rm + rn = r(m+n)$ parameters and $r(m+n)$ flops. If we take the rank to be very low $r << m, n$, the number of parameters in $\mathbf{U}$ and $\mathbf{V}$ are much smaller compared to $\mathbf{W}$. The general objective function is given as

$$\underset{m \times n}{\mathbf{W}} = \underset{m \times r}{\mathbf{U}} \; \underset{r \times n}{\mathbf{V}} \quad \Longrightarrow \quad \underset{U,V}{\text{minimize}} \quad ||\mathbf{W} - \mathbf{UV}||_F^2 \tag{3}$$

There are various constrained versions for the low-rank matrix factorization in Equation 3. In the following sections, we explain two most principal and prominent versions with orthogonality and sign constraints.

2.1.2 Truncated Singular Value Decomposition (SVD)

One of the constrained matrix factorization method is based on Singular Value Decomposition (SVD) which produces a factorization by applying orthogonal constraints on the $\mathbf{U}$ and $\mathbf{V}$ factors. These approaches aim to find a linear combination of the basis vectors which restrict to the orthogonal vectors in feature space that minimize reconstruction error. In the case of the SVD, there are no restrictions on the signs of $\mathbf{U}$ and $\mathbf{V}$ factors. Moreover, the data matrix $\mathbf{W}$ is also unconstrained.

$$\mathbf{W} = \mathbf{USV} \quad \Longrightarrow \quad \underset{\mathbf{U},\mathbf{S},\mathbf{V}}{\text{minimize}} \quad ||\mathbf{W} - \mathbf{USV}||_F^2 \quad \text{s.t. } \mathbf{U} \text{ and } \mathbf{V} \text{ are orthogonal, } \mathbf{S} \text{ is diagonal} \tag{4}$$

The optimal values $\mathbf{U}^r_{m \times r}$, $\mathbf{S}^r_{r \times r}$, $\mathbf{V}^r_{r \times n}$ for $\mathbf{U}_{m \times n}$, $\mathbf{S}_{n \times n}$, and $\mathbf{V}_{n \times n}$ are obtained by taking the top r singular values from the diagonal matrix $\mathbf{S}$ and the corresponding singular vectors from $\mathbf{U}$ and $\mathbf{V}$.

2.1.3 Semi Non-negative Matrix Factorization (Semi-NMF)

Another important method, Semi-NMF generalizes Non-negative Matrix Factorization (NMF) by relaxing some of the sign constraints on negative values for $\mathbf{U}$ and $\mathbf{W}$ ($\mathbf{V}$ has to be kept positive). Semi-NMF is more preferable in application to Neural Networks because of this generic capability of having negative values. For detailed explanations of NMF and Semi-NMF, interested readers can refer to Appendix A.

To elaborate, when the input matrix $\mathbf{W}$ is unconstrained (i.e., contains mixed signs), we consider a factorization, in which we restrict $\mathbf{V}$ to be non-negative, while having no restriction on the signs of $\mathbf{U}$. We minimize the objective function as in Equation 9.

$$\mathbf{W}_{\pm} \approx \mathbf{U}_{\pm}\mathbf{V}_{+} \quad \Longrightarrow \quad \underset{\mathbf{U},\mathbf{V}}{\text{minimize}} \quad ||\mathbf{W} - \mathbf{UV}||_F^2 \quad \text{s.t. } \mathbf{V} \geq 0 \tag{5}$$

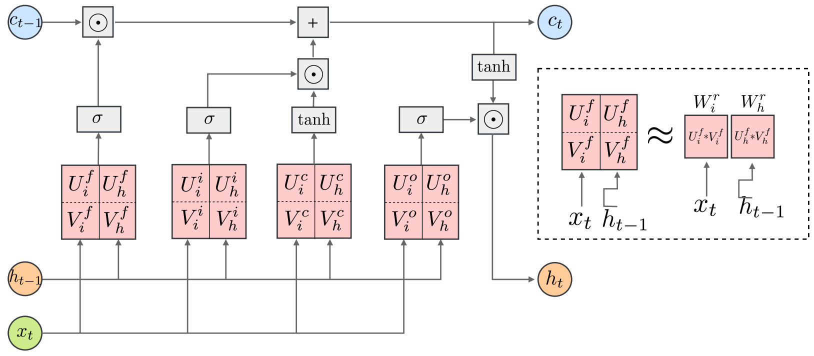
c_{t-1}
c_t
h_{t-1}
h_t
x_t
σ
tanh
U_i^f U_h^f V_i^f V_h^f
U_i^i U_h^i V_i^i V_h^i
U_i^c U_h^c V_i^c V_h^c
U_i^o U_h^o V_i^o V_h^o
$\approx$
W_i^r W_h^r
$U_i^f{*}V_i^f$ $U_h^f{*}V_h^f$

Figure 1: Factorized LSTM Cell

The optimization algorithm iteratively alternates between the update of $\mathbf{U}$ and $\mathbf{V}$ using coordinate descent (Luo & Tseng, 1992). For interested readers, more details on the optimization method and the relation between the NMF-based algorithms and the clustering methods can be found in Appendix A.

2.2 LSTM Low-Rank Factorization

As elaborated in Equation 1, a basic LSTM cell includes four gates: input, forget, output, and cell state, performing a linear combination on input at time t and hidden state at time $t-1$ as in Equation 6.

$$\begin{pmatrix}\mathbf{i}_t\\ \mathbf{f}_t\\ \mathbf{o}_t\\ \hat{\mathbf{c}}_t\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}(\mathbf{W}_i^r \quad \mathbf{W}_h^r)\begin{pmatrix}\mathbf{x}_t\\ \mathbf{h}_{t-1}\end{pmatrix}, \qquad \mathbf{W}_i^r = \begin{pmatrix}\mathbf{U}_i^i\mathbf{V}_i^i\\ \mathbf{U}_i^f\mathbf{V}_i^f\\ \mathbf{U}_i^o\mathbf{V}_i^o\\ \mathbf{U}_i^c\mathbf{V}_i^c\end{pmatrix}, \mathbf{W}_h^r = \begin{pmatrix}\mathbf{U}_h^i\mathbf{V}_h^i\\ \mathbf{U}_h^f\mathbf{V}_h^f\\ \mathbf{U}_h^o\mathbf{V}_h^o\\ \mathbf{U}_h^c\mathbf{V}_h^c\end{pmatrix} \tag{6}$$

For large scale data, having four $\mathbf{W}_i$ matrices and four $\mathbf{W}_h$ matrices demand huge memory and computational power. Hence, we propose to replace $\mathbf{W}_i$, $\mathbf{W}_h$ pair for each gate with their low-rank decomposition, leading to a significant reduction in memory and computational cost requirement, as discussed earlier. The scheme of this operation is shown in the dashed box at the right side of Figure 1 and the complete factorized LSTM cell is shown in the left side of Figure 1.

3 Experiments and Results

We mainly have two means of evaluation using five different publicly available datasets: 1) Perplexity in two different Language Modeling (LM) datasets, 2) Accuracy/F1 in three downstream NLP tasks that ELMo achieved state-of-the-art single-model performance. We benchmark the language modeling capability using both Penn Treebank (Marcus et al., 1993, PTB), with standard preprocessing methods as in Mikolov et al. (2010), and WikiText-2 (Merity et al., 2017, WT2). For the downstream NLP tasks, we evaluate our method in the Stanford Question Answering Dataset (Rajpurkar et al., 2016, SQuAD) the Stanford Natural Language Inference (Bowman et al., 2015, SNLI) corpus, and the Stanford Sentiment Treebank (Socher et al., 2013, SST-5) dataset. Detailed statistics of the tasks are summarized in Appendix Table 6.

For all datasets, we run experiments across different levels of low-rank approximation *r* with different factorization methods, i.e. Semi-NMF and SVD. We also compare the factorization efficiency when only one of $\mathbf{W}_i$ or $\mathbf{W}_h$ was factorized, and when both were compressed (denoted as $\mathbf{W}_{all}$). This is done in order to see which recurrence type (additive or multiplicative) is more suitable for compression. As a compression baseline, we compare matrix factorization with the best pruning methodologies used in LSTMs (Han et al., 2015; See et al., 2016). To elaborate, for each weight matrix $\mathbf{W}_{i,h}$, we mask the low-magnitude weights to zero, according to the compression ratio of the

Table 1: The table shows the total parameters and perplexity (average among all weights and across all ranks) of the model with the best efficiency on test sets for PTB (left) and WT2 (right) for Language Modeling tasks. We choose a 3-layer AWD LSTM (tied) as our baseline (Merity et al., 2018). Lower ***E(r)*** and **PPL** are better. We reproduced the baseline model results. † Parameter numbers are estimated with reference to (Grachev et al., 2017).

	Param.	**PPL (avg)**	***E(r)***
Baseline (ours): AWD-LSTM	24M	58.98	-
(Merity et al., 2018) AWD-LSTM	24M	58.3	-
(Melis et al., 2018) 4-layer LSTM	24M	58.3	-
(Grachev et al., 2017) Tensor Train LSTM	12M	168.6	2.92†
Semi-NMF W$_h$ (r = 400)	18M	59.7 (108.9)	4.19
SVD W$_h$ (r = 400)	18M	**59.3** (**104.9**)	**2.12**
Prune W$_h$ (r = 400)	18M	59.5 (202)	3.04
(Wen et al., 2018) ISS from scratch	11M	65.4	0
(Merity et al., 2018) Fine-tuning	24M	57.3	-
Semi-NMF W$_h$ (r = 200) Fine-tuning	18M	57.84 (64.6)	-0.08
SVD W$_h$ (r = 400) Fine-tuning	18M	57.8 (65.8)	-0.08
Prune W$_h$ (r = 400) Fine-tuning	18M	**57.1** (**61.6**)	**-0.12**
SVD W$_h$ (r = 10) Fine-tuning	9M	58.18	0.0002

	Param.	**PPL (avg)**	***E(r)***
Baseline (ours): AWD-LSTM	33M	65.67	-
(Merity et al., 2018) AWD-LSTM	33M	66	-
(Melis et al., 2018) 2-layer LSTM	24M	65.9	-
Semi-NMF W$_h$ (r = 400)	27M	66.5 (114.2)	4.33
SVD W$_h$ (r = 400)	27M	**66.1** (**111.6**)	**2.28**
Prune W$_h$ (r = 400)	27M	66.23 (251.8)	3.94
SVD W$_h$ (r = 10)	13M	99.92	62.5
Prune W$_h$ (r = 10)	13M	109.16	72.6

low rank factorization[1]. In Appendix Table 12, we report the corresponding compression ratio for each rank used in the experiment.

In addition to standard metrics (e.g. Perplexity, Accuracy, F1), we report the following: number of parameters, efficiency $E(r)$ (ratio of loss in performance vs loss in parameters - the lower the better; refer to Appendix B), L1 norm, and inference time[2] in test set for matrix factorization methods and uncompressed models.

3.1 Language Modeling (LM)

Given a sequence of *N* words, $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, the task of language modeling is to compute the probability of x_T given context $(x_1, x_2, \ldots, x_{T-1})$, for each token.

$$P(\mathbf{x}) = P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, x_2, \ldots, x_{t-1}) \tag{7}$$

We can naturally model this task with any many-to-many Recurrent Neural Network (RNN) variants, and, in particular, we train a 3-layer LSTM Language Model proposed by Merity et al. (2018), following the same model architecture, hyper-parameters, and training details for both datasets, using their released code[3]. After training the LSTM, we compressed the trained weights of each layer with Semi-NMF, SVD, and pruning, with different levels of low-rank approximations, and compare the perplexity. Since, PTB is a small size dataset, we finetuned the compressed version of the model for several epochs[4], interested reader can refer to Appendix for more details. Table 1 summarizes the results which are reported in extensive form in Appendix Tables 7 and 8.

From the tables, compressing $\mathbf{W}_h$ is always more efficient and performing better than compressing $\mathbf{W}_i$ for all the compression methods. When we compare different compression methods, SVD has the lowest, and average, perplexity and the best efficiency among others, in both PTB and WT2. This difference is not very noticeable for high rank (e.g. r=400), but it becomes more evident for higher compression, as shown in Appendix Figures 4 and 7. Moreover, all the methods perform better than the result reported by Grachev et al. (2017) which used Tensor Train (Oseledets, 2011) for compressing the LSTM layers.

In Table 1, we report the results after fine-tuning. The results shows that MF methods and pruning works better that existing reported compression (Wen et al., 2018) and is very close to the uncompressed fine-tuned version of our baseline reported in Merity et al. (2018). In this setting, by factorizing W_i with rank 10 we achieve a small improvement compared to the baseline with a 2.13x speedup. Notice that with rank 10 pruning also comparably works (57.94 PPL).

[1]We align the pruning rate with the rank with $\frac{r(m+n)}{mn}$.

[2]Using an Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz, and averaged over 5 runs.

[3]`https://github.com/salesforce/awd-lstm-lm`

[4]For pruning during training we blind the weights using a static mask.

[6]Timing information was impossible to obtain from test set of SQuAD, as it is not externally provided.

Table 2: The table shows the total parameters and accuracy (average among all weights and across ranks) of the model with the best efficiency on test sets for SST-5 (left), SNLI (right), and dev set[6] for SQuAD (bottom). We reproduced the results of all the baseline models.

	Param.	Acc.	*E(r)*
Baseline (ours): BCN + ELMo	121M	54.5	-
(Peters et al., 2018b) BCN + ELMo	-	54.7 ± 0.5	-
(McCann et al., 2017) BCN	-	53.7	-
Semi-NMF W_h (r = 350)	111M	54.16 (51.11)	5.06
SVD W_i (r = 350)	111M	54.39 (**51.48**)	2.25
Prune W_h (r = 350)	111M	**54.89** (51.36)	**-3.93**
SVD W_h (r = 10)	88M	50.41	15.57
Prune W_h (r = 10)	88M	50.81	14.05

	Param.	Acc.	*E(r)*
Baseline (ours): ESIM + ELMo	99M	88.5	-
(Peters et al., 2018b) ESIM + ELMo	-	88.7 ± 0.17	-
(Chen et al., 2017) ESIM	-	88.6	-
Semi-NMF W_h (r = 300)	86M	88.48 (86.99)	0.23
SVD W_h (r = 250)	82M	88.45 (87.06)	0.32
Prune W_h (r = 400)	111M	**88.54** (**87.18**)	**-0.24**
SVD W_h (r = 10)	66M	87.28	2.86
Prune W_h (r = 10)	66M	87.51	2.32

	Param.	F1 (avg)	*E(r)*
Baseline (ours): BiDAF + ELMo	112M	81.75	-
Wen et al. (2018) ISS from scratch	1.03M	75.78	0.04
Wen et al. (2018) ISS fine-tuning	1.48M	76.4	0.04
(Seo et al., 2016) BiDAF	-	77.3	-
Semi-NMF W_i (r = 400)	105M	81.78 (78.47)	-0.03
SVD W_i (r = 400)	105M	**81.78** (**78.6**)	**-0.38**
Prune W_i (r = 400)	105M	81.73 (77.8)	0.3
SVD W_h (r = 10)	79M	76.71	12.57
Prune W_h (r = 10)	79M	76.54	13.01

3.2 NLP Tasks with ELMo

To highlight the practicality of our proposed method, we also measure the factorization performances with models using pre-trained ELMo (Peters et al., 2018a), as ELMo is essentially a 2-layer bidirectional LSTM Language Model that captures rich contextualized representations. Using the same publicly released pre-trained ELMo weights[7] as the input embedding layer of all three tasks, we train publicly available state-of-the-art models as in Peters et al. (2018a): BiDAF (Seo et al., 2016) for *SQuAD*, ESIM (Chen et al., 2017) for *SNLI*, and BCN (McCann et al., 2017) for *SST-5*. Similar to the Language Modeling tasks, we low-rank factorize the pre-trained ELMo layer only, and compare the accuracy and F1 scores across different levels of low-rank approximation. Note that although many of these models are based on RNNs, we factorize only the ELMo layer in order to show that our approach can effectively compress pre-trained transferable knowledge. As we only compress the ELMo weights, and other layers of each model also have large number of parameters, the inference time is affected less than in Language Modeling tasks. The percentage of parameters in the ELMo layer for BiDAF (*SQuAD*) is 59.7%, for ESIM (*SNLI*) 67.4%, and for BCN (*SST-5*) 55.3%.

From Table 2, for *SST-5* and *SNLI*, we can see that compressing $\mathbf{W}_h$ is in general more efficient and better performing than compressing $\mathbf{W}_i$, except for SVD in *SST-5*. However, from Appendix Table 9, we can see that the difference in accuracy drop is not that big. On the other hand, for the results on SQuAD, Table 2 shows the opposite trend, in which compressing $\mathbf{W}_i$ constantly outperforms compressing $\mathbf{W}_h$ for all methods we experimented with. Notice that, for SQuAD, even with using a very low rank $r = 10$, we see better results in compression than ISS of Wen et al. (2018). In fact, we can see that, in average, using highly compressed ELMo with BiDAF still performs better than without. Overall, we can see that for all datasets, we achieve performances that are not significantly different from the baseline results even after compressing over more than 10M parameters.

4 Norm Analysis

In the previous section, we observe two interesting points: 1) Matrix Factorization (MF) works consistently better in PTB and Wiki-Text 2, but Pruning works better in ELMo for $\mathbf{W}_h$, 2) Factorizing $\mathbf{W}_h$ is generally better than factorizing $\mathbf{W}_i$. To answer these questions, we collected the L1 norm and Nuclear norm statistics, defined in Appendix B, and comparing among $\mathbf{W}_h$ and $\mathbf{W}_i$ for both PTB and ELMo. Following the definitions, L1 and its standard deviation (*std*) together describe the sparsity of a matrix; a matrix with higher L1 and higher *std* is considered to be inherently sparser. On the other hand, Nuclear norm approximates the rank of a matrix. Using these measures, we

[7]`https://allennlp.org/elmo`

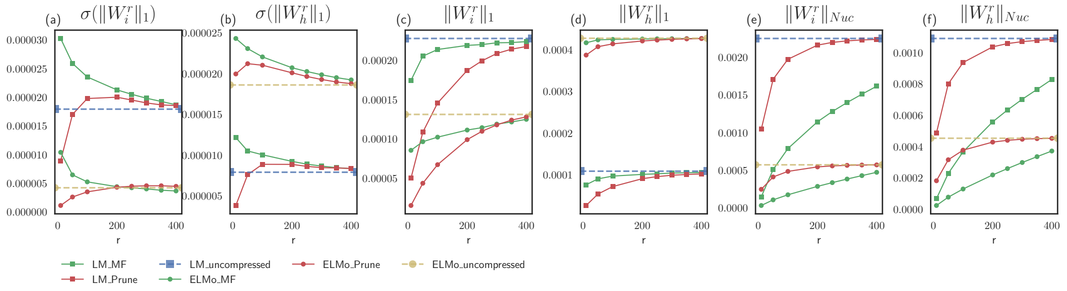

Figure 2: Norm analysis comparisons between MF and Pruning in Language Modeling (PTB) and ELMo. Rank versus (a) $\sigma(\|\mathbf{W}_i\|_1)$ (b) $\sigma(\|\mathbf{W}_h\|_1)$ (c) $\|\mathbf{W}_i\|_1$ (d) $\|\mathbf{W}_h\|_1$ (e) $\|\mathbf{W}_i\|_{Nuc}$ (f) $\|\mathbf{W}_h\|_{Nuc}$.

compare pruning and MF with different compression rates, or factorization ranks, reveal interesting property of the compression algorithms.

MF versus Pruning in $\mathbf{W}_i$ From the results in Section 3, we observed that MF performs better than Pruning in compressing $\mathbf{W}_i$ for high compression ratios. Figure 2 shows rank r versus L1 norm and its standard deviation, in both PTB and ELMo [8]. The first notable pattern from Panel (a) is that MF and Pruning have diverging values from $r \leq 200$. We can see that Pruning makes the *std* of L1 lower than the uncompressed, while MF monotonically increases the *std* from uncompressed baseline. This means that as we approximate to lower ranks ($r \leq 200$), MF retains more salient information, while Pruning loses some of those salient information. This can be clearly shown from Panel (c), in which Pruning always drops significantly more in L1 than MF does. These statistics explain why MF consistently outperforms Pruning for higher compression ratios in $\mathbf{W}_i$.

MF versus Pruning in $\mathbf{W}_h$ The explanation and results for $\mathbf{W}_h$ are also consistent in both PTB and WT2; MF works better than Pruning for higher compression ratios. On the other hand, results from Table 1 show that Pruning works better than MF in $\mathbf{W}_h$ of ELMo even in higher compression ratios. We can explain this seemingly anomalous behavior with Panels (b) and (d). We can see from Panel (d) that L1 norms of MF and Pruning do not significantly deviate nor decrease much from the uncompressed baseline. Meanwhile, Panel (b) reveals an interesting pattern, in which the *std* actually increases for Pruning and is always kept above the uncompressed baseline. This means that Pruning retains salient information for $\mathbf{W}_h$, while keeping the matrix sparse.

This behavior of $\mathbf{W}_h$ can be explained by the nature of the compression and with inherent matrix sparsity. In this setting, pruning is zeroing values already close to zero, so it is able to keep the L1 stable while increasing the *std*. On the other hand, MF instead reduces noise by pushing lower values to be even lower (or zero) and keeps salient information by pushing larger values to be even larger. This pattern is more evident in Figure 3, in which you can see a clear salient red line in $\mathbf{W}_h$ that gets even stronger after factorization ($\mathbf{U}_h \times \mathrm{V}_h$). Naturally, when the compression rate is low (e.g. r=300) pruning is more efficient strategy then MF.

$\mathbf{W}_i$ versus $\mathbf{W}_h$ Another notable pattern we observed is that compressing $\mathbf{W}_h$, in general, shows better performance than compressing $\mathbf{W}_h$. We show the change in Nuclear norm and their corresponding starting points (i.e. uncompressed) in Figure 2 Panels (e) and (f) to answer this question. Notably, $\mathbf{W}_h$ have consistently lower nuclear norm in both LM and ELMo compared to $\mathbf{W}_i$. This difference is larger for LM (PTB), in which $\|\mathbf{W}_i\|_{Nuc}$ is twice of that of $\|\mathbf{W}_h\|_{Nuc}$. As mentioned above, having a lower nuclear norm is often an indicator of low-rank in a matrix; hence, we hypothesize that $\mathbf{W}_h$ is inherently low-rank than $\mathbf{W}_i$ in general. We confirm this from Panel (d), in which even with very high compression ratio (e.g. $r = 10$), the L1 norm does not decrease that much. This explains the large gap in performance between the compression of $\mathbf{W}_i$ and $\mathbf{W}_h$. On the other hand, in ELMo, this gap in norm is lower, which also shows smaller differences in performance between $\mathbf{W}_i$ and $\mathbf{W}_h$, and also sometimes even the opposite (i.g. SQuAD). Hence, we believe that smaller

[8]Note that we refer to ELMo as a single dataset, since the ELMo weights do not change across different datasets.

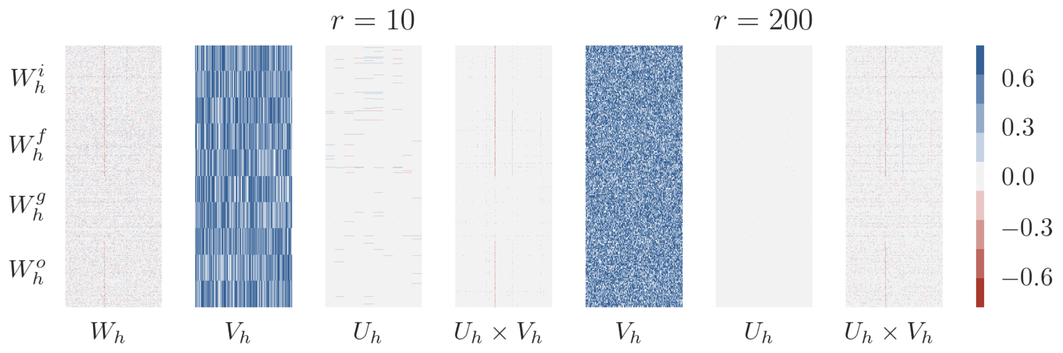

Figure 3: Heatmap of ELMO forward weights.

nuclear norms lead to better performance for all compression methods. These results are, in fact, consistent with the results of Levy et al. (2018) which claims that multiplicative recurrences are of less importance than additive recurrence.

5 Related Work

The current approaches of model compression are mainly focused on matrix factorization, pruning, and quantization. The effectiveness of these approaches were shown and applied in different modalities. In speech processing, Wilson et al. (2008); Mohammadiha et al. (2013); Geiger et al. (2014); Fan et al. (2014) studied the effectiveness of Non-Matrix Factorization (NMF) on speech enhancement by reducing the noisy speech interference. Matrix factorization-based techniques were also applied in image captioning (Hong et al., 2016; Li et al., 2017) by exploiting the clustering intepretations of NMF. Semi-NMF, proposed by Ding et al. (2010), relaxed the constraints of NMF to allow mixed signs and extend the possibility to be applied in non-negative cases. Trigeorgis et al. (2014) proposed a variant of the Semi-NMF to learn low-dimensional representation through a multi-layer structure. Miceli Barone (2018) proposed to replace GRUs with low-rank and diagonal weights to enable low-rank parameterization of LSTMs. Kuchaiev & Ginsburg (2017) modifed LSTM structure by replacing input and hidden weights with two smaller partitions to boost the training and inference time.

On the other hand, compression techniques can also be applied as post-processing steps. Grachev et al. (2017) investigated low-rank factorization on standard LSTM model. The Tensor-Train method has been used to train end-to-end high-dimensional sequential video data with LSTM and GRU (Yang et al., 2017; Tjandra et al., 2017). In another line of work, See et al. (2016) explored pruning in order to reduce the number of parameters in Neural Machine Translation. Wen et al. (2018) proposed to zero out the weights in the network learning blocks to remove insignificant weights of the RNN. Meanwhile, Liu et al. (2018) proposed to binarize LSTM Language Models. Finally, Han et al. (2016) proposed to use all pruning, quantization, and Huffman coding to the weights on AlexNet.

6 Conclusion

In conclusion, we exhaustively explored the limits of compressing LSTM gates using low-rank matrix factorization and pruning in four different NLP tasks. Our experiment results and norm analysis show that show that Low-Rank Matrix Factorization works better in general than pruning, but if the matrix is particularly sparse, Pruning works better. We also discover that inherent low-rankness and low nuclear norm correlate well, explaining why compressing multiplicative recurrence works better than compressing additive recurrence. In future works, we plan to factorize all LSTMs in the model, e.g. BiDAF model, and try to combine both Pruning and Matrix Factorization.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. Factorized recurrent neural architectures for longer range dependence. In *International Conference on Artificial Intelligence and Statistics*, pp. 1522–1530, 2018.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 1657–1668, 2017.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pp. 2148–2156, 2013.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 551–556. ACM, 2004.

Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 606–610. SIAM, 2005.

Chris HQ Ding, Tao Li, and Michael I Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):45–55, 2010.

Hao-Teng Fan, Jeih-weih Hung, Xugang Lu, Syu-Siang Wang, and Yu Tsao. Speech enhancement using segmental nonnegative matrix factorization. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 4483–4487. IEEE, 2014.

Maryam Fazel. *Matrix rank minimization with applications*. PhD thesis, PhD thesis, Stanford University, 2002.

Jürgen T Geiger, Jort F Gemmeke, Björn Schuller, and Gerhard Rigoll. Investigating nmf speech enhancement for neural network based acoustic models. In *Proc. INTERSPEECH 2014, ISCA, Singapore, Singapore*, 2014.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL `http://dx.doi.org/10.1162/089976600300015015`.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Artem M Grachev, Dmitry I Ignatov, and Andrey V Savchenko. Neural networks compression for language modeling. In *International Conference on Pattern Recognition and Machine Intelligence*, pp. 351–357. Springer, 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1135–1143. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf`.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and whats next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 473–483, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Seunghoon Hong, Jonghyun Choi, Jan Feyereisl, Bohyung Han, and Larry S Davis. Joint image clustering and labeling by matrix factorization. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1411–1424, 2016.

Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

Caiming Xiong James Bradbury, Stephen Merity and Richard Socher. Quasi-recurrent neural networks. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=H1zJ-v5xl`.

Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for lstm networks. *ICLR Workshop*, 2017.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 188–197, 2017.

Omer Levy, Kenton Lee, Nicholas FitzGerald, and Luke Zettlemoyer. Long short-term memory as a dynamically computed element-wise weighted sum. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 732–739. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/P18-2116`.

Xuelong Li, Guosheng Cui, and Yongsheng Dong. Graph regularized non-negative low-rank matrix factorization for image clustering. *IEEE transactions on cybernetics*, 47(11):3840–3853, 2017.

Xuan Liu, Di Cao, and Kai Yu. Binarized lstm language model. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2113–2121. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/N18-1192`.

Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. Learning compact recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 5960–5964. IEEE, 2016.

Zhi-Quan Luo and Paul Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=972470.972475`.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6294–6305, 2017.

Gbor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ByJHuTgA-`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *ICLR*, 2017.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyyGPP0TZ`.

Antonio Valerio Miceli Barone. Low-rank passthrough neural networks. In *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*, pp. 77–86. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/W18-3410`.

Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Nasser Mohammadiha, Paris Smaragdis, and Arne Leijon. Supervised and unsupervised speech enhancement using nonnegative matrix factorization. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(10):2140–2151, 2013.

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

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237. Association for Computational Linguistics, 2018a. URL `http://aclweb.org/anthology/N18-1202`.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pp. 2227–2237, 2018b.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1264. URL `http://www.aclweb.org/anthology/D16-1264`.

Abigail See, Minh-Thang Luong, and Christopher D Manning. Compression of neural machine translation models via pruning. *CoNLL 2016*, pp. 291, 2016.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *ICLR 2017*, 2016.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, 2013. URL `http://www.aclweb.org/anthology/D13-1170`.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Compressing recurrent neural network with tensor train. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 4451–4458. IEEE, 2017.

George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. A deep semi-nmf model for learning hidden representations. In *International Conference on Machine Learning*, pp. 1692–1700, 2014.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk6cfpRjZ`.

Kevin W Wilson, Bhiksha Raj, Paris Smaragdis, and Ajay Divakaran. Speech denoising using nonnegative matrix factorization with priors. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pp. 4029–4032. IEEE, 2008.

Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pp. 267–273. ACM, 2003.

Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*, pp. 3891–3900, 2017.

A MATRIX FACTORIZATION

A.1 SEMI-NMF OPTIMIZATION

In this section, we provide the semi-NMF algorithm and we elaborate the optimization and the aim of each step. This algorithm is an extension of NMF, where the data matrix is remained unconstrained Ding et al. (2010). The original NMF optimization function shows in 8. Semi-NMF ignores the constraint in U as showed in 9.

$$\mathbf{W}_+ \approx \mathbf{U}_+\mathbf{V}_+ \quad \implies \quad \underset{U,V}{\text{minimize}} \quad ||\mathbf{W} - \mathbf{U}\mathbf{V}||_F^2 \quad \text{s.t. } U \geq 0,\ V \geq 0 \tag{8}$$

$$\mathbf{W}_\pm \approx \mathbf{U}_\pm\mathbf{V}_+ \quad \implies \quad \underset{U,V}{\text{minimize}} \quad ||\mathbf{W} - \mathbf{U}\mathbf{V}||_F^2 \quad \text{s.t. } \mathbf{V} \geq 0 \tag{9}$$

Exploring the relationships between matrix factorization and K-means clustering has implications for the interpretability of matrix factors Ding et al. (2005); Xu et al. (2003); Dhillon et al. (2004).

1. Initialize $\mathbf{U}$ and run k-means clustering Hartigan & Wong (1979). $\mathbf{V}_{ki} = 1$ if x_i belongs to cluster k. Otherwise, $\mathbf{V}_{ki} = 0$.
   The objective function of k-means clustering
   $$J_{k-means} = \sum_{i=1}^{m}\sum_{k=1}^{K} v_{ki}||w_i - u_k||^2 = ||\mathbf{W} - \mathbf{U}\mathbf{V}||^2 \tag{10}$$
   We can relax the range of v_{ki} over the values in $(0, 1)$ or $(0, \infty)$. This restricts $\mathbf{V}$ to accept only nonnegative values and allow $\mathbf{U}$ to have mixed signs values.
2. Update $\mathbf{U}$ by fixing $\mathbf{V}$ using this constraint. By fixing $\mathbf{V}$, the solution of $\boldsymbol{U}$ can be obtained by calculating the derivative of $d\mathbf{J}/d\boldsymbol{U} = -2\mathbf{W}\mathbf{V}^T + 2\boldsymbol{U}\mathbf{V}\mathbf{V}^T = 0$. Then we can get the $\mathbf{U} = \mathbf{W}\mathbf{V}^T(\mathbf{V}\mathbf{V}^T)^{-1}$.
3. Update $\mathbf{V}$ by fixing $\boldsymbol{U}$
   $$\mathbf{V}_{ki} = \mathbf{V}_{ki}\sqrt{\frac{(\mathbf{W}^T\mathbf{U})^+_{ik} + [\mathbf{V}^T(\mathbf{U}^T\mathbf{U})^-]_{ik}}{(\mathbf{W}^T\mathbf{U})^-_{ik} + [\mathbf{V}^T(\mathbf{U}^T\mathbf{U})^+]_{ik}}} \tag{11}$$
   The positive and negative parts are computed $\mathbf{A}^+_{ik} = \frac{(|\mathbf{A}_{ik}|+\mathbf{A}_{ik})}{2}$, $\mathbf{A}^-_{ik} = \frac{(|\mathbf{A}_{ik}|-\mathbf{A}_{ik})}{2}$

According to Ding et al. (2010), this method will reach convergence. By fixing $\mathbf{U}$, the residual $||\mathbf{W} - \mathbf{U}\mathbf{V}^T||^2$ will decrease monotonically, and after fixing $\mathbf{V}$, we get the optimal solution for the objective function.

The algorithm is computed by using an iterative updating algorithm that alternates between the update of $\mathbf{U}$ and $\mathbf{V}$ (Ding et al., 2010). The steps are very similar to coordinate descent Luo & Tseng (1992) with some modifications. The optimization is convex in $\mathbf{U}$ or $\mathbf{V}$, not both.

In the latent space derived by the NMF factorization family, each axis captures the centroid of a particular cluster, and each sample is represented as an additive combination of the centroids. The cluster membership of each document can be easily determined by finding the corresponding cluster centroid (the axis) with which the document has the largest projection value. Note in particular that the result of a K-means clustering run can be written as a matrix factorization $\mathbf{W} = \mathbf{U}\mathbf{V}$, where $\mathbf{W} \in \mathbb{R}^{nm}$ is the data matrix, $\mathbf{U} \in \mathbb{R}^{nr}$ contains the cluster centroids, and $\mathbf{V} \in \mathbb{R}^{rm}$ contains the cluster membership indicators.

- Perform the NMF or semi-NMF on $\mathbf{W}$ to obtain the two non-negative matrices $\mathbf{U}$ and $\mathbf{V}$.
- Matrix $\mathbf{U}$ contains r $n-$dimensional cluster centers and matrix $\mathbf{V}$ contains membership weight for each of the m samples in each of the r clusters. One can assign data i to the cluster c if $c = argmax_j\mathbf{V}_{ij}$.

Table 3: Summary of Op, Cost and Memory consumption of compression method used in the paper.

$\mathbf{A} \in \mathbb{R}^{z \times m}, \mathbf{W} \in \mathbb{R}^{m \times n}$ $\mathbf{U} \in \mathbb{R}^{m \times r}, \mathbf{V} \in \mathbb{R}^{r \times n}$	**Op**	**Cost**	**Memory**
Uncompressed	$\mathbf{A} \cdot \mathbf{W}$	$O(z \times m \times n)$	$O(z \times m + m \times n)$
Pruning[9]	$\mathbf{A} \cdot (\frac{r(m+n)}{mn}\mathbf{W})$	$\Omega(\mathrm{z} \times r(m+n))$	$O(z \times m + r(m+n))$
Matrix Factorization	$(\mathbf{A} \cdot \mathbf{U}) \cdot \mathbf{V}$	$\Theta(\mathrm{z} \times r(m+n))$	$O(z \times m + r(m+n))$

A.2 Complexity

The algorithm complexity in terms of time and memory is shown in Table 3.

B Measures

B.1 L1 Norm

The L1 Norm of any matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, induced from the vector L1 Norm and called the maximum absolute column sum norm, and the standard deviation of the L1 norm are defined as such,

$$\mathcal{L}_j = \sum_{i=1}^{m} |w_{ij}|, \qquad \|\mathbf{W}\|_1 = \max_j \mathcal{L}_j, \qquad \bar{\mathcal{L}} = \sum_{j=1}^{n} \mathcal{L}_j, \qquad \sigma(\|\mathbf{W}\|_1) = \sum_{j=1}^{n} (\bar{\mathcal{L}} - \mathcal{L}_j)^2 \quad (13)$$

L1 norm is basically the maximum of the L1 vector norms of all column vectors in $\mathbf{W}$. The standard deviation of L1 norm calculates the standard deviation across the column-wise L1 norms.. These values indicate how high the values are in terms of magnitude, and how much variance the column vector norms have. In other words, if the $\sigma(\|\mathbf{W}\|_1)$ is high and $\|\mathrm{W}\|_1$ is high, the matrix can be considered to be sparser.

B.2 Nuclear Norm

The nuclear norm of any matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, is defined as the sum of singular values as following,

$$\|\mathbf{W}\|_{nuc} = \sqrt{\mathbf{W} * \mathbf{W}} = \sum_{i=1}^{\min m,n} \sigma_i(\mathbf{W}) \quad (14)$$

The nuclear norm is often an approximation of the rank of the given matrix; a low nuclear norm indicates low rank.

B.3 Efficiency $E(r)$

For evaluating the performance of the compression we define efficiency measure as:

$$E(r) = \frac{R(M, M^r)}{R(P, P^r)} \quad (15)$$

where M represent any evaluation metric (i.e. Accuracy, F1-score, Perplexity[10]), P represents the number of parameters[11], and $R(a,b) = \frac{a-b}{a}$ where $a = max(a,b)$, i.e. the ration. This indicator shows the ratio of loss in performance versus the loss in number of parameter. Hence, an efficient compression holds a very small E since the denominator, $P - P^r$, became large just when the number of parameter decreases, and the numerator, $M - M^r$, became small only if there is no loss in the considered measure. In some cases E became negative if there is an improvement.

[9]Pruning speed and memory depends on the desired sparsity defined as $\frac{r(m+n)}{mn}$

[10]Note that for Perplexity, we use $R(M^r, M)$ instead, because lower is better.

[11]P^r and M^r are the parameter and the measure after semi-NMF of rank r

C Fine-tuning

In this section, we explain the fine-tuning steps and how we tune our hyper-parameters. This step is intended to improve the performance of our models. After several attempts of fine-tuning, we figured out that we need to apply different initial learning rates for every rank to quickly reach the convergence point, especially after matrix factorization. During every step, Asynchronous Stochastic Gradient Descent Merity et al. (2018) is used as the optimizer. Table 4 shows the hyper-parameters setting.

Table 4: Hyper-parameters setting

	r	***initial lr***
Semi-NMF	10	10
	200	1
	400	0.1
SVD	10	10
	200	1
	400	1
Prune	10	30
	200	30
	400	30

We achieved lower perplexity after fine-tuning steps and the results are shown in Figure 5.

Table 5: Results of fine-tuning on PTB.

	Rank	***Semi-NMF***	***SVD***	***Prune***
Baseline	r	58.98	58.98	58.98
$\mathbf{W}_i$	10	81.4	88.12	82.23
	200	58.57	58.49	57.97
	400	58.47	58.04	57.65
$\mathbf{W}_h$	10	58.11	58.18	57.94
	200	57.76	58.03	57.2
	400	57.84	**57.81**	57.19
$\mathbf{W}_{all}$	10	92.47	97.26	-
	200	58.59	58.82	-
	400	58.58	57.99	-

D Tables

Table 6: Datasets used for the evaluation. In PTB and WikiText-2 we show the number of tokens, in the other three the number of samples.

Dataset	Train	Validation	Test	Vocabulary
PTB	929,590	73,761	82,431	10,000
WT2	2,088,628	217,646	245,569	33,278
SST-5	8,544	1,101	2,210	19,500
SNLI	550,152	10,000	10,000	84,487
SQuAD	90,000	10,000	10,000	115,613

Table 7: Penn Tree Bank (PTB) language modeling results.

PTB									
Rank		***Speedup***	***PPL Increase***		$E(r)$		***Param.***	***PPL Increase***	$E(r)$
r		$\frac{T_{Unc}}{T_r}$	**Semi-NMF**	**SVD**	**Semi-NMF**	**SVD**	**M(%)**	**Pruning**	
uncompressed		495.43 (ms)	58.98		-		20.21M	58.98	-
$\mathbf{W}_i$	*10*	1.6	426.42	403.21	200.97	199.57	11.38M (43.7)	617.34	208.82
	50	1.58	76.57	72.24	137.64	134.14	11.92M (41.0)	205.84	189.39
	100	1.47	28.24	24.24	85.88	77.25	12.59M (37.7)	50.22	121.98
	200	1.33	10.61	6.86	49.15	33.58	13.94M (31.0)	9.36	44.15
	250	1.27	7.64	4.5	41.43	25.61	14.62M (27.7)	4.87	27.55
	300	1.24	5.96	3.09	37.7	20.46	15.29M (24.3)	2.74	18.24
	350	1.17	4.7	2.2	35.14	17.14	15.97M (21.0)	1.58	12.42
	400	1.16	3.72	1.61	33.59	15.08	16.64M (17.7)	**0.92**	8.7
$\mathbf{W}_h$	*10*	2.13	19.52	19.1	45.34	44.6	9.13M (54.8)	24.18	53.02
	50	2.06	6.96	6.69	20.23	19.52	9.67M (52.2)	12.93	34.46
	100	1.94	3.64	3.19	11.9	10.51	10.34M (48.8)	7.83	24.0
	200	1.58	1.65	1.16	6.46	4.58	11.69M (42.2)	3.63	13.75
	250	1.47	1.28	0.81	5.47	3.48	12.37M (38.8)	2.54	10.64
	300	1.42	1.05	0.6	4.93	2.84	13.04M (35.5)	1.26	5.9
	350	1.29	0.89	0.46	4.63	2.43	13.72M (32.1)	0.79	4.11
	400	1.26	0.72	**0.36**	4.19	2.12	14.39M (28.8)	0.52	3.04
$\mathbf{W}_{all}$	*10*	8.19	433.15	415.08	89.3	88.84	291.60K (98.6)	654.96	93.08
	50	7.63	86.17	81.99	63.69	62.4	1.37M (93.2)	274.56	88.31
	100	5.82	34.82	30.22	42.9	39.15	2.72M (86.5)	89.22	69.57
	200	3.43	14.55	9.85	27.04	19.56	5.42M (73.2)	21.62	36.66
	250	2.42	10.85	6.28	23.37	14.48	6.77M (66.5)	13.08	27.3
	300	2.14	8.28	4.13	20.58	10.93	8.12M (59.8)	7.83	19.59
	350	1.71	6.6	2.82	18.94	8.6	9.47M (53.1)	5.56	16.21
	400	1.59	5.22	**2.07**	17.5	7.28	10.82M (46.5)	4.33	14.72

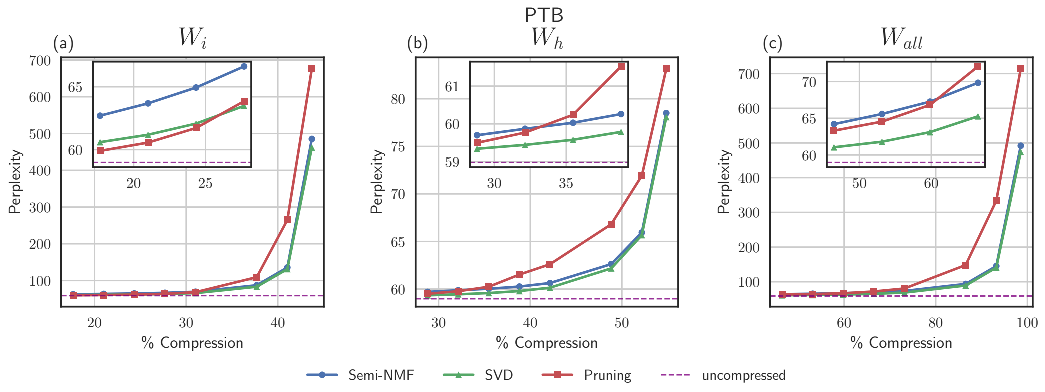

Figure 4: PTB: comparison between Semi-NMF, SVD and pruning. Perplexity versus LSTM compression rate, in (a) $\mathbf{W}_i$ (b) $\mathbf{W}_h$ (c) $\mathbf{W}_{all}$.

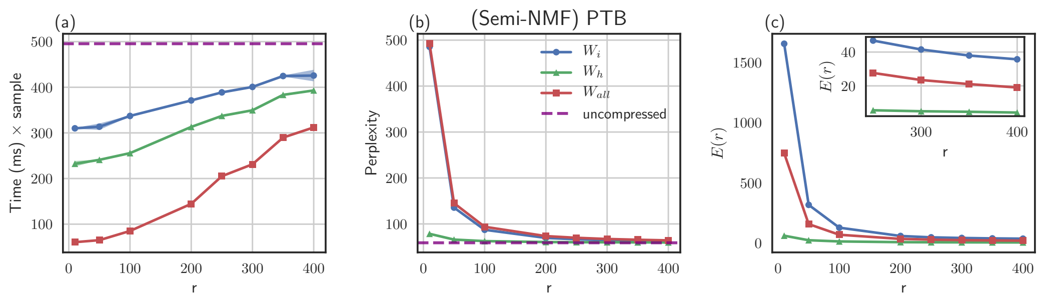

Figure 5: Semi-NMF compression results in PTB for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

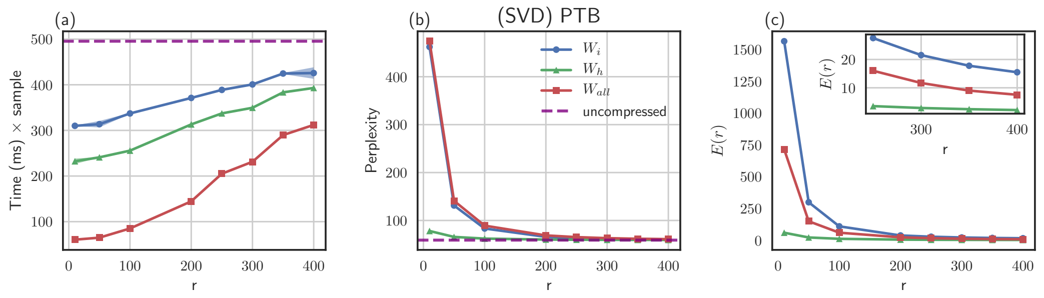

Figure 6: SVD compression results in PTB for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

Table 8: WikiText-2 (WT2) language modeling results.

Wiki-Text 2									
Rank		***Speedup***	***PPL Increase***		$E(r)$		***Param.***	***PPL Increase***	$E(r)$
r		$\frac{T_{Unc}}{T_r}$	***Semi-NMF***	***SVD***	***Semi-NMF***	***SVD***	***M(%)***	***Pruning***	
uncompressed		516.97 (ms)	65.67		-		20.21M	65.67	-
$\mathbf{W}_i$	*10*	1.6	415.94	377.82	197.57	194.89	11.38M (43.7)	791.2	211.23
	50	1.53	63.01	59.37	119.31	115.69	11.92M (41.0)	137.67	164.97
	100	1.43	19.29	16.63	60.22	53.6	12.59M (37.7)	35.47	93.02
	200	1.33	6.99	5.07	31.01	23.11	13.94M (31.0)	6.71	29.88
	250	1.29	5.24	3.45	26.69	18.03	14.62M (27.7)	3.54	18.48
	300	1.23	4.14	2.49	24.36	15.02	15.29M (24.3)	2.01	12.2
	350	1.18	3.29	1.88	22.72	13.22	15.97M (21.0)	1.14	8.12
	400	1.19	2.74	1.44	22.68	12.18	16.64M (17.7)	0.7	5.97
$\mathbf{W}_h$	*10*	2.09	36.5	34.25	65.14	62.49	9.13M (54.8)	43.49	72.64
	50	1.97	7.87	7.58	20.51	19.84	9.67M (52.2)	19.43	43.76
	100	1.9	3.68	3.39	10.87	10.05	10.34M (48.8)	10.07	27.23
	200	1.56	1.76	1.36	6.19	4.81	11.69M (42.2)	3.59	12.3
	250	1.48	1.41	0.97	5.42	3.75	12.37M (38.8)	2.22	8.42
	300	1.41	1.14	0.73	4.81	3.09	13.04M (35.5)	1.41	5.93
	350	1.32	0.96	0.55	4.48	2.6	13.72M (32.1)	0.9	4.21
	400	1.29	0.83	0.43	4.33	2.28	14.39M (28.8)	0.56	2.94
$\mathbf{W}_{all}$	*10*	6.23	448.05	422.28	88.49	87.81	291.60K (98.6)	822.8	93.96
	50	5.44	83.22	78.65	59.96	58.47	1.37M (93.2)	220.76	82.68
	100	4.75	28.39	24.08	34.88	31.01	2.72M (86.5)	73.61	61.07
	200	3.19	10.08	7.16	18.18	13.44	5.42M (73.2)	16.61	27.59
	250	2.45	7.32	4.82	15.08	10.29	6.77M (66.5)	8.62	17.45
	300	1.93	5.84	3.45	13.65	8.34	8.12M (59.8)	4.75	11.28
	350	1.68	4.7	2.57	12.57	7.07	9.47M (53.1)	2.64	7.27
	400	1.58	3.86	1.96	11.95	6.23	10.82M (46.5)	1.55	4.96

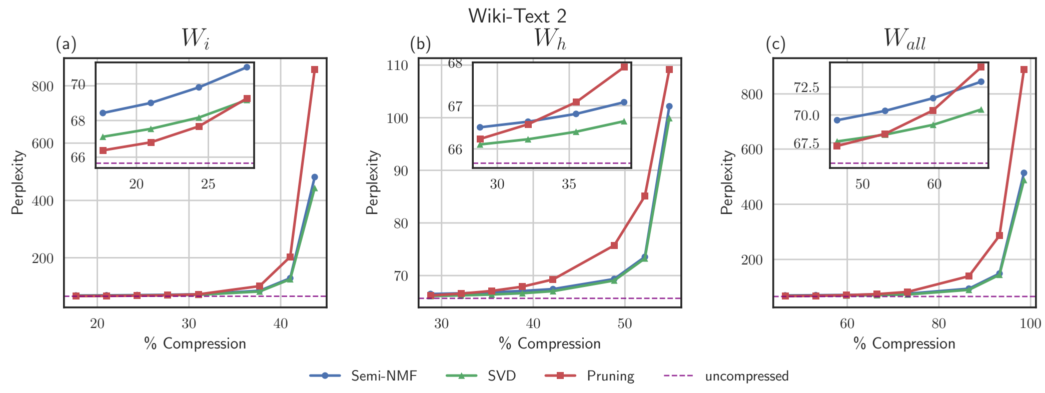

Figure 7: WikiText-2: comparison between Semi-NMF, SVD and pruning. Perplexity versus LSTM compression rate, in (a) $\mathbf{W}_i$ (b) $\mathbf{W}_h$ (c) $\mathbf{W}_{all}$.

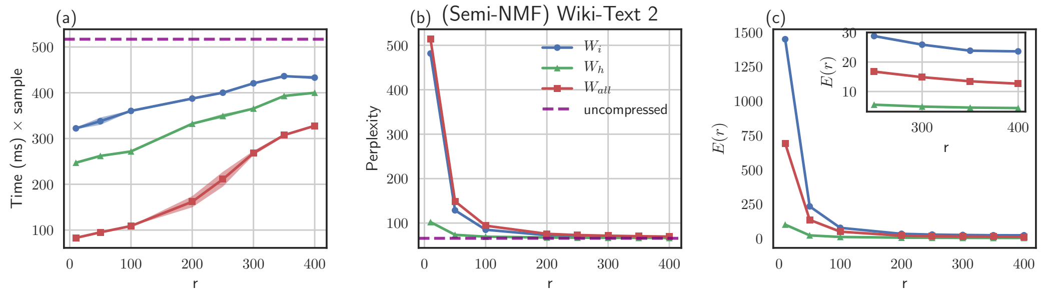

Figure 8: Semi-NMF compression results in WikiText-2 for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

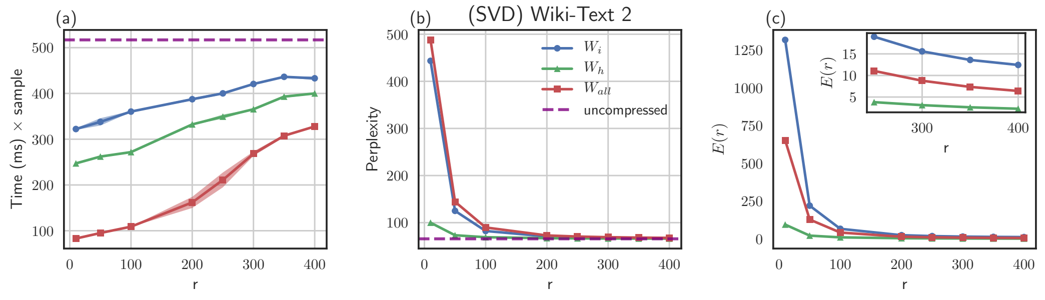

Figure 9: SVD compression results in WikiText-2 for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

Table 9: SST-5 sentiment analysis results.

SST-5									
Rank		***Speedup***	***Acc Drop***		$E(r)$		***Param.***	***Acc Drop***	$E(r)$
r		$\frac{T_{Unc}}{T_r}$	***Semi-NMF***	***SVD***	***Semi-NMF***	***SVD***	***M(%)***	***Pruning***	
uncompressed		90.17 (ms)	54.57		-		67.11M	54.57	-
$\mathbf{W}_i$	*10*	1.28	16.33	13.98	61.1	52.3	34.23M (49.0)	20.0	74.81
	50	1.25	6.83	6.56	27.85	26.74	36.93M (45.0)	11.4	46.47
	100	1.23	3.53	2.94	16.2	13.5	40.31M (39.9)	3.62	16.61
	200	1.16	1.99	1.54	12.22	9.44	47.07M (29.9)	0.86	5.28
	250	1.13	1.36	0.9	10.02	6.68	50.45M (24.8)	0.41	3.01
	300	1.1	0.72	0.68	6.7	6.29	53.83M (19.8)	0.27	2.51
	350	1.08	0.86	0.18	10.68	2.25	57.21M (14.8)	-0.09	-1.12
	400	1.05	0.45	0.23	8.53	4.27	60.59M (9.7)	-0.05	-0.85
$\mathbf{W}_h$	*10*	1.26	4.39	4.16	16.42	15.57	34.23M (49.0)	3.76	14.05
	50	1.25	2.35	2.35	9.59	9.59	36.93M (45.0)	2.26	9.22
	100	1.21	2.13	2.4	9.76	11.01	40.31M (39.9)	1.9	8.72
	200	1.16	1.4	1.18	8.61	7.22	47.07M (29.9)	0.27	1.67
	250	1.13	1.13	1.31	8.35	9.69	50.45M (24.8)	0.09	0.67
	300	1.09	0.68	0.86	6.29	7.96	53.83M (19.8)	-0.05	-0.42
	350	1.08	0.41	1.09	5.06	13.49	57.21M (14.8)	-0.32	-3.93
	400	1.03	0.63	0.45	11.95	8.53	60.59M (9.7)	-0.09	-1.71
$\mathbf{W}_{all}$	*10*	1.72	14.48	13.67	27.08	25.56	1.35M (98.0)	14.03	26.23
	50	1.68	7.6	7.42	15.49	15.12	6.76M (89.9)	10.0	20.38
	100	1.56	5.25	4.48	12.04	10.28	13.52M (79.9)	5.88	13.5
	200	1.39	3.35	2.67	10.28	8.19	27.03M (59.7)	1.99	6.11
	250	1.32	2.58	1.76	9.52	6.51	33.79M (49.6)	0.86	3.17
	300	1.21	1.45	1.49	6.7	6.91	40.55M (39.6)	0.32	1.47
	350	1.15	1.95	0.81	12.08	5.06	47.31M (29.5)	-0.18	-1.12
	400	1.08	1.18	1.0	11.09	9.39	54.07M (19.4)	-0.14	-1.28

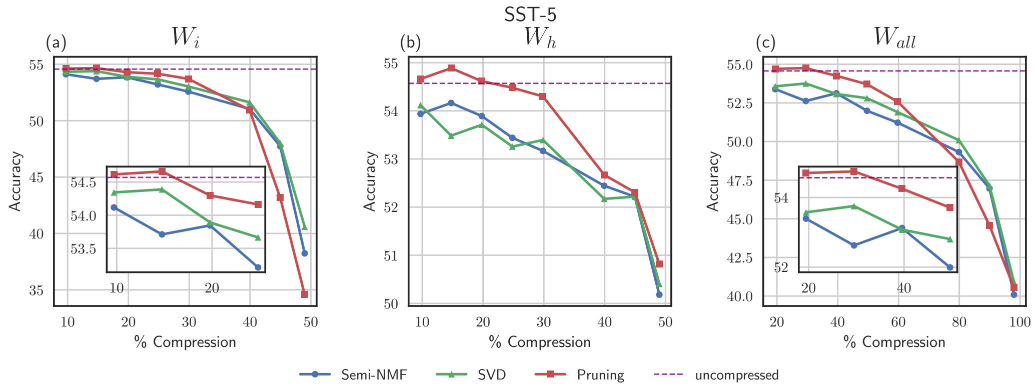

Figure 10: SST-5: comparison between Semi-NMF, SVD and pruning. Perplexity versus LSTM compression rate, in (a) $\mathbf{W}_i$ (b) $\mathbf{W}_h$ (c) $\mathbf{W}_{all}$.

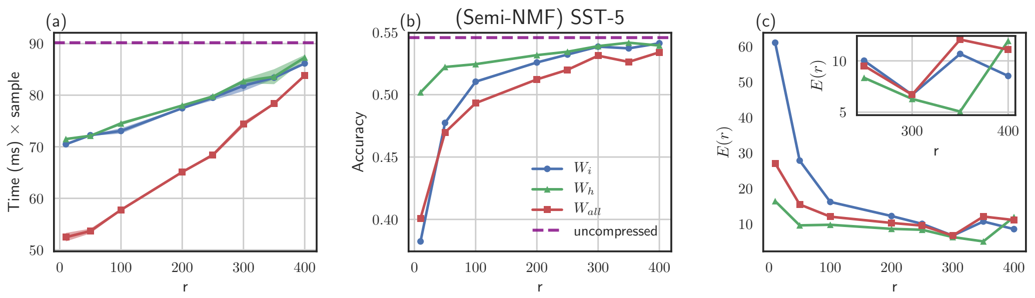

Figure 11: Semi-NMF compression results in SST-5 for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

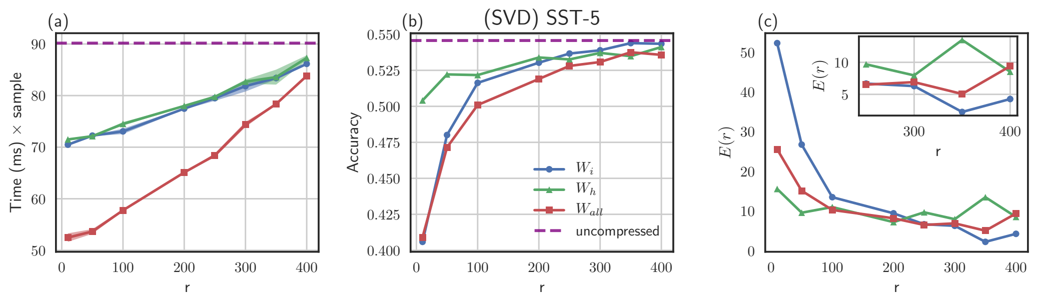

Figure 12: SVD compression results in SST-5 for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

Table 10: SNLI textual entailment results.

SNLI									
Rank		***Speedup***	***Acc Drop***		$E(r)$		***Param.***	***Acc Drop***	$E(r)$
r		$\frac{T_{Unc}}{T_r}$	***Semi-NMF***	***SVD***	***Semi-NMF***	***SVD***	***M(%)***	***Pruning***	
uncompressed		153.12 (ms)	88.52		-		67.11M	88.52	-
$\mathbf{W}_i$	*10*	1.4	11.43	10.36	26.36	23.89	34.23M (49.0)	14.84	34.22
	50	1.37	1.98	1.7	4.99	4.27	36.93M (45.0)	4.7	11.82
	100	1.37	0.79	0.97	2.25	2.74	40.31M (39.9)	1.13	3.2
	200	1.25	0.41	0.42	1.54	1.58	47.07M (29.9)	0.26	1.0
	250	1.26	0.23	0.23	1.07	1.07	50.45M (24.8)	0.2	0.93
	300	1.16	0.36	0.11	2.03	0.64	53.83M (19.8)	0.12	0.7
	350	1.11	0.15	0.14	1.17	1.09	57.21M (14.8)	0.01	0.08
	400	1.07	0.07	0.03	0.83	0.35	60.59M (9.7)	0.03	0.35
$\mathbf{W}_h$	*10*	1.33	1.27	1.24	2.93	2.86	34.23M (49.0)	1.01	2.32
	50	1.31	0.62	0.54	1.56	1.36	36.93M (45.0)	0.57	1.43
	100	1.3	0.32	0.42	0.89	1.18	40.31M (39.9)	0.36	1.01
	200	1.2	0.21	0.19	0.81	0.73	47.07M (29.9)	0.19	0.73
	250	1.17	0.11	0.07	0.51	0.32	50.45M (24.8)	0.06	0.28
	300	1.11	0.04	0.1	0.23	0.58	53.83M (19.8)	0.01	0.06
	350	1.08	0.05	0.08	0.39	0.62	57.21M (14.8)	0.05	0.39
	400	1.06	0.06	0.05	0.71	0.59	60.59M (9.7)	-0.02	-0.24
$\mathbf{W}_{all}$	*10*	2.16	12.44	10.38	14.34	11.97	1.35M (98.0)	10.84	12.5
	50	1.95	2.54	2.28	3.2	2.86	6.76M (89.9)	3.8	4.77
	100	1.85	1.41	1.32	2.0	1.87	13.52M (79.9)	1.64	2.32
	200	1.59	0.53	0.52	1.0	0.98	27.03M (59.7)	0.4	0.75
	250	1.5	0.44	0.27	1.0	0.63	33.79M (49.6)	0.14	0.32
	300	1.32	0.39	0.25	1.1	0.73	40.55M (39.6)	0.05	0.15
	350	1.2	0.32	0.13	1.21	0.51	47.31M (29.5)	0.04	0.16
	400	1.15	0.33	0.19	1.89	1.12	54.07M (19.4)	0.02	0.12

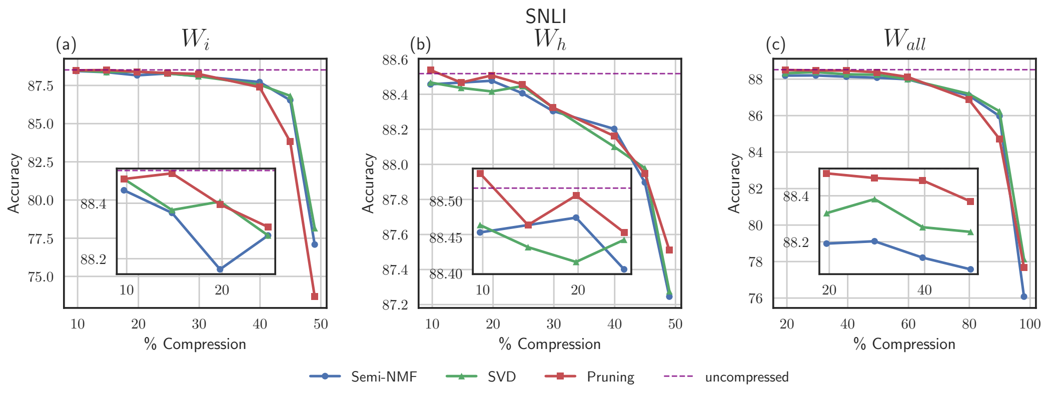

Figure 13: SNLI: comparison between Semi-NMF, SVD and pruning. Perplexity versus LSTM compression rate, in (a) $\mathbf{W}_i$ (b) $\mathbf{W}_h$ (c) $\mathbf{W}_{all}$.

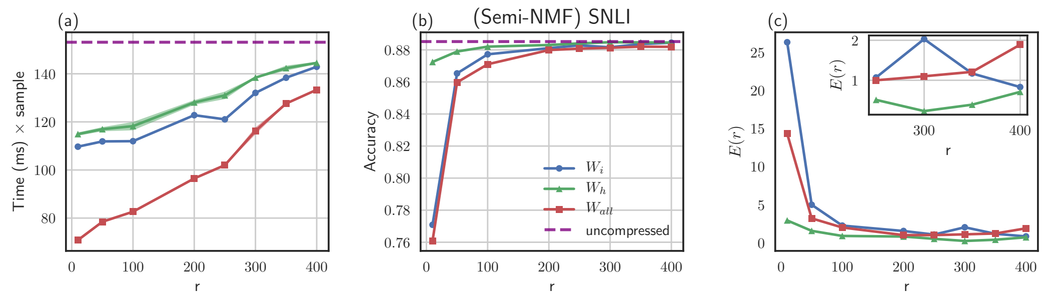

Figure 14: Semi-NMF compression results in SNLI for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

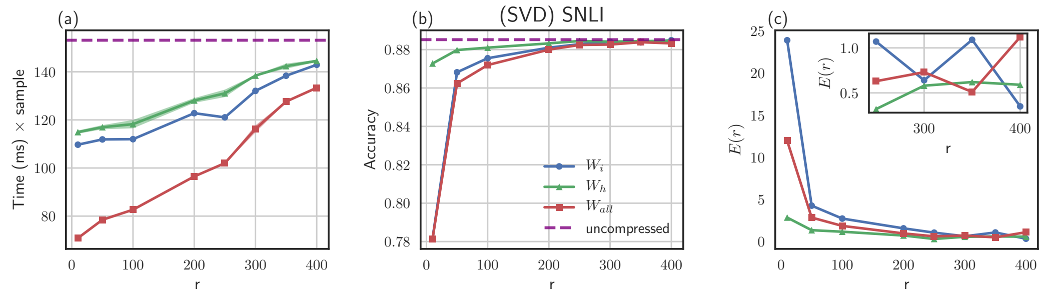

Figure 15: SVD compression results in SNLI for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

Table 11: SQuAD question answering results.

SQuAD									
Rank		***Speedup***	***F1 Drop***		$E(r)$		***Param.***	***F1 Drop***	$E(r)$
r		$\frac{T_{Unc}}{T_r}$	***Semi-NMF***	***SVD***	***Semi-NMF***	***SVD***	***M(%)***	***Pruning***	
uncompressed		747.85 (ms)	81.76		-		67.11M	81.76	-
$\mathbf{W}_i$	*10*	1.31	21.06	24.62	52.59	61.45	34.23M (49.0)	29.73	74.23
	50	1.28	5.46	5.28	14.86	14.36	36.93M (45.0)	10.88	29.59
	100	1.24	2.82	1.83	8.65	5.6	40.31M (39.9)	3.65	11.18
	200	1.18	0.65	0.49	2.68	1.99	47.07M (29.9)	0.66	2.72
	250	1.16	0.41	0.32	2.0	1.57	50.45M (24.8)	0.28	1.36
	300	1.1	0.18	0.06	1.09	0.36	53.83M (19.8)	0.22	1.33
	350	1.07	0.04	-0.0	0.35	-0.02	57.21M (14.8)	0.1	0.85
	400	1.05	-0.03	-0.03	-0.33	-0.38	60.59M (9.7)	0.02	0.3
$\mathbf{W}_h$	*10*	1.29	5.16	5.04	12.89	12.57	34.23M (49.0)	5.21	13.01
	50	1.26	2.43	2.38	6.61	6.47	36.93M (45.0)	2.16	5.87
	100	1.23	1.55	1.37	4.74	4.21	40.31M (39.9)	1.03	3.14
	200	1.16	0.76	0.56	3.12	2.29	47.07M (29.9)	0.22	0.91
	250	1.15	0.53	0.35	2.6	1.72	50.45M (24.8)	0.19	0.92
	300	1.09	0.58	0.29	3.56	1.81	53.83M (19.8)	0.17	1.07
	350	1.07	0.24	0.21	2.02	1.76	57.21M (14.8)	0.14	1.14
	400	1.06	0.2	0.13	2.55	1.62	60.59M (9.7)	0.1	1.3
$\mathbf{W}_{all}$	10	1.97	21.36	20.18	26.66	25.19	1.35M (98.0)	23.64	29.51
	50	1.83	6.65	6.67	9.04	9.08	6.76M (89.9)	10.11	13.75
	100	1.71	4.13	3.36	6.33	5.15	13.52M (79.9)	4.61	7.06
	200	1.51	1.75	1.03	3.59	2.1	27.03M (59.7)	1.06	2.17
	250	1.41	1.24	0.79	3.06	1.93	33.79M (49.6)	0.39	0.96
	300	1.26	0.63	0.39	1.95	1.21	40.55M (39.6)	0.18	0.55
	350	1.19	0.43	0.31	1.79	1.29	47.31M (29.5)	0.15	0.61
	400	1.14	0.44	0.14	2.75	0.86	54.07M (19.4)	0.06	0.36

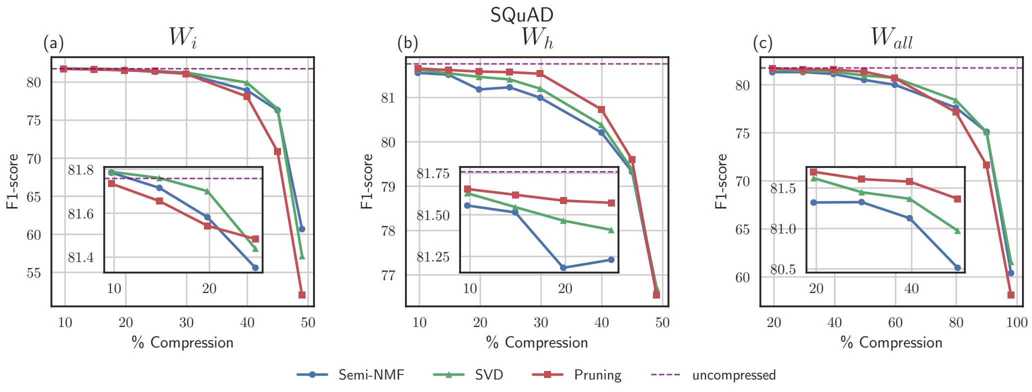

Figure 16: SQuAD: comparison between Semi-NMF, SVD and pruning. Perplexity versus LSTM compression rate, in (a) $\mathbf{W}_i$ (b) $\mathbf{W}_h$ (c) $\mathbf{W}_{all}$.

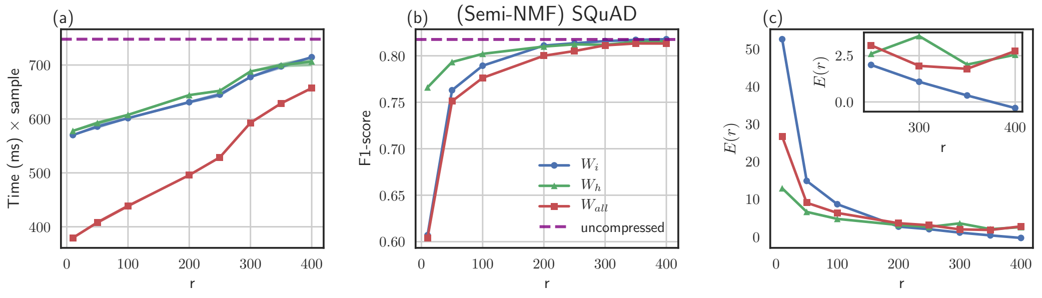

Figure 17: Semi-NMF compression results in SQuAD for $\mathbf{W}_i$, $\mathbf{W}_h$, and $\mathbf{W}_{all}$. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

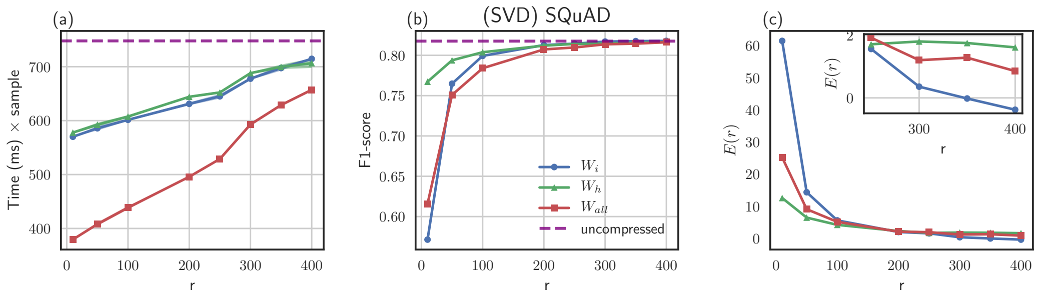

Figure 18: SVD compression results in SQuAD for W_i, W_h, and W_{all}. Rank r versus (a) Time per sample (b) Perplexity (c) Efficiency $E(r)$.

Table 12: Conversion from rank to pruning.

Rank	***PTB/WT2***		***ELMo***
	$\mathbf{W}_i$	$\mathbf{W}_h$	$\mathbf{W}_i$/$\mathbf{W}_h$
10	98.4%	98.7%	97.9%
50	92.4%	93.9%	89.9%
100	84.9%	87.9%	79.8%
200	69.8%	75.9%	59.7%
250	62.3%	69.9%	49.6%
300	54.8%	63.9%	39.5%
350	47.3%	57.8%	29.5%
400	39.7%	51.8%	19.4%

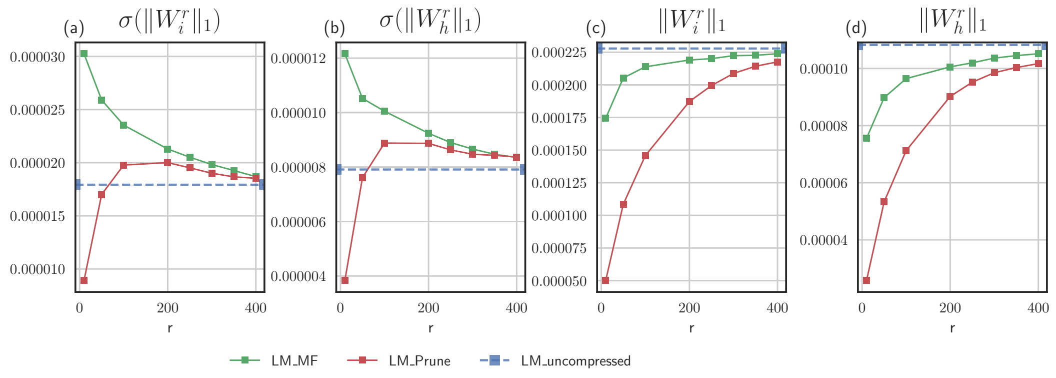

Figure 19: Norm analysis comparisons between MF and Pruning in Language Modeling (PTB). Rank versus (a)$\sigma(\|\mathbf{W}_i\|_1)$ (b) $\sigma(\|\mathbf{W}_h\|_1)$ (c) $\|\mathbf{W}_i\|_1$ (d) $\|\mathbf{W}_h\|_1$.

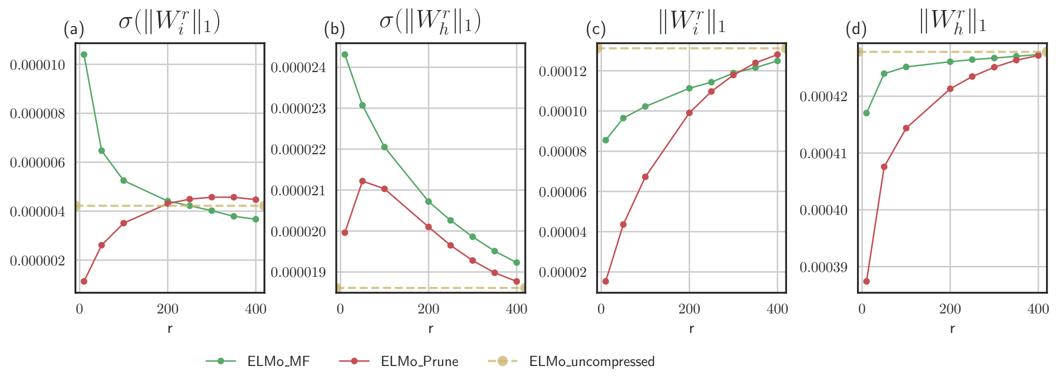

Figure 20: Norm analysis comparisons between MF and Pruning in ELMo. Rank versus (a)$\sigma(\|\mathbf{W}_i\|_1)$ (b) $\sigma(\|\mathbf{W}_h\|_1)$ (c) $\|\mathbf{W}_i\|_1$ (d) $\|\mathbf{W}_h\|_1$.

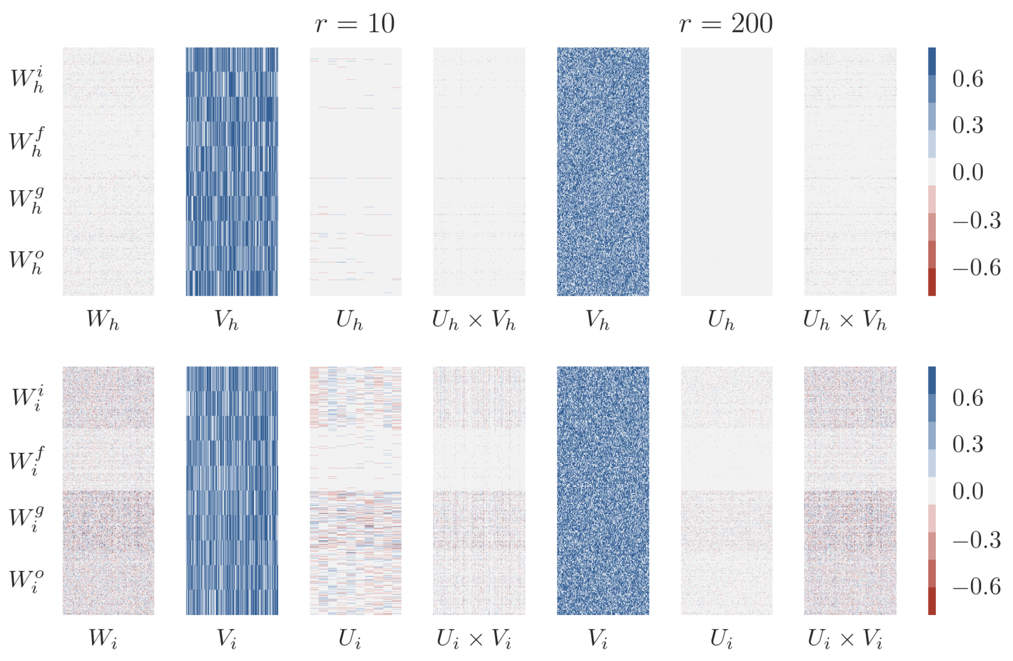

Figure 21: Heatmap LSTM weights on PTB

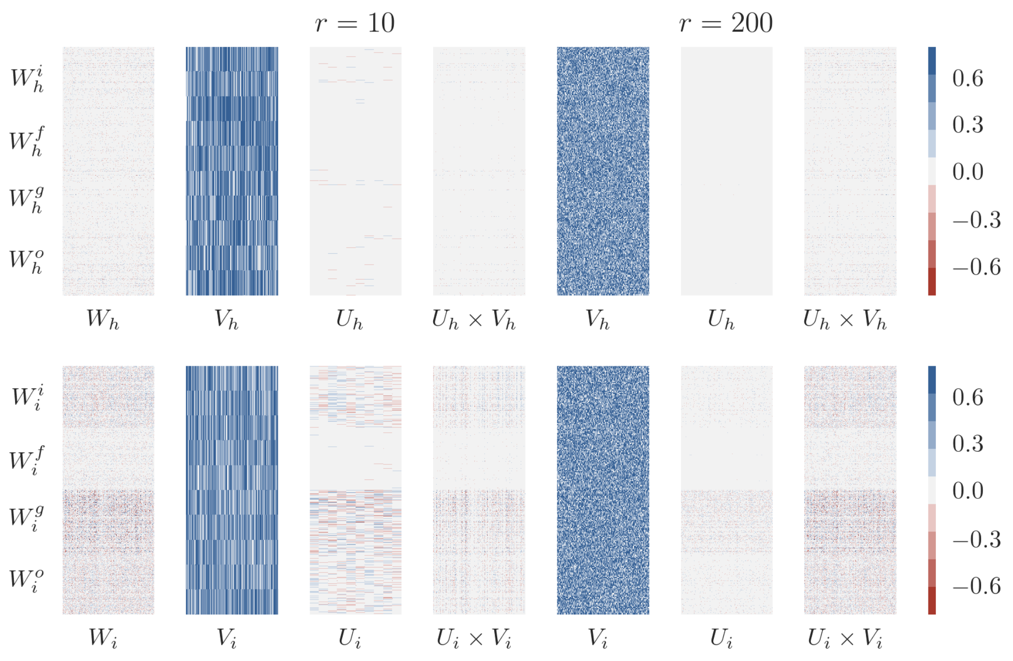

Figure 22: Heatmap of LSTM weights on WikiText-2

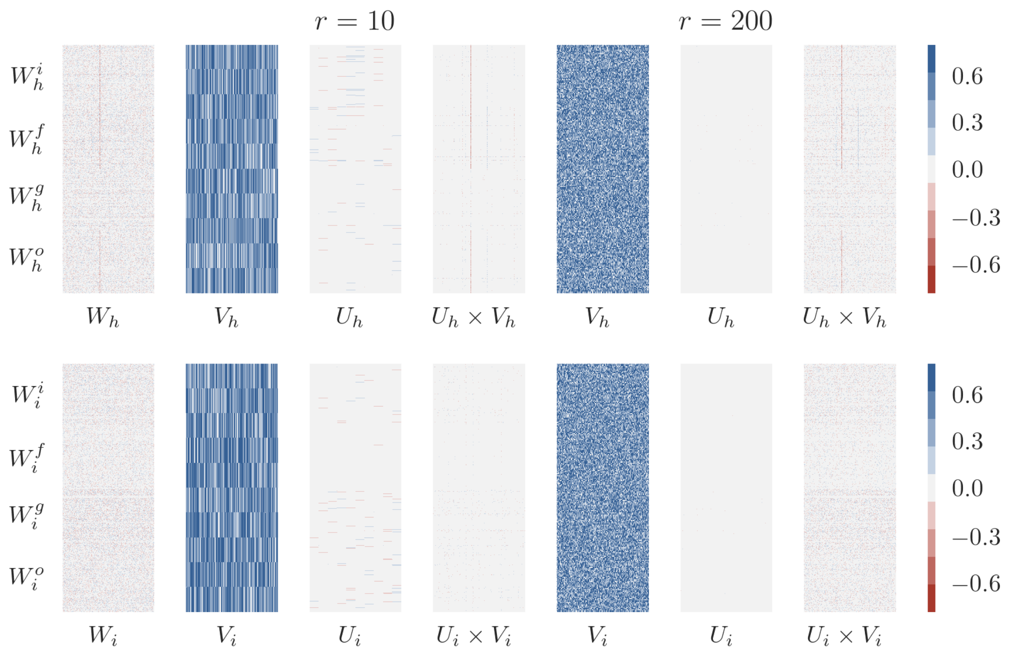

Figure 23: Heatmap of ELMo forward weights

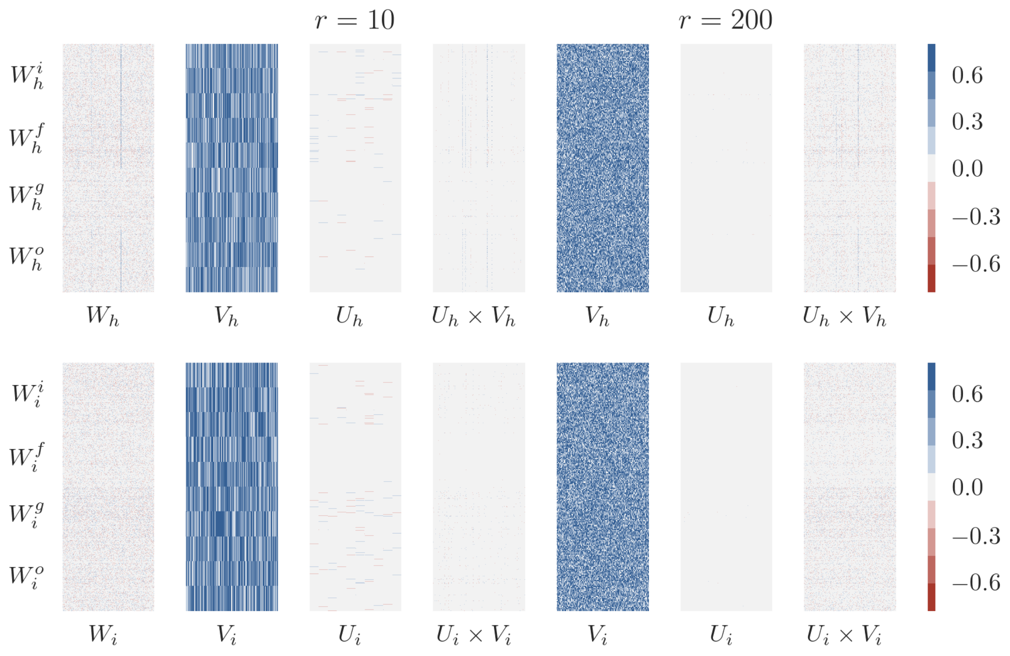

Figure 24: Heatmap of ELMo backward weights