IN-TRAINING MATRIX FACTORIZATION FOR PARAMETER-FRUGAL NEURAL MACHINE TRANSLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose the use of in-training matrix factorization to reduce the model size for neural machine translation. Using in-training matrix factorization, parameter matrices may be decomposed into the products of smaller matrices, which can compress large machine translation architectures by vastly reducing the number of learnable parameters. We apply in-training matrix factorization to different layers of standard neural architectures and show that in-training factorization is capable of reducing nearly 50% of learnable parameters without any associated loss in BLEU score. Further, we find that in-training matrix factorization is especially powerful on embedding layers, providing a simple and effective method to curtail the number of parameters with minimal impact on model performance, and, at times, an increase in performance.

1 INTRODUCTION

While neural models for machine translation have realized considerable breakthroughs in recent years and are now state-of-the-art in many contexts, they are frequently expensive in resources, owing to their large number of parameters. In a context of democratization of deep learning tools, having smaller models that can be learned and/or applied offline on small devices would have immediate and important applications, for example for privacy reasons. However, it is often necessary to leverage deep networks with large layers in order to capture abstract and complex patterns and interactions over the data. We posit that the need for such large parameter matrices is not because they are essential to the representational power of the network. Rather, our hypothesis is that having many parameters can help neural architectures overcome limitations introduced with approximate optimization methods, noisy data supervision, and sub-optimal model architectures. Moreover, recent work has suggested many parameters in large neural architectures are largely superfluous, serving solely to accommodate the stochastic nature of modern machine learning optimization algorithms (Frankle & Carbin, 2019).

Motivated by those hypotheses, we study the application of in-training matrix factorization to neural machine translation. Traditional matrix factorization methods are a simple but powerful technique that has been used after training in other deep learning systems, such as Sainath et al. (2013) for speech recognition and Kim et al. (2015) for computer vision. In contrast to traditional matrix factorization techniques, in-training matrix factorization reduces the number of learnable parameters at training time, lessening the need for computational resources.

The main contributions of this work are:

1. We formally define in-training matrix factorization and present a technique to utilize matrix factorization during training time.
2. We conduct sets of experiments on two standard neural machine translation architectures: the LSTM encoder-decoder and the transformer network.
3. We show that in-training factorization can decrease a model's size by half with no impact on performance, and at times, can improve model performance

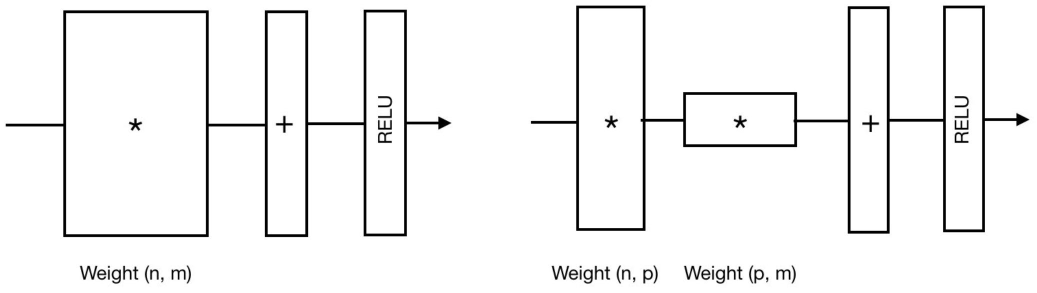

Figure 1: A diagram of in-training factorization. A weight matrix is replaced by the product of two weight matrices, followed by the bias and activation.

2 Approach

2.1 Traditional post-training matrix factorization

Traditionally, matrix factorization has been used as a compression method for a pretrained model. In this case one must obtain a low-rank approximation of a matrix M. A solution is to follow the Eckart-Young-Mirsky theorem (Eckart & Young, 1936) and use singular-value decomposition (SVD).

If $M = UDV^{\top}$ with $U \in \mathbb{R}^{n,n}$, $V \in \mathbb{R}^{m,m}$ and $D = Diag(\lambda_1, ..., \lambda_m) \in \mathbb{R}^{n,m}$ with (λ_i) in decreasing order, then by keeping the top p diagonal elements of D and the corresponding matrix blocks in U and V we get the best p-rank approximation of M for the L_2 (Frobenius) norm.

In practice, because this factorization disturbs the computation graph, the result of this factorization does not always reproduce the non-factorized performance. Thus, practitioners generally must, additionally, refine their compressed models by relearning the pretrained and factorized parameter matrices.

This form of post-training matrix factorization suffers from two core limitations. First, the original uncompressed model must be learned. This results in increased training time, computational resources, and cost to model development. Second, the factorized model must be retrained before it can be deployed, which, again, increases training time, computational resources, and cost.

2.2 In-training factorization

It is, however, even easier to initialize the neural network in factorized form, and perform training from there. The idea of in-training matrix factorization is straightforward: wherever the computation graph involves multiplication by a (n, m) parameter matrix M, we replace M by the product of two matrices of sizes (n, p) and (p, m), where $p << min(n, m)$. A diagram of in-training factorization is provided in Figure 1. Algebraically speaking, this is a p-rank approximation of M. We refer to n, m, and p as the *left*, *right* and *inner* sizes, respectively, of the in-training factorized matrix.

In terms of implementation, we replace one linear layer with two, with a single output bias and activation after the second layer. Thus, this method replaces (nm)-many parameters with $p*(n+m)$-many parameters. Hence, when $p < \frac{nm}{n+m}$, we reduce the number of parameters. For square matrices where $n = m$, the limit is $p < \frac{m}{2}$. In our experiments, we always set $p \leq \frac{min(n,m)}{2}$.

Even though it may seem that the structure of the network has not changed, as multiplying the two matrices would result in a standard layer with a lower-rank weight matrix, this form seems very adapted to optimization constraints. Applied on embedding and projection layers especially, this simple technique is surprisingly effective, and the change is almost transparent. It can also be combined with weight tying (Press & Wolf, 2017) without difficulty.

3 Experiments

3.1 Dataset

Unless specified otherwise, we conduct experiments using the English-to-German task of the IWSLT 2014 dataset. It consists of transcripts of TED talks translated into several languages, with sentence-to-sentence correspondences. The training, validation, and test sets contain 153326, 6969, and 6750 sentences, respectively. We perform additional experiments using the English-to-Portuguese and English-to-Turkish language pairs from the same dataset.

3.2 Models

We run separate sets of experiments on two standard neural machine translation architectures: an LSTM encoder-decoder and a transformer network.

3.2.1 LSTM encoder-decoder

This architecture mostly follows Luong et al. (2015). We use a 3-layer bidirectional encoder with hidden size 256 in each direction, and a 3-layer decoder with size 512. The embedding size is 256 and the embedding layer of the decoder is tied with the projection layer, following Press & Wolf (2017). Beam search is used for inference. The vocabulary size is 30k for the source, 46k for the target.

3.2.2 Transformer

This model (Vaswani et al., 2017) makes use of multi-head attention and self attention rather than a recurrent architecture. We adapt the implementation provided by Klein et al. (2017), including the learning rate schedule, with smaller parameters, which are provided in Table 4. We accumulate 8 mini-batches of 12 to 16 sentences between gradient updates to arrive at the total batch sizes we're reporting. The reported experiments do not tie the embedding and projection layers – which when tried did not yield satisfying results. This model uses subwords and has 15k subword vocabulary for source and target language, respectively.

3.3 Methodology

For a given model, we run in-training factorization experiments with different inner sizes p, on different layers. We add comparisons with non-factorized models of equivalent size p. We compare our method with a compression baseline proposed by See et al. (2016), based solely on the pruning of small elements for compression purposes. However, our method has the crucial advantage of being compatible with modern GPU computation, while pruning creates sparse matrices that do not benefit as much from GPU acceleration. Additionally, we run post-training factorization with a variety of schemes and compare results with pruning and in-training factorization.

We hold batch sizes constant when training across the various compression paradigms. We do so in order to compare compression methods in training environments that are as similar as possible. We note, however, that in-training factorization provides the ability to increase the batch size by compressing the model during training. It has been shown that larger batch sizes can decrease training time and increase performance, especially on the transformer model (Popel & Bojar, 2018). Larger batch sizes can be simulated by accumulating mini-batch gradients before performing the gradient update, which aids in performance; however, accumulating gradients does not efficiently parallelize operations as can a GPU with larger batches.

4 Results and Analysis

4.1 In-training factorization

Table 1 provides results and comparisons for in-training factorization with the LSTM encoder-decoder model. Tables 2 and 5 provide results and comparisons for the transformer model. We observe a number of findings from the results.

Table 1: Results of factorization methods for the LSTM encoder-decoder model. “Pruned” refers to a model where the smallest x% of parameters have been pruned. “In-training” refers to the in-training factorization method described in Section 2.2 performed on the embedding/projection layer.

Compression method	**Size reduction**	**BLEU**	**BLEU non-factorized**
None, embedding=256 (baseline)	0%	-	26.70
Pruned, embedding=256	-40%	26.61	-
Pruned, embedding=256	-56%	24.91	-
In-training, inner size=128	-32%	26.0	26.47
In-training, inner size=64	-48%	26.75	25.34
In-training, inner size=32	-56%	26.63	22.69
In-training, inner size=16	-60%	24.72	11.74

Table 2: Results of in-training factorization for the transformer architecture. “Pruned” refers to a model where the smallest x% of parameters have been pruned. “In-training (embed)” refers to in-training factorization performed on the embedding and projection layers. “In-training (+feed-forward)” refers to in-training factorization performed on the embedding, projection, and feed forward layers. “In-training (+attention)” refers to in-training factorization performed on the embedding, projection, feed forward, and attention layers.

Compression method	**Size reduction**	**BLEU**	**BLEU non-factorized**
None, embedding=512 (baseline)	0%	-	25.95
Pruned, embedding=512	-27%	25.67	-
Pruned, embedding=512	-51%	23.15	-
In-training (embed), inner size=256	-19%	26.22	25.65
In-training (embed), inner size=128	-27%	26.11	23.88
In-training (embed), inner size=64	-51%	25.55	20.01
In-training (embed), inner size=32	-55%	24.75	12.42
In-training (+feed-forward), inner size=256	-32%	25.58	25.65
In-training (+feed-forward), inner size=128	-54%	25.39	23.88
In-training (+feed-forward), inner size=64	-64%	25.24	20.01
In-training (+feed-forward), inner size=32	-70%	22.43	12.42
In-training (+attention), inner size=256	-22%	26.12	23.58

4.1.1 PARAMETER REDUCTION

We observe that in-training factorization can bring a significant reduction of the number of parameters with minimal impact to performance. With the LSTM we can reduce model size by 48% with no drop in BLEU score, and we can reduce model size by 56% and lose less than 0.1 BLEU points. With the transformer model, we can reduce model size by 27% with no drop in BLEU score, and we can reduce the model size by 51% with only a 0.4 drop in BLEU score. By comparison, the same size reductions through pruning decreases the BLEU score by 1.8 BLEU points on the LSTM model and 2.8 BLEU points on the transformer model.

4.1.2 TRANSFORMER PERFORMANCE IMPROVEMENT

We further observe that limited amounts of in-training factorization seem to improve performance with the transformer model. Reducing the model size by 19% and 27% via in-training factorization on the embedding and projection layers improves the BLEU score of the model by 0.3 and 0.2 points, respectively. Additionally, compressing the model by 22% via in-training factorization across the embedding, projection, feed-forward, and attention layers yields a 0.2 increase in BLEU score.

We hypothesize that, for the transformer model, in-training factorization serves as a form of regularization to reduce overfitting. As the transformer model is more sensitive to hyperparameter tuning

Table 3: Factorized and non-factorized transformer performance by language. The non-factorized model uses an embedding dimension of 512, and the factorized model uses an inner size of 256.

	BLEU with in-training factorization	BLEU without factorization
German	**26.22**	25.95
Portuguese	**22.63**	21.56
Turkish	**12.05**	11.95

Table 4: Transformer model parameters for each language.

	Layers	**Embedding dim.**	**Feed-forward dim.**	**Attention dim.**	**Total batch size**
German	3	512	1024	512	128
Portuguese	4	512	512	512	96
Turkish	5	512	512	512	96

than the LSTM model, as well as showing to be more prone to overfitting, this behavior, while unexpected, is not entirely surprising.

To test our hypothesis that in-training factorization can increase the performance of the transformer model, we perform the same experiment comparing a transformer model with factorized embeddings of dimension 512 and inner size of 256, with a non-factorized baseline model with embeddings of dimension 512 on two other languages of the IWSLT dataset: Portuguese (79525 training sentences) and Turkish (193734 training sentences). We report our results in Table 3. We observe that in all language pairs evaluated, in-training factorization improves the performance of the transformer model. This evidences that the performance gains observed with in-training factorization generalize across languages.

4.1.3 INFLUENCE ON TRAINING TIME

We additionally examine the impact of in-training factorization on training time for the transformer model. Compressing the model size allows for larger batches to to be stored in the same amount of memory. Thus, if we compress a model by greater than 50%, then we should be able to double the batch size. We compare the time to convergence for our baseline non-factorized transformer model with an embedding dimension of 512 with an in-training factorized model. The factorized model has a 51% reduction in model size, and thus we can double the batch size than that of the non-factorized model. The results of this experiment are provided in Figure 2. We observe that the factorized transformer model converges about 20% faster than the baseline model. This confirms the findings from Popel & Bojar (2018) that increasing the batch size can decrease training time.

4.2 POST-TRAINING COMPRESSION

4.2.1 POST-TRAINING FACTORIZATION

After performing the in-training experiments, we compare these results with traditional post-training factorization methods. To select rank values for post-training factorization, we compute the singular values of the weight matrices of the transformer model in its different layer functions (embedding and projection, attention, and feed-forward) to determine which are easiest to factorize. We find that the singular values of the weight matrices have very different structures. Those of the embedding and projection layers are dominated by the biggest singular value, around 5 times bigger than the second biggest. On the contrary, the attention and feed-forward matrices are more balanced. As the main difference between the two kinds of matrices is that there is no bias vector added after the embeddings, our hypothesis is that this large singular value is a form of learned bias that skews the parameters heavily.

We consider a singular value relevant if it is bigger than 10% of the dominant one. As the embedding and projection layers are dominated by one value that is much bigger than the rest, we find that no more than 1% of values are within those bounds, which is coherent with the better results we obtain

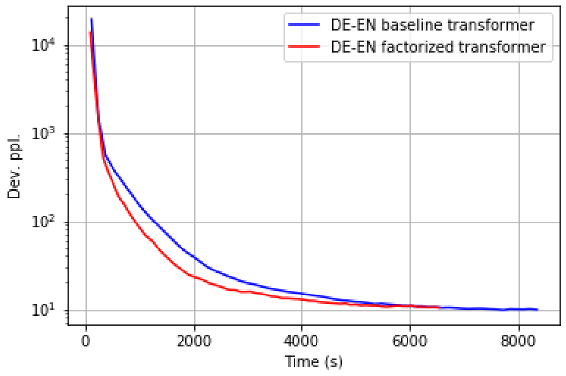

Figure 2: Time evolution of validation perplexity for baseline and in-training factorized transformers with the same GPU memory usage. The factorized model has half the parameters and double the batch size.

Table 5: Comparison of in-training and post-training factorization for the transformer model on the German test set. “In-training (+attention)” refers to in-training factorization applied to the embedding, projection, feed-forward, and attention layers. “Post-training” refers to post-training factorization applied to all weights. “Post-training then pruned” refers to pruning occurring after post-training factorization.

Compression method	**Size reduction**	**BLEU**	**BLEU non-factorized**
None, embedding=512 (baseline)	0%	-	25.95
In-training (+attention)	-22%	26.12	-
Post-training	-22%	26.44	-
Post-training then pruned	-46.8%	21.48	-

on those layers. The attention and feed-forward layers, in contrast, seem well suited to halving their rank (for both, around 35% to 60% of the singular values are relevant according to our heuristic).

We use a rank of 256 (half of the singular values) and attempt to factorize the whole model. We experiment with a variety of compression schemes on the transformer model with those values, and report the results in Table 5. We observe that, similar to the in-training factorization, the post-training factorization also improves the performance of the baseline transformer model. This provides more evidence that matrix factorization can function as a regularizer for models that are prone to overfitting. We additionally observe that the performance of the post-training factorization seems marginally better than the performance of the in-training factorization, although this might be due to additional training.

4.2.2 Post-training factorization and pruning

Lastly, as pruning and factorization have similar goals, we attempt to combine their gains by pruning a post-training factorized model (Table 5). We observe that pruning an already factorized model decreases BLEU score by nearly 5 points from the score of the factorized model (from 26.38 to 21.48). This strongly suggests that the parameter reductions for both methods are, at least partly, redundant.

5 Related work

5.1 Neural Machine Translation

Neural networks have taken an increasing share of sequence-to-sequence models in the past few years. Bahdanau et al. (2015) first use attentional encoder-decoders for machine translation, establishing a new baseline for many incremental additions (Luong et al., 2015; Liu et al., 2016; Gu et al., 2017). A few years later Vaswani et al. (2017) introduced transformer networks using self attention as an efficient alternative for machine translation tasks. State-of-the-art versions of these models are usually applied on subword units obtained through methods like Byte-Pair Encoding (Britz et al., 2017).

5.2 Model compression

In order to save on parameters, See et al. (2016) propose to use *parameter pruning* to reduce model size by replacing the n% smallest elements in the network with 0. They show that they can reduce the number of weights in a LSTM encoder-decoder by 40% without significant drop in performance – which we reproduced. Murray & Chiang (2015) is the equivalent of our training time compression for pruning, learning which weights are necessary as training goes on and pruning them accordingly. Other compression methods involve *quantization* at inference (Wu et al., 2016; Devlin, 2017; Quinn & Ballesteros, 2018) or in training (Micikevicius et al., 2017) ; or for example involve *model distillation* (Kim & Rush, 2016).

5.3 Parameter matrix factorization

Matrix factorization is a wide family of algorithms (Yang, 2015). Accordingly, it has been used in very diverse ways to help compress neural networks. The most common is to use a low-rank approximation to compress Convolutional Neural Networks without loss of performance, as Sainath et al. (2013) do for speech recognition and Kim et al. (2015) for computer vision. Prabhavalkar et al. (2016) use low-rank factorization on inner layers of the LSTM for speech recognition, as does Povey et al. (2018) in a constrained version. The work in Stahlberg & Byrne (2017) is closer to ours, as they factorize trained embedding matrices in encoder-decoders with SVD.

However, all of these uses are post-training compression techniques with a simple application of SVD. We focus on in-training factorization, and compare that approach with the existing post-training factorization paradigm. The closest work that we know of is in Kuchaiev & Ginsburg (2017), which factorizes the LSTM cell matrices during training in a highly optimized framework; however, they do so at the cost of an increase of perplexity. On the other hand, we focus on embedding and attention matrices, for which this work hasn't been performed before to the best of our knowledge, with no cost, or even small gains, in performance.

6 Future work

A limitation of our paper is that it does not explore how to select the hidden sizes for in-training factorized models. Exploring a priori strategies for selecting hidden sizes for in-training factorization could be a valuable extension of our work. Another extension of our work could include analyzing why transformer models consistently improve performance when learned with in-training factorization. It may also be constructive to examine whether our results generalize to larger machine translation corpora. Other potential areas of research could include studying the impact of in-training factorization on other natural language processing tasks, including classification tasks and word2vec embeddings. Finally, a robust comparison between in-training factorization and post-training factorization on model performance would likely be productive.

7 Conclusion

We have introduced a method to use matrix factorization at training time to reduce the parameter footprint of neural machine translation models. We compare in-training factorization to existing post-hoc parameter reduction methods, including parameter pruning and post-training factorization.

We find that using factorization comes with significant gains in both final performance and number of required parameters. Lastly, we demonstrate the effectiveness of our in-training factorization technique to learn a model with fewer parameters, improved accuracy, and decreased training time.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.0473`.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1442–1451, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1151. URL `https://www.aclweb.org/anthology/D17-1151`.

Jacob Devlin. Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the cpu. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2820–2825. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1300. URL `http://aclweb.org/anthology/D17-1300`.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, Sep 1936. ISSN 1860-0980. doi: 10.1007/BF02288367. URL `https://doi.org/10.1007/BF02288367`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 1053–1062, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-1099`.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *CoRR*, abs/1511.06530, 2015. URL `http://arxiv.org/abs/1511.06530`.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1139. URL `http://aclweb.org/anthology/D16-1139`.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017. doi: 10.18653/v1/P17-4012. URL `https://doi.org/10.18653/v1/P17-4012`.

Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for lstm networks. *CoRR*, abs/1703.10722, 2017.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. Neural machine translation with supervised attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 3093–3102, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL `https://www.aclweb.org/anthology/C16-1291`.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1166. URL `http://aclweb.org/anthology/D15-1166`.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017. URL `http://arxiv.org/abs/1710.03740`.

Kenton Murray and David Chiang. Auto-sizing neural networks: With applications to n-gram language models. *CoRR*, abs/1508.05051, 2015. URL `http://arxiv.org/abs/1508.05051`.

Martin Popel and Ondrej Bojar. Training tips for the transformer model. *CoRR*, abs/1804.00247, 2018. URL `http://arxiv.org/abs/1804.00247`.

Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. pp. 3743–3747, 09 2018. doi: 10.21437/Interspeech.2018-1417.

Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Ian McGraw. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pp. 5970–5974, 2016. doi: 10.1109/ICASSP.2016.7472823. URL `https://doi.org/10.1109/ICASSP.2016.7472823`.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163. Association for Computational Linguistics, 2017. URL `http://aclweb.org/anthology/E17-2025`.

Jerry Quinn and Miguel Ballesteros. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pp. 114–120. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-3014. URL `http://aclweb.org/anthology/N18-3014`.

T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659, May 2013. doi: 10.1109/ICASSP.2013.6638949.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 291–301. Association for Computational Linguistics, 2016. doi: 10.18653/v1/K16-1029. URL `http://aclweb.org/anthology/K16-1029`.

Felix Stahlberg and Bill Byrne. Unfolding and shrinking neural machine translation ensembles. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1946–1956. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1208. URL `http://aclweb.org/anthology/D17-1208`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Jie Yang. Notes on low-rank matrix factorization. *CoRR*, abs/1507.00333, 2015. URL `http://arxiv.org/abs/1507.00333`.