Compression of Neural Machine Translation Models via Pruning

Anonymous ACL submission

Abstract

Neural Machine Translation (NMT), like many other deep learning domains, typically suffers from over-parameterization, resulting in large storage sizes. This paper examines three simple magnitude-based pruning schemes to compress NMT models, namely *class-blind*, *class-uniform*, and *class-distribution*, which differ in terms of how pruning thresholds are computed for the different classes of weights in the NMT architecture. We demonstrate the efficacy of weight pruning as a compression technique for a state-of-the-art NMT system. We show that an NMT model with over 200 million parameters can be pruned by 40% with very little performance loss as measured on the WMT'14 English-German translation task. This sheds light on the distribution of redundancy in the NMT architecture. Our main result is that with *retraining*, we can recover and even surpass the original performance with an 80%-pruned model.

1 Introduction

Neural Machine Translation (NMT) is a simple new architecture for translating texts from one language into another (Sutskever et al., 2014; Cho et al., 2014). NMT is a single deep neural network that is trained end-to-end, holding several advantages such as the ability to capture long-range dependencies in sentences, and generalization to unseen texts. Despite being relatively new, NMT has already achieved state-of-the-art translation results for several language pairs including English-French (Luong et al., 2015b), English-German (Jean et al., 2015a; Luong et al., 2015a; Luong and Manning, 2015; Sennrich et al., 2015), English-Turkish (Sennrich et al., 2015), and English-Czech (Jean et al., 2015b; Luong and Manning, 2016). Figure 1 gives an example of an NMT system.

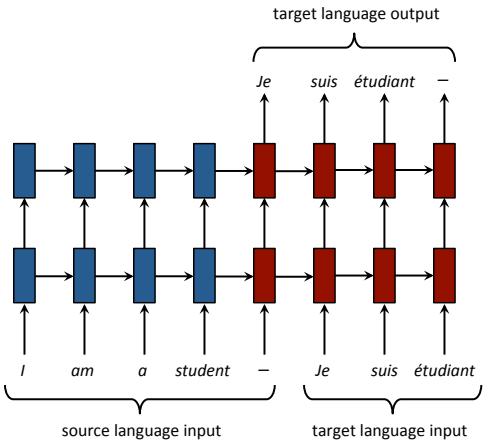

Figure 1: A simplified diagram of NMT.

While NMT has a significantly smaller memory footprint than traditional phrase-based approaches (which need to store gigantic phrase-tables and language models), the model size of NMT is still prohibitively large for mobile devices. For example, a recent state-of-the-art NMT system requires over 200 million parameters, resulting in a storage size of hundreds of megabytes (Luong et al., 2015a). Though the trend for bigger and deeper neural networks has brought great progress, it has also introduced over-parameterization, resulting in long running times, overfitting, and the large storage size discussed above. Thus a solution to the over-parameterization problem could potentially aid all three issues.

Our contribution. In this paper we investigate the efficacy of weight pruning for NMT as a means of compression. We show that despite its simplicity, magnitude-based pruning with retraining is highly effective, and we compare three

magnitude-based pruning schemes—*class-blind*, *class-uniform* and *class-distribution*. Though recent work has chosen to use the latter two, we find the first and simplest scheme—*class-blind*—the most successful. We are able to prune 40% of the weights of a state-of-the-art NMT system with negligible performance loss, and by adding a retraining phase after pruning, we can prune 80% with no performance loss. Our pruning experiments also reveal some patterns in the distribution of redundancy in NMT. In particular we find that higher layers, attention and softmax weights are the most important, while lower layers and the embedding weights hold a lot of redundancy. For the Long Short-Term Memory (LSTM) architecture, we find that at lower layers the parameters for the input are most crucial, but at higher layers the parameters for the gates also become important.

2 Related Work

Pruning the parameters from a neural network, referred to as *weight pruning* or *network pruning*, is a well-established idea though it can be implemented in many ways. Among the most popular are the Optimal Brain Damage (OBD) (Le Cun et al., 1989) and Optimal Brain Surgeon (OBS) (Hassibi and Stork, 1993) techniques, which involve computing the Hessian matrix of the loss function with respect to the parameters, in order to assess the *saliency* of each parameter. Parameters with low saliency are then pruned from the network and the remaining sparse network is retrained. Both OBD and OBS were shown to perform better than the so-called 'naive magnitude-based approach', which prunes parameters according to their magnitude (deleting parameters close to zero). However, the high computational complexity of OBD and OBS compare unfavorably to the computational simplicity of the magnitude-based approach, especially for large networks (Augasta and Kathirvalavakumar, 2013).

In recent years, the deep learning renaissance has prompted a re-investigation of network pruning for modern models and tasks. Magnitude-based pruning (with iterative retraining) has yielded strong results for Convolutional Neural Nets (CNNs) performing visual tasks. (Collins and Kohli, 2014) prune 75% of AlexNet parameters with small accuracy loss on the ImageNet task, while (Han et al., 2015b) prune 89% of AlexNet parameters with no accuracy loss on the ImageNet task.

Other approaches focus on pruning neurons rather than parameters, via sparsity-inducing regularizers (Murray and Chiang, 2015) or 'wiring together' pairs of neurons with similar input weights (Srinivas and Babu, 2015). These approaches are much more constrained than weight-pruning schemes; they necessitate finding entire zero rows of weight matrices, or (near-) identical pairs of rows, before a single neuron can be pruned. By contrast weight-pruning approaches allow weights to be pruned freely and independently of each other. The neuron-pruning approach of (Srinivas and Babu, 2015) was shown to perform poorly (it suffered performance loss after removing only 35% of AlexNet parameters) compared to the weight-pruning approach of (Han et al., 2015b). Though (Murray and Chiang, 2015) demonstrates neuron-pruning for language modeling as part of a (non-neural) Machine Translation pipeline, their approach is more geared towards architecture selection than compression.

There are many other compression techniques for neural networks, including approaches based on on low-rank approximations for weight matrices (Jaderberg et al., 2014; Denton et al., 2014), or weight sharing via hash functions (Chen et al., 2015). Several methods involve reducing the precision of the weights or activations (Courbariaux et al., 2015), sometimes in conjunction with specialized hardware (Gupta et al., 2015), or even using binary weights (Lin et al., 2016). The 'knowledge distillation' technique of (Hinton et al., 2015) involves training a small 'student' network on the soft outputs of a large 'teacher' network. Some approaches use a sophisticated pipeline of several techniques to achieve impressive feats of compression (Han et al., 2015a; Iandola et al., 2016).

Most of the above work has focused on compressing CNNs for vision tasks. We extend the magnitude-based pruning approach of (Han et al., 2015b) to recurrent neural networks (RNN), in particular LSTM architectures for NMT, and to our knowledge we are the first to do so. There has been some recent work on compression for RNNs (Lu et al., 2016; Prabhavalkar et al., 2016), but it focuses on other, non-pruning compression techniques. Nonetheless, our general observations on the distribution of redundancy in a LSTM are corroborated by (Lu et al., 2016).

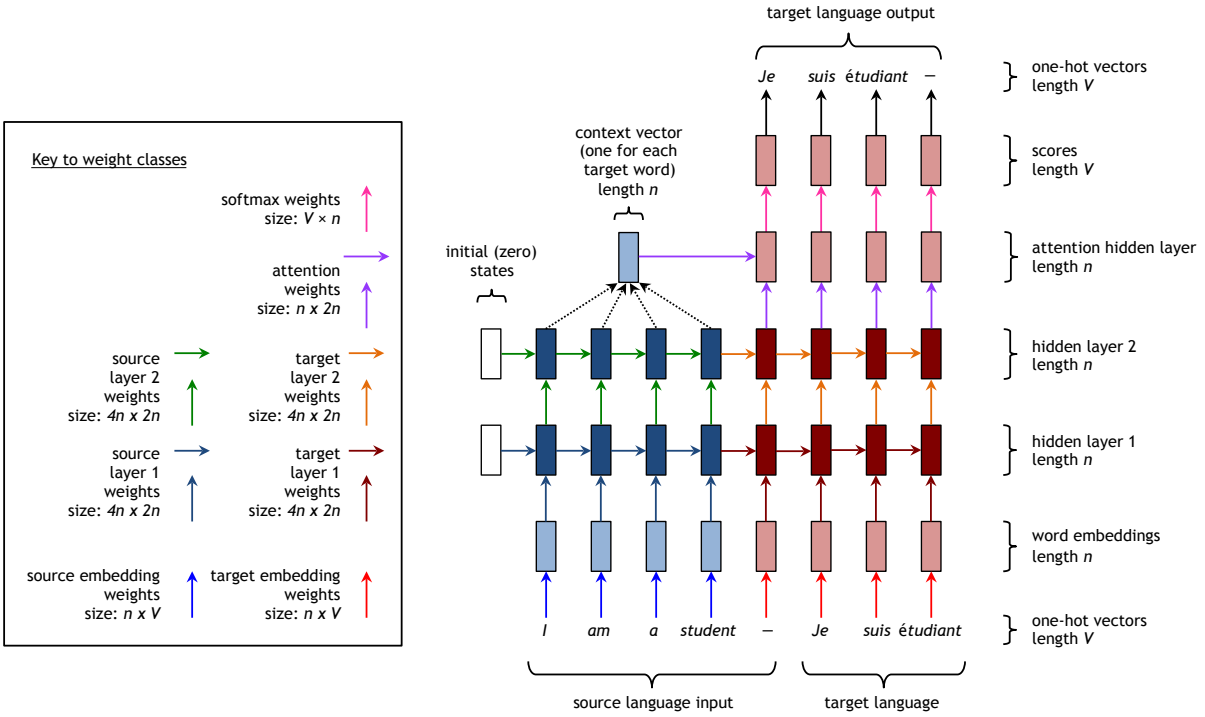

Figure 2: NMT architecture. This example has two layers, but our system has four. The different weight classes are indicated by arrows of different color (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). Best viewed in color.

3 Our Approach

We first give a brief overview of Neural Machine Translation before delving into a model architecture of interest, the deep multi-layer recurrent model with LSTM. We then explain the different types of NMT weights together with our approaches to pruning and retraining.

3.1 Neural Machine Translation

Neural machine translation aims to directly model the conditional probability $p(y|x)$ of translating a source sentence, $x_1, \ldots, x_n$, to a target sentence, $y_1, \ldots, y_m$. It accomplishes this goal through an *encoder-decoder* framework (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). The *encoder* computes a representation $\boldsymbol{s}$ for each source sentence. Based on that source representation, the *decoder* generates a translation, one target word at a time, and hence, decomposes the log conditional probability as:

$$\log p(y|x) = \sum_{t=1}^{m} \log p\left(y_t|y_{<t}, \boldsymbol{s}\right) \quad (1)$$

Most NMT work uses RNNs, but approaches differ in terms of: (a) architecture, which can be unidirectional, bidirectional, or deep multi-layer RNN; and (b) RNN type, which can be Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or the gated recurrent unit (Cho et al., 2014).

In this work, we specifically consider the *deep multi-layer recurrent* architecture with *LSTM* as the hidden unit type. Figure 1 illustrates an instance of that architecture during training in which the source and target sentence pair are input for supervised learning. During testing, the target sentence is not known in advance; instead, the most probable target words predicted by the model are fed as inputs into the next timestep. The network stops when it emits the end-of-sentence symbol—a special 'word' in the vocabulary, represented by a dash in Figure 1.

3.2 Understanding NMT Weights

We show in Figure 2 the same system in more detail, highlighting the different types of parameters, or weights, in the model. We will go through the architecture from bottom to top. First, a vocabulary is chosen for each language, assuming that the top V frequent words are selected. Thus, every word in the source or target vocabulary can be represented by a one-hot vector of length V. The source input sentence and target input sentence, represented as a sequence of one-hot vec-

tors, are transformed into a sequence of word embeddings by the *embedding* weights. These embedding weights, which are learned during training, are different for the source words and the target words. The word embeddings are vectors of length n, the *dimension* of the network.

The word embeddings are then fed as input into the main network, which consists of two multilayer RNNs 'stuck together'—an encoder for the source language and a decoder for the target language, each with their own weights. The *feed-forward* (vertical) weights connect the hidden unit from the layer below to the upper RNN block, and the *recurrent* (horizontal) weights connect the hidden unit from the previous time-step RNN block to the current time-step RNN block.

The hidden state at the top layer of the decoder is fed through an *attention* layer, which guides the translation by 'paying attention' to relevant parts of the source sentence; for more information see Section 3 of (Luong et al., 2015a). Finally, for each target word, the top layer hidden unit is transformed by the *softmax* weights into a score vector of length V. The target word with the highest score is selected as the output translation.

Weight Subgroups in LSTM – For the aforementioned RNN block, we choose to use LSTM as a hidden unit type. To facilitate our discussion later on the different subgroups of weights within LSTM, we first revise details of an LSTM suggested by Zaremba et al. (2014) as follows:

$$\begin{pmatrix} i \\ f \\ o \\ \hat{h} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{4n,2n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix} \quad (2)$$

$$c_t^l = f \circ c_{t-1}^l + i \circ \hat{h} \quad (3)$$

$$h_t^l = o \circ \tanh(c_t^l) \quad (4)$$

Here, each LSTM block at time t and layer l computes as outputs a pair of hidden and memory vectors $(h_t^l,\ c_t^l)$ given the previous pair $(h_{t-1}^l,\ c_{t-1}^l)$ and an input vector h_t^{l-1} (either from the below LSTM block or the embedding weights if $l = 1$). All of these vectors have dimensions of n.

The core of an LSTM block is the weight matrix $T_{4n,2n}$ of size $4n \times 2n$. This matrix can be decomposed into 8 subgroups that are responsible for the interractions between $\{$input gate i, forget gate f, output gate o, input signal $\hat{h}\} \times \{$feed-forward input h_t^{l-1}, recurrent input $h_{t-1}^l\}$.

3.3 Pruning Schemes

We follow the general magnitude-based approach of (Han et al., 2015b), which consists of pruning weights with smallest absolute value. However, we question the authors' pruning scheme with respect to the different weight classes, and experiment with three pruning schemes. Suppose we wish to prune $x\%$ of the total parameters in the model. How do we distribute the pruning over the different weight classes (illustrated in Figure 2) of our model? We propose to examine three different pruning schemes:

1. *Class-blind*: Take all parameters, sort them by magnitude and prune the $x\%$ with smallest magnitude, regardless of weight class. (So some classes are pruned proportionally more than others).

2. *Class-uniform*: Within each class, sort the weights by magnitude and prune the $x\%$ with smallest magnitude. (So all classes have exactly $x\%$ of their parameters pruned).

3. *Class-distribution*: For each class c, weights with magnitude less than $\lambda\sigma_c$ are pruned. Here, σ_c is the standard deviation of that class and λ is a universal parameter chosen such that in total, $x\%$ of all parameters are pruned. This is used by (Han et al., 2015b).

All these schemes have their seeming advantages. Class-blind pruning is the simplest and adheres to the principle that pruning weights (or equivalently, setting them to zero) is least damaging when those weights are small, regardless of their locations in an architecture. Class-uniform pruning and class-distribution pruning both seek to prune proportionally within each weight class, either absolutely, or relative to the standard deviation of that class. We find that class-blind pruning outperforms both other schemes (see Section 4.1).

3.4 Retraining

In order to prune NMT models aggressively without performance loss, we retrain our pruned networks. In our implementation, we keep "mask" matrices, which represent the sparse structure of a network, so as to ignore weights at pruned locations. We detail in Section 4.1 a successful "formula" to retrain pruned NMT models.

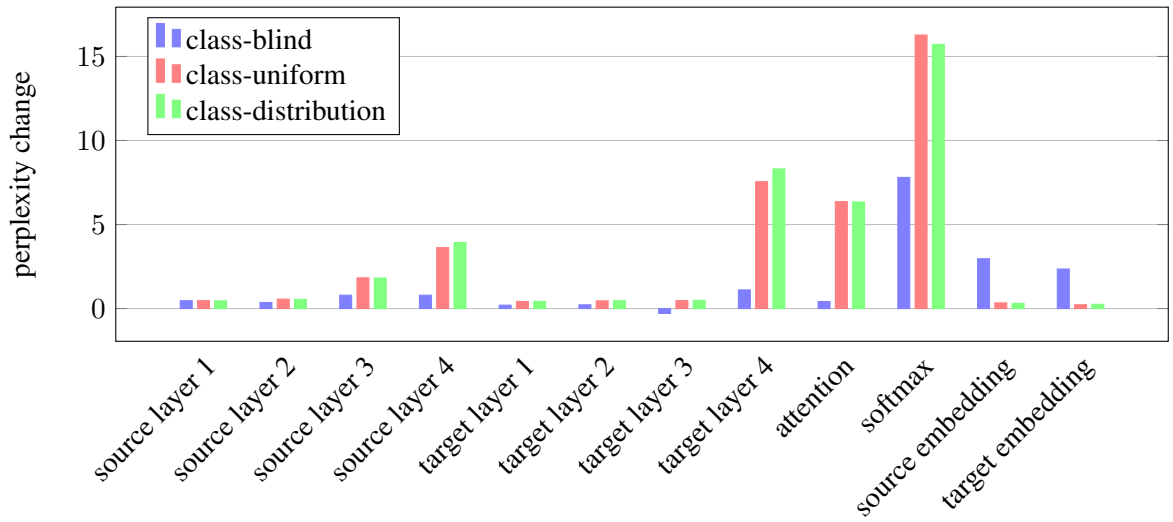

Figure 3: 'Breakdown' of performance loss (i.e., perplexity increase) by weight class, when pruning 90% of weights using each of the three pruning schemes. Each of the first eight classes have 8 million weights, attention has 2 million, and the last three have 50 million weights each.

4 Experiments

We evaluate the effectiveness of our pruning approaches on a state-of-the-art NMT model.[1] Specifically, an attention-based English-German NMT system from (Luong et al., 2015a) is considered. Training data was obtained from WMT'14 consisting of 4.5M sentence pairs (116M English words, 110M German words). For more details on training hyperparameters, we refer readers to Section 4.1 of (Luong et al., 2015a). All models are tested on newstest2014 (2737 sentences). The model achieves a perplexity of 6.1 and a BLEU score of 20.5 (after unknown word replacement).[2]

When *retraining* pruned NMT systems, we use the following settings: (a) we start with a smaller learning rate of 0.5 (the original model uses a learning rate of 1.0), (b) we train for fewer epochs, 4 instead of 12, using plain SGD, (c) a simple learning rate schedule is employed; after 2 epochs, we begin to halve the learning rate every half an epoch, and (d) all other hyperparameters are the same, such as mini-batch size 128, maximum gradient norm 5, and dropout with probability 0.2.

4.1 Comparing pruning schemes

Despite its simplicity, we observe in Figure 4 that *class-blind* pruning outperforms both other

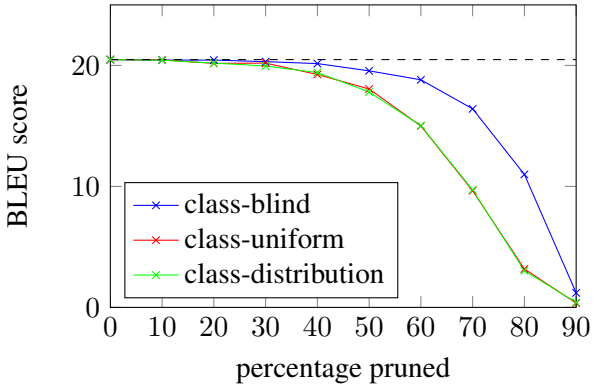

Figure 4: Effects of different pruning schemes.

schemes in terms of translation quality at all pruning percentages. Furthermore, to explain the poor performance of class-uniform and class-distribution pruning, for each of the three pruning schemes, we pruned each class separately and recorded the effect on performance (as measured by perplexity). Figure 3 shows that with class-uniform pruning, the overall performance loss is caused disproportionately by a few classes: target layer 4, attention and softmax weights. Looking at Figure 5, we see that the most damaging classes to prune also tend to be those with weights of greater magnitude—these classes have much larger weights than others at the same percentile, so pruning them under the class-uniform pruning scheme is more damaging. The situation is similar for class-distribution pruning.

By contrast, Figure 3 shows that under class-

[1] We thank the authors of (Luong et al., 2015a) for providing their trained models and assistance in using the codebase at `https://github.com/lmthang/nmt.matlab`.

[2] The performance of this model is reported under row *global (dot)* in Table 4 of (Luong et al., 2015a).

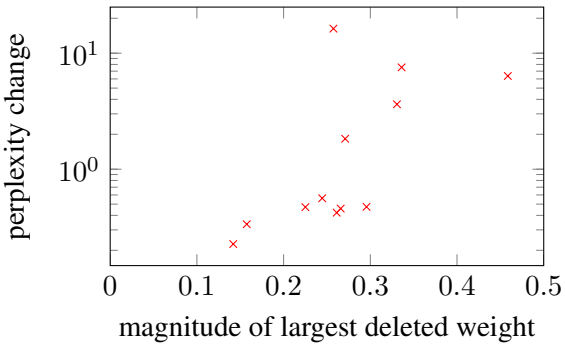

Figure 5: Magnitude of largest deleted weight vs. perplexity change, for the 12 different weight classes when pruning 90% of parameters by class-uniform pruning.

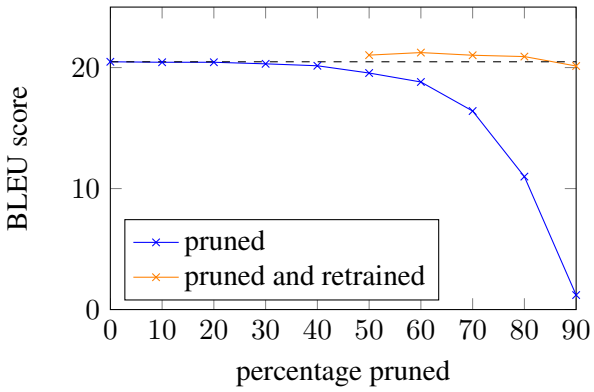

Figure 6: Performance of pruned models, immediately after pruning and after retraining.

blind pruning, the damage caused by pruning softmax, attention and target layer 4 weights is greatly decreased, and the contribution of each class towards the performance loss is overall more uniform. In fact, the distribution begins to reflect the number of parameters in each class—for example, the source and target embedding classes have larger contributions because they have more weights. We use only class-blind pruning for the rest of the experiments.

Figure 3 also reveals some interesting information about the distribution of redundancy in NMT architectures—namely it seems that higher layers are more important than lower layers, and that attention and softmax weights are crucial. We will explore the distribution of redundancy further in Section 4.3.

4.2 Pruning and retraining

Pruning has an immediate negative impact on performance (as measured by BLEU score on the validation set) that is exponential in pruning percentage; this is demonstrated by the blue line in Figure 6. However we find that up to about 40% pruning, performance is mostly unaffected, indicating a large amount of redundancy and over-parameterization in NMT.

We now consider the effect of retraining pruned models. The orange line in Figure 6 shows that after retraining the pruned models, baseline performance (20.48 BLEU) is both recovered and improved upon, up to 80% pruning (20.91 BLEU), with only a small performance loss at 90% pruning (20.13 BLEU). This may seem surprising, as we might not expect a sparse model to significantly out-perform a model with five times as many parameters. There are several possible explanations, two of which are given below.

Firstly, we found that the less-pruned models perform better on the training set than the validation set, whereas the more-pruned models have closer performance on the two sets. This indicates that pruning has a regularizing effect on the retraining phase, though clearly more is not always better, as the 50% pruned and retrained model performs better than the 90% pruned and retrained model. Nonetheless, this regularization effect may explain why the pruned and retrained models outperform the baseline.

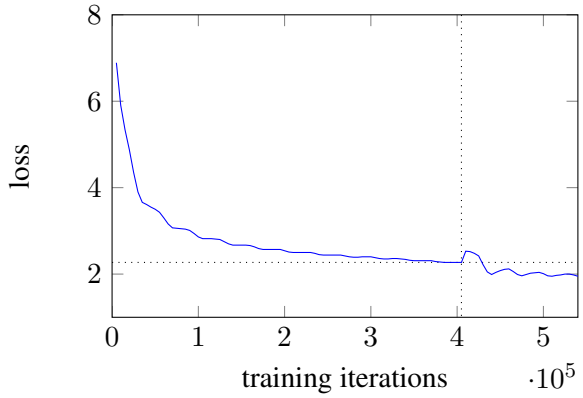

Figure 7: The loss function during training, pruning and retraining. The vertical dotted line marks the point when 80% of the parameters are pruned. The horizontal dotted line marks the best performance of the unpruned baseline.

Alternatively, pruning may serve as a means to escape a local optimum. Figure 7 shows the loss function over time during the training, pruning and retraining process. During the original training

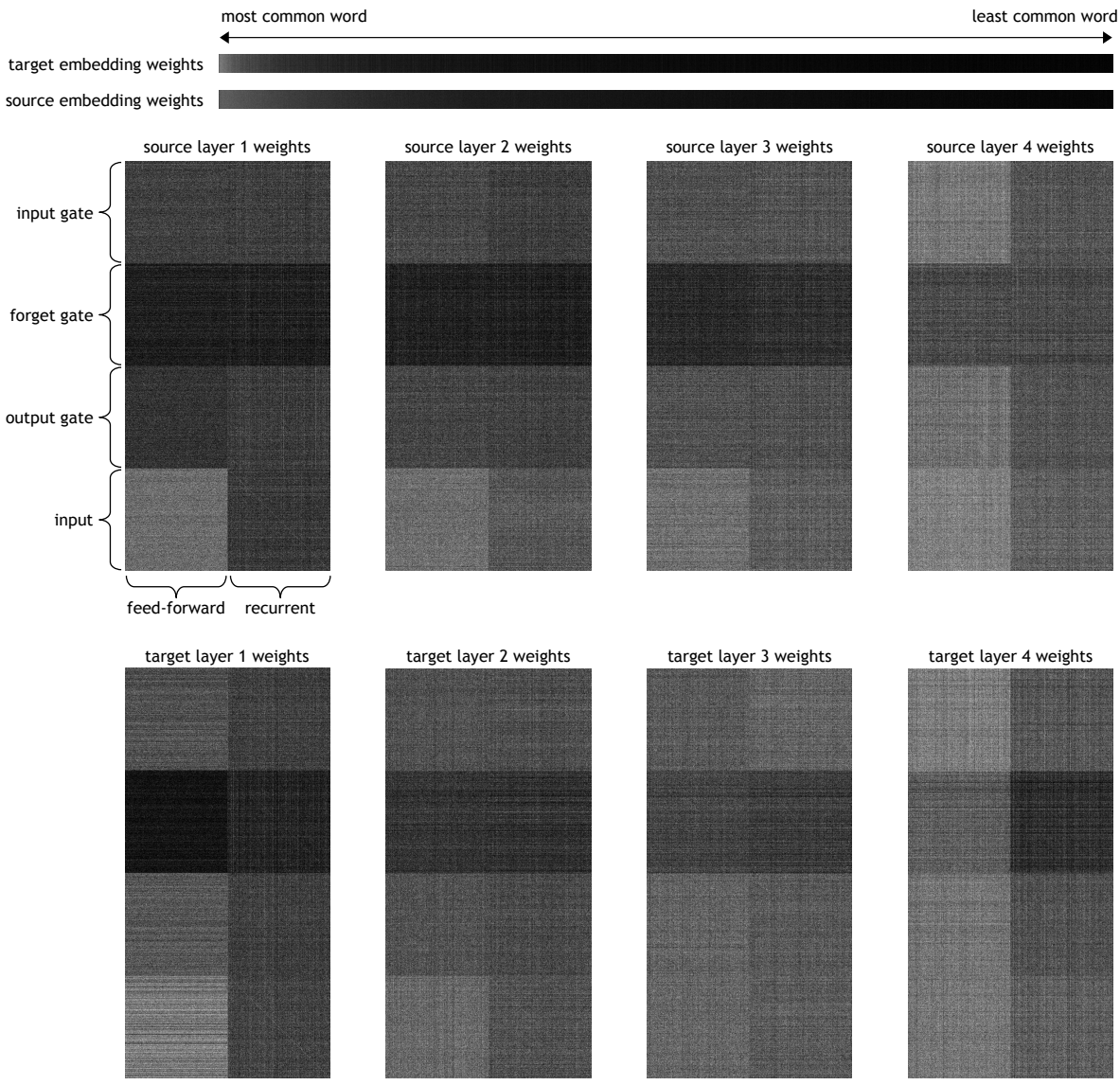

Figure 8: Graphical representation of the location of small weights in the model. Black pixels represent weights with absolute size in the bottom 80%; white pixels represent those with absolute size in the top 20%. Equivalently, these pictures illustrate which parameters remain after pruning 80% using our class-blind pruning scheme.

process, the loss curve flattens out and seems to converge (note that we use early stopping to obtain our baseline model, so the original model was trained for longer than shown in Figure 7). Pruning causes an immediate increase in the loss function, but enables further gradient descent, allowing the retraining process to find a new, better local optimum. It seems that the disruption caused by pruning is beneficial in the long-run.

4.3 Distribution of redundancy in NMT

We visualize in Figure 8 the redundancy structore of our NMT baseline model. *Black* pixels represent weights near to zero; *white* pixels represent larger ones. First we consider the embedding weight matrices, whose columns correspond to words in the vocabulary. Unsurprisingly, in Figure 8, we see that the parameters corresponding to the less common words are more dispensable. In fact, at the 80% pruning rate, for 100 uncommon source words and 1194 uncommon target words, we delete *all* parameters corresponding to that word. This is not quite the same as removing the word from the vocabulary—true out-of-vocabulary words are mapped to the embedding for the 'unknown word' symbol, whereas these 'pruned-out' words are mapped to a zero embedding. However in the original unpruned model these uncommon words already had near-zero embeddings, indicating that the model was unable to learn sufficiently distinctive representations.

Returning to Figure 8, now look at the eight weight matrices for the source and target connections at each of the four layers. Each matrix corresponds to the $4n \times 2n$ matrix $T_{4n,2n}$ in Equation (1). In all eight matrices, we observe—as does (Lu et al., 2016)—that the weights connecting to the input $\hat{h}$ are most crucial, followed by the input gate i, then the output gate o, then the forget gate f. This is particularly true of the lower layers, which focus primarily on the input $\hat{h}$. However for higher layers, especially on the target side, weights connecting to the gates are as important as those connecting to the input $\hat{h}$. The gates represent the LSTM's ability to add to, delete from or retrieve information from the memory cell. Figure 8 therefore shows that these sophisticated memory cell abilities are most important at the *end* of the NMT pipeline (the top layer of the decoder). This is reasonable, as we expect higher-level features to be learned later in a deep learning pipeline.

We also observe that for lower layers, the feed-forward input is much more important than the recurrent input, whereas for higher layers the recurrent input becomes more important. This makes sense: lower layers concentrate on the low-level information from the current word embedding (the feed-forward input), whereas higher layers make use of the higher-level representation of the sentence so far (the recurrent input).

Lastly, on close inspection, we notice several white diagonals emerging within the subsquares of the matrices in Figure 8, indicating that even without initializing the weights to identity matrices, an identity-like weight matrix is learned. At higher pruning percentages, these diagonals become more pronounced.

5 Future Work

The pruning method described in (Han et al., 2015b) includes several iterations of pruning and retraining. Implementing this for NMT would likely result in further compression and performance improvements. If possible it would be highly valuable to exploit the sparsity of the pruned models to speed up training and runtime, perhaps through sparse matrix representations and multiplications. Though we have found magnitude-based pruning to perform very well, it would be instructive to revisit the original claim that other pruning methods (for example Optimal Brain Damage and Optimal Brain Surgery) are more principled, and perform a comparative study.

6 Conclusion

We have shown that weight pruning with retraining is a highly effective method of compression and regularization on a state-of-the-art NMT system, compressing the model to 20% of its size with no loss of performance. Though we are the first to apply compression techniques to NMT, we obtain a similar degree of compression to other current work on compressing state-of-the-art deep neural networks, with an approach that is simpler than most. We have found that the absolute size of parameters is of primary importance when choosing which to prune, leading to an approach that is extremely simple to implement, and can be applied to any neural network. Lastly, we have gained insight into the distribution of redundancy in the NMT architecture.

References

M Gethsiyal Augasta and T Kathirvalavakumar. 2013. Pruning algorithms of neural networksa comparative study. *Central European Journal of Computer Science*, 3(3):105–115.

Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.

Maxwell D Collins and Pushmeet Kohli. 2014. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Low precision arithmetic for deep learning. In *International Conference on Learning Representations (ICLR) Workshop Contribution*.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. *arXiv preprint arXiv:1502.02551*.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR'16 oral)*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143.

Babak Hassibi and David G Stork. 1993. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *Advances in Neural Information Processing Systems (NIPS) Deep Learning Workshop*.

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

Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 1mb model size. *arXiv preprint arXiv:1602.07360*.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Advances in Neural Information Processing Systems (NIPS)*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015a. On using very large target vocabulary for neural machine translation. In *ACL*.

Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015b. Montreal neural machine translation systems for WMT'15. In *WMT*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *EMNLP*.

Yann Le Cun, John S Denker, and Sara A Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural networks with few multiplications. In *International Conference on Learning Representations (ICLR)*.

Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. 2016. Learning compact recurrent neural networks. *arXiv preprint arXiv:1604.02594*.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domain. In *IWSLT*.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015a. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2015b. Addressing the rare word problem in neural machine translation. In *ACL*.

Kenton Murray and David Chiang. 2015. Auto-sizing neural networks: With applications to n-gram language models. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Ian McGraw. 2016. On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.