CS224N Final Project: Exploiting the Redundancy in Neural Machine Translation

Abi See
Stanford University
abisee@stanford.edu

Abstract

Neural Machine Translation (NMT) has enjoyed success in recent years, but like many other deep learning domains, typically suffers from over-parameterization, resulting in overfitting and large storage size. We demonstrate the efficacy of weight pruning as a compression and regularization technique. In particular we find that our baseline model can be immediately compressed to 40% of its previous size with no performance loss, indicating a high level of redundancy, and that with retraining, our baseline model can be compressed to 10% of its previous size with some performance *improvement*. In addition, we investigate the distribution of redundancy in the NMT architecture and the interaction of pruning with other forms of regularization such as dropout.

1 Introduction

Neural Machine Translation (NMT), although a very recent approach, has matched the performance of more established methods such as phrase-based Statistical Machine Translation (Jean et al., 2014; Luong et al., 2015; Sutskever et al., 2014). The advantages of NMT over phrase-based SMT are its simplicity (there is just one system to be optimized, rather than a pipeline of several components) and its smaller storage requirement (NMT does not require the storage of large phrase-tables or language models).

Nonetheless, the storage requirement of NMT, and of deep neural networks in general, is relatively large. For example, the state-of-the-art system described in (Luong et al., 2015) requires over 200 million parameters, resulting in a storage size of hundreds of megabytes. For the potential goal of running neural networks on mobile devices, this is prohibitively large, as an average smartphone in 2015 has a storage size of 16GB or 32GB, with little of that to spare. In addition, deep neural networks tend to be over-parameterized, resulting in long running times, overfitting, and the large storage size described above. Thus a solution to the over-parameterization problem could potentially aid all three issues.

In this paper we investigate the efficacy of weight pruning on NMT, concentrating on its potential as a means of regularization and compression. We also investigate the nature of redundancy in NMT, yielding domain-specific observations regarding the distribution of redundancy in a NMT model.

2 Related Work

Weight pruning is essentially a simple concept, though it can be performed in a variety of ways. It was proposed over two decades ago, for example by (Hassibi and Stork, 1993) in their 'Optimal Brain Surgeon' technique. With the recent resurgence of deep neural networks has come a renewed interest in pruning, both of weights (Han et al., 2015b; Collins and Kohli, 2014), and of neurons (Murray and Chiang, 2015; Srinivas and Babu, 2015).

Other compression techniques have also seen renewed interest, such as weight binarization (Lin et al., 2015) and low-precision multiplications (Courbariaux et al., 2014). (Chen et al., 2015) describe a weight-sharing scheme called HashedNets that randomly groups weights into hash buckets. (Hinton et al., 2015) describes a technique called 'distillation' by which a compact neural network is trained on the output of a larger neural network.

We closely follow the approach of (Han et al., 2015b) to pruning, though while that work (and indeed most of the work mentioned in this section) focuses on Convolutional Neural Networks and their applications to image-processing, we fo-

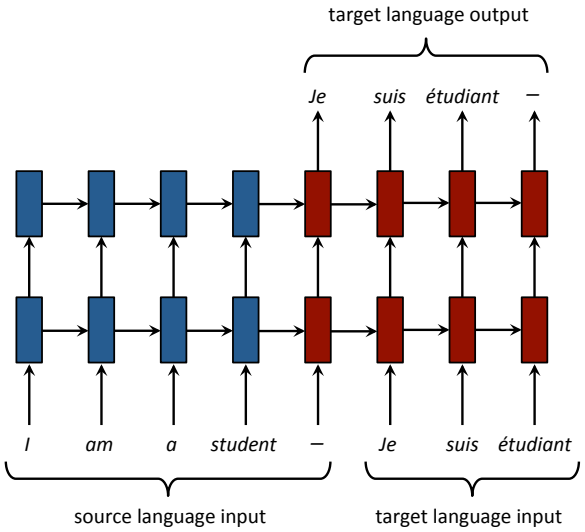

Figure 1: Simplified diagram of NMT architecture.

cus on Long Short-Term Memory and its application to Neural Machine Translation.

3 Neural Machine Translation

The NMT model of (Luong et al., 2015) uses a recurrent neural network architecture to translate sentences to sentences (see Figure 1 for a simplified representation). During training, the source and target sentence pair are fed in as input, and the target output sentence produced by the network is used to evaluate the performance of the model, and perform stochastic gradient descent. During runtime, just the source sentence is fed in—the 'target language input' depicted in Figure 1 is simply the output of the network shifted by one. That is, each word emitted by the network is fed back into the network on 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.

Figure 2 shows the same system in more detail, and in particular highlights the different parameters, or weights, used to describe the model. We will describe it bottom to top. First, a vocabulary is chosen for both the source and target languages (we assume these vocabularies have a common size, V). Thus every word in the source or target vocabulary can be represented by a one-hot vector of length V (actually $V + 2$, to accommodate the 'unknown word' and 'end of sentence' symbols). The source input sentence and target input sentence, represented as a sequence of one-hot vectors, 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 is two Recurrent Neural Networks (RNN) 'stuck together'—one for the source language and one for the target language, each with their own weights. In this example there are two layers. The RNNs have Long Short-Term Memory architecture, which has shown to aid information retention over long sequences (Hochreiter and Schmidhuber, 1997); for more detail, see Section 8. The *feed-forward weights* connect the hidden unit from the layer below (or the word embeddings) to the next LSTM block, and the *recurrent weights* connect the hidden unit from the previous word to the LSTM block. Finally, for each target word, the top layer hidden unit is transformed by the *top layer weights* into a score vector of length V. The target word with the highest score is selected as the output translation.

4 Evaluation metrics

We evaluate our models by two measures: BLEU score and perplexity. BLEU score compares the output target sentence with the gold target sentence, and is measured on a scale from 0 (worst) to 100 (best). Perplexity is the exponential of the average cost per word, measured on a scale from 1 (best) to infinity (worst). For each output target word, the model produces scores for each word in the vocabulary, which are converted to a probability distribution over the vocabulary. The *cost* is the negative log probability of the gold word—that is, the lower the system's certainty in choosing the correct word, the higher the cost.

Both evaluation metrics are valuable. BLEU measures a system's performance on the end-goal of machine translation, translation quality, whereas perplexity is the quantity minimized during training. BLEU is a 'hard' measure of performance, as it is calculated based on the sentences produced by the network, whereas perplexity is a 'softer' measure that takes into account not just whether the correct target word was produced, but the probability of producing the correct target word. For scaled-down systems such as our base-

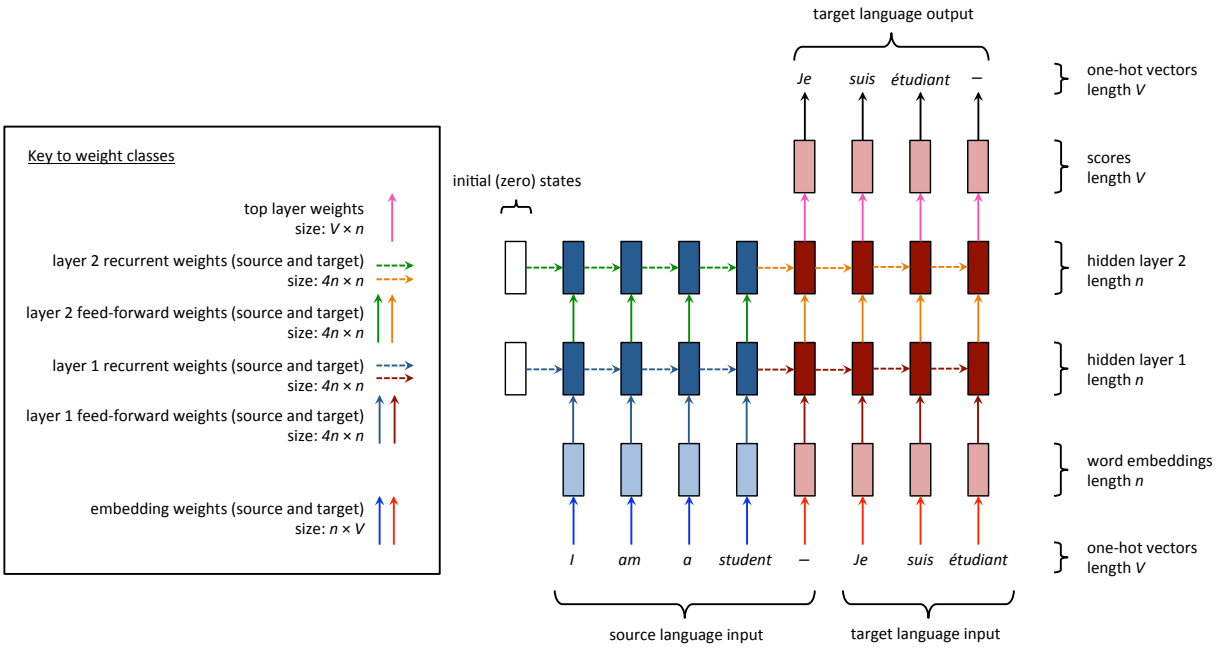

Figure 2: Neural Machine Translation architecture. The eleven weight classes are indicated by arrows with different formatting (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). As there are three weight classes of size Vn, and eight classes of size $4n^2$ (see Section 8 for more explanation), it follows that for our baseline with $V = 10,000$ and $n = 500$, each of the three large classes has 5 million weights, and each of the eight small classes has 1 million weights, for a total of 23 million weights overall.

line, the BLEU scores are quite low, so perplexity is useful to capture a more fine-grained measure of performance. We use both BLEU and perplexity in this paper.

5 Our Baseline

Our models are trained and tested on the WIT3 Vietnamese-to-English dataset (Cettolo et al., 2012), which consists of transcribed and translated TED and TEDX talks. We took a training set size of approximately 133,000 training sentences, and a validation set size of 1553 sentences.

We closely follow the state-of-the-art approach of (Luong et al., 2015) in all aspects, though we make the following changes for practicality. While (Luong et al., 2015) uses a training set with 4.5 million sentence pairs, a vocabulary size V of 50,000, 4 layers, and dimension n of 1000, our baseline has a training set size of 133,000, $V = 10,000$, 2 layers, and $n = 500$. We also omit the 'attention' mechanism described in (Luong et al., 2015). Consequently our models have 23 million parameters (see Figure 2), compared to the approximately 214 million parameters of the state-of-the-art model described in (Luong et al., 2015).

5.1 Dropout

We trained our baselines with the regularization technique dropout (Zaremba et al., 2014), which temporarily sets some random subset of a hidden unit to zero during the forward propagation phase of training. We trained two baselines: one with a more aggressive dropout rate of 50%, and one with a less aggressive rate of 20%. The less-aggressively regularized model achieved a better BLEU score (9.83 compared to 9.61) but the more-aggressively regularized model achieved a better perplexity (19.99 compared to 19.88). We selected the more-aggressively regularized model as our baseline, so all results in this paper are with respect to this baseline unless otherwise stated. However, we did repeat our experiments with the alternative baseline, with interesting results (see Section 9).

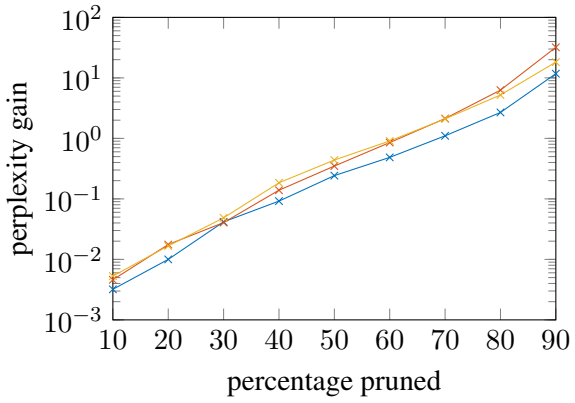

Figure 3: Immediate effect of different pruning schemes on perplexity. We see that delete-smallest pruning (blue) has a smaller impact on performance than either uniform pruning (red) or standard-deviation pruning (yellow), at all pruning percentages.

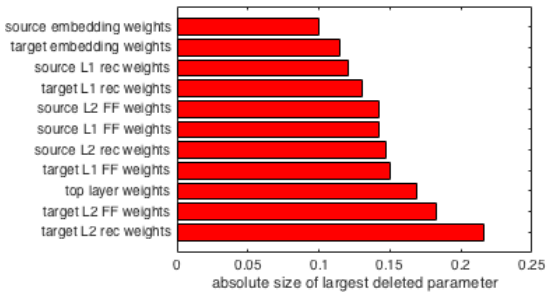

Figure 4: Absolute size of largest deleted parameter in each weight class, when pruning 90% of parameters using uniform pruning. The distribution is similar for other pruning percentages.

6 Choosing a pruning scheme

Before investigating the effect of pruning on performance, we must decide *how* to prune the weights from the eleven weight classes of our model (illustrated in Figure 2). That is, suppose we want to prune $x\%$ of the parameters. There are several possible schemes:

1. Delete-smallest pruning: Prune the $x\%$ of parameters with smallest absolute value, regardless of weight class. (So some classes are pruned proportionally more than others, though $x\%$ of total parameters are pruned).

2. Uniform pruning: For each class, prune the $x\%$ of parameters with smallest absolute value. (So all classes have exactly $x\%$ of their parameters pruned).

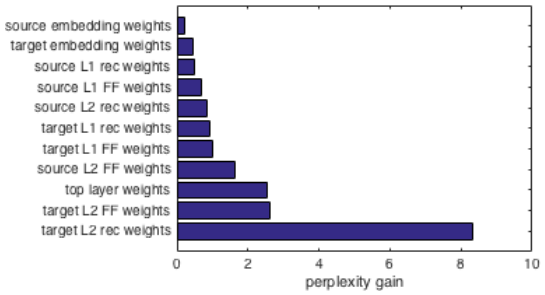

Figure 5: 'Breakdown' of perplexity gain (i.e. performance loss) when deleting 90% of parameters using uniform pruning. Equivalently, perplexity gain resulting from deleting 90% from each class separately. The distribution is similar for other pruning percentages.

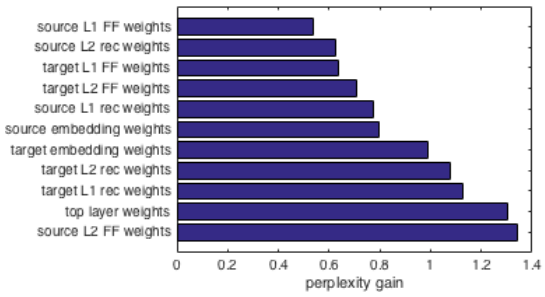

Figure 6: 'Breakdown' of perplexity gain (i.e. performance loss) when deleting 90% of parameters using delete-smallest pruning. The distribution is similar for other pruning percentages.

3. Standard-deviation pruning: Find some quality parameter $\lambda > 0$ such that if within each class, all parameters with absolute value less than $\lambda\sigma$ are pruned (where σ is the standard deviation of that class's weights), then in total $x\%$ of all parameters are pruned. This method is used by (Han et al., 2015b).

All these schemes have their seeming advantages. Delete-smallest pruning is the simplest, and adheres to the principle that pruning weights with smallest absolute value should least impact performance. Uniform pruning and standard-deviation pruning both seek to prune proportionally within each weight class, either absolutely, or relative to that class's standard deviation. However, we find that delete-smallest pruning outperforms both other schemes in terms of immediate effect on performance after pruning (Figure 3).

The poor performance of uniform pruning can be explained by Figure 4, which illustrates the

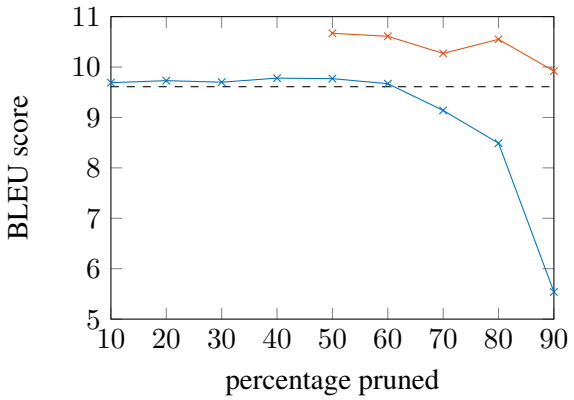

Figure 7: BLEU scores of pruned models (blue line), and of retrained pruned models (red line). The dashed line indicates baseline performance. The plot for perplexity is similar.

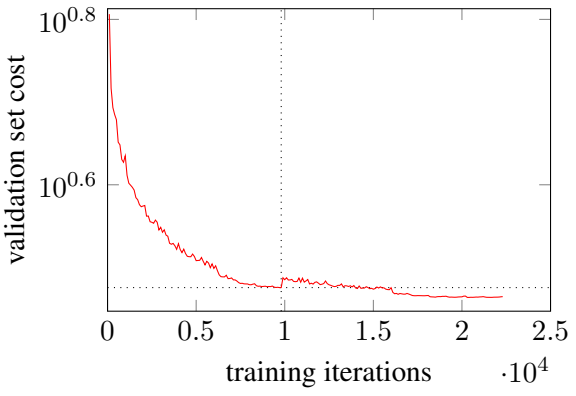

Figure 8: Cost minimization during training, pruning, and retraining. The vertical dotted line marks the point when 50% of parameters are pruned; the switchover from baseline training to retraining. The horizontal dotted line marks the baseline lowest cost, which is surpassed during retraining.

sizes of the largest weights deleted by the uniform pruning scheme, and Figure 5, which gives a 'breakdown' of the performance loss of uniform pruning by weight class. The two are highly correlated—in fact, the ordering of the weight classes is almost the same in the two figures. This indicates that, as some weight classes clearly have significantly larger weights than others at the same percentile, pruning them creates a greater performance loss. The standard-deviation pruning scheme suffers from the same problem.

Figure 5 also shows that for uniform pruning, the performance loss breakdown by weight class is highly uneven; the effect of pruning one class dominates the others. In comparison, Figure 6 shows that for delete-smallest pruning, performance loss is much more equally distributed between the weight classes. However, it is not completely uniform, indicating that factors other than absolute weight size may have an impact when pruning.

7 Effect of pruning on performance

Pruning has an immediate negative impact on performance (as measured by BLEU score), that is exponential in pruning percentage; this is demonstrated by the blue line in Figure 7. We also observe that for up to 60% pruning percentage, performance is slightly *increased*, indicating a regularizing effect on the model. While it may seem that we are getting away with throwing away over half the model, it is likely that the information of the model is stored in the weights with *large* absolute value, whereas the weights close to zero constitute noise, thus they can be safely (and in fact advantageously) pruned. So, though we are indeed reducing the model storage size by 60%, we are not deleting 60% of the important information. In any case, this is evidence for the large amount of redundancy and over-parameterization in NMT.

The red line in Figure 7 shows that after retraining the pruned models, baseline performance is both recovered and improved upon, even for the most pruned model (90%). This may seem surprising, as (notwithstanding the small improvement from noise-canceling discussed above) we might not expect a sparse model to significantly out-perform a model with ten times as many parameters. There are several possible explanations of this, two of which are given below.

Firstly, we observe that the less-pruned models perform better on the training set than the validation set, whereas the more-pruned models have more similar 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 significantly outperform the baseline.

Alternatively, pruning may serve as a means for the model to escape a local optimum. Figure 8 shows the validation set cost over time during the training, pruning and retraining process. The original baseline is trained to convergence—the gradi-

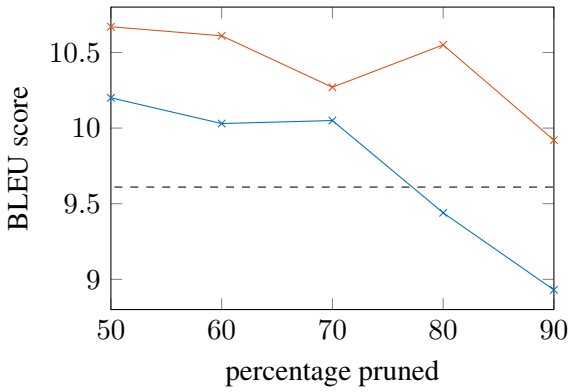

Figure 9: Comparison of the performance of sparse-from-scratch models (blue line) and the pruned and retrained models (red line). The dashed line indicates baseline performance. The plot for perplexity is similar.

ent at the end of training is zero. Pruning causes an immediate increase in cost, but the gradient becomes negative again, allowing the retraining process to find a new, better local optimum. It may be that the disruption caused by pruning is beneficial in the long-run.

7.1 Starting with sparse models

The favourable performance of the pruned and retrained models raises the question: can we get a shortcut to this performance by *starting* with sparse models? That is, rather than train, prune, and retrain, what if we simply prune then train? We took the sparsity structure of our pruned models, and trained new models from scratch with the same sparsity structure. Figure 9 shows that the sparse-from-scratch models do not perform as well as the pruned and retrained models, but they do, for smaller pruning percentages, outperform the baseline. This implies that in a trained and pruned model, the sparsity structure itself (even without the values of the remaining weights) contains important information.

8 Distribution of redundancy in NMT

In Figure 10, we see the location of small weights (i.e. redundancy) within the weight classes of our NMT baseline model. Black pixels represent weights near to zero; white pixels represent large weights.

First we consider the embedding weights and top layer weights. Recall that the embedding weight matrices and top layer weight matrices have rows (or columns) corresponding to words in the vocabulary. Unsurprisingly, in Figure 10 we see that the parameters corresponding to the less common weights are more dispensable. In fact, at the 90% pruning rate, for many uncommon source words we delete *all* source embedding parameters corresponding to that word, giving the word a constant zero embedding. 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, given that the pruned weights were already very small, in the original unpruned model these words already had near-zero embeddings. This is perhaps an indication that for the less common words in our vocabulary, the model was unable to learn sufficiently distinctive representations.

To understand the pictures in Figure 10 corresponding to the feed-forward and recurrent weights, we must first refer to the Long Short-Term Memory forward-propagation equations, for which we follow (Graves and others, 2012). Each LSTM block is indexed by time t (which flows left to right in Figure 1) and layer l (bottom to top). The main output of the LSTM block is the hidden state h_t^l (a vector of length n), which is passed to the LSTM blocks one layer above, and one timestep forwards. In addition, each LSTM block has a memory cell c_t^l (also length n), which is intended to store the 'long term' memory of the network. Finally, the hidden state h_t^l and cell c_t^l are calculated from h_t^{l-1}, h_{t-1}^l and c_{t-1}^l with the use of three 'gates' that control the flow of information into and through the LSTM block, like so:

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

$$c_t^l = f \circ c_{t-1}^l + i \circ \hat{h} \quad (2)$$

$$h_t^l = o \circ \tanh(c_t^l) \quad (3)$$

where $T_{2n,4n}$ is a $4n \times 2n$ weight matrix, the functions sigm and tanh are applied element-wise, i is the input gate, f the forget gate, o the output gate and $\hat{h}$ is the input.

Equation (1) describes how the feed-forward connections and recurrent connections define the gates and the input $\hat{h}$, from previous hidden states h_t^{l-1} and h_{t-1}^l. Equation (2) describes how the

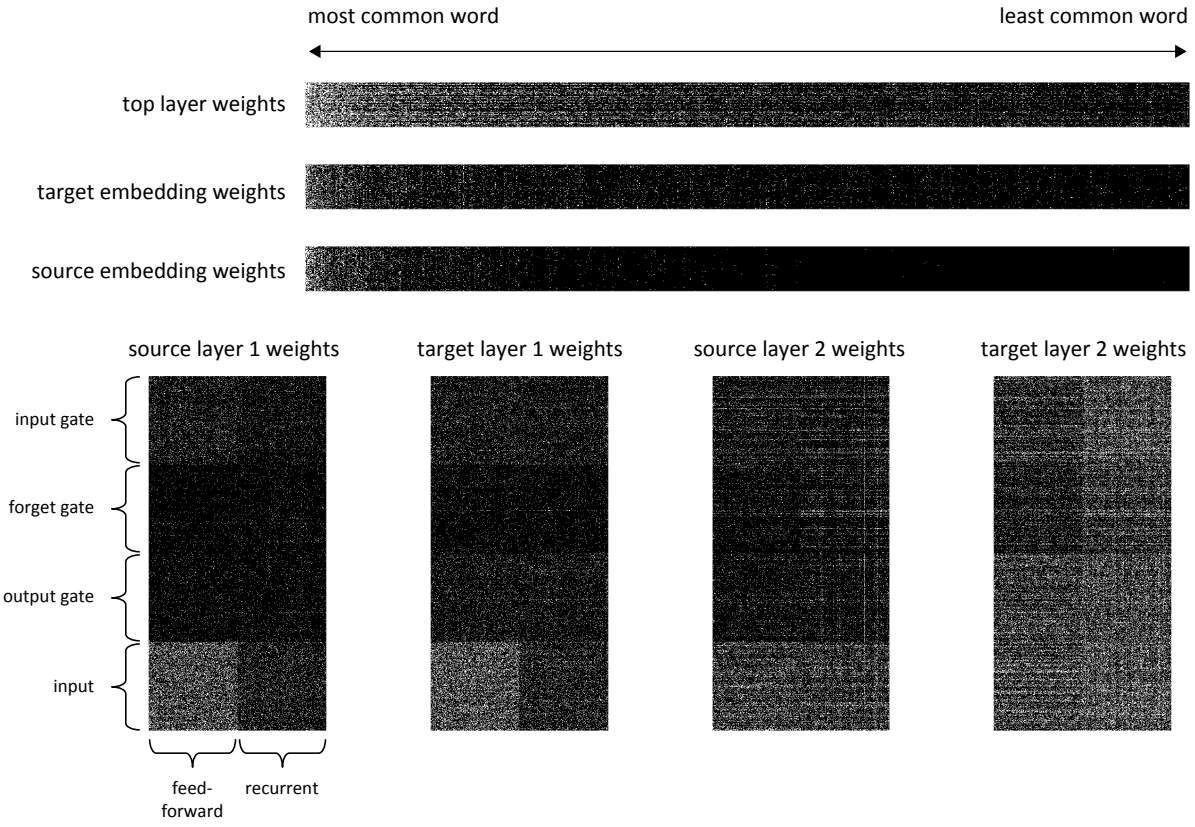

Figure 10: Graphical representation of the location of small weights in the model. Black pixels represent weights with absolute size in the bottom 90%; white pixels represent those with absolute size in the top 10%. Equivalently, these pictures illustrate which parameters remain after pruning 90% using our delete-smallest pruning scheme.

the input $\hat{h}$ 'passes through' the input gate and the previous memory cell 'passes through' the forget gate to define the new memory cell. Equation (3) describes how the memory cell 'passes through' the output gate to describe the hidden state.

Returning to Figure 10, we now see that the four pictured weight matrices correspond to the $4n \times 2n$ weight matrix $T_{2n,4n}$ in Equation (1). In all four matrices, we can observe 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.

For layer 1, the feed-forward input is more important than the recurrent input, whereas for layer 2 the recurrent input is more important. This makes sense: layer 1 concentrates on the low-level information from the current word embedding (the feed-forward input), whereas layer 2 concentrates on the higher-level representation of the sentence so far (the recurrent input).

For layer 2, there is more redundancy in the source weights than the target weights. In particular, for target layer 2, the 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 10 therefore shows that these sophisticated memory cell abilities are most important at the *end* of the NMT pipeline (the top level of the target RNN). This is reasonable, as we expect higher-level features to be learnt later in a deep learning pipeline.

9 Generalizability of our results

Though we endeavored to choose a well-regularized baseline with sensible choices of hyperparameters that could be considered 'representative', the usefulness of our results depends nonetheless on their generalizability to other NMT models. Some are easily generalizable: for example, the distribution of weight sizes as illustrated in Figures 4 and 10, and the effect of each weight class on uniform pruning as illustrated in Figure 5, are very similar across models with one or two

layers, with varying dimension n, varying embedding size, and varying dropout rates.

Other results require more expensive tests and are therefore less quickly verified. In Section 7 we found that up to 60% of our baseline can be pruned without retraining with no loss of BLEU score (see Figure 7). While we found that other models (with different dropout rates, and with n increased and decreased from our baseline) sometimes benefitted immediately from pruning, it was not always consistent, and not usually as much as 60%. In addition we found that BLEU tends to immediately benefit from pruning more than perplexity.

We successfully repeated our central result—that we can prune 90% of parameters then recover baseline performance through retraining—on a 1-layer version of our baseline (obtaining a perplexity of 19.81 for the compressed model vs baseline 20.78). The same is true for our 'alternative baseline' described in Section 5.1, which was trained with a gentler dropout rate of 20%. However, we found that for this gently-regularized baseline, we could improve even more by using a more aggressive dropout rate of 50% for the retraining. This raised the question: was the performance increase due to the pruning, or the change in dropout? To find out, we trained a new model with a dropout rate of 20%, pruned nothing, then retrained with a dropout rate of 50%. This model (perplexity 17.54) out-performed the models trained with either dropout rate alone (19.99 and 19.88). This is an interesting discovery that is orthogonal to pruning.

10 Future Work

To properly establish the efficacy of weight pruning for NMT, the results of this paper should be repeated on a state-of-the-art NMT model, for example (Luong et al., 2015). Secondly, the pruning method described in (Han et al., 2015b) includes *several* iterations of pruning and retraining. Implementing this would likely result in further compression and performance improvements. Thirdly, if possible it would be highly valuable to exploit the sparsity of the pruned models to speed up training and runtime.

This work uncovered several unexpected discoveries that merit further investigation, orthogonal to pruning. In Section 9 we observed that training with a gentle dropout rate followed by a more aggressive dropout rate resulted in significantly better performance than from using either dropout rate alone. If this is generally true, it could form an improvement to the dropout technique, or help us to understand the technique better. In Section 7.1 we observed that, for smaller pruning percentages, pruning models *before* training (i.e. training sparse models from scratch) performs better than the densely-trained baseline. This could potentially be a useful compression or regularization technique, alongside or as an alternative to the train-prune-retrain method explored in this paper. Lastly, in Section 7 we hypothesized that the disruption caused by pruning may enable the model to escape local optima during training, thus reaching better performances. This could be tested by performing other types of 'disruption' to converged models, and observing whether the immediate performance loss is recovered and improved upon, as we observed here.

11 Conclusion

We have shown that weight pruning (with retraining) is a highly effective method of compression and regularization on our NMT baseline, and have some evidence to indicate that our results are generalizable to other NMT models. We have found that the absolute size of parameters is of primary importance when choosing which to prune, and have gained insight into the distribution of redundancy in the NMT architecture. In addition the investigation has uncovered several unexpected results that challenge, and may deepen, our understanding of related topics such as dropout, sparse models and stochastic gradient descent.

Acknowledgments

Thanks to my advisor Christopher Manning for his guidance during this project, and to Thang Minh Luong for the use of his NMT code, and extensive help in using it.

References

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy, May.

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*.

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. 2014. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*.

Alex Graves et al. 2012. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding.

Song Han, Jeff Pool, John Tran, and William J Dally. 2015b. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

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. *arXiv preprint arXiv:1503.02531*.

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

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2015. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Kenton Murray and David Chiang. 2015. Auto-sizing neural networks: With applications to n-gram language models. *arXiv preprint arXiv:1508.05051*.

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 VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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