

# Unfolding and Shrinking Neural Machine Translation Ensembles

Felix Stahlberg and Bill Byrne

{fs439, wjb31}@cam.ac.uk

Department of Engineering

University of Cambridge, UK

## Abstract

Ensembling is a well-known technique in neural machine translation (NMT) to improve system performance. Instead of a single neural net, multiple neural nets with the same topology are trained separately, and the decoder generates predictions by averaging over the individual models. Ensembling often improves the quality of the generated translations drastically. However, it is not suitable for production systems because it is cumbersome and slow. This work aims to reduce the runtime to be on par with a single system without compromising the translation quality. First, we show that the ensemble can be unfolded into a single large neural network which imitates the output of the ensemble system. We show that unfolding can already improve the runtime in practice since more work can be done on the GPU. We proceed by describing a set of techniques to shrink the unfolded network by reducing the dimensionality of layers. On Japanese-English we report that the resulting network has the size and decoding speed of a single NMT network but performs on the level of a 3-ensemble system.

## 1 Introduction

The top systems in recent machine translation evaluation campaigns on various language pairs use ensembles of a number of NMT systems (Bogdan et al., 2016; Sennrich et al., 2016a; Chung et al., 2016; Neubig, 2016; Wu et al., 2016; Cromieres et al., 2016; Durrani et al., 2017). Ensembling (Dietterich, 2000; Hansen and Salamon, 1990) of neural networks is a simple yet very effective technique to improve the accuracy of NMT.

The decoder makes use of  $K$  NMT networks which are either trained independently (Sutskever et al., 2014; Chung et al., 2016; Neubig, 2016; Wu et al., 2016) or share some amount of training iterations (Sennrich et al., 2016b,a; Cromieres et al., 2016; Durrani et al., 2017). The ensemble decoder computes predictions from each of the individual models which are then combined using the arithmetic average (Sutskever et al., 2014) or the geometric average (Cromieres et al., 2016).

Ensembling consistently outperforms single NMT by a large margin. However, the decoding speed is significantly worse since the decoder needs to apply  $K$  NMT models rather than only one. Therefore, a recent line of research transfers the idea of *knowledge distillation* (Bucilu et al., 2006; Hinton et al., 2014) to NMT and trains a smaller network (the student) by minimizing the cross-entropy to the output of the ensemble system (the teacher) (Kim and Rush, 2016; Freitag et al., 2017). This paper presents an alternative to knowledge distillation as we aim to speed up decoding to be comparable to single NMT while retaining the boost in translation accuracy from the ensemble. In a first step, we describe how to construct a single large neural network which imitates the output of an ensemble of multiple networks with the same topology. We will refer to this process as *unfolding*. GPU-based decoding with the unfolded network is often much faster than ensemble decoding since more work can be done on the GPU. In a second step, we explore methods to reduce the size of the unfolded network. This idea is justified by the fact that ensembled neural networks are often over-parameterized and have a large degree of redundancy (LeCun et al., 1989; Hassibi et al., 1993; Srinivas and Babu, 2015). Shrinking the unfolded network leads to a smaller model which consumes less space on the disk and in the memory; a crucial factor on mobile devices. More importantly, the

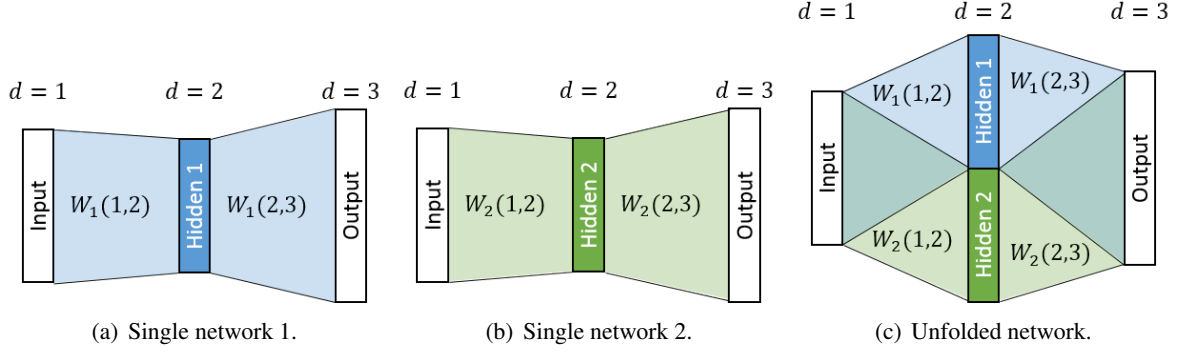


Figure 1: Unfolding mimics the output of the ensemble of two single layer feedforward networks.

decoding speed on all platforms benefits greatly from the reduced number of neurons. We find that the dimensionality of linear embedding layers in the NMT network can be reduced heavily by low-rank matrix approximation based on singular value decomposition (SVD). This suggests that high dimensional embedding layers may be needed for training, but do not play an important role for decoding. The NMT network, however, also consists of complex layers like gated recurrent units (Cho et al., 2014, GRUs) and attention (Bahdanau et al., 2015). Therefore, we introduce a novel algorithm based on linear combinations of neurons which can be applied either during training (*data-bound*) or directly on the weight matrices without using training data (*data-free*). We report that with a mix of the presented shrinking methods we are able to reduce the size of the unfolded network to the size of the single NMT network while keeping the boost in BLEU score from the ensemble. Depending on the aggressiveness of shrinking, we report either a gain of 2.2 BLEU at the same decoding speed, or a  $3.4\times$  CPU decoding speed up with only a minor drop in BLEU compared to the original single NMT system. Furthermore, it is often much easier to stage a single NMT system than an ensemble in a commercial MT workflow, and it is crucial to be able to optimize quality at specific speed and memory constraints. Unfolding and shrinking address these problems directly.

## 2 Unfolding $K$ Networks into a Single Large Neural Network

The first concept of our approach is called *unfolding*. Unfolding is an alternative to ensembling of multiple neural networks with the same topology. Rather than averaging their predictions, unfolding constructs a single large neural net out of the indi-

vidual models which has the same number of input and output neurons but larger inner layers. Our main motivation for unfolding is to obtain a single network with ensemble level performance which can be shrunk with the techniques in Sec. 3.

Suppose we ensemble two single layer feedforward neural nets as shown in Fig. 1. Normally, ensembling is implemented by performing an isolated forward pass through the first network (Fig. 1(a)), another isolated forward pass through the second network (Fig. 1(b)), and averaging the activities in the output layers of both networks. This can be simulated by merging both networks into a single large network as shown in Fig. 1(c). The first neurons in the hidden layer of the combined network correspond to the hidden layer in the first single network, and the others to the hidden layer of the second network. A single pass through the combined network yields the same output as the ensemble if the output layer is linear (up to a factor 2). The weight matrices in the unfolded network can be constructed by stacking the corresponding weight matrices (either horizontally or vertically) in network 1 and 2. This kind of aggregation of multiple networks with the same topology is not only possible for single-layer feedforward architectures but also for complex networks consisting of multiple GRU layers and attention.

For a formal description of unfolding we address layers with indices  $d = 0, 1, \dots, D$ . The special layer 0 has a single neuron for modelling bias vectors. Layer 1 holds the input neurons and layer  $D$  is the output layer. We denote the size of a layer in the individual models as  $s(d)$ . When combining  $K$  networks, the layer size  $s'(d)$  in the unfolded network is increased by factor  $K$  if  $d$  is an inner layer, and equal to  $s(d)$  if  $d$  is the in-

$$W'(d_1, d_2) = \begin{cases} \begin{pmatrix} W_1(d_1, d_2) & 0 & \cdots & 0 \\ 0 & W_2(d_1, d_2) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & & W_K(d_1, d_2) \end{pmatrix} & \text{if } d_1 \in \text{InnerLayers and } d_2 \in \text{InnerLayers} \\ \frac{1}{K} \begin{pmatrix} W_1(d_1, d_2) \\ \vdots \\ W_K(d_1, d_2) \end{pmatrix} & \text{if } d_1 \in \text{InnerLayers and } d_2 \notin \text{InnerLayers} \\ \begin{pmatrix} W_1(d_1, d_2) & \cdots & W_K(d_1, d_2) \end{pmatrix} & \text{if } d_1 \notin \text{InnerLayers and } d_2 \in \text{InnerLayers} \end{cases}$$

Figure 2: General formula for unfolding weight matrices. The set  $\text{InnerLayers} := [2, D - 1]$  includes all layers except the input, output, and bias layer.

put or output layer. We denote the weight matrix between two layers  $d_1, d_2 \in [0, D]$  in the  $k$ -th individual model ( $k \in [1, K]$ ) as  $W_k(d_1, d_2) \in \mathbb{R}^{s(d_1) \times s(d_2)}$ , and the corresponding weight matrix in the unfolded network as  $W'(d_1, d_2) \in \mathbb{R}^{s'(d_1) \times s'(d_2)}$ . We explicitly allow  $d_1$  and  $d_2$  to be non-consecutive or reversed to be able to model recurrent networks. We use the zero-matrix if layers  $d_1$  and  $d_2$  are not connected. The construction of the unfolded weight matrix  $W'(d_1, d_2)$  from the individual matrices  $W_k(d_1, d_2)$  depends on whether the connected layers are inner layers or not. The complete formula is listed in Fig. 2.

Unfolded NMT networks approximate but do not exactly match the output of the ensemble due to two reasons. First, the unfolded network synchronizes the attentions of the individual models. Each decoding step in the unfolded network computes a single attention weight vector. In contrast, ensemble decoding would compute one attention weight vector for each of the  $K$  input models. A second difference is that the ensemble decoder first applies the softmax at the output layer, and then averages the prediction probabilities. The unfolded network averages the neuron activities (i.e. the logits) first, and then applies the softmax function. Interestingly, as shown in Sec. 4, these differences do not have any impact on the BLEU score but yield potential speed advantages of unfolding since the computationally expensive softmax layer is only applied once.

### 3 Shrinking the Unfolded Network

After constructing the weight matrices of the unfolded network we reduce the size of it by iteratively shrinking layer sizes. In this section we denote the incoming weight matrix of the layer to

shrink as  $U \in \mathbb{R}^{m_{in} \times m}$  and the outgoing weight matrix as  $V \in \mathbb{R}^{m \times m_{out}}$ . Our procedure is inspired by the method of Srinivas and Babu (2015). They propose a criterion for removing neurons in inner layers of the network based on two intuitions. First, similarly to Hebb’s learning rule, they detect redundancy by the principle *neurons which fire together, wire together*. If the incoming weight vectors  $U_{:,i}$  and  $U_{:,j}$  are exactly the same for two neurons  $i$  and  $j$ , we can remove the neuron  $j$  and add its outgoing connections to neuron  $i$  ( $V_{i,:} \leftarrow V_{i,:} + V_{j,:}$ ) without changing the output.<sup>1</sup> This holds since the activity in neuron  $j$  will always be equal to the activity in neuron  $i$ . In practice, Srinivas and Babu use a distance measure based on the difference of the incoming weight vectors to search for similar neurons as exact matches are very rare.

The second intuition of the criterion used by Srinivas and Babu (2015) is that neurons with small outgoing weights contribute very little overall. Therefore, they search for a pair of neurons  $i, j \in [1, m]$  according to the following term and remove the  $j$ -th neuron.<sup>2</sup>

$$\arg \min_{i,j \in [1,m]} \|U_{:,i} - U_{:,j}\|_2^2 \|V_{j,:}\|_2^2 \quad (1)$$

Neuron  $j$  is selected for removal if (1) there is another neuron  $i$  which has a very similar set of incoming weights and if (2)  $j$  has a small outgoing weight vector. Their criterion is *data-free* since

<sup>1</sup>We denote the  $i$ -th row vector of a matrix  $A$  with  $A_{i,:}$  and the  $i$ -th column vector as  $A_{:,i}$ .

<sup>2</sup>Note that the criterion in Eq. 1 generalizes the criterion of Srinivas and Babu (2015) to multiple outgoing weights. Also note that Srinivas and Babu (2015) propose some heuristic improvements to this criterion. However, these heuristics did not work well in our NMT experiments.

it does not require any training data. For further details we refer to Srinivas and Babu (2015).

### 3.1 Data-Free Neuron Removal

Srinivas and Babu (2015) propose to add the outgoing weights of  $j$  to the weights of a similar neuron  $i$  to compensate for the removal of  $j$ . However, we have found that this approach does not work well on NMT networks. We propose instead to compensate for the removal of a neuron by a linear combination of the remaining neurons in the layer. Data-free shrinking assumes for the sake of deriving the update rule that the neuron activation function is linear. We now ask the following question: How can we compensate as well as possible for the loss of neuron  $j$  such that the impact on the output of the whole network is minimized? Data-free shrinking represents the incoming weight vector of neuron  $j$  ( $U_{:,j}$ ) as linear combination of the incoming weight vectors of the other neurons. The linear factors can be found by satisfying the following linear system:

$$U_{:,-j}\lambda = U_{:,j} \quad (2)$$

where  $U_{:,-j}$  is matrix  $U$  without the  $j$ -th column. In practice, we use the method of ordinary least squares to find  $\lambda$  because the system may be overdetermined. The idea is that if we mix the outputs of all neurons in the layer by the  $\lambda$ -weights, we get the output of the  $j$ -th neuron. The row vector  $V_{j,:}$  contains the contributions of the  $j$ -th neuron to each of the neurons in the next layer. Rather than using these connections, we approximate their effect by adding some weight to the outgoing connections of the other neurons. How much weight depends on  $\lambda$  and the outgoing weights  $V_{j,:}$ . The factor  $D_{k,l}$  which we need to add to the outgoing connection of the  $k$ -th neuron to compensate for the loss of the  $j$ -th neuron on the  $l$ -th neuron in the next layer is:

$$D_{k,l} = \lambda_k V_{j,l} \quad (3)$$

Therefore, the update rule for  $V$  is:

$$V \leftarrow V + D \quad (4)$$

In the remainder we will refer to this method as *data-free* shrinking. Note that we recover the update rule of Srinivas and Babu (2015) by setting  $\lambda$  to the  $i$ -th unit vector. Also note that the error introduced by our shrinking method is due to the

fact that we ignore the non-linearity, and that the solution for  $\lambda$  may not be exact. The method is error-free on linear layers as long as the residuals of the least-squares analysis in Eq. 2 are zero.

**GRU layers** The terminology of *neurons* needs some further elaboration for GRU layers which rather consist of update and reset gates and states (Cho et al., 2014). On GRU layers, we treat the states as neurons, i.e. the  $j$ -th neuron refers to the  $j$ -th entry in the GRU state vector. Input connections to the gates are included in the incoming weight matrix  $U$  for estimating  $\lambda$  in Eq. 2. Removing neuron  $j$  in a GRU layer means deleting the  $j$ -th entry in the states and both gate vectors.

### 3.2 Data-Bound Neuron Removal

Although we find our data-free approach to be a substantial improvement over the methods of Srinivas and Babu (2015) on NMT networks, it still leads to a non-negligible decline in BLEU score when applied to recurrent GRU layers. Our data-free method uses the incoming weights to identify similar neurons, i.e. neurons expected to have similar activities. This works well enough for simple layers, but the interdependencies between the states and the gates inside gated layers like GRUs or LSTMs are complex enough that redundancies cannot be found simply by looking for similar weights. In the spirit of Babaeizadeh et al. (2016), our *data-bound* version records neuron activities during training to estimate  $\lambda$ . We compensate for the removal of the  $j$ -th neuron by using a linear combination of the output of remaining neurons with similar activity patterns. In each layer, we prune 40 neurons each 450 training iterations until the target layer size is reached. Let  $A$  be the matrix which holds the records of neuron activities in the layer since the last removal. For example, for the decoder GRU layer, a batch size of 80, and target sentence lengths of 20,  $A$  has  $20 \cdot 80 \cdot 450 = 720K$  rows and  $m$  (the number of neurons in the layer) columns.<sup>3</sup> Similarly to Eq. 2 we find interpolation weights  $\lambda$  using the method of least squares on the following linear system.

$$A_{:,-j}\lambda = A_{:,j} \quad (5)$$

The update rule for the outgoing weight matrix is the same as for our data-free method (Eq. 4).

<sup>3</sup>In practice, we use a random sample of 50K rows rather than the full matrix to keep the complexity of the least-squares analysis under control.



The key difference between data-free and data-bound shrinking is the way  $\lambda$  is estimated. Data-free shrinking uses the similarities between incoming weights, and data-bound shrinking uses neuron activities recorded during training. Once we select a neuron to remove, we estimate  $\lambda$ , compensate for the removal, and proceed with the shrunk network. Both methods are prior to any decoding and result in shrunk parameter files which are then loaded to the decoder. Both methods remove neurons rather than single weights.

The data-bound algorithm runs gradient-based optimization on the unfolded network. We use the AdaGrad (Duchi et al., 2011) step rule, a small learning rate of 0.0001, and aggressive step clipping at 0.05 to avoid destroying useful weights which were learned in the individual networks prior to the construction of the unfolded network.

Our data-bound algorithm uses a data-bound version of the neuron selection criterion in Eq. 1 which operates on the activity matrix  $A$ . We search for the pair  $i, j \in [1, m]$  according the following term and remove neuron  $j$ .

$$\arg \min_{i, j \in [1, m]} \|A_{:,i} - A_{:,j}\|_2^2 \|A_{:,j}\|_2^2 \quad (6)$$

### 3.3 Shrinking Embedding Layers with SVD

The standard attention-based NMT network architecture (Bahdanau et al., 2015) includes three linear layers: the embedding layer in the encoder, and the output and feedback embedding layers in the decoder. We have found that linear layers are particularly easy to shrink using low-rank matrix approximation. As before we denote the incoming weight matrix as  $U \in \mathbb{R}^{m_{in} \times m}$  and the outgoing weight matrix as  $V \in \mathbb{R}^{m \times m_{out}}$ . Since the layer is linear, we could directly connect the previous layer with the next layer using the product of both weight matrices  $X = U \cdot V$ . However,  $X$  may be very large. Therefore, we approximate  $X$  as a product of two low rank matrices  $Y \in \mathbb{R}^{m_{in} \times m'}$  and  $Z \in \mathbb{R}^{m' \times m_{out}}$  ( $X \approx YZ$ ) where  $m' \ll m$  is the desired layer size. A very common way to find such a matrix factorization is using truncated singular value decomposition (SVD). The layer is eventually shrunk by replacing  $U$  with  $Y$  and  $V$  with  $Z$ .

## 4 Results

The individual NMT systems we use as source for constructing the unfolded networks are trained us-

ing AdaDelta (Zeiler, 2012) on the Blocks/Theano implementation (van Merriënboer et al., 2015; Bastien et al., 2012) of the standard attention-based NMT model (Bahdanau et al., 2015) with: 1000 dimensional GRU layers (Cho et al., 2014) in both the decoder and bidirectional encoder; a single maxout output layer (Goodfellow et al., 2013); and 620 dimensional embedding layers. We follow Sennrich et al. (2016b) and use subword units based on byte pair encoding rather than words as modelling units. Our SGNMT decoder (Stahlberg et al., 2017)<sup>4</sup> with a beam size of 12 is used in all experiments. Our primary corpus is the Japanese-English (Ja-En) ASPEC data set (Nakazawa et al., 2016). We select a subset of 500K sentence pairs to train our models as suggested by Neubig et al. (2015). We report cased BLEU scores calculated with Moses' `multi-bleu.pl` to be strictly comparable to the evaluation done in the Workshop of Asian Translation (WAT). We also apply our method to the WMT data set for English-German (En-De), using the *news-test2014* as a development set, and keeping *news-test2015* and *news-test2016* as test sets. En-De BLEU scores are computed using `mteval-v13a.pl` as in the WMT evaluation. We set the vocabulary sizes to 30K for Ja-En and 50K for En-De. We also report the *size factor* for each model which is the total number of model parameters (sum of all weight matrix sizes) divided by the number of parameters in the original NMT network (86M for Ja-En and 120M for En-De). We choose a widely used, simple ensembling method (prediction averaging) as our baseline. We feel that the prevalence of this method makes it a reasonable baseline for our experiments.

**Shrinking the Unfolded Network** First, we investigate which shrinking methods are effective for which layers. Tab. 1 summarizes our results on a 2-unfold network for Ja-En, i.e. two separate NMT networks are combined in a single large network as described in Sec. 2. The layers in the combined network are shrunk to the size of the original networks using the methods discussed in Sec. 3.

Shrinking the linear embedding layers with SVD (Sec. 3.3) is very effective. The unfolded model with shrunk embedding layers performs at the same level as the ensemble (compare rows (b) and (c)). In our initial experiments, we applied the method of Srinivas and Babu (2015) to

<sup>4</sup>‘vanilla’ decoding strategy

	Base	Shrinking Methods					Size Factor	BLEU	
		Encoder Embed.	GRUs	Attention Match	GRU	Decoder Maxout Embeds.		dev	test
(a)	Single	-	-	-	-	-	1.00	20.8	23.5
(b)	2-Ens.	-	-	-	-	-	$2 \times 1.00$	22.7	25.2
(c)	2-Unfold	SVD	-	-	-	SVD	1.85	22.7	25.1
(d)	2-Unfold	SVD	-	Data-Free	-	SVD	1.77	22.7	25.1
(e)	2-Unfold	SVD	Data-Free	Data-Free	Data-Free	-	1.05	21.6	24.2
(f)	2-Unfold	SVD	Data-Bound	Data-Free	Data-Bound	-	1.05	22.4	<b>25.3</b>
(g)	2-Unfold	SVD	Data-Bound	Data-Free	Data-Bound	Data-Free	1.00	16.9	19.3
(h)	2-Unfold	SVD	Data-Bound	Data-Free	Data-Bound	Data-Bound	1.00	21.9	24.6

Table 1: Shrinking layers of the unfolded network on Ja-En to their original size.

shrink the other layers, but their approach performed very poorly on this kind of network: the BLEU score dropped down to 15.5 on the development set when shrinking all layers except the decoder maxout and embedding layers, and to 9.9 BLEU when applying their method only to embedding layers.<sup>5</sup> Row (e) in Tab. 1 shows that our data-free algorithm from Sec. 3.1 is better suited for shrinking the GRU and attention layers, leading to a drop of only 1 BLEU point compared to the ensemble (b) (i.e. 0.8 BLEU better than the single system (a)). However, using the data-bound version of our shrinking algorithm (Sec. 3.2) for the GRU layers performs best.<sup>6</sup> The shrunk model yields about the same BLEU score as the ensemble on the test set (25.2 in (b) and 25.3 in (f)). Shrinking the maxout layer remains more of a challenge (rows (g) and (h)), but the number of parameters in this layer is small. Therefore, shrinking all layers except the maxout layer leads to almost the same number of parameters (factor 1.05 in row (f)) as the original NMT network (a), and thus to a similar storage size, memory consumption, and decoding speed, but with a 1.8 BLEU gain. Based on these results we fix the shrinking method used for each layer for all remaining experiments as follows: We shrink linear embedding layers with our SVD-based method, GRU layers with our data-bound method, the attention layer with our data-free method, and do not shrink the maxout layer.

Our data-bound algorithm from Sec. 3.2 has two mechanisms to compensate for the removal of a neuron. First, we use a linear combination of the remaining neurons to update the outgoing weight matrix by imitating its activations (Eq. 4). Second, stochastic gradient descent (SGD) fine-tunes all

	Compensation Method		BLEU	
	Linear Combination	SGD	dev	test
(a)			16.3	18.0
(b)	✓		22.1	24.3
(c)		✓	21.7	24.4
(d)	✓	✓	22.4	25.3

Table 2: Compensating for neuron removal in the data-bound algorithm. Row (d) corresponds to row (f) in Tab. 1.

weights during this process. Tab. 2 demonstrates that both mechanisms are crucial for minimizing the effect of shrinking on the BLEU score.

**Decoding Speed** Our testing environment is an Ubuntu 16.04 with Linux 4.4.0 kernel, 32 GB RAM, an Intel® Core i7-6700 CPU at 3.40 GHz and an Nvidia GeForce GTX Titan X GPU. CPU decoding uses a single thread. We used the first 500 sentences of the Ja-En WAT development set for the time measurements.

Our results in Tab. 3 show that decoding with ensembles (rows (b) and (e)) is slow: combining the predictions of the individual models on the CPU is computationally expensive, and ensemble decoding requires  $K$  passes through the softmax layer which is also computationally expensive. Unfolding the ensemble into a single network and shrinking the embedding and attention layers improves the runtimes on the GPU significantly without noticeable impact on BLEU (rows (c) and (f)). This can be attributed to the fact that unfolding can reduce the communication overhead between CPU and GPU. Comparing rows (d) and (g) with row (a) reveals that shrinking the unfolded networks even further speeds up CPU and GPU decoding almost to the level of single system decoding. However, more aggressive shrinking yields a BLEU score of 25.3 when combining three systems (row (g)) – 1.8 BLEU better than the single system, but 0.6 BLEU worse than the 3-

<sup>5</sup>Results with the original method of Srinivas and Babu (2015) are not included in Tab. 1.

<sup>6</sup>If we apply different methods to different layers of the same network, we first apply SVD-based shrinking, then the data-free method, and finally the data-bound method.

	System	Words/Min.		Size Factor	BLEU	
		CPU	GPU		dev	test
(a)	Single	323.4	2993.6	1.00	20.8	23.5
(b)	2-Ensemble	163.7	1641.1	$2 \times 1.00$	22.7	25.2
(c)	2-Unfold, shrunk embed.& attention	157.2	2592.2	1.77	22.7	25.1
(d)	2-Unfold, shrunk all except maxout	308.3	2961.4	1.05	22.4	25.3
(e)	3-Ensemble	110.9	1158.2	$3 \times 1.00$	23.4	25.9
(f)	3-Unfold, shrunk embed.& attention	95.4	2182.1	2.99	23.2	25.9
(g)	3-Unfold, shrunk all except maxout	301.6	3024.4	1.09	22.2	25.3

Table 3: Time measurements on Ja-En. Layers are shrunk to their size in the original NMT model.

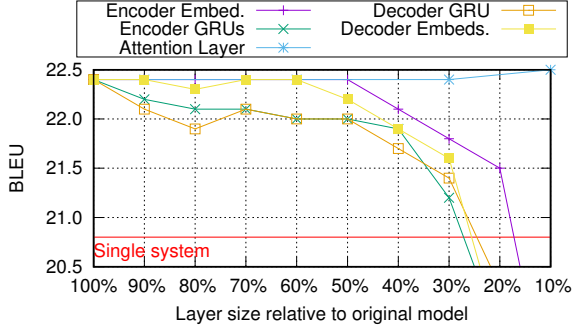


Figure 3: Impact of shrinking on the BLEU score.

	Single	3-Unfold		
		Normal	Small	Tiny
Enc. Embed.	620	410	310	170
Enc. GRUs	1000	1300	580	580
Attention	1000	100	100	100
Dec. GRU	1000	1350	590	590
Dec. Maxout	500	1500	1500	1500
Dec. Embeds.	620	430	320	170
Size Factor	1.00	1.00	0.50	0.33

Table 4: Layer sizes of our setups for Ja-En.

ensemble. Therefore, we will investigate the impact of shrinking on the different layers in the next sections more thoroughly.

### Degrees of Redundancy in Different Layers

We applied our shrinking methods to isolated layers in the 2-Unfold network of Tab. 1 (f). Fig. 3 plots the BLEU score when isolated layers are shrunk even below their size in the original NMT network. The attention layer is very robust against shrinking and can be reduced to 100 neurons (10% of the original size) without impacting the BLEU score. The embedding layers can be reduced to 60% but are sensitive to more aggressive pruning. Shrinking the GRU layers affects the BLEU score the most but still outperforms the single system when the GRU layers are shrunk to 30%.

**Adjusting the Target Sizes of Layers** Based on our previous experiments we revise our approach to shrink the 3-Unfold system in Tab. 3. Instead

	System	Words/Min.		BLEU	
		CPU	GPU	dev	test
(a)	Single	323.4	2993.6	20.8	23.5
(b)	3-Ensemble	110.9	1158.2	23.4	25.9
(c)	3-Unfold-Normal	445.2	3071.1	22.9	25.7
(d)	3-Unfold-Small	946.1	3572.0	21.7	23.9
(e)	3-Unfold-Tiny	1102.5	3483.7	20.6	23.2

Table 5: Our best models on Ja-En.

of shrinking all layers except the maxout layer to the same degree, we adjust the aggressiveness of shrinking for each layer. We suggest three different setups (*Normal*, *Small*, and *Tiny*) with the layer sizes specified in Tab. 4. *3-Unfold-Normal* has the same number of parameters as the original NMT networks (size factor: 1.0), *3-Unfold-Small* is only half their size (size factor: 0.5), and *3-Unfold-Tiny* reduces the size by two thirds (size factor: 0.33). When comparing rows (a) and (c) in Tab. 5 we observe that *3-Unfold-Normal* yields a gain of 2.2 BLEU with respect to the original single system and a slight improvement in decoding speed at the same time.<sup>7</sup> Networks with the size factor 1.0 like *3-Unfold-Normal* are very likely to yield about the same decoding speed as the *Single* network regardless of the decoder implementation, machine learning framework, and hardware. Therefore, we think that similar results are possible on other platforms as well.

CPU decoding speed directly benefits even more from smaller setups – *3-Unfold-Tiny* is only 0.3 BLEU worse than *Single* but decoding on a single CPU is 3.4 times faster (row (a) vs. row (e) in Tab. 5). This is of great practical use: batch decoding with only two CPU threads surpasses production speed which is often set to 2000 words per minute (Beck et al., 2016). Our initial experiments in Tab. 6 suggest that the *Normal* setup is applicable to En-De as well, with substantial improve-

<sup>7</sup>To validate that the gains come from ensembling and unfolding and not from the layer sizes in *3-Unfold-Normal* we trained a network from scratch with the same dimensions. This network performed similarly to our *Single* system.

System	Wrds/Min. (GPU)	BLEU on news-test*		
		2014	2015	2016
Single	2128.7	19.6	21.9	24.6
2-Ensemble	1135.3	20.5	22.9	26.1
2-Unfold-Norm.	2099.1	20.7	23.1	25.8

Table 6: Our best models on En-De.

ments in BLEU compared to *Single* with about the same decoding speed.

## 5 Related Work

The idea of pruning neural networks to improve the compactness of the models dates back more than 25 years (LeCun et al., 1989). The literature is therefore vast (Augasta and Kathirvalavakumar, 2013). One line of research aims to remove unimportant network connections. The connections can be selected for deletion based on the second-derivative of the training error with respect to the weight (LeCun et al., 1989; Hassibi et al., 1993), or by a threshold criterion on its magnitude (Han et al., 2015). See et al. (2016) confirmed a high degree of weight redundancy in NMT networks.

In this work we are interested in removing neurons rather than single connections since we strive to shrink the unfolded network such that it resembles the layout of an individual model. We argued in Sec. 4 that removing neurons rather than connections does not only improve the model size but also the memory footprint and decoding speed. As explained in Sec. 3.1, our data-free method is an extension of the approach by Srinivas and Babu (2015); our extension performs significantly better on NMT networks. Our data-bound method (Sec. 3.2) is inspired by Babaeizadeh et al. (2016) as we combine neurons with similar activities during training, but we use linear combinations of multiple neurons to compensate for the loss of a neuron rather than merging pairs of neurons.

Using low rank matrices for neural network compression, particularly approximations via SVD, has been studied widely in the literature (Denil et al., 2013; Denton et al., 2014; Xue et al., 2013; Prabhavalkar et al., 2016; Lu et al., 2016). These approaches often use low rank matrices to approximate a full rank weight matrix in the original network. In contrast, we shrink an entire linear layer by applying SVD on the product of the incoming and outgoing weight matrices (Sec. 3.3).

In this paper we mimicked the output of the high performing but cumbersome ensemble by constructing a large unfolded network, and shrank this

network afterwards. Another approach, known as *knowledge distillation*, uses the large model (the teacher) to generate soft training labels for the smaller student network (Bucilu et al., 2006; Hinton et al., 2014). The student network is trained by minimizing the cross-entropy to the teacher. This idea has been applied to sequence modelling tasks such as machine translation and speech recognition (Wong and Gales, 2016; Kim and Rush, 2016; Freitag et al., 2017). Our approach can be computationally more efficient as the training set does not have to be decoded by the large teacher network.

Junczys-Dowmunt et al. (2016a; 2016b) reported gains from averaging the weight matrices of multiple checkpoints of the same training run. However, our attempts to replicate their approach were not successful. Averaging might work well when the behaviour of corresponding units is similar across networks, but that cannot be guaranteed when networks are trained independently.

## 6 Conclusion

We have described a generic method for improving the decoding speed and BLEU score of single system NMT. Our approach involves unfolding an ensemble of multiple systems into a single large neural network and shrinking this network by removing redundant neurons. Our best results on Japanese-English either yield a gain of 2.2 BLEU compared to the original single NMT network at about the same decoding speed, or a  $3.4\times$  CPU decoding speed up with only a minor drop in BLEU.

The current formulation of unfolding works for networks of the same topology as the concatenation of layers is only possible for analogous layers in different networks. Unfolding and shrinking diverse networks could be possible, for example by applying the technique only to the input and output layers or by some other scheme of finding associations between units in different models, but we leave this investigation to future work as models in NMT ensembles in current research usually have the same topology (Bojar et al., 2016; Senrich et al., 2016a; Chung et al., 2016; Neubig, 2016; Wu et al., 2016; Durrani et al., 2017).

## Acknowledgments

This work was supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC grant EP/L027623/1).



## Appendix: Probabilistic Interpretation of Data-Free and Data-Bound Shrinking

Data-free and data-bound shrinking can be interpreted as setting the expected difference between network outputs before and after a removal operation to zero under different assumptions.

For simplicity, we focus our probabilistic treatment of shrinking on single layer feedforward networks. Such a network maps an input  $\mathbf{x} \in \mathbb{R}^{m_{in}}$  to an output  $\mathbf{y} \in \mathbb{R}^{m_{out}}$ . The  $l$ -th output  $y_l$  is computed according the following equation

$$y_l = \sum_{k \in [1, m]} \sigma(\mathbf{x} \mathbf{u}_k^T) V_{k,l} \quad (7)$$

where  $\mathbf{u}_k \in \mathbb{R}^{m_{in}}$  is the incoming weight vector of the  $k$ -th hidden neuron (denoted as  $U_{:,k}$  in the main paper) and  $V \in \mathbb{R}^{m \times m_{out}}$  the outgoing weight matrix of the  $m$ -dimensional hidden layer. We now remove the  $j$ -th neuron in the hidden layer and modify the outgoing weights to compensate for the removal:

$$y'_l = \sum_{k \in [1, m] \setminus \{j\}} \sigma(\mathbf{x} \mathbf{u}_k^T) V'_{k,l} \quad (8)$$

where  $y'_l$  is the output after the removal operation and  $V' \in \mathbb{R}^{m \times m_{out}}$  are the modified outgoing weights. Our goal is to choose  $V'$  such that the expected error introduced by removing neuron  $j$  is zero:

$$\mathbb{E}_{\mathbf{x}}(y_l - y'_l) = 0 \quad (9)$$

**Data-free shrinking** Data-free shrinking makes two assumptions to satisfy Eq. 9. First, we assume that the incoming weight vector  $\mathbf{u}_j$  can be represented as linear combination of the other weight vectors.

$$\mathbf{u}_j = \sum_{k \in [1, m] \setminus \{j\}} \lambda_k \mathbf{u}_k \quad (10)$$

Second, it assumes that the neuron activation function  $\sigma(\cdot)$  is linear. Starting with Eqs. 7 and 8 we can write  $\mathbb{E}_{\mathbf{x}}(y_l - y'_l)$  as

$$\begin{aligned} & \mathbb{E}_{\mathbf{x}} \left( \sigma(\mathbf{x} \mathbf{u}_j^T) V_{j,l} + \underbrace{\sum_{k \in [1, m] \setminus \{j\}} \sigma(\mathbf{x} \mathbf{u}_k^T) (V_{k,l} - V'_{k,l})}_{:=R} \right) \\ \stackrel{\text{Eq. 10}}{=} & \mathbb{E}_{\mathbf{x}} \left( \sigma(\mathbf{x} \left( \sum_{k \in [1, m] \setminus \{j\}} \lambda_k \mathbf{u}_k \right)^T) V_{j,l} + R \right) \\ \stackrel{\sigma(\cdot) \text{ lin.}}{=} & \mathbb{E}_{\mathbf{x}} \left( \sum_{k \in [1, m] \setminus \{j\}} \sigma(\mathbf{x} \mathbf{u}_k^T) \lambda_k V_{j,l} + R \right) \\ = & \sum_{k \in [1, m] \setminus \{j\}} \mathbb{E}_{\mathbf{x}} \left( \sigma(\mathbf{x} \mathbf{u}_k^T) \right) (V_{k,l} - V'_{k,l} + \lambda_k V_{j,l}) \end{aligned}$$

We set this term to zero (and thus satisfy Eq. 9) by setting each component of the sum to zero.

$$\forall k \in [1, m] \setminus \{j\} : V'_{k,l} = V_{k,l} + \lambda_k V_{j,l} \quad (11)$$

This condition is directly implemented by the update rule in our shrinking algorithm (Eq. 3 and 4).

**Data-bound shrinking** Data-bound shrinking does not require linearity in  $\sigma(\cdot)$ . It rather assumes that the expected value of the neuron activity  $j$  is a linear combination of the expected values of the other activities:

$$\mathbb{E}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_j^T)) = \sum_{k \in [1, m] \setminus \{j\}} \lambda_k \mathbb{E}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_k^T)) \quad (12)$$

$\mathbb{E}_{\mathbf{x}}(\cdot)$  is estimated using importance sampling:

$$\hat{\mathbb{E}}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_k^T); \mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}' \in \mathcal{X}} \sigma(\mathbf{x}' \mathbf{u}_k^T) \quad (13)$$

In practice, the samples in  $\mathcal{X}$  are collected in the activity matrix  $A$  from Sec. 3.2. We can satisfy Eq. 9 by using the  $\lambda$ -values from Eq. 12, so that  $\mathbb{E}_{\mathbf{x}}(y_l - y'_l)$  becomes

$$\begin{aligned} & \stackrel{\text{Eqs. 7,8}}{=} \mathbb{E}_{\mathbf{x}} \left( \sigma(\mathbf{x} \mathbf{u}_j^T) V_{j,l} + \sum_{k \in [1, m] \setminus \{j\}} \sigma(\mathbf{x} \mathbf{u}_k^T) (V_{k,l} - V'_{k,l}) \right) \\ = & \mathbb{E}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_j^T) V_{j,l}) + \sum_{k \in [1, m] \setminus \{j\}} \mathbb{E}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_k^T)) (V_{k,l} - V'_{k,l}) \\ \stackrel{\text{Eq. 12}}{=} & \sum_{k \in [1, m] \setminus \{j\}} \mathbb{E}_{\mathbf{x}}(\sigma(\mathbf{x} \mathbf{u}_k^T)) (V_{k,l} - V'_{k,l} + \lambda_k V_{j,l}) \end{aligned}$$

Again, we set this to zero using Eq. 11.

## References

- M. Gethsiyal Augasta and Thangairulappan Kathirvalavakumar. 2013. Pruning algorithms of neural networks – a comparative study. *Central European Journal of Computer Science*, 3(3):105–115.
- Mohammad Babaeizadeh, Paris Smaragdis, and Roy H. Campbell. 2016. NoiseOut: A simple way to prune neural networks. In *Proceedings of the 1st International Workshop on Efficient Methods for Deep Neural Networks (EMDNN)*.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*, Toulon, France.
- Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. Theano: new features and speed improvements. In *NIPS*, South Lake Tahoe, Nevada, USA.
- Daniel Beck, Adrià de Gispert, Gonzalo Iglesias, Aurelien Waite, and Bill Byrne. 2016. [Speed-constrained tuning for statistical machine translation using Bayesian optimization](#). In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 856–865, San Diego, California. Association for Computational Linguistics.
- Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. [Findings of the 2016 conference on machine translation](#). In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.
- Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. [Learning phrase representations using RNN encoder-decoder for statistical machine translation](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. [A character-level decoder without explicit segmentation for neural machine translation](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703, Berlin, Germany. Association for Computational Linguistics.
- Fabien Cromieres, Chenhui Chu, Toshiaki Nakazawa, and Sadao Kurohashi. 2016. [Kyoto university participation to WAT 2016](#). In *Proceedings of the 3rd Workshop on Asian Translation (WAT2016)*, pages 166–174, Osaka, Japan. The COLING 2016 Organizing Committee.
- Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. 2013. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156.
- Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277.
- Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.
- John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pages 2121–2159.
- Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and Stephan Vogel. 2017. QCRI machine translation systems for IWSLT 16. *arXiv preprint arXiv:1701.03924*.
- Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation. *arXiv preprint arXiv:1702.01802*.
- Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Max-out networks. In *ICML*, pages 1319–1327.
- Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143.
- Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.
- Babak Hassibi, David G. Stork, et al. 1993. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pages 164–164.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*.

- Marcin Junczys-Dowmunt, Tomasz Dwojak, and Hieu Hoang. 2016a. Is neural machine translation ready for deployment? A case study on 30 translation directions. In *International Workshop on Spoken Language Translation IWSLT*.
- Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. 2016b. [The AMU-UEDIN submission to the WMT16 news translation task: Attention-based NMT models as feature functions in phrase-based SMT](#). In *Proceedings of the First Conference on Machine Translation*, pages 319–325, Berlin, Germany. Association for Computational Linguistics.
- Yoon Kim and Alexander M. Rush. 2016. [Sequence-level knowledge distillation](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.
- Yann LeCun, John S. Denker, Sara A. Solla, Richard E. Howard, and Lawrence D. Jackel. 1989. Optimal brain damage. In *NIPS*, volume 2, pages 598–605.
- Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. 2016. Learning compact recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5960–5964. IEEE.
- Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitry Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. 2015. Blocks and Fuel: Frameworks for deep learning. *CoRR*.
- Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016. ASPEC: Asian scientific paper excerpt corpus. In *LREC*, pages 2204–2208, Portoroz, Slovenia.
- Graham Neubig. 2016. [Lexicons and minimum risk training for neural machine translation: NAIST-CMU at WAT2016](#). In *Proceedings of the 3rd Workshop on Asian Translation (WAT2016)*, pages 119–125, Osaka, Japan. The COLING 2016 Organizing Committee.
- Graham Neubig, Makoto Morishita, and Satoshi Nakamura. 2015. Neural reranking improves subjective quality of machine translation: NAIST at WAT2015. In *Workshop on Asian Translation*, pages 35–41.
- Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Lan McGraw. 2016. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5970–5974. IEEE.
- Abigail See, Minh-Thang Luong, and D. Christopher Manning. 2016. [Compression of neural machine translation models via pruning](#). In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 291–301. Association for Computational Linguistics.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. [Edinburgh neural machine translation systems for WMT 16](#). In *Proceedings of the First Conference on Machine Translation*, pages 371–376, Berlin, Germany. Association for Computational Linguistics.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. [Neural machine translation of rare words with subword units](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.
- Suraj Srinivas and R. Venkatesh Babu. 2015. [Data-free parameter pruning for deep neural networks](#). In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 31.1–31.12. BMVA Press.
- Felix Stahlberg, Eva Hasler, Danielle Saunders, and Bill Byrne. 2017. SGNMT – A flexible NMT decoding platform for quick prototyping of new models and search strategies. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 3104–3112. MIT Press.
- Jeremy HM. Wong and Mark JF. Gales. 2016. Sequence student-teacher training of deep neural networks. *Interspeech 2016*, pages 2761–2765.
- Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- Jian Xue, Jinyu Li, and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369.
- Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.