Convolutional Sequence to Sequence Learning

Abstract

The prevalent approach to sequence to sequence learning maps an input sequence to a variable length output sequence via long short term memory networks. We introduce an architecture based entirely on convolutional neural networks. Compared to recurrent models, computations over all elements can be fully parallelized during training and optimization is easier since the number of non-linearities is fixed and independent of the input length. Our use of gated linear units eases gradient propagation and we equip each decoder layer with a separate attention module. We evaluate our approach on machine translation and text summarization. On WMT'16 English-Romanian translation we achieve a new state of the art and on both WMT'14 English-German and WMT'14 English-French translation we perform on par to the best published LSTM results or outperform them.

1. Introduction

Sequence to sequence learning has been successful on many tasks such as machine translation, speech recognition (Sutskever et al., 2014; Chorowski et al., 2015) and text summarization (Rush et al., 2015; Nallapati et al., 2016; Shen et al., 2016). The dominant approach to date encodes the input sequence with a series of bi-directional recurrent neural networks (RNN) and generates a variable length output with another set of decoder RNNs, both of which interface via a soft-attention mechanism (Bahdanau et al., 2014; Luong et al., 2015a). In machine translation, this architecture has been demonstrated to outperform traditional phrase-based models by large margins (Sennrich et al., 2016b; Zhou et al., 2016; Wu et al., 2016; §2).

Convolutional neural networks are less common for sequence modeling, despite several advantages (Waibel et al., 1989; LeCun & Bengio, 1995). Compared to recurrent layers, convolutions create representations for fixed size contexts, however, the effective context size of the network can easily be made larger by stacking several layers on top of each other. This allows to precisely control the maximum length of dependencies to be modeled. Convolutional networks do not depend on the computations of the previous time step and therefore allow parallelization over every element in a sequence. This contrasts with RNNs which maintain a hidden state of the entire past that prevents parallel computation within a sequence.

Multi-layer convolutional neural networks create hierarchical representations over the input sequence in which nearby input elements interact at lower layers while distant elements interact at higher layers. This resembles syntactic grammar formalisms which view sentences as hierarchical phrase-structure trees with noun-phrases and verb-phrases that have further internal structure (Manning & Schütze, 1999). Hierarchical structure provides a shorter path to capture long-range dependencies between inputs compared to the chain structure modeled by recurrent networks, e.g. we can obtain a feature representation capturing relationships within a window of n words by applying only $\mathcal{O}(\frac{n}{k})$ convolutional operations for a kernel of width k, compared to a linear number $\mathcal{O}(n)$ for recurrent neural networks. Inputs to a convolutional network are fed through a constant number of kernels and non-linearities, whereas recurrent networks apply up to n operations and non-linearities to the first word and only a single set of operations to the last word. Fixing the number of non-linearities applied to the inputs also eases learning.

Recent work has applied convolutional neural networks to sequence modeling such as Bradbury et al. (2016) who introduce recurrent pooling between a succession of convolutional layers or Kalchbrenner et al. (2016) who tackle neural translation without attention. However, none of these approaches has been demonstrated to improve over recurrent architectures on large benchmark datasets. Gated convolutions have been previously explored for machine translation by Meng et al. (2015) but their evaluation was restricted to a small dataset and the model was used in tandem with a traditional count-based model. Architectures which are partially convolutional have shown strong performance on larger tasks but their decoder is still recurrent (Gehring et al., 2016).

In this paper we propose an architecture for sequence to sequence modeling that is entirely convolutional. Our model is equipped with gated linear units (Dauphin et al., 2016) and residual connections (He et al., 2015a). We also use attention in every decoder layer and the combination of these choices enables us to tackle large scale problems (§3).

We evaluate our approach on several large datasets for ma-

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

chine translation as well as summarization and compare to the current best recurrent architectures reported in the literature. On WMT'16 English-Romanian translation we achieve a new state-of-the-art, outperforming the previous best result by over 1.3 BLEU and on WMT'14 English-German we perform on par to the current best LSTM setup of Wu et al. (2016). On a subset of the WMT'14 English-French task we outperform the best previous result by 1.6 BLEU (Zhou et al., 2016) on the full dataset we outperform the word-based system of Wu et al. (2016) by 0.5 BLEU with models trained in 14 days on 8 GPUs compared to their 96 GPU setup (§4, §5).

2. Recurrent Sequence to Sequence Learning

Sequence to sequence learning has been addressed with encoder-decoder architectures based on recurrent neural networks (Sutskever et al., 2014; Bahdanau et al., 2014). The encoder RNN processes an input sequence $\mathbf{x} = (x_1, \ldots, x_m)$ of m elements and returns state representations $\mathbf{z} = (z_1 \ldots, z_m)$. The decoder RNN takes $\mathbf{z}$ and generates the output sequence $\mathbf{y} = (y_1, \ldots, y_n)$ left to right, one element at a time. To generate output y_{i+1}, the decoder computes a new hidden state h_{i+1} based on the previous state h_i, an embedding g_i of the previous target language word y_i, as well as a conditional input c_i derived from the encoder output $\mathbf{z}$. Based on this generic formulation, various encoder-decoder architectures have been proposed, which differ mainly in the conditional input and the type of RNN.

Models without attention consider only the final encoder state z_m by setting $c_i = z_m$ for all i (Cho et al., 2014), or simply initialize the first decoder state with z_m (Sutskever et al., 2014), in which case c_i is not used. Architectures with attention (Bahdanau et al., 2014; Luong et al., 2015a) compute c_i as a weighted sum of $(z_1 \ldots, z_m)$ at each time step. The weights of the sum are referred to as attention scores and allow the network to focus on different parts of the input sequence as it generates the output sequences. Attention scores are computed by essentially comparing each encoder state z_j to a combination of the previous decoder state h_i and the last prediction y_i; the result is normalized to be a distribution over input elements.

Popular choices for recurrent networks in encoder-decoder models are long short term memory networks (LSTM; Hochreiter & Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014). Both extend Elman RNNs (Elman, 1990) with a gating mechanism that allows the memorization of information from previous time steps in order to model long-term dependencies. Most recent approaches also rely on bi-directional encoders to build representations of both past and future contexts (Bahdanau et al., 2014; Zhou et al., 2016; Wu et al., 2016). Models with many layers often rely on shortcut or residual connections (He et al., 2015a; Zhou et al., 2016; Wu et al., 2016).

3. A Convolutional Architecture

Next we introduce an entirely convolutional architecture for sequence to sequence modeling. Instead of relying on recurrent networks to compute intermediate states $\mathbf{z}$ and $\mathbf{h}$ we use convolutional neural networks (CNN).

3.1. Position Embeddings

First, we embed input elements $\mathbf{x} = (x_1, \ldots, x_m)$ in distributional space as $\mathbf{w} = (w_1, \ldots, w_m)$, where $w_j \in \mathbb{R}^f$ is a column in an embedding matrix $\mathcal{D} \in \mathbb{R}^{V \times f}$. We also equip our model with a sense of order by embedding the absolute position of input elements $\mathbf{p} = (p_1, \ldots, p_m)$ where $p_j \in \mathbb{R}^f$. Both are combined to obtain element representations $\mathbf{e} = (w_1 + p_1, \ldots, w_m + p_m)$. We proceed similarly for output elements that were already generated by the decoder network to yield $\mathbf{g} = (g_1, \ldots, g_n)$. Position embeddings are important in our architecture since they give our model a sense of which portion of the sequence in the input or output it is currently dealing with (§5.2).

3.2. Convolutional Block Structure

Both encoder and decoder networks share a simple block structure that computes intermediate states based on a fixed number of input elements. We denote the output of the l-th block as $\mathbf{h}^l = (h_1^l, \ldots, h_n^l)$ for the decoder network, and $\mathbf{z}^l = (z_1^l, \ldots, z_m^l)$ for the encoder network; we refer to blocks and layers interchangeably. Each block contains a one dimensional convolution followed by a non-linearity. For a network with a single block and kernel width k, each resulting state h_i^1 contains information over k input elements. Stacking several blocks on top of each other increases the number of input elements represented in a state. For instance, stacking 6 blocks with $k = 5$ results in an input field of 25 elements, i.e. each output depends on 25 inputs. Non-linearities allow the networks to exploit the full input field, or to focus on fewer elements if needed.

Each convolution kernel is parameterized as $W \in \mathbb{R}^{2d \times kd}$, $b_w \in \mathbb{R}^{2d}$ and takes as input $X \in \mathbb{R}^{k \times d}$ which is a concatenation of k input elements embedded in d dimensions and maps them to a single output element $Y \in \mathbb{R}^{2d}$ that has twice the dimensionality of the input elements; subsequent layers operate over the k output elements of the previous layer. We choose gated linear units (GLU; Dauphin et al., 2016) as non-linearity which implement a simple gating mechanism over the output of the convolution $Y = [A\ B] \in \mathbb{R}^{2d}$:

$$v([A\ B]) = A \otimes \sigma(B)$$

where $A, B \in \mathbb{R}^d$ are the inputs to the non-linearity, $\otimes$ is the point-wise multiplication and the output $v([A\ B]) \in \mathbb{R}^d$ is half the size of Y. The gates $\sigma(B)$ control which inputs A of the current context are relevant. A similar non-linearity has been introduced in Oord et al. (2016b) who

apply $\tanh$ to A but Dauphin et al. (2016) shows that GLUs perform better for language modelling.

To enable deep convolutional networks, we add residual connections from the input of each convolution to the output of the block (He et al., 2015a).

$$h_i^l = v(W^l[h_{i-k/2}^{l-1}, \ldots, h_{i+k/2}^{l-1}] + b_w^l) + h_i^{l-1}$$

For encoder networks we ensure that the output of the convolutional layers matches the input length by padding the input at each layer. However, for decoder networks we have to take care that no future information is available to the decoder (Oord et al., 2016a). Specifically, we pad the input by $k-1$ elements on both the left and right side by zero vectors, and then remove k elements from the end of the convolution output.

We also add linear mappings to project between the embedding size f and the hidden size $2d$. We apply such a transform to $\mathbf{w}$ when feeding it to the encoder network, to the encoder output z_j^u, to the final layer of the decoder just before the softmax $\mathbf{h^L}$, and to all decoder layers $\mathbf{h^l}$ before computing attention scores in (1).

Finally, the model computes a distribution over the T possible next target words y_{i+1} by transforming the top decoder output h_i^L via a linear layer with weights W_o and bias b_o:

$$p(y_{i+1}|y_1, \ldots, y_i, \mathbf{x}) = \text{softmax}(W_o h_{i+1}^L + b_o) \in \mathbb{R}^T$$

3.3. Multi-step Attention

We introduce a separate attention mechanism for each decoder layer. To compute the attention, we summarize the current decoder state h_i^l and an embedding g_i of the previous prediction y_i (Figure 1, bottom left):

$$d_i^l = W_d^l h_i^l + b_d^l + g_i \quad (1)$$

For decoder layer l the attention a_{ij}^l of state i and source element j is computed as a dot-product between the decoder state summary d_i^l and each output z_j^u of the last encoder block u:

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^m \exp\left(d_i^l \cdot z_t^u\right)}$$

The conditional input c_i^l for the current decoder layer is a weighted sum of the encoder outputs as well as the raw input element embeddings (Figure 1, bottom right).

$$c_i^l = \sum_{j=1}^m a_{ij}^l (z_j^u + e_j) \quad (2)$$

This is slightly different to recurrent approaches which compute both the attention and the weighted sum over z_j^u only. We found adding e^j beneficial and it resembels key-value memory networks where the keys are the z_j^u and the values are the $z_j^u + e_j$ (Miller et al., 2016). Encoder outputs z_j^u represent potentially large input contexts and e_j provides point information about a specific word that is useful when making a prediction.

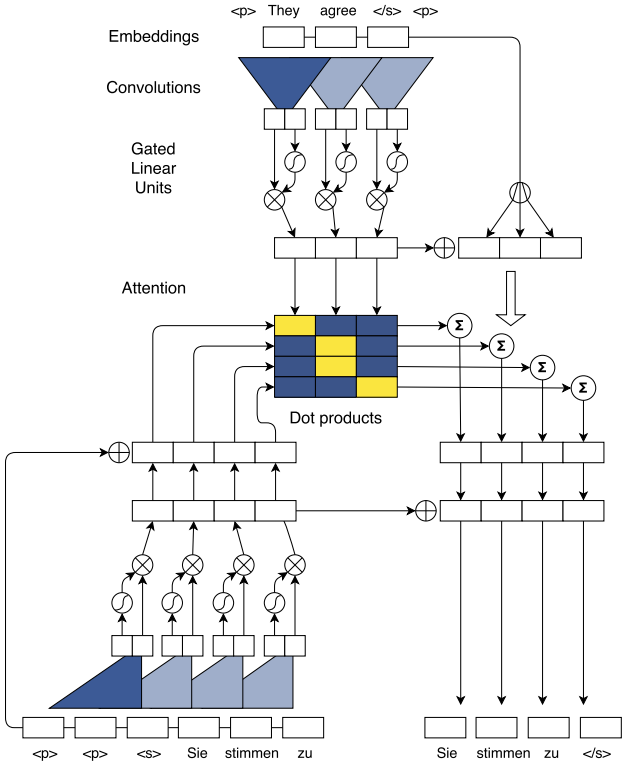

Figure 1. Convolutional architecture with the encoder network (top), attention (center) and the decoder network (bottom). During training, computation in the decoder and the attention can be parallelized over the target sequence unlike for recurrent networks.

Once c_i^l has been computed, it is simply added to the output of the corresponding decoder layer h_i^l. This can be seen as attention with multiple 'hops' (Sukhbaatar et al., 2015) compared to single step attention (Bahdanau et al., 2014; Luong et al., 2015a; Zhou et al., 2016; Wu et al., 2016). In particular, the attention of the first layer determines a useful source context which is then fed to the second layer which takes this information into account when computing attention etc. The decoder has also immediate access to the attention history of the $k-1$ previous time steps because the conditional inputs $c_{i-k}^{l-1}, \ldots, c_i^{l-1}$ are part of $h_{i-k}^{l-1}, \ldots, h_i^{l-1}$ which are input to h_i^l. This makes it easier for the model to take into account which previous inputs have been attended to already compared to recurrent nets where this information is in the recurrent state and needs to survive several non-linearities. Overall, our attention mechanism considers which words we previously attended

to (Yang et al., 2016) *and* also performs multiple attention 'hops' per time step. In §C, we plot attention scores for a deep decoder and show that at different layers, significantly different portions of the source sentence encoding are attended to.

Our convolutional architecture also allows to batch the attention computation across all elements of a sequence compared to RNNs (Figure 1, middle). For multi-hop attention we batch the computations of each pass individually.

3.4. Normalization Strategy

Batch-normalization stabilizes and accelerates learning by scaling activations to have zero mean and unit variance (Ioffe & Szegedy, 2015). For structured prediction tasks samples depend on each other and we need to take care not to access any future information in the decoder network during training. However, batch-normalization does access future information to compute the statistics for the scaling. Kalchbrenner et al. (2016) circumvents this issue by using part of the batch solely to estimate the relevant statistics. This has the disadvantage that some of the samples are not actually used for back-propagation. Furthermore, Kalchbrenner et al. (2016) report that the statistics learned by sub-batch normalization are specific to each length bucket in their data batching strategy which prevents them from sharing statistics between target sequences that would fall in different buckets. This greatly complicates search as the final length of a target sequence is not known at the beginning of generation and determining the length beforehand is less flexible.

We adopt a different strategy to stabilize learning which allows to train on all samples in a batch. In particular, we carefully initialize the model weights (§3.5) and scale parts of the network to ensure that the variance throughout the network does not increase or decrease. The idea is to scale the output of residual blocks as well as the conditional input from the attention to preserve the variance of activations. We multiply the sum of the input and the output of a residual block by $\sqrt{0.5}$ which halves the variance of the sum. This assumes that both summands have the same variance which is not always true but effective in practice.

The conditional input c_i^l is a weighted sum of m vectors (2) and we counteract a change in variance through scaling by $m\sqrt{1/m}$; we multiply by m to scale up the inputs to their original size, assuming the attention scores are uniformly distributed. This is generally not the case but we found this strategy to work well in practice.

For convolutional decoders with multiple attention, we scale the gradients for the encoder layers by the number of attention mechanisms we use; we exclude source word embeddings. We found this to stabilize learning since the encoder received too much gradient otherwise.

3.5. Initialization

Normalizing activations in the network when adding the output of different layers, e.g. residual connections, requires a careful weight initialization. The motivation for our initialization is the same as for the normalization: maintain variances of activations throughout the forward and backward passes. All embeddings are initialized from a normal distribution with mean 0 and standard deviation 0.1. For layers whose output is not subject to activation with gated linear units, we initialize weights from $\mathcal{N}(0, \sqrt{1/n_l})$ where n_l is the number of input connections for each neuron. This ensures that the variance of a normally distributed input is retained.

For layers with an immediately succeeding GLU activation, we propose a weight initialization scheme by adapting the derivations in (He et al., 2015b; Glorot & Bengio, 2010; Appendix A). If the GLU inputs are distributed with mean 0 and have sufficiently small variance, then we can approximate the output variance with $1/4$ of the input variance (Appendix A.1). Hence, we initialize the weights so that the input to the GLU activations have 4 times the variance of the layer input. This is achieved by drawing their initial values from $\mathcal{N}(0, \sqrt{4/n_l})$. Biases are uniformly set to zero when the network is constructed.

We apply dropout to the input of some layers so that inputs are retained with a probability of p. This can be seen as multiplication with a Bernoulli random variable taking value $1/p$ with probability p and 0 otherwise (Srivastava et al., 2014). The application of dropout will then cause the variance to be scaled by $1/p$. We aim to restore the incoming variance by initializing the respective layers with larger weights. Specifically, we use $\mathcal{N}(0, \sqrt{4p/n_l})$ for layers whose output is subject to GLU and $\mathcal{N}(0, \sqrt{p/n_l})$ otherwise (Appendix A.3).

4. Experimental Setup

4.1. Datasets

We consider three major WMT translation tasks as well as a text summarization task. Unless otherwise stated, we use a source vocabulary of 200K types for word-based models.

WMT'16 English-Romanian. We use the same data and pre-processing as Sennrich et al. (2016b) but remove sentences with more than 175 words. This results in 2.8M sentence pairs for training.[1] Our model is word-based instead of relying on byte-pair encoding (Sennrich et al., 2016a;b). We evaluate on newstest2016 and use a target vocabulary of 80K words.

[1]We followed the pre-processing of https://github.com/rsennrich/wmt16-scripts/blob/80e21e5/sample/preprocess.sh and added the back-translated data from http://data.statmt.org/rsennrich/wmt16_backtranslations/en-ro.

WMT'14 English-German. We use the same setup as Luong et al. (2015a) which comprises 4.5M sentence pairs and we test on newstest2014[2]. We consider models based on words and byte-pair encoding (BPE). For word models we use a target vocabulary of 160K words as well as vocabulary selection (Mi et al., 2016; L'Hostis et al., 2016) to restrict the output vocabulary to a small subset which speeds up training and testing. The average vocabulary size for each training batch is about 20K target words. For BPE models the source and target vocabularies consist of 80K sub-word tokens which were estimated using byte-pair encoding (Sennrich et al., 2016a).

WMT'14 English-French (12M). We use a commonly used subset of 12M sentence pairs (Schwenk, 2014), and remove sentences longer than 150 words. This results in 10.7M sentence-pairs for training. Results are reported on *ntst14*. For this task we set the target vocabulary to 30K types to be comparable with previous work.

WMT'14 English-French (36M). We also consider the full training set of 36M sentence pairs, and remove sentences longer than 175 words as well as pairs with a source/target length ratio exceeding 1.5. This results in 35.5M sentence-pairs for training. Results are reported on *ntst14*. We consider target vocabularies with 80K types for both words and BPE.

A small subset of the training data serves as validation set (about 0.5-1% for each dataset) for early stopping and learning rate annealing.

Abstractive summarization. We train on the Gigaword corpus (Graff et al., 2003) and pre-process it identically to Rush et al. (2015) resulting in 3.8M training examples and 190K for validation. We evaluate on the DUC-2004 test data comprising 500 article-title pairs (Over et al., 2007) and report three variants of recall-based ROUGE (Lin, 2004), namely, ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (longest-common substring). We also evaluate on a Gigaword test set of 2000 pairs which is identical to the one used by Rush et al. (2015) and we report F1 ROUGE similar to prior work. Similar to Shen et al. (2016) we use a source and target vocabulary of 30K words and require outputs to be at least 14 words long.

4.2. Model Parameters and Optimization

We use 512 hidden units for both encoders and decoders. All embeddings, including the output produced by the decoder before the final linear layer, have dimensionality 512; we use the same dimensionalities for linear layers mapping between the hidden and embedding sizes (§3.2). Convolutional layers are initialized as described in §3.5.

We train our convolutional models with Nesterov's accelerated gradient method (Sutskever et al., 2013) using a momentum value of 0.99 and renormalize gradients if their norm exceeds 0.1 (Pascanu et al., 2013). We use a learning rate of 0.25 and once the validation perplexity stops improving, we reduce the learning rate by an order of magnitude each epoch until it falls below 10^{-4}.

For all models, we use mini-batches of 64 sentences. We restrict the maximum number of words in a mini-batch to make sure that batches with long sentences still fit in GPU memory. If the threshold is exceeded, we simply split the batch in two and process each half separatedly. Gradients are normalized by the number of non-padding tokens per mini-batch. We also use weight normalization for all layers except for lookup tables (Salimans & Kingma, 2016).

Besides dropout on the embeddings and the decoder output, we also apply dropout to the input of the convolutional blocks (Srivastava et al., 2014). All models are implemented in Torch (Collobert et al., 2011) and trained on a single Nvidia M40 GPU except for WMT'14 English-French for which we use a multi-GPU setup on a single machine. We train on up to eight GPUs synchronously by maintaining copies of the model on each card and split the batch so that each worker computes 1/8-th of the gradients; at the end we sum the gradients via Nvidia NCCL. In this case, mini-batches consist of 512 sentences in total.

4.3. Evaluation

We report average results over three runs of each model, where runs differ in the initial random seed only. Translations are generated by a beam search and we normalize log-likelihood scores by sentence length. We use a beam of width 5. For WMT'14 English-German and WMT'14 English-French we tune a word penalty on separate dev sets (*newstest2015* and *ntst1213*). The word penalty adds $|\mathbf{y}|^{\alpha}$ to the log-likelihoods.

For word-based models, we perform unknown word replacement based on attention scores after generation (Jean et al., 2015). Unknown words are replaced by looking up the source word with the maximum attention score in a pre-computed dictionary. If the dictionary contains no translation, then we simply copy the source word. Dictionaries were extracted from the aligned training data that was aligned with `fast_align` (Dyer et al., 2013). Each source word is mapped to the target word it is most frequently aligned to. In our multi-step attention (§3.3) we simply average the attention scores over all layers. Finally, we compute case-sensitive tokenized BLEU, except for WMT'16 English-Romanian where we use detokenized BLEU to be comparable with Sennrich et al. (2016b).[3]

[2] http://nlp.stanford.edu/projects/nmt

[3] https://github.com/moses-smt/mosesdecoder/blob/617e8c8/scripts/generic/{multi-bleu.perl,mteval-v13a.pl}

5. Results

5.1. Recurrent vs. Convolutional Models

We first evaluate our convolutional model on three translation tasks. On WMT'16 English-Romanian translation we compare to Sennrich et al. (2016b) which is the winning entry on this language pair at WMT'16 (Bojar et al., 2016). Their model implements the attention-based sequence to sequence architecture of Bahdanau et al. (2014) and uses GRU cells both in the encoder and decoder. The vocabulary of this model is based on byte pair encoding (BPE); we use the same pre-processing but no BPE (§4).

Table 1 shows that our fully convolutional model (ConvS2S) outperforms the WMT'16 winning entry for English-Romanian by over 1.3 BLEU; the best random seed according to validation perplexity achieved BLEU 29.67. Our models are purely word-based and this particular instance has 20 layers in the encoder and 20 layers in the decoder and uses a kernel width of 3 for both. We expect that BPE would improve our results further. We trained for six days on a single GPU.

On WMT'14 English to German translation we compare to the following prior work: Kalchbrenner et al. (2016) propose a convolutional model based on characters without attention with 15 layers each in the encoder and decoder, Luong et al. (2015a) is based on a four layer LSTM attention model; Wu et al. (2016) represents the state of the art on this dataset and they use 8 encoder LSTMs as well as 8 decoder LSTMs, we quote their result for a word-based model, such as ours, as well as a word-piece model (Schuster & Nakajima, 2012) that bears similarity to BPE.

The results (Table 1) show that our convolutional model with a byte-pair encoding vocabulary achieves virtually the same accuracy as the current state-of-the-art on WMT'14 English-German which relies on word-piece tokenization. If we switch the vocabulary to words, then we match the accuracy of the best word-based model of Wu et al. (2016).[4] The very small gap to the word-piece model is likely due to the slightly better tokenization enabled by word-piece modeling. Word piece modeling is orthogonal to the underlying model architecture and we expect it to be equally beneficial to our model. For our BPE model, the encoder has 13 layers and the decoder has 8 layers with kernels of size 3. We trained this model on a single GPU in ten days.

Next, we test on the much larger WMT'14 English-French corpus where we consider both a commonly used subset of 12M sentence pairs for training as well as the full setup with 36M sentence pairs. On the subset, we compare to the following prior work: Bahdanau et al. (2014) is the original attention model based on a single-layer GRU in the encoder and decoder, Luong et al. (2015a) have a four-layer LSTM model, Jean et al. (2014) use a very large output vocabulary of 500K words and the very deep LSTM setup of Zhou et al. (2016) features 9 bi-directional LSTMs in the encoder and a 7-layer decoder. For the sbuset, we report single system accuracy similar to prior work and we choose the random seed which achieved the best validation perplexity over three runs.

Table 1 shows that we outperform the best prior work on this dataset by 1.6 BLEU which demonstrates that our fully convolutional model can improve over very strong LSTM setups. Specifically, the encoder of our model has 19 layers and kernel width 3 and the decoder consists of 9 layers with kernels of size 3. The model was trained on 4 GPUs over 10 days (§4).

Finally, we train on all data of the WMT'14 English-French task. We compare to the deep LSTM setup of Wu et al. (2016) who also average accuracy over several runs. Our models are still training but Table 1 shows that they already outperform the equivalent word-based model of Wu et al. (2016) by 0.5 BLEU. This is based on 20 layers in the encoder and 20 layers in the decoder. Their word piece approach performs better but this technique is orthogonal to the underlying model architecture and we expect it to be equally beneficial to our model, similar to WMT'14 English-German. Our results are based on training on 8 GPUs for two weeks compared to the 96 GPU setup of Wu et al. (2016). Zhou et al. (2016) report a very strong result of 39.2 BLEU for a single run. Our best single system result compares at 38.81 BLEU for a smaller configuration with 10 decoder layers.

5.2. Position Embeddings

In the following sections, we analyze the design choices we made in our architecture to give an intuition why they are important. The remaining results are on the WMT'14 English-German task with 13 encoder layers at kernel size 3 and 5 decoder layers at kernel size 5. All figures are averaged over three runs (§4) and BLEU is reported on newstest2014 before unknown word replacement.

We start with an experiment that removes the position embeddings from the encoder and decoder (§3.1). These embeddings allow our model to identify which portion of the source and target sequence it is dealing with. Table 2 shows that position embeddings are helpful but that our model still performs well without them. Removing the source position embeddings results in a larger accuracy decrease than target position embeddings. However, removing both source and target positions decreases accuracy only by 0.5 BLEU. We had assumed that the model would not be able to calibrate the length of the output sequences very well without explicit position information, however, the output lengths of models without position embeddings closely matches models with position information. This indicates that the

[4] Our setup uses a larger vocabulary, however, this makes only a very small difference because the additional words only represent a tiny fraction of the actual words in the data.

WMT'16 English-Romanian	BLEU
Sennrich et al. (2016b) GRU (BPE 90K)	28.1
ConvS2S (Word 80K)	29.45

WMT'14 English-German	BLEU
Kalchbrenner et al. (2016) ByteNet	18.9
Luong et al. (2015a) LSTM (Word 50K)	20.9
Wu et al. (2016) LSTM (Word 80K)	23.12
Wu et al. (2016) LSTM (Word pieces 32K)	24.61
ConvS2S (Word 160K)	23.13
ConvS2S (BPE 80K)	24.54

WMT'14 English-French (12M)	BLEU
Bahdanau et al. (2014) GRU (Word 30K)	28.45
Luong et al. (2015b) LSTM (Word 40K)	32.7
Jean et al. (2014) GRU (Word 500K)	34.60
Zhou et al. (2016) LSTM (Word 30K)	35.9
ConvS2S (Word 30K)	37.5

WMT'14 English-French (36M)	BLEU
Wu et al. (2016) LSTM (Word 80K)	37.90
Wu et al. (2016) LSTM (Word pieces 32K)	38.95
ConvS2S (Word 80K)	38.39

Table 1. Accuracy on WMT tasks comapred to previous work.

models can learn relative position information within the contexts visible to the encoder and decoder networks which can observe up to 27 and 25 words respectively.

Recurrent models typically do not use explicit position embeddings since they can learn where they are in the sequence through the recurrent hidden state computation. In our setting, the use of position embeddings requires only a simple addition to the input word embeddings which is a negligible overhead.

5.3. Multi-step Attention

The multiple attention mechanism (§3.3) computes a separate source context vector for each decoder layer. The computation also takes into account contexts computed for preceding decoder layers of the current time step as well as previous time steps that are within the receptive field of the decoder. How does multiple attention compare to attention in fewer layers or even only in a single layer as is usual? Table 3 shows that attention in all decoder layers achieves the best validation perplexity (Valid PPL) which is the metric we use to select models for final BLEU evaluation. Removing attention in the top layer (5) leads to comparable

	PPL	BLEU
ConvS2S	6.64	21.7
-source position	6.69	21.3
-target position	6.63	21.5
-source & target position	6.68	21.2

Table 2. Effect of removing position embeddings from our model in terms of validation perplexity (valid PPL) and BLEU.

Attn Layers	PPL	BLEU
1,2,3,4,5	6.65	21.4
1,2,3,4	6.70	21.4
1,2,3	6.95	21.2
1,2	6.92	21.4
1,3,5	6.97	20.9
1	7.15	21.2
2	7.09	21.2
3	7.66	20.3
4	7.66	20.3
5	7.66	20.3

Table 3. Multi-step attention in all five decoder layers or fewer layers in terms of validation perplexity and test BLEU.

accuracy but other configurations are significantly worse, either in terms of BLEU or validation perplexity, or both.

The computational overhead for attention is very small compared to the rest of the network. Training with attention in all five decoder layers processes 3624 target words per second on average on a single GPU, compared to 3772 words per second for attention in a single layer. This is only a 4% slow down when adding 4 attention modules. Most neural machine translation systems only use a single module. This demonstrates that attention is not the bottleneck in neural machine translation, even though it is quadratic in the sequence length (cf. Kalchbrenner et al. (2016)). Part of the reason for the low impact of attention on speed is that we batch the computation of an attention module over all target words, similar to Kalchbrenner et al. (2016). Batching in RNNs is restricted to the current time step. We use vocabulary selection for these experiments and the average output vocabulary contains 20K words per batch.

5.4. Kernel size and Depth

Figure 2 shows accuracy when we change the number of layers in the encoder or decoder. The kernel width for layers in the encoder is 3 and for the decoder it is 5. Deeper architectures are particularly beneficial for the encoder but less so for the decoder. Decoder setups with two layers already perform well whereas for the encoder accuracy keeps increasing steadily with more layers until up to 9 layers when accuracy starts to plateau.

	DUC-2004			Gigaword		
	RG-1 (R)	RG-2 (R)	RG-L (R)	RG-1 (F)	RG-2 (F)	RG-L (F)
RNN MLE (Shen et al., 2016)	24.92	8.60	22.25	32.67	15.23	30.56
RNN MRT (Shen et al., 2016)	30.41	**10.87**	26.79	**36.54**	16.59	33.44
WFE (Suzuki & Nagata, 2017)	**32.28**	10.54	**27.80**	36.30	17.31	**33.88**
ConvS2S	30.44	10.84	26.90	35.88	17.48	33.29

Table 4. Accuracy on two summarization tasks in terms of Rouge-1 (RG-1), Rouge-2 (RG-2), and Rouge-L (RG-L).

Kernel width	Encoder layers		
	5	9	13
3	20.6	21.2	21.6
5	20.8	21.0	21.4
7	20.8	21.3	21.1

Table 5. Encoder with different kernel width in terms of BLEU.

Kernel width	Decoder layers		
	3	5	7
3	21.1	21.7	21.6
5	21.1	21.6	21.2
7	21.4	21.3	21.3

Table 6. Decoder with different kernel width in terms of BLEU.

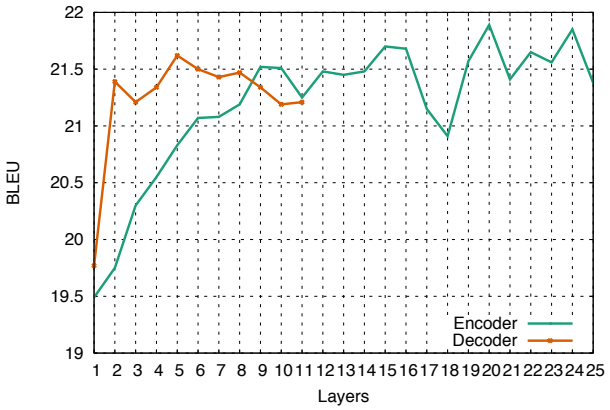

Figure 2. Encoder and decoder with different number of layers.

Aside from increasing the depth of the networks, we can also change the kernel width. Table 5 shows that encoders with narrow kernels and many layers perform better than wider kernels. These networks can also be faster since the amount of work to compute a kernel operating over 3 input elements is less than half compared to kernels over 7 elements. We see a similar picture for decoder networks with large kernel sizes (Table 6). Similar to above, more layers stop improving accuracy after a certain point. Dauphin et al. (2016) shows that context sizes of 20 words are often sufficient to achieve very good accuracy on language modeling for English.

5.5. Summarization

Finally, we evaluate our model on abstractive sentence summarization which takes a long sentence as input and outputs a shortened version. The current best models on this task are recurrent neural networks which either optimize the the evaluation metric (Shen et al., 2016) or address specific problems to summarization such as avoiding repeated words (Suzuki & Nagata, 2017). We use standard likelihood training for our model and use a simple setup with six layers in the encoder and decoder each, hidden size 256, batch size 128, and we trained on a single GPU in one night. Table 4 shows that our likelihood trained model outperforms the likelihood trained model (RNN MLE) of Shen et al. (2016) and is not far behind the best models on this task which benefit from task-specific optimization and model structure. We expect our model to benefit from these improvements as well.

6. Conclusion and Future Work

We introduce the first fully convolutional model for sequence to sequence learning. Compared to recurrent networks, our convolutional approach allows to discover compositional structure in the sequences more easily since representations are built hierarchically. Our model performs multiple attention steps and has access to previous attention vectors. We propose a normalization and initialization scheme that allows to train on the entire batch compared to sub-batch normalization which does not use all samples for gradient computation (Kalchbrenner et al., 2016).

To our knowledge we are the first to demonstrate that a convolutional architecture can outperform strong recurrent models on large benchmarks. We achieve a new state of the art on the WMT'16 English-Romanian translation task and outperform the previous best system by over 1.3 BLEU. On a subset of WMT'14 English-French we outperform the best previous result which is based on a very deep 16 layer LSTM model by 1.6 BLEU and we perform on par to the best word-based model on the WMT'14 English-German task. In future work, we would like to apply convolutional architectures to other sequence to sequence learning problems which may benefit from learning hierarchical representations as well.

References

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

Bojar, Ondej, Chatterjee, Rajen, Federmann, Christian, Graham, Yvette, Haddow, Barry, Huck, Matthias, Jimeno-Yepes, Antonio, Koehn, Philipp, Logacheva, Varvara, Monz, Christof, Negri, Matteo, Névéol, Aurélie, Neves, Mariana L., Popel, Martin, Post, Matt, Rubino, Raphaël, Scarton, Carolina, Specia, Lucia, Turchi, Marco, Verspoor, Karin M., and Zampieri, Marcos. Findings of the 2016 conference on machine translation. In *Proc. of WMT*, 2016.

Bradbury, James, Merity, Stephen, Xiong, Caiming, and Socher, Richard. Quasi-Recurrent Neural Networks. *arXiv preprint arXiv:1611.01576*, 2016.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of EMNLP*, 2014.

Chorowski, Jan K, Bahdanau, Dzmitry, Serdyuk, Dmitriy, Cho, Kyunghyun, and Bengio, Yoshua. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pp. 577–585, 2015.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clement. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*, 2011. URL http://torch.ch.

Dauphin, Yann N., Fan, Angela, Auli, Michael, and Grangier, David. Language modeling with gated linear units. *arXiv preprint arXiv:1612.08083*, 2016.

Dyer, Chris, Chahuneau, Victor, and Smith, Noah A. A Simple, Fast, and Effective Reparameterization of IBM Model 2. In *Proc. of ACL*, 2013.

Elman, Jeffrey L. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990.

Gehring, Jonas, Auli, Michael, Grangier, David, and Dauphin, Yann N. A Convolutional Encoder Model for Neural Machine Translation. *arXiv preprint arXiv:1611.02344*, 2016.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. *The handbook of brain theory and neural networks*, 2010.

Graff, David, Kong, Junbo, Chen, Ke, and Maeda, Kazuaki. English gigaword. *Linguistic Data Consortium, Philadelphia*, 2003.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep Residual Learning for Image Recognition. In *Proc. of CVPR*, 2015a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015b.

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

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

Jean, Sébastien, Cho, Kyunghyun, Memisevic, Roland, and Bengio, Yoshua. On Using Very Large Target Vocabulary for Neural Machine Translation. *arXiv preprint arXiv:1412.2007v2*, 2014.

Jean, Sébastien, Firat, Orhan, Cho, Kyunghyun, Memisevic, Roland, and Bengio, Yoshua. Montreal Neural Machine Translation systems for WMT15. In *Proc. of WMT*, pp. 134–140, 2015.

Kalchbrenner, Nal, Espeholt, Lasse, Simonyan, Karen, van den Oord, Aaron, Graves, Alex, and Kavukcuoglu, Koray. Neural Machine Translation in Linear Time. *arXiv*, 2016.

LeCun, Yann and Bengio, Yoshua. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

L'Hostis, Gurvan, Grangier, David, and Auli, Michael. Vocabulary Selection Strategies for Neural Machine Translation. *arXiv preprint arXiv:1610.00072*, 2016.

Lin, Chin-Yew. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pp. 74–81, 2004.

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

Luong, Minh-Thang, Sutskever, Ilya, Le, Quoc V, Vinyals, Oriol, and Zaremba, Wojciech. Addressing the Rare Word Problem in Neural Machine Translation. In *Proc. of ACL*, 2015b.

Manning, Christopher D and Schütze, Hinrich. Foundations of statistical natural language processing, 1999.

Meng, Fandong, Lu, Zhengdong, Wang, Mingxuan, Li, Hang, Jiang, Wenbin, and Liu, Qun. Encoding Source Language with Convolutional Neural Network for Machine Translation. In *Proc. of ACL*, 2015.

Mi, Haitao, Wang, Zhiguo, and Ittycheriah, Abe. Vocabulary Manipulation for Neural Machine Translation. In *Proc. of ACL*, 2016.

Miller, Alexander H., Fisch, Adam, Dodge, Jesse, Karimi, Amir-Hossein, Bordes, Antoine, and Weston, Jason. Key-value memory networks for directly reading documents. In *Proc. of EMNLP*, 2016.

Nallapati, Ramesh, Zhou, Bowen, Gulcehre, Caglar, Xiang, Bing, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proc. of EMNLP*, 2016.

Oord, Aaron van den, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016a.

Oord, Aaron van den, Kalchbrenner, Nal, Vinyals, Oriol, Espeholt, Lasse, Graves, Alex, and Kavukcuoglu, Koray. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016b.

Over, Paul, Dang, Hoa, and Harman, Donna. Duc in context. *Information Processing & Management*, 43(6): 1506–1520, 2007.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1310–1318, 2013.

Rush, Alexander M, Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence summarization. In *Proc. of EMNLP*, 2015.

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Schuster, Mike and Nakajima, Kaisuke. Japanese and korean voice search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 5149–5152. IEEE, 2012.

Schwenk, Holger. http://www-lium.univ-lemans.fr/~schwenk/cslm_joint_paper/, 2014. Accessed: 2016-10-15.

Sennrich, Rico, Haddow, Barry, and Birch, Alexandra. Neural Machine Translation of Rare Words with Subword Units. In *Proc. of ACL*, 2016a.

Sennrich, Rico, Haddow, Barry, and Birch, Alexandra. Edinburgh Neural Machine Translation Systems for WMT 16. In *Proc. of WMT*, 2016b.

Shen, Shiqi, Zhao, Yu, Liu, Zhiyuan, Sun, Maosong, et al. Neural headline generation with sentence-wise optimization. *arXiv preprint arXiv:1604.01904*, 2016.

Srivastava, Nitish, Hinton, Geoffrey E., Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent Neural Networks from overfitting. *JMLR*, 15:1929–1958, 2014.

Sukhbaatar, Sainbayar, Weston, Jason, Fergus, Rob, and Szlam, Arthur. End-to-end Memory Networks. In *Proc. of NIPS*, pp. 2440–2448, 2015.

Sutskever, Ilya, Martens, James, Dahl, George E., and Hinton, Geoffrey E. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to Sequence Learning with Neural Networks. In *Proc. of NIPS*, pp. 3104–3112, 2014.

Suzuki, Jun and Nagata, Masaaki. Cutting-off redundant repeating generations for neural abstractive summarization. *arXiv preprint arXiv:1701.00138*, 2017.

Waibel, Alex, Hanazawa, Toshiyuki, Hinton, Geoffrey, Shikano, Kiyohiro, and Lang, Kevin J. Phoneme Recognition using Time-delay Neural Networks. *IEEE transactions on acoustics, speech, and signal processing*, 37 (3):328–339, 1989.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V, Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yang, Zichao, Hu, Zhiting, Deng, Yuntian, Dyer, Chris, and Smola, Alex. Neural Machine Translation with Recurrent Attention Modeling. *arXiv preprint arXiv:1607.05108*, 2016.

Zhou, Jie, Cao, Ying, Wang, Xuguang, Li, Peng, and Xu, Wei. Deep Recurrent Models with Fast-Forward Connections for Neural Machine Translation. *arXiv preprint arXiv:1606.04199*, 2016.

A. Weight Initialization

We derive a weight initialization scheme tailored to the GLU activation function similar to Glorot & Bengio (2010); He et al. (2015b) by focusing on the variance of activations within the network for both forward and backward passes. We also detail how we modify the weight initialization for dropout.

A.1. Forward Pass

Assuming that the inputs $\mathbf{x}_l$ of a convolutional layer l and its weights W_l are independent and identically distributed (i.i.d.), the variance of its output, computed as $\mathbf{y}_l = W_l \mathbf{x}_l + \mathbf{b}_l$, is

$$Var\big[y_l\big] = n_l Var\big[w_l x_l\big] \tag{3}$$

where n_l is the number inputs to the layer. For one-dimensional convolutional layers with kernel width k and input dimension c, this is kc. We adopt the notation in (He et al., 2015b), i.e. y_l, w_l and x_l represent the random variables in $\mathbf{y}_l$, W_l and $\mathbf{x}_l$. With w_l and x_l independent from each other and normally distributed with zero mean, this amounts to

$$Var\big[y_l\big] = n_l Var\big[w_l\big] Var\big[x_l\big]. \tag{4}$$

$\mathbf{x}_l$ is the result of the GLU activation function $\mathbf{y}^a_{l-1}\, \sigma(\mathbf{y}^b_{l-1})$ with $\mathbf{y}_{l-1} = (\mathbf{y}^a_{l-1}, \mathbf{y}^b_{l-1})$, and $\mathbf{y}^a_{l-1}, \mathbf{y}^b_{l-1}$ i.i.d. Next, we formulate upper and lower bounds in order to approximate $Var[x_l]$. If $\mathbf{y}_{l-1}$ follows a symmetric distribution with mean 0, then

$$Var\big[x_l\big] = Var\big[y^a_{l-1}\, \sigma(\mathbf{y}^b_{l-1})\big] \tag{5}$$

$$= E\big[\big(y^a_{l-1}\, \sigma(\mathbf{y}^b_{l-1})\big)^2\big] - E^2\big[y^a_{l-1}\, \sigma(\mathbf{y}^b_{l-1})\big] \tag{6}$$

$$= Var[y^a_{l-1}] E\big[\sigma(\mathbf{y}^b_{l-1})^2\big]. \tag{7}$$

A lower bound is given by $(1/4)Var[y^a_{l-1}]$ when expanding (6) with $E^2[\sigma(\mathbf{y}^b_{l-1})] = 1/4$:

$$Var\big[x_l\big] = Var\big[y^a_{l-1}\, \sigma(\mathbf{y}^b_{l-1})\big] \tag{8}$$

$$= Var\big[y^a_{l-1}\big] E^2\big[\sigma(\mathbf{y}^b_{l-1})\big] + Var\big[y^a_{l-1}\big] Var\big[\sigma(\mathbf{y}^b_{l-1})\big] \tag{9}$$

$$= \frac{1}{4} Var\big[y^a_{l-1}\big] + Var\big[y^a_{l-1}\big] Var\big[\sigma(\mathbf{y}^b_{l-1})\big] \tag{10}$$

and $Var[y^a_{l-1}]Var[\sigma(\mathbf{y}^b_{l-1})] > 0$. We utilize the relation $\sigma(x)^2 \leq (1/16)x^2 - 1/4 + \sigma(x)$ (Appendix B) to provide an upper bound on $E[\sigma(x)^2]$:

$$E[\sigma(x)^2] \leq E\big[\frac{1}{16}x^2 - \frac{1}{4} + \sigma(x)\big] \tag{11}$$

$$= \frac{1}{16} E[x^2] - \frac{1}{4} + E[\sigma(x)] \tag{12}$$

With $x \sim \mathcal{N}(0, std(x))$, this yields

$$E\big[\sigma(x)^2\big] \leq \frac{1}{16} E\big[x^2\big] - \frac{1}{4} + \frac{1}{2} \tag{13}$$

$$= \frac{1}{16} Var\big[x\big] + \frac{1}{4}. \tag{14}$$

With (7) and $Var[y^a_{l-1}] = Var[y^b_{l-1}] = Var[y_{l-1}]$, this results in

$$Var\big[x_l\big] \leq \frac{1}{16} Var\big[y_{l-1}\big]^2 + \frac{1}{4} Var\big[y_{l-1}\big]. \tag{15}$$

We initialize the embedding matrices in our network with small variances (around 0.01), which allows us to dismiss the quadratic term and approximate the GLU output variance with

$$Var[x_l] \approx \frac{1}{4} Var[y_{l-1}]. \tag{16}$$

If L network layers of equal size and with GLU activations are combined, the variance of the final output $\mathbf{y}_L$ is given by

$$Var[y_L] \approx Var[y_1] \left(\prod_{l=2}^{L} \frac{1}{4} n_l Var[w_l] \right). \tag{17}$$

Following (He et al., 2015b), we aim to satisfy the condition

$$\frac{1}{4} n_l Var\big[w_l\big] = 1, \forall l \tag{18}$$

so that the activations in a network are neither exponentially magnified nor reduced. This is achieved by initializing W_l from $\mathcal{N}(0, \sqrt{4/n_l})$.

A.2. Backward Pass

The gradient of a convolutional layer is computed via back-propagation as $\Delta \mathbf{x}_l = \hat{W}_l \mathbf{y}_l$. Considering separate gradients $\Delta \mathbf{y}^a_l$ and $\Delta \mathbf{y}^b_l$ for GLU, the gradient of $\mathbf{x}$ is given by

$$\Delta \mathbf{x}_l = \hat{W}^a_l \Delta \mathbf{y}^a_l + \hat{W}^b_l \Delta \mathbf{y}^b_l. \tag{19}$$

$\hat{W}$ corresponds to W with re-arranged weights to enable back-propagation. Analogously to the forward pass, Δx_l, $\hat{w}_l$ and Δy_l represent the random variables for the values in $\Delta \mathbf{x_l}$, $\hat{W}_l$ and $\Delta \mathbf{y_l}$, respectively. Note that W and $\hat{W}$ contain the same values, i.e. $\hat{w} = w$. Similar to (3), the variance of Δx_l is

$$Var[\Delta x_l] = \hat{n}_l \left(Var[w^a_l] Var[\Delta y^a_l] + Var[w^b_l] Var[\Delta y^b_l] \right). \tag{20}$$

Here, $\hat{n}_l$ is the number of inputs to layer $l+1$. The gradients for the GLU inputs are:

$$\Delta \mathbf{y}^a_l = \Delta \mathbf{x}_{l+1}\, \sigma(\mathbf{y}^b_l) \quad \text{and} \tag{21}$$

$$\Delta \mathbf{y}^b_l = \Delta \mathbf{x}_{l+1} \mathbf{y}^a_l\, \sigma'(\mathbf{y}^b_l). \tag{22}$$

The approximation for the forward pass can be used for $Var[\Delta y_l^a]$, and for estimating $Var[\Delta y_l^b]$ we assume an upper bound on $E[\sigma'(y_l^b)^2]$ of $1/16$ since $\sigma'(y_l^b) \in [0, \frac{1}{4}]$. Hence,

$$Var[\Delta y_l^a] - \frac{1}{4} Var[\Delta x_{l+1}] \leq \frac{1}{16} Var[\Delta x_{l+1}] Var[y_l^b])] \tag{23}$$

$$Var[\Delta y_l^b] \leq \frac{1}{16} \Delta Var[\Delta x_{l+1}] Var[y_l^a] \tag{24}$$

We observe relatively small gradients in our network, typically around 0.001 at the start of training. Therefore, we approximate by discarding the quadratic terms above, i.e.

$$Var[\Delta y_l^a] \approx \frac{1}{4} Var[\Delta x_{l+1}] \tag{25}$$

$$Var[\Delta y_l^b] \approx 0 \tag{26}$$

$$Var[\Delta x_l] \approx \frac{1}{4} \hat{n}_l Var[w_l^a] Var[\Delta x_{l+1}] \tag{27}$$

As for the forward pass, the above result can be generalized to backpropagation through many successive layers, resulting in

$$Var[\Delta x_2] \approx Var[\Delta x_{L+1}] \left(\prod_{l=2}^{L} \frac{1}{4} \hat{n}_l Var[w_l^a] \right) \tag{28}$$

and a similar condition, i.e. $(1/4)\hat{n}_l Var[w_l^a] = 1$. In the networks we consider, successions of convolutional layers usually operate on the same number of inputs so that most cases $n_l = \hat{n}_l$. Note that W_l^b is discarded in the approximation; however, for the sake of consistency we use the same initialization for W_l^a and W_l^b.

For arbitrarily large variances of network inputs and activations, our approximations are invalid; in that case, the initial values for W_l^a and W_l^b would have to be balanced for the input distribution to be retained. Alternatively, methods that explicitly control the variance in the network, e.g. batch normalization (Ioffe & Szegedy, 2015) or layer normalization (Ba et al., 2016) could be employed.

A.3. Dropout

Dropout retains activations in a neural network with a probability p and sets them to zero otherwise (Srivastava et al., 2014). It is common practice to scale the retained activations by $1/p$ during training so that the weights of the network do not have to be modified at test time when p is set to 1. In this case, dropout amounts to multiplying activations $\mathbf{x}$ by a Bernoulli random variable r where $\Pr[r = 1/p] = p$ and $\Pr[r = 0] = 1 - p$ (Srivastava et al., 2014). It holds that $E[r] = 1$ and $Var[r] = (1-p)/p$. If x is independent of r and $E[x] = 0$, the variance after dropout is

$$Var[xr] = E[r]^2 Var[x] + Var[r] Var[x] \tag{29}$$

$$= \left(1 + \frac{1-p}{p}\right) Var[x] \tag{30}$$

$$= \frac{1}{p} Var[x] \tag{31}$$

Assuming that a the *input* of a convolutional layer has been subject to dropout with a retain probability p, the variations of the forward and backward activations from §A.1 and §A.2 can now be approximated with

$$Var[x_{l+1}] \approx \frac{1}{4p} n_l Var[w_l] Var[x_l] \quad \text{and} \tag{32}$$

$$Var[\Delta x_l] \approx \frac{1}{4p} n_l Var[w_l^a] Var[\Delta x_{l+1}]. \tag{33}$$

This amounts to a modified initialization of W_l from a normal distribution with zero mean and a standard deviation of $\sqrt{4p/n}$. For layers without a succeeding GLU activation function, we initialize weights from $\mathcal{N}(0, \sqrt{p/n})$ to calibrate for any immediately preceding dropout application.

B. Upper Bound on Squared Sigmoid

The sigmoid function $\sigma(x)$ can be expressed as a hyperbolic tangent by using the identity $\tanh(x) = 2\,\sigma(2x) - 1$. The derivative of $\tanh$ is $\tanh'(x) = 1 - \tanh^2(x)$, and with $\tanh(x) \in [0, 1], x \geq 0$ it holds that

$$\tanh'(x) \leq 1, x \geq 0 \tag{34}$$

$$\int_0^x \tanh'(x)\,\mathrm{d}x \leq \int_0^x 1\,\mathrm{d}x \tag{35}$$

$$\tanh(x) \leq x, x \geq 0 \tag{36}$$

We can express this relation with $\sigma(x)$ as follows:

$$2\,\sigma(x) - 1 \leq \frac{1}{2} x, x \geq 0 \tag{37}$$

Both terms of this inequality have rotational symmetry w.r.t 0, and thus

$$\left(2\,\sigma(x) - 1\right)^2 \leq \left(\frac{1}{2} x\right)^2 \quad \forall x \tag{38}$$

$$\Leftrightarrow \sigma(x)^2 \leq \frac{1}{16} x^2 - \frac{1}{4} + \sigma(x). \tag{39}$$

C. Attention Visualization

Figure 3 shows attention scores for a generated sentence from the WMT'14 English to German task. The model that was used to generate these scores uses 8 decoder layers and a 80k BPE vocabulary. The attention passes in different

decoder layers capture different portions of the source sentence. Here, layer 1, 3 and 6 exhibit a linear alignment, with the first alignment appearing the clearest although it is slightly off and frequently attends to the correct source word for the previously generated target word. Layer 2 and 8 do not show a clear structure, presumably capturing information from the whole sentence. The fourth layer shows high alignment scores on nouns like "festival", "way" and "work" for both the generated target nouns as well as their preceding words. Note that in German, those preceding words depend on gender and object relationship of the respective noun. Finally, the attention scores in layer 5 and 7 focus on "built", which is subject to reordering in the German sentence and moved from the beginning to the very end of the sentence. One interpretation for this is that as generation progresses, the model repeatedly tries to perform the re-ordering. "aufgebaut" can be generated after a noun or pronoun only, which is reflected in the higher scores at position 2, 5, 8, 11 and 13.

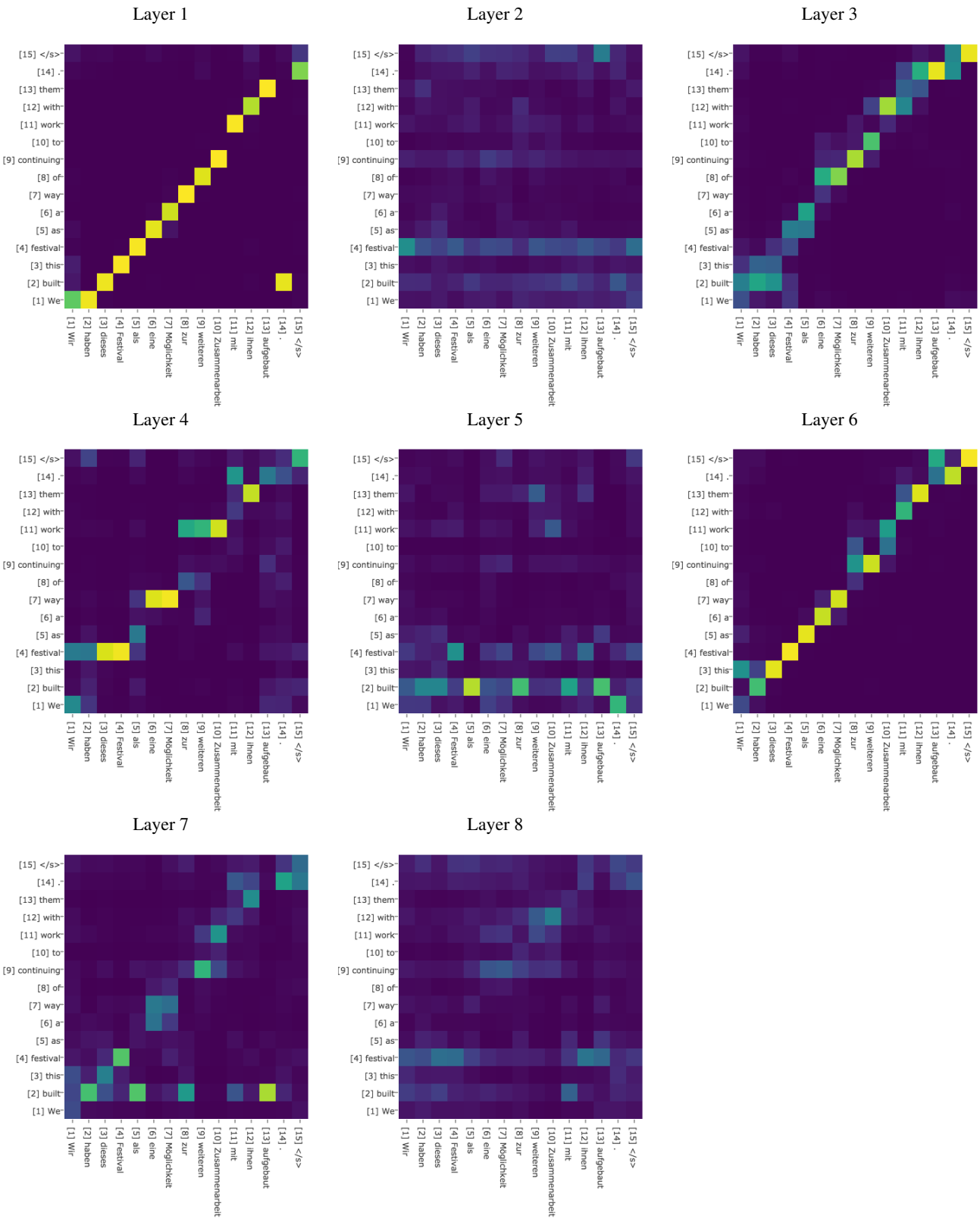

Figure 3. Attention scores for different decoder layers for an when translation from English (y-axis) to German (x-axis). This model uses 8 decoder layers and a 80k BPE vocabulary.