How Much Attention Do You Need?
A Granular Analysis of Neural Machine Translation Architectures

Tobias Domhan
Amazon
Berlin, Germany
domhant@amazon.com

Abstract

With recent advances in network architectures for Neural Machine Translation (NMT) recurrent models have effectively been replaced by either convolutional or self-attentional approaches, such as in the Transformer. While the main innovation of the Transformer architecture is its use of self-attentional layers, there are several other aspects, such as attention with multiple heads and the use of many attention layers, that distinguish the model from previous baselines. In this work we take a fine-grained look at the different architectures for NMT. We introduce an Architecture Definition Language (ADL) allowing for a flexible combination of common building blocks. Making use of this language, we show in experiments that one can bring recurrent and convolutional models very close to the Transformer performance by borrowing concepts from the Transformer architecture, but not using self-attention. Additionally, we find that self-attention is much more important for the encoder side than for the decoder side, where it can be replaced by a RNN or CNN without a loss in performance in most settings. Surprisingly, even a model without any target side self-attention performs well.

1 Introduction

Since the introduction of attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015) Neural Machine Translation (NMT) (Sutskever et al., 2014) has shown some impressive results. Initially, approaches to NMT mainly relied on Recurrent Neural Networks (RNNs) (Kalchbrenner and Blunsom, 2013; Bahdanau et al., 2014; Luong et al., 2015; Wu et al., 2016) such as Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) or the Gated Rectified Unit (GRU) (Cho et al., 2014).

Recently, other approaches relying on convolutional networks (Kalchbrenner et al., 2016; Gehring et al., 2017) and self-attention (Vaswani et al., 2017) have been introduced. These approaches remove the dependency between source language time steps, leading to considerable speed-ups in training time and improvements in quality. The Transformer, however, contains other differences besides self-attention, including layer normalization across the entire model, multiple source attention mechanisms, a multi-head dot attention mechanism, and the use of residual feed-forward layers. This raises the question of how much each of these components matters.

To answer this question we first introduce a flexible Architecture Definition Language (ADL) (§2). In this language we standardize existing components in a consistent way making it easier to compare structural differences of architectures. Additionally, it allows us to efficiently perform a granular analysis of architectures, where we can evaluate the impact of individual components, rather than comparing entire architectures as a whole. This ability leads us to the following observations:

- Source attention on lower encoder layers brings no additional benefit (§4.2).
- Multiple source attention layers and residual feed-forward layers are key (§4.3).
- Self-attention is more important for the source than for the target side (§4.4).

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1799–1808
Melbourne, Australia, July 15 - 20, 2018.

2 Flexible Neural Machine Translation Architecture Combination

In order to experiment easily with different architecture variations we define a domain specific NMT Architecture Definition Language (ADL), consisting of combinable and nestable building blocks.

2.1 Neural Machine Translation

NMT is formulated as a sequence to sequence prediction task in which a source sentence $X = x_1, ..., x_n$ is translated auto-regressively into a target sentence $Y = y_1, ..., y_m$ one token at a time as

$$p(y_t|Y_{1:t-1}, X; \boldsymbol{\theta}) = \text{softmax}(\mathbf{W}_o \mathbf{z}^L + \mathbf{b}_o), \quad (1)$$

where $\mathbf{b}_o$ is a bias vector, $\mathbf{W}_o$ projects a model dependent hidden vector $\mathbf{z}^L$ of the Lth decoder layer to the dimension of the target vocabulary $\mathbf{V}_{trg}$ and $\boldsymbol{\theta}$ denotes the model parameters. Typically, during training $Y_{1:t-1}$ consists of the reference sequence tokens, rather then the predictions produced by the model, which is known as teacher-forcing. Training is done by minimizing the cross-entropy loss between the predicted and the reference sequence.

2.2 Architecture Definition Language

In the following we specify the ADL which can be used to define any standard NMT architecture and combinations thereof.

Layers The basic building block of the ADL is a layer l. Layers can be nested, meaning that a layer can consist of several sub-layers. Layers optionally take set of named arguments $l(k_1{=}v_1, k_2{=}v_2, ...)$ with names k_1, k_2, ... and values v_1, v_2, ... or positional arguments $l(v_1, v_2, ...)$.

Layer definitions For each layer we have a corresponding layer definition based on the hidden states of the previous layer and any additional arguments. Specifically, each layer takes T hidden states $\mathbf{h}_1^i, ..., \mathbf{h}_T^i$, which in matrix form are $\mathbf{H}^i \in \mathbb{R}^{T \times d_i}$, and produces a new set of hidden states $\mathbf{h}_1^{i+1}, ..., \mathbf{h}_T^{i+1}$ or $\mathbf{H}^{i+1}$. While each layer can have a different number of hidden units d_i, in the following we assume them to stay constant across layers and refer to the model dimensionality as d_{model}. We distinguish the hidden states on the source side $\mathbf{U}^0, ..., \mathbf{U}^{L_s}$ from the hidden states of the target side $\mathbf{Z}^0, ..., \mathbf{Z}^L$. These are produced by the source and target embeddings and L_s source layers and L target layers.

Source attention layers play a special role in that their definition additionally makes use of any of the source hidden states $\mathbf{U}^0, ..., \mathbf{U}^{L_s}$.

Layer chaining Layers can be chained, feeding the output of one layer as the input to the next. We denote this as $l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_L$. This is equivalent to writing $l_L(... \, l_2(l_1(\mathbf{H}^0)))$ if none of the layers is a source attention layer.

In layer chains layers may also contain layers that themselves take arguments. As an example $l_1(k{=}v) \rightarrow l_2 \rightarrow ... \rightarrow l_L$ is equivalent to $l_L(... \, l_2(l_1(\mathbf{H}^0, k{=}v)))$. Note that unlike in the layer definition hidden states are not explicitly stated in the layer chain, but rather implicitly defined through the preceding layers.

Encoder/Decoder structure A NMT model is fully defined through two layer chains, namely one describing the encoder and another describing the decoder. The first layer hidden states on the source $\mathbf{U}^0$ are defined through the source embedding as

$$\mathbf{u}_t^0 = \mathbf{E}_{src} \mathbf{x}_t \quad (2)$$

where $\mathbf{x}_t \in \{0, 1\}^{|\mathbf{V}_{src}|}$ is the one-hot representation of x_t and $\mathbf{E}_S \mathbf{x}_t \in \mathbb{R}^{e \times |\mathbf{V}_{src}|}$ an embedding matrix with embedding dimensionality e. Similarly, $\mathbf{Z}^0$ is defined through the target embedding matrix $\mathbf{E}_{tgt}$.

Given the final decoder hidden state $\mathbf{Z}^L$ the next word predictions are done according to Equation 1.

Layer repetition Networks often consist of substructures that are repeated several times. In order to support this we define a repetition layer as

$$\textit{repeat}(n, l) = l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n,$$

where l represents a layer chain and each one of $l_1, ..., l_n$ an instantiation of that layer chain with a separate set of weights.

2.3 Layer Definitions

In this section we will introduce the concrete layers and their definitions, which are available for composing NMT architectures. They are based on building blocks common to many current NMT models.

Dropout A dropout (Srivastava et al., 2014) layer, denoted as $\textit{dropout}(\mathbf{h}_t)$, can be applied to hidden states as a form of regularization.

Fixed positional embeddings Fixed positional embeddings (Vaswani et al., 2017) add information about the position in the sequence to the hidden states. With $\mathbf{h}_t \in \mathbb{R}^d$ the positional embedding layer is defined as

$$\begin{aligned} \textit{pos}(\mathbf{h}_t) &= \textit{dropout}(\sqrt{d} \cdot \mathbf{h}_t + \mathbf{p}_t) \\ p_{t,j} &= \sin(t/10000^{2j/d}) \\ p_{t,2j+1} &= \cos(t/10000^{2j/d}). \end{aligned}$$

Linear We define a linear projection layer as

$$\textit{linear}(\mathbf{h}_t, d_o) = \mathbf{W}\mathbf{h}_t + \mathbf{b},$$

where $\mathbf{W} \in \mathbb{R}^{d_o \times d_{in}}$.

Feed-forward Making use of the linear projection layer a feed-forward layer with ReLU activation and dropout is defined as

$$\textit{ff}(\mathbf{h}_t, d_o) = \textit{dropout}(\max(0, \textit{linear}(\mathbf{h}_t, d_o)))$$

and a version which temporarily upscales the number of hidden units, as done by Vaswani et al. (2017), can be defined as

$$\textit{ffl}(\mathbf{h}_t) = \textit{ff}(4d_{in}) \rightarrow \textit{linear}(d_{in})$$

where $\mathbf{h}_t \in \mathbb{R}^{d_{in}}$.

Convolution Convolutions run a small feed-forward network on a sliding window over the input. Formally, on the encoder side this is defined as

$$\textit{cnn}(\mathbf{H}, v, k) = v(\mathbf{W}[\mathbf{h}_{i-\lfloor k/2 \rfloor}; ...; \mathbf{h}_{i+\lfloor k/2 \rfloor}] + \mathbf{b})$$

where k is the kernel size, and v is a non-linearity. The input is padded so that the number of hidden states does not change.

To preserve the auto-regressive property of the decoder we need to make sure to never take future decoder time steps into account, which can be achieved by adding $k-1$ padding vectors $\mathbf{h}_{-k+1} = \mathbf{0}, \ldots, \mathbf{h}_{-1} = \mathbf{0}$ such that the decoder convolution is given as

$$\textit{cnn}(\mathbf{H}, v, k) = v(\mathbf{W}[\mathbf{h}_{t-k+1}; ...; \mathbf{h}_t] + \mathbf{b}).$$

The non-linearity v can either be a ReLU or a Gated Linear Unit (GLU) (Dauphin et al., 2016). With the GLU we set $d_i = 2d$ such that we can split $\mathbf{h} = [\mathbf{h}_A; \mathbf{h}_B] \in \mathbb{R}^{2d}$ and compute the non-linearity as

$$\textit{glu}([\mathbf{h}_A; \mathbf{h}_B]) = \mathbf{h}_A \otimes \sigma(\mathbf{h}_B).$$

Identity We define an identity layer as

$$\textit{id}(\mathbf{h}_t) = \mathbf{h}_t.$$

Concatenation To concatenate the output of p layer chains we define

$$\textit{concat}(\mathbf{h}_t, l_1, ..., l_p) = [l_1(\mathbf{h}_t); ...; l_p(\mathbf{h_t})].$$

Recurrent Neural Network An RNN layer is defined as

$$\begin{aligned} \textit{rnn}(\mathbf{h}_t) &= f_{rnn\_o}(\mathbf{h}_t, \mathbf{s}_{t-1}) \\ \mathbf{s}_t &= f_{rnn\_h}(\mathbf{h}_t, \mathbf{s}_{t-1}) \end{aligned}$$

where f_{rnn_o} and f_{rnn_h} could be defined through either a GRU (Cho et al., 2014) or a LSTM (Hochreiter and Schmidhuber, 1997) cell. In addition, a bidirectional RNN layer *birnn* is available, which runs one *rnn* in forward and another in reverse direction and concatenates both results.

Attention All attention mechanisms take a set of query vectors $\mathbf{q}_0, ..., \mathbf{q}_M$, key vectors $\mathbf{k}_0, ..., \mathbf{k}_N$ and value vectors $\mathbf{v}_0, ..., \mathbf{v}_N$ in order to produce one context vector per query, which is a linear combination of the value vectors. We define $\mathbf{Q} \in \mathbb{R}^{M \times d}$, $\mathbf{V} \in \mathbb{R}^{N \times d}$ and $\mathbf{K} \in \mathbb{R}^{N \times d}$ as the concatenation of these vectors. What is used as the query, key and value vectors depends on attention type and is defined below.

Dot product attention The scaled dot product attention (Vaswani et al., 2017) is defined as

$$\textit{dot\_att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, s) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{s}}\right)\mathbf{V},$$

where the scaling factor s is implicitly set to d unless noted otherwise. Adding a projection to the queries, keys and values we get the projected dot attention as

$$\begin{aligned} &\textit{proj\_dot\_att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, d_p, s) = \\ &\textit{dot\_att}(\mathbf{Q}\mathbf{W}^Q, \mathbf{K}\mathbf{W}^K, \mathbf{V}\mathbf{W}^V, s) \end{aligned}$$

where d_p is dimensionality of the projected vectors such that $\mathbf{W}^Q \in \mathbb{R}^{d_q \times d_p}$, $\mathbf{W}^K \in \mathbb{R}^{d_k \times d_p}$ and $\mathbf{W}^V \in \mathbb{R}^{d_v \times d_p}$.

Vaswani et al. (2017) further introduces a multi-head attention, which applies multiple attentions at a reduced dimensionality. With h heads multi-head attention is computed as

$$\textit{mh\_dot\_att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, h, s) = [\mathbf{C}_0; ...; \mathbf{C}_h],$$

$$\mathbf{C}_i = \textit{proj\_dot\_att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, d/h, s).$$

Note that with $h = 1$ we recover the projected dot attention.

MLP attention The MLP attention (Bahdanau et al., 2014) computes the scores with a one-layer neural network as

$$\textit{mlp\_att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{S})\,\mathbf{V},$$
$$S_{ij} = \mathbf{w}_o^T \tanh(\mathbf{W}_q \mathbf{q}_i + \mathbf{W}_k \mathbf{k}_j).$$

Source attention Using the source hidden vectors $\mathbf{U}$, the source attentions are computed as

$$\textit{mh\_dot\_src\_att}(\mathbf{H}, \mathbf{U}, h, s) = \textit{mh\_dot\_att}(\mathbf{H}, \mathbf{U}, \mathbf{U}, h, s),$$
$$\textit{mlp\_src\_att}(\mathbf{H}, \mathbf{U}) = \textit{mlp\_att}(\mathbf{H}, \mathbf{U}, \mathbf{U}),$$
$$\textit{dot\_src\_att}(\mathbf{H}, \mathbf{U}, s) = \textit{mh\_dot\_att}(\mathbf{H}, \mathbf{U}, \mathbf{U}, 1, s).$$

Self-attention Self-attention (Vaswani et al., 2017) uses the hidden states as queries, keys and values such that

$$\textit{mh\_dot\_self\_att}(\mathbf{H}, s) = \textit{mh\_dot\_att}(\mathbf{H}, \mathbf{H}, \mathbf{H}, s).$$

Please note that on the target side one needs to make sure to preserve the auto-regressive property by only attending to hidden states at the current or past steps $\mathbf{h} < t$, which is achieved by masking the attention mechanism.

Layer normalization Layer normalization (Ba et al., 2016) uses the mean and standard deviation for normalization. It is computed as

$$\textit{norm}(\mathbf{h}_t) = \frac{\mathbf{g}}{\sigma_t} \otimes (\mathbf{h}_t - \mu_t) + \mathbf{b}$$
$$\mu_t = \frac{1}{d}\sum_{i=1}^{d} \mathbf{h}_{t,j} \quad \sigma_t = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\mathbf{h}_{t,j} - \mu_j)^2}$$

where $\mathbf{g}$ and $\mathbf{b}$ are learned scale and shift parameters with the same dimensionality as $\mathbf{h}$.

Residual layer A residual layer adds the output of an arbitrary layer chain l to the current hidden states. We define this as

$$\textit{res}(\mathbf{h_t}, l) = \mathbf{h_t} + l(\mathbf{h_t}).$$

For convenience we also define

$$\textit{res\_d}(\mathbf{h}_t, l) = \textit{res}(l(\mathbf{h}_t) \rightarrow \textit{dropout}) \quad \text{and}$$
$$\textit{res\_nd}(\mathbf{h}_t, l) = \textit{res}(\text{norm} \rightarrow l(\mathbf{h}_t) \rightarrow \textit{dropout}).$$

2.4 Standard Architectures

Having defined the common building blocks we now show how standard NMT architectures can be constructed.

RNMT As RNNs have been around the longest in NMT, several smaller architecture variations exist. Similar to Wu et al. (2016) in the following we use a bi-directional RNN followed by a stack of uni-directional RNNs with residual connections on the encoder side. Using the ADL an n layer encoder can be expressed as

$$\mathbf{U}^{L_s} = \textit{dropout} \rightarrow \textit{birnn} \rightarrow \textit{repeat}(n-1, \textit{res\_d}(\textit{rnn})).$$

For the decoder we use the architecture by Luong et al. (2015), which first runs a stacked RNN and then combines the context provided by a single attention mechanism with the hidden state provided by the RNN. This can be expressed by

$$\mathbf{Z}^L = \textit{dropout} \rightarrow \textit{repeat}(n, \textit{res\_d}(\textit{rnn})) \rightarrow \textit{concat}(\textit{id}, \textit{mlp\_att}) \rightarrow \textit{ff}.$$

If input feeding (Luong et al., 2015) is used the first layer hidden states are redefined as

$$\mathbf{z}_t^0 = [\mathbf{z}_{t-1}^L; \mathbf{E}_{tgt}\mathbf{y}_t].$$

Note that this inhibits any parallelism across decoder time steps. This is only an issue when using models other than RNNs, as RNNs already do not allow for parallelizing over decoder time steps.

ConvS2S Gehring et al. (2017) introduced a NMT model that fully relies on convolutions, both on the encoder and on the decoder side. The encoder is defined as

$$\mathbf{U}^{L_s} = \textit{pos} \rightarrow \textit{repeat}(n, \textit{res}(\textit{cnn}(\textit{glu}) \rightarrow \textit{dropout}))$$

and the decoder, which uses an unscaled single head dot attention is defined as

$$\mathbf{Z}^L = \textit{pos} \rightarrow \textit{res}(\textit{dropout} \rightarrow \textit{cnn}(\textit{glu}) \rightarrow \textit{dropout} \rightarrow \textit{res}(\textit{dot\_src\_att}(s{=}1))).$$

Note that unlike (Gehring et al., 2017) we do not project the query vectors before the attention and do not add the embeddings to the attention values.

Transformer The Transformer (Vaswani et al., 2017) makes use of self-attention, instead of RNNs or Convolutional Neural Networks (CNNs), as the basic computational block. Note that we use a slightly updated residual structure as implemented by *tensor2tensor*[1] than proposed originally. Specifically, layer normalization is applied to the input of the residual block instead of applying it between blocks. The Transformer uses a combination of self-attention and feed-forward layers on the encoder and additionally source attention layers on the decoder side. When defining the Transformer encoder block as

$$t_{\text{enc}} = \mathit{res\_nd}(\mathit{mh\_dot\_self\_att}) \rightarrow \mathit{res\_nd}(\mathit{fff}),$$

and the decoder block as

$$t_{\text{dec}} = \mathit{res\_nd}(\mathit{mh\_dot\_self\_att}) \rightarrow \mathit{res\_nd}(\mathit{mh\_dot\_src\_att}) \rightarrow \mathit{res\_nd}(\mathit{fff}).$$

the Transformer encoder is given as

$$\mathbf{U}^{L_s} = \mathit{pos} \rightarrow \mathit{repeat}(n, t_{\text{enc}}) \rightarrow \mathit{norm}$$

and the decoder as

$$\mathbf{Z}^{L} = \mathit{pos} \rightarrow \mathit{repeat}(n, t_{\text{dec}}) \rightarrow \mathit{norm}.$$

3 Related Work

The dot attention mechanism, now heavily used in the Transformer models, was introduced by (Luong et al., 2015) as part of an exploration of different attention mechanisms for RNN based NMT models.

Britz et al. (2017) performed an extensive exploration of hyperparameters of RNN based NMT models. The variations explored include different attention mechanisms, RNN cells types and model depth.

Similar to our work, Schrimpf et al. (2017) define a language for exploring architectures. In this case the architectures are defined for RNN cells and not for the higher level model architecture. Using the language they perform an automatic search of RNN cell architectures.

For the application of image classification there have been several recent successful efforts of automatically searching for successful architectures (Zoph and Le, 2016; Negrinho and Gordon, 2017; Liu et al., 2017).

[1] https://github.com/tensorflow/tensor2tensor

4 Experiments

What follows is an extensive empirical analysis of current NMT architectures and how certain sublayers as defined through our ADL affect performance.

4.1 Setup

All experiments were run with an adapted version of SOCKEYE (Hieber et al., 2017), which can parse arbitrary model definitions that are expressed in the language described in Section 2.3. The code and configuration are available at https://github.com/awslabs/sockeye/tree/acl18 allowing researchers to easily replicate the experiments and to quickly try new NMT architectures by either making use of existing building blocks in novel ways or adding new ones.

In order to get data points on corpora of different sizes we ran experiments on both WMT and IWSLT data sets. For WMT we ran the majority of our experiments on the most recent WMT'17 data consisting of roughly 5.9 million training sentences for English-German (EN→DE) and 4.5 million sentences for Latvian-English (LV→EN). We used newstest2016 as validation data and report metrics calculated on newstest2017. For the smaller IWSLT'16 English-German corpus, which consists of roughly 200 thousand training sentences, we used TED.tst2013 as validation data and report numbers for TED.tst2014.

For both WMT'17 and IWSLT'16 we preprocessed all data using the Moses[2] tokenizer and apply Byte Pair Encoding (BPE) (Sennrich et al., 2015) with 32,000 merge operations. Unless noted otherwise we run each experiment three times with different random seeds and report the mean and standard deviation of the BLEU and METEOR (Lavie and Denkowski, 2009) scores across runs. Evaluation scores are based on tokenized sequences and calculated with MultEval (Clark et al., 2011).

Model	WMT'14
Vaswani et al. (2017)	27.3
Our Transformer$_{\text{base}}$ impl.	27.5

Table 1: BLEU scores on WMT'14 EN→DE.

In order to compare to previous work, we also ran an additional experiment on WMT'14 using the same data as Vaswani et al. (2017)

[2] https://github.com/moses-smt/mosesdecoder/

as provided in preprocessed form through *tensor2tensor*.[3] This data set consists of WMT'16 training data, which has been tokenized and byte pair encoded with 32,000 merge operations. Evaluation is done on tokenized and compound split newstest2014 data using multi-bleu.perl in order to get scores comparable to Vaswani et al. (2017). As seen in Table 1, our Transformer implementation achieves a score equivalent to the originally reported numbers.

On the smaller IWSLT data we use $d_{model} = 512$ and on WMT $d_{model} = 256$ for all models. Models are trained with 6 encoder and 6 decoder blocks, where in the Transformer model a layer refers to a full encoder or decoder block. All convolutional layers use a kernel of size 3 and a ReLU activation, unless noted otherwise. RNNs use LSTM cells. For training we use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0002. The learning rate is decayed by a factor of 0.7, whenever the validation perplexity does not improve for 8 consecutive checkpoints, where a checkpoint is created every 4,000 updates on WMT and 1,000 updates on IWSLT. All models use label smoothing (Szegedy et al., 2016) with $\epsilon_{ls} = 0.1$.

4.2 What to attend to?

Source attention is typically based on the top encoder block. With multiple source attention layers one could hypothesize that it could be beneficial to allow attention encoder blocks other than the top encoder block. It might for example be beneficial for lower decoder blocks to use encoder blocks from the same level as they represent the same level of abstraction. Inversely, assuming that the translation is done in a coarse to fine manner it might help to first use the uppermost encoder block and use gradually lower level representations.

Encoder block	IWSLT	WMT'17
upper	25.4 ± 0.2	27.6 ± 0.0
increasing	25.4 ± 0.1	27.3 ± 0.1
decreasing	25.3 ± 0.2	27.1 ± 0.1

Table 2: BLEU scores when varying the encoder block used in the source attention mechanism of a Transformer on the EN→DE IWSLT and WMT'17 datasets.

[3] https://github.com/tensorflow/tensor2tensor/blob/765d33bb/tensor2tensor/data_generators/translate_ende.py

The result of modifying the source attention mechanism to use different encoder blocks is shown in Table 2. The variations include using the result of the encoder Transformer block at the same level as the decoder Transformer block (*increasing*) and using the upper encoder Transformer block in the first decoder block and then gradually using the lower blocks (*decreasing*).

We can see that attention on the upper encoder block performs best and no gains can be observed by attention on different encoder layers in the source attention mechanism.

4.3 Network Structure

The Transformer sets itself apart from both standard RNN models and convolutional model by more than just the multi-head self-attention blocks.

RNN to Transformer The differences to the RNN include the multiple source attention layers, multi-head attention, layer normalization and the residual upscaling feed-forward layers. Additionally, RNN models typically use single head MLP attention instead of the dot attention. This raises the question of what aspect contributes most to the performance of the Transformer.

Table 3 shows the result of taking an RNN and step by step changing the architecture to be similar to the Transformer architecture. We start with a standard RNN architecture with MLP attention similar to Luong et al. (2015) as described in Section 2.4 with and without input feeding denoted as RNMT.

Next, we take a model with a residual connection around the encoder bi-RNN such that the encoder is defined as

$$dropout \rightarrow res\_d(birnn) \rightarrow repeat(5, res\_d(rnn)).$$

The decoder uses a residual single head dot attention and no input feeding and is defined as

$$dropout \rightarrow repeat(6, res\_d(rnn)) \rightarrow res\_d(dot\_src\_att) \rightarrow res\_d(\mathit{fff}).$$

We denote this model as RNN in Table 3. This model is then changed to use multi-head attention (mh), positional embeddings (pos), layer normalization on the inputs of the residual blocks (norm), an attention mechanism in a residual block after every RNN layer with multiple (multi-att) and a single head (multi-add-1h), and finally a residual

Model	IWSLT EN→DE BLEU	WMT'17 EN→DE BLEU	WMT'17 EN→DE METEOR	WMT'17 LV→EN BLEU	WMT'17 LV→EN METEOR
Transformer	25.4 ± 0.1	27.6 ± 0.0	47.2 ± 0.1	18.5 ± 0.0	51.3 ± 0.1
RNMT	23.2 ± 0.2	25.5 ± 0.2	45.1 ± 0.1	-	-
- input feeding	23.1 ± 0.2	24.6 ± 0.1	43.8 ± 0.2	-	-
RNN	22.8 ± 0.2	23.8 ± 0.1	43.3 ± 0.1	15.2 ± 0.1	45.9 ± 0.1
+ mh	23.7 ± 0.4	24.4 ± 0.1	43.9 ± 0.1	16.0 ± 0.1	47.1 ± 0.1
+ pos	23.9 ± 0.2	24.1 ± 0.1	43.5 ± 0.2	-	-
+ norm	23.7 ± 0.1	24.0 ± 0.2	43.2 ± 0.1	15.2 ± 0.1	46.3 ± 0.2
+ multi-att-1h	24.5 ± 0.0	25.2 ± 0.1	44.9 ± 0.1	16.6 ± 0.2	49.1 ± 0.2
/ multi-att	24.4 ± 0.3	25.5 ± 0.0	45.3 ± 0.0	17.0 ± 0.2	49.4 ± 0.1
+ ff	25.1 ± 0.1	26.7 ± 0.1	46.4 ± 0.2	17.8 ± 0.1	50.5 ± 0.1

Table 3: Transforming an RNN into a Transformer style architecture. + shows the incrementally added variation. / denotes an alternative variation to which the subsequent + is relative to.

upscaling feed-forward layer is added after each attention block (ff). The final architecture of the encoder after applying these variations is

$$pos \rightarrow res\_nd(birnn) \rightarrow res\_nd(ffl) \rightarrow repeat(5, res\_nd(rnn) \rightarrow res\_nd(ffl)) \rightarrow norm$$

and of the decoder

$$pos \rightarrow repeat(6, res\_nd(rnn) \rightarrow res\_nd(mh\_dot\_src\_att) \rightarrow res\_nd(ffl)) \rightarrow norm.$$

Comparing this to the Transformer as defined in Section 2.4 we note that the model is identical to the Transformer, except that each self-attention has been replaced by an RNN or bi-RNN.

Table 3 shows that not using input feeding has a negative effect on the result, which however can be compensated by the explored model variations. With just a single attention mechanism the model benefits from multiple attention heads. The gains are even larger when an attention mechanism is added to every layer. With multiple source attention mechanisms the benefit of multiple heads decreases. Layer normalization on the inputs of the residual blocks has a small negative effect in all settings and metrics. As RNNs can learn to encode positional information positional embeddings are not strictly necessary. Indeed, we can observe no gains but rather even a small drop in BLEU and METEOR for WMT'17 EN→DE when using them. Adding feed-forward layers leads to large and consistent performance boost. While the final model, which is a Transformer model where each self-attention has been replaced by an RNN, is able to make up for a large amount of the difference between the baseline and the Transformer, it is still outperformed by the Transformer. The largest gains come from multiple attention mechanisms and residual feed-forward layers.

CNN to Transformer While the convolutional models have much more in common with the Transformer than the RNN based models, there are still some notable differences. Like the Transformer, convolutional models have no dependency between decoder time steps during training, use multiple source attention mechanisms and use a slightly different residual structure, as seen in Section 2.4. The Transformer uses a multi-head scaled dot attention while the ConvS2S model uses an unscaled single head dot attention. Other differences include the use of layer normalization as well as residual feed-forward blocks in the Transformer.

The result of making a CNN based architecture more and more similar to the Transformer can be seen in Table 4. As a baseline we use a simple residual CNN structure with a residual single head dot attention. This is denoted as CNN in Table 4. On the encoder side we have

$$pos \rightarrow repeat(6, res\_d(cnn))$$

and for the decoder

$$pos \rightarrow repeat(6, res\_d(cnn) \rightarrow res\_d(dot\_src\_att)).$$

This is similar to, but slightly simpler than, the ConvS2S model described in Section 2.4. In the experiments we explore both the GLU and ReLU as non-linearities for the CNN.

Adding layer normalization (norm), multi-head attention (mh) and upsampling residual feed-forward layers (ff) we arrive at a model that is

Model	IWSLT EN-DE BLEU	WMT'17 EN→DE BLEU	WMT'17 EN→DE METEOR	WMT'17 LV→EN BLEU	WMT'17 LV→EN METEOR
Transformer	25.4 ± 0.1	27.6 ± 0.0	47.2 ± 0.1	18.5 ± 0.0	51.3 ± 0.1
CNN GLU	24.3 ± 0.4	25.0 ± 0.3	44.4 ± 0.2	16.0 ± 0.5	47.4 ± 0.4
+ norm	24.1 ± 0.1	-	-	-	-
+ mh	24.2 ± 0.2	25.4 ± 0.1	44.8 ± 0.1	16.1 ± 0.1	47.6 ± 0.2
+ ff	25.3 ± 0.1	26.8 ± 0.1	46.0 ± 0.1	16.4 ± 0.2	47.9 ± 0.2
CNN ReLU	23.6 ± 0.3	23.9 ± 0.1	43.4 ± 0.1	15.4 ± 0.1	46.4 ± 0.3
+ norm	24.3 ± 0.1	24.3 ± 0.2	43.6 ± 0.1	16.0 ± 0.2	47.1 ± 0.5
+ mh	24.2 ± 0.2	24.9 ± 0.1	44.4 ± 0.1	16.1 ± 0.1	47.5 ± 0.2
+ ff	25.3 ± 0.3	26.9 ± 0.1	46.1 ± 0.0	16.4 ± 0.2	47.9 ± 0.1

Table 4: Transforming a CNN based model into a Transformer style architecture.

identical to a Transformer where the self-attention layers have been replaced by CNNs. This means that we have the following architecture on the encoder

$$pos \rightarrow repeat(6, res\_nd(cnn) \rightarrow res\_nd(\mathit{fffl})) \rightarrow norm.$$

Whereas for the decoder we have

$$pos \rightarrow repeat(6, res\_nd(cnn) \rightarrow res\_nd(mh\_dot\_src\_att) \rightarrow res\_nd(\mathit{fffl})) \rightarrow norm.$$

While in the baseline the GLU activation works better than the ReLU activation, when layer normalization, multi-head attention attention and residual feed-forward layers are added, the performance is similar. Except for IWSLT multi-head attention gives consistent gains over single head attention. The largest gains can however be observed by the addition of residual feed-forward layers. The performance of the final model, which is very similar to a Transformer where each self-attention has been replaced by a CNN, matches the performance of the Transformer on IWSLT EN→DE but is still 0.7 BLEU points worse on WMT'17 EN→DE and two BLEU points on WMT'17 LV→EN.

4.4 Self-attention variations

At the core of the Transformer are self-attentional layers, which take the role previously occupied by RNNs and CNNs. Self-attention has the advantage that any two positions are directly connected and that, similar to CNNs, there are no dependencies between consecutive time steps so that the computation can be fully parallelized across time. One disadvantage is that relative positional information is not directly represented and one needs to rely on the different heads to make up for this. In a CNN information is constrained to a local window which grows linearly with depth. Relative positions are therefore taken into account. While an RNN keeps an internal state, which can be used in future time steps, it is unclear how well this works for very long range dependencies (Koehn and Knowles, 2017; Bentivogli et al., 2016). Additionally, having a dependency on the previous hidden state inhibits any parallelization across time.

Given the different advantages and disadvantages we selectively replace self-attention on the encoder and decoder side in order to see where the model benefits most from self-attention.

We take the encoder and decoder block defined in Section 2.4 and try out different layers in place of the self-attention. Concretely, we have

$$t_{\text{enc}} = res\_nd(\mathbf{x}_{\text{enc}}) \rightarrow res\_nd(\mathit{fffl}),$$

on the encoder side and

$$t_{\text{dec}} = res\_nd(\mathbf{x}_{\text{dec}}) \rightarrow res\_nd(mh\_dot\_src\_att) \rightarrow res\_nd(\mathit{fffl}).$$

on the decoder side. Table 5 shows the result of replacing $\mathbf{x}_{\text{enc}}$ and $\mathbf{x}_{\text{dec}}$ with either self-attention, a CNN with ReLU activation or an RNN. Notice that with self-attention used in both $\mathbf{x}_{\text{enc}}$ and $\mathbf{x}_{\text{dec}}$ we recover the Transformer model. Additionally, we remove the residual block on the decoder side entirely (*none*). This results in a decoder block which only has information about the previous target word y_t through the word embedding that is fed as the input to the first layer. The decoder block is reduced to

$$t_{\text{dec}} = res\_nd(mh\_dot\_src\_att) \rightarrow res\_nd(\mathit{fffl}).$$

		IWSLT EN→DE	WMT'17 EN→DE		WMT'17 LV→EN	
Encoder	Decoder	BLEU	BLEU	METEOR	BLEU	METEOR
self-att	self-att	25.4 ± 0.2	27.6 ± 0.0	47.2 ± 0.1	18.3 ± 0.0	51.1 ± 0.1
self-att	RNN	25.1 ± 0.1	27.4 ± 0.1	47.0 ± 0.1	18.4 ± 0.2	51.1 ± 0.1
self-att	CNN	25.4 ± 0.4	27.6 ± 0.2	46.7 ± 0.1	18.0 ± 0.3	50.3 ± 0.3
RNN	self-att	25.8 ± 0.1	27.2 ± 0.1	46.7 ± 0.1	17.8 ± 0.1	50.6 ± 0.1
CNN	self-att	25.7 ± 0.1	26.6 ± 0.3	46.3 ± 0.1	16.8 ± 0.4	49.4 ± 0.4
RNN	RNN	25.1 ± 0.1	26.7 ± 0.1	46.4 ± 0.2	17.8 ± 0.1	50.5 ± 0.1
CNN	CNN	25.3 ± 0.3	26.9 ± 0.1	46.1 ± 0.0	16.4 ± 0.2	47.9 ± 0.2
self-att	*combined*	25.1 ± 0.2	27.6 ± 0.2	47.2 ± 0.2	18.3 ± 0.1	51.1 ± 0.1
self-att	*none*	23.7 ± 0.2	25.3 ± 0.2	43.1 ± 0.1	15.9 ± 0.1	45.1 ± 0.2

Table 5: Different variations of the encoder and decoder self-attention layer.

In addition to that, we try a combination where the first and fourth block use self-attention, the second and fifth an RNN, the third and sixth a CNN (*combined*).

Replacing the self-attention on both the encoder and the decoder side with an RNN or CNN results in a degradation of performance. In most settings, such as WMT'17 EN→DE for both variations and WMT'17 LV→EN for the RNN, the performance is comparable when replacing the decoder side self-attention. For the encoder however, except for IWSLT, we see a drop in performance of up to 1.5 BLEU points when not using self-attention. Therefore, self-attention seems to be more important on the encoder side than on the decoder side. Despite the disadvantage of having a limited context window, the CNN performs as well as self-attention on the decoder side on IWLT and WMT'17 EN→DE in terms of BLEU and only slightly worse in terms of METEOR. The combination of the three mechanisms (*combined*) on the decoder side performs almost identical to the full Transformer model, except for IWSLT where it is slightly worse.

It is surprising how well the model works without any self-attention as the decoder essentially looses any information about the history of generated words. Translations are entirely based on the previous word, provided through the target side word embedding, and the current position, provided through the positional embedding.

5 Conclusion

We described an ADL for specifying NMT architectures based on composable building blocks. Instead of committing to a single architecture, the language allows for combining architectures on a granular level. Using this language we explored how specific aspects of the Transformer architecture can successfully be applied to RNNs and CNNs. We performed an extensive evaluation on IWSLT EN→DE, WMT'17 EN→DE and LV→EN, reporting both BLEU and METEOR over multiple runs in each setting.

We found that RNN based models benefit from multiple source attention mechanisms and residual feed-forward blocks. CNN based models on the other hand can be improved through layer normalization and also feed-forward blocks. These variations bring the RNN and CNN based models close to the Transformer. Furthermore, we showed that one can successfully combine architectures. We found that self-attention is much more important on the encoder side than it is on the decoder side, where even a model without self-attention performed surprisingly well. For the data sets we evaluated on, models with self-attention on the encoder side and either an RNN or CNN on the decoder side performed competitively to the Transformer model in most cases.

We make our implementation available so that it can be used for exploring novel architecture variations.

References

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

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

Luisa Bentivogli, Arianna Bisazza, Mauro Cettolo, and Marcello Federico. 2016. Neural versus phrase-

based machine translation quality: a case study. *arXiv preprint arXiv:1608.04631*.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

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. *arXiv preprint arXiv:1406.1078*.

Jonathan H Clark, Chris Dyer, Alon Lavie, and Noah A Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 176–181. Association for Computational Linguistics.

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

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A Toolkit for Neural Machine Translation. *ArXiv preprint arXiv:1712.05690*.

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

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*.

Alon Lavie and Michael J Denkowski. 2009. The meteor metric for automatic evaluation of machine translation. *Machine translation*, 23(2-3):105–115.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

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

Renato Negrinho and Geoff Gordon. 2017. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*.

Martin Schrimpf, Stephen Merity, James Bradbury, and Richard Socher. 2017. A flexible approach to automated rnn architecture generation. *arXiv preprint arXiv:1712.07316*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

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*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.