

# Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction

Maha Elbayad<sup>1,2</sup>    Laurent Besacier<sup>1</sup>    Jakob Verbeek<sup>2</sup>

Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, LJK, F-38000 Grenoble France

<sup>1</sup> `firstname.lastname@univ-grenoble-alpes.fr`

<sup>2</sup> `firstname.lastname@inria.fr`

## Abstract

Current state-of-the-art machine translation systems are based on encoder-decoder architectures, that first encode the input sequence, and then generate an output sequence based on the input encoding. Both are interfaced with an attention mechanism that recombines a fixed encoding of the source tokens based on the decoder state. We propose an alternative approach which instead relies on a single 2D convolutional neural network across both sequences. Each layer of our network re-codes source tokens on the basis of the output sequence produced so far. Attention-like properties are therefore pervasive throughout the network. Our model yields results that are competitive with state-of-the-art encoder-decoder systems, while being conceptually simpler and having fewer parameters.

## 1 Introduction

Deep neural networks have made a profound impact on natural language processing technology in general, and machine translation in particular (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014; Jean et al., 2015; LeCun et al., 2015). Machine translation (MT) can be seen as a sequence-to-sequence prediction problem, where the source and target sequences are of different and variable length. Current state-of-the-art approaches are based on encoder-decoder architectures (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015). The encoder “reads” the variable-length source sequence and maps it into a vector representation. The decoder takes this vector as input and “writes” the target sequence, updating its state each step with the most recent word that it generated. The basic encoder-decoder model is generally equipped with an attention model (Bahdanau et al., 2015), which repetitively

re-accesses the source sequence during the decoding process. Given the current state of the decoder, a probability distribution over the elements in the source sequence is computed, which is then used to select or aggregate features of these elements into a single “context” vector that is used by the decoder. Rather than relying on the global representation of the source sequence, the attention mechanism allows the decoder to “look back” into the source sequence and focus on salient positions. Besides this inductive bias, the attention mechanism bypasses the problem of vanishing gradients that most recurrent architectures encounter.

However, the current attention mechanisms have limited modeling abilities and are generally a simple weighted sum of the source representations (Bahdanau et al., 2015; Luong et al., 2015), where the weights are the result of a shallow matching between source and target elements. The attention module re-combines the same source token codes and is unable to re-encode or re-interpret the source sequence while decoding.

To address these limitations, we propose an alternative neural MT architecture, based on deep 2D convolutional neural networks (CNNs). The product space of the positions in source and target sequences defines the 2D grid over which the network is defined. The convolutional filters are masked to prohibit accessing information derived from future tokens in the target sequence, obtaining an autoregressive model akin to generative models for images and audio waveforms (Oord et al., 2016a,b). See Figure 1 for an illustration.

This approach allows us to learn deep feature hierarchies based on a stack of 2D convolutional layers, and benefit from parallel computation during training. Every layer of our network computes features of the the source tokens, based on the target sequence produced so far, and uses these to predict the next output token. Our model therefore

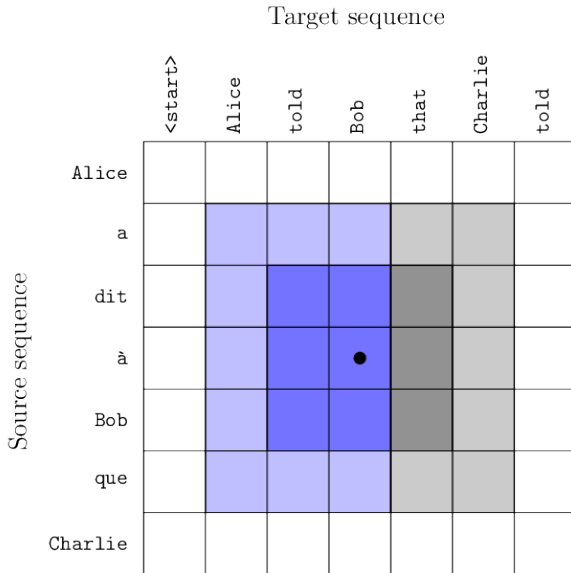


Figure 1: Convolutional layers in our model use masked  $3 \times 3$  filters so that features are only computed from previous output symbols. Illustration of the receptive fields after one (dark blue) and two layers (light blue), together with the masked part of the field of view of a normal  $3 \times 3$  filter (gray).

has attention-like capabilities by construction, that are pervasive throughout the layers of the network, rather than using an “add-on” attention model.

We validate our model with experiments on the IWSLT 2014 German-to-English (De-En) and English-to-German (En-De) tasks. We improve on state-of-the-art encoder-decoder models with attention, while being conceptually simpler and having fewer parameters.

In the next section we will discuss related work, before presenting our approach in detail in Section 3. We present our experimental evaluation results in Section 4, and conclude in Section 5.

## 2 Related work

The predominant neural architectures in machine translation are recurrent encoder-decoder networks (Graves, 2012; Sutskever et al., 2014; Cho et al., 2014). The encoder is a recurrent neural network (RNN) based on gated recurrent units (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) to map the input sequence into a vector representation. Often a bi-directional RNN (Schuster and Paliwal, 1997) is used, which consists of two RNNs that process the input in opposite directions, and the final states of both RNNs are concatenated as the input encoding. The decoder consists of a

second RNN, which takes the input encoding, and sequentially samples the output sequence one token at a time whilst updating its state.

While best known for their use in visual recognition models, (Oord et al., 2016a; Salimans et al., 2017; Reed et al., 2017; Oord et al., 2016c). Recent works also introduced convolutional networks to natural language processing. The first convolutional approaches to encoding variable-length sequences consist of stacking word vectors, applying 1D convolutions then aggregating with a max-pooling operator over time (Collobert and Weston, 2008; Kalchbrenner et al., 2014; Kim, 2014). For sequence generation, the works of Ranzato et al. (2016); Bahdanau et al. (2017); Gehring et al. (2017a) mix a convolutional encoder with an RNN decoder. The first entirely convolutional encoder-decoder models were introduced by Kalchbrenner et al. (2016b), but they did not improve over state-of-the-art recurrent architectures. Gehring et al. (2017b) outperformed deep LSTMs for machine translation 1D CNNs with gated linear units (Meng et al., 2015; Oord et al., 2016c; Dauphin et al., 2017) in both the encoder and decoder modules.

Such CNN-based models differ from their RNN-based counterparts in that temporal connections are placed between layers of the network, rather than within layers. See Figure 2 for a conceptual illustration. This apparently small difference in connectivity has two important consequences. First, it makes the field of view grow linearly across layers in the convolutional network, while it is unbounded within layers in the recurrent network. Second, while the activations in the RNN can only be computed in a sequential manner, they can be computed in parallel across the temporal dimension in the convolutional case.

In all the recurrent or convolutional models mentioned above, each of the input and output sequences are processed separately as a one-dimensional sequence by the encoder and decoder respectively. Attention mechanisms (Bahdanau et al., 2015; Luong et al., 2015; Xu et al., 2015) were introduced as an interface between the encoder and decoder modules. During encoding, the attention model finds which hidden states from the source code are the most salient for generating the next target token. This is achieved by evaluating a “context vector” which, in its most basic form, is a weighted average of the source features. The

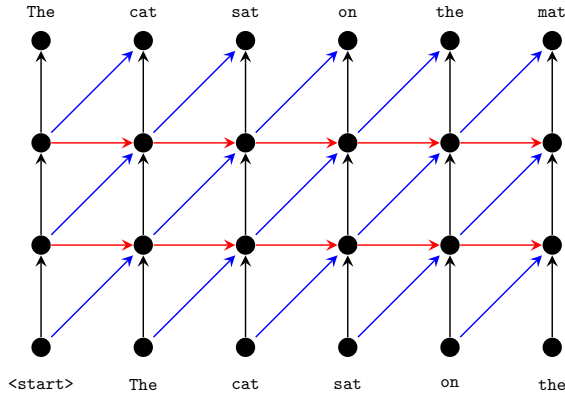


Figure 2: Illustration of decoder network topology with two hidden layers, nodes at bottom and top represent input and output respectively. Horizontal connections are used for RNNs, diagonal connections for convolutional networks. Vertical connections are used in both cases. Parameters are shared across time-steps (horizontally), but not across layers (vertically).

weights of the summation are predicted by a small neural network that scores these features conditioning on the current decoder state.

Vaswani et al. (2017) propose an architecture relying entirely on attention. Positional input coding together with self-attention (Parikh et al., 2016; Lin et al., 2017) replaces recurrent and convolutional layers. Huang et al. (2018) use an attention-like gating mechanism to alleviate an assumption of monotonic alignment in the phrase-based translation model of Wang et al. (2017). Deng et al. (2018) treat the sentence alignment as a latent variable which they infer using a variational inference network during training to optimize a variational lower-bound on the log-likelihood.

### Beyond uni-dimensional encoding/decoding.

The idea of building a 2D grid from parallel sequences (as in Figure 1) is used in different NLP tasks especially for scoring parallel texts. This includes works on semantic matching, paraphrase identification and machine translation. ARC-II of Hu et al. (2014) has 1D convolutions applied to each sequence separately before a series of 2D convolutions and max-poolings are followed by an MLP to estimate the matching score. They interestingly highlighted the desirable property of letting the sequences ‘meet’ before their representations mature. He and Lin (2016); Wan et al. (2016) first encode the sequences with Bi-LSTMs then evaluate pairwise similarities between the words

of the two sequences to build an interaction grid. While He and Lin (2016) process the grid with a two-dimensional CNN, Wan et al. (2016) directly use k-max pooling to aggregate and then score the pair. Similarly, for sequence alignment, Levy and Wolf (2017) use LSTM hidden states as tokens representations and, similar to our work, concatenate pairwise representations and feed their input grid to a 2D convolutional network followed by a soft-max to estimate soft-alignment probabilities. Recently in question-answering, Raison et al. (2018) weaved two Bi-LSTMs, one along the context dimension and the other along the question dimension in order to identify a response span in the context.

More related to our work on machine translation, Kalchbrenner et al. (2016a) proposed the ‘reencoder’ network where a Grid LSTM processes both sequences along its first and second dimension, allowing the model to re-encode the source sequence as it advances along the target dimension. They also observed that such a structure implements an implicit form of attention. Wu et al. (2017) used a CNN over the 2D source-target representation, but only as a discriminator in an adversarial training setup. Similar to semantic matching models, they do not use masked convolutions, since their CNN is used to predict if a given source-target pair is a human or machine translation. Concurrently with our work, Bahar et al. (2018) used a 2DLSTM layer to jointly process the source and target sequences with a similar two-dimensional layout.

### 3 Translation by 2D Convolution

In this section we present our 2D CNN translation model in detail.

**Input source-target tensor.** Given the source and target pair  $(s, t)$  of lengths  $|s|$  and  $|t|$  respectively, we first embed the tokens in  $d_s$  and  $d_t$  dimensional spaces via look-up tables. The word embeddings  $\{x_1, \dots, x_{|s|}\}$  and  $\{y_1, \dots, y_{|t|}\}$  are then concatenated to form a 3D tensor  $X \in \mathbb{R}^{|t| \times |s| \times f_0}$ , with  $f_0 = d_t + d_s$ , where

$$X_{ij} = [y_i \ x_j]. \quad (1)$$

This joint unigram encoding is the input to our convolutional network.

**Convolutional layers.** We use the DenseNet (Huang et al., 2017) convolutional

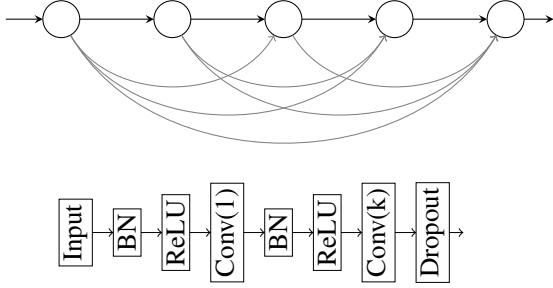


Figure 3: Architecture of the DenseNet at block level (top), and within each block (bottom).

architecture, which is the state of the art for image classification tasks. Layers are densely connected, meaning that each layer takes as input the activations of all the preceding layers, rather than just the last one, to produce its  $g$  feature maps. The parameter  $g$  is called the “growth rate” as it is the number of appended channels to the network’s output at each layer. The long-distance connections in the network improve gradient flow to early network layers during training, which is beneficial for deeper networks.

Each layer first batch-normalizes (Ioffe and Szegedy, 2015) its input and apply a ReLU (Nair and Hinton, 2010) non-linearity. To reduce the computation cost, each layer first computes  $4g$  channels using a  $1 \times 1$  convolution from the  $f_0 + (l - 1)g$  input channels to layer  $l \in \{1, \dots, L\}$ . This is followed by a second batch-normalization and ReLU non-linearity. The second convolution has  $(k \times \lceil \frac{k}{2} \rceil)$  kernels, *i.e.* masked as illustrated in Figure 1, and generates the  $g$  output features maps to which we apply dropout (Srivastava et al., 2014). The architecture of the densely connected network is illustrated in Figure 3.

We optionally use gated linear units (Dauphin et al., 2017) in both convolutions, these double the number of output channels, and we use half of them to gate the other half.

**Target sequence prediction.** Starting from the initial  $f_0$  feature maps, each layer  $l \in \{1, \dots, L\}$  of our DenseNet produces a tensor  $H^l$  of size  $|t| \times |s| \times f_l$ , where  $f_l$  is the number of output channels of that layer. To compute a distribution over the tokens in the output vocabulary, we need to collapse the second dimension of the tensor, which is given by the variable length of the input sequence, to retrieve a unique encoding for each target position.

The simplest aggregation approach is to apply max-pooling over the input sequence to obtain a tensor  $H^{\text{pool}} \in \mathbb{R}^{|t| \times f_L}$ , *i.e.*

$$H_{id}^{\text{pool}} = \max_{j \in \{1, \dots, |s|\}} H_{ijd}^L. \quad (2)$$

Alternatively, we can use average-pooling over the input sequence:

$$H_{id}^{\text{pool}} = \frac{1}{\sqrt{|s|}} \sum_{j \in \{1, \dots, |s|\}} H_{ijd}^L. \quad (3)$$

The scaling with the inverse square-root of the source length acts as a variance stabilization term, which we find to be more effective in practice than a simple averaging.

The pooled features are then transformed to predictions over the output vocabulary  $\mathcal{V}$ , by linearly mapping them with a matrix  $E \in \mathbb{R}^{|\mathcal{V}| \times f_L}$  to the vocabulary dimension  $|\mathcal{V}|$ , and then applying a soft-max. Thus the probability distribution over  $\mathcal{V}$  for the  $i$ -th output token is obtained as

$$p_i = \text{SoftMax}(EH_i^{\text{pool}}). \quad (4)$$

Alternatively, we can use  $E$  to project to dimension  $d_t$ , and then multiply with the target word embedding matrix used to define the input tensor. This reduces the number of parameters and generally improves the performance.

**Implicit sentence alignment.** For a given output token position  $i$ , the max-pooling operator of Eq. (2) partitions the  $f_L$  channels by assigning them across the source tokens  $j$ . Let us define

$$B_{ij} = \{d \in \{1, \dots, f_L\} \mid j = \arg \max(H_{ijd}^L)\}$$

as the channels assigned to source token  $j$  for output token  $i$ . The energy that enters into the soft-max to predict token  $w \in \mathcal{V}$  for the  $i$ -th output position is given by

$$e_{iw} = \sum_{d \in \{1, \dots, f_L\}} E_{wd} H_{id}^{\text{pool}} \quad (5)$$

$$= \sum_{j \in \{1, \dots, |s|\}} \sum_{d \in B_{ij}} E_{wd} H_{ijd}^L. \quad (6)$$

The total contribution of the  $j$ -th input token is thus given by

$$\alpha_{ij} = \sum_{d \in B_{ij}} E_{wd} H_{ijd}^L, \quad (7)$$

where we dropped the dependence on  $w$  for simplicity. As we will show experimentally in the next section, visualizing the values  $\alpha_{ij}$  for the ground-truth output tokens, we can recover an implicit sentence alignment used by the model.



**Self attention.** Besides pooling we can collapse the source dimension of the feature tensor with an attention mechanism. This mechanism will generate a tensor  $H^{\text{att}}$  that can be used instead of, or concatenated with,  $H^{\text{Pool}}$ .

We use the self-attention approach of Lin et al. (2017), which for output token  $i$  computes the attention vector  $\rho_i \in \mathbb{R}^{|s|}$  from the activations  $H_i^L$ :

$$\rho_i = \text{SoftMax}(H_i^L w + b \mathbb{1}_{|s|}), \quad (8)$$

$$H_i^{\text{att}} = \sqrt{|s|} \rho_i^\top H_i^L, \quad (9)$$

where  $w \in \mathbb{R}^{f_L}$  and  $b \in \mathbb{R}$  are parameters of the attention mechanism. Scaling of attention vectors with the square-root of the source length was also used by Gehring et al. (2017b), and we found it effective here as well as in the average-pooling case.

## 4 Experimental evaluation

In this section, we present our experimental setup, followed by quantitative results, qualitative examples of implicit sentence alignments from our model, and a comparison to the state of the art.

### 4.1 Experimental setup

**Data and pre-processing.** We experiment with the IWSLT 2014 bilingual dataset (Cettolo et al., 2014), which contains transcripts of TED talks aligned at sentence level, and translate between German (De) and English (En) in both directions. Following the setup of (Edunov et al., 2018), sentences longer than 175 words and pairs with length ratio exceeding 1.5 were removed from the original data. There are 160+7K training sentence pairs, 7K of which are separated and used for validation/development. We report results on a test set of 6,578 pairs obtained by concatenating TED.dev2010, TEDX.dev2012 and TED.tst2010-2012. We tokenized and lower-cased all data using the standard scripts from the Moses toolkit (Koehn et al., 2007).

For open-vocabulary translation, we segment sequences using byte pair encoding (Sennrich et al., 2016) with 14K merge operations following two approaches. The first (V1), similar to Edunov et al. (2018); Deng et al. (2018), is a joint encoding i.e. applied to the concatenation of source and target texts. This results in a German and English vocabularies of around 12K and 8.8K types respectively. The second approach (V2) encodes each language independently resulting in a German and

English vocabularies of 13.3K and 13.8K respectively.

**Implementation details.** Unless stated otherwise, we use DenseNets with masked convolutional filters of size  $5 \times 3$ , as given by the light blue area in Figure 1. To train our models for the ablation study, we use maximum likelihood estimation (MLE) with Adam ( $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-8}$ ) starting with a learning rate of  $5e^{-4}$  that we scale by a factor of 0.8 if no improvement is noticed on the validation loss after three evaluations; we evaluate every 8K updates. For faster training and due to the increased computational requirements, from  $\mathcal{O}(|x| + |y|)$  of encoder-decoder models to  $\mathcal{O}(|x| \cdot |y|)$ , we only read sequences up to 80 positions. We also downsample the initial grid channels by half to reduce the number of input channels to every dense block, thus requiring less memory. After training all models for 40 epochs, the best performing model on the validation set is used to decode with a beam-search of width 5. We measure translation quality using the BLEU metric (Papineni et al., 2002).

**Baselines.** For comparison with state-of-the-art architectures, we implemented a bidirectional LSTM encoder-decoder model with dot-product attention (Bahdanau et al., 2015; Luong et al., 2015) using PyTorch (Paszke et al., 2017), and used Facebook AI Research Sequence-to-Sequence Toolkit (Gehring et al., 2017b) to train the ConvS2S and Transformer (Vaswani et al., 2017) models on our data.

For the Bi-LSTM encoder-decoder, the encoder is a single layer bidirectional LSTM with input embeddings of size 128 and a hidden state of size 256 (128 in each direction). The decoder is a single layer LSTM with similar input size and a hidden size of 256, the target input embeddings are also used in the pre-softmax projection. For regularization, we apply a dropout of rate 0.2 to the inputs of both encoder and decoder and to the output of the decoder prior to softmax. As in (Bahdanau et al., 2015), we refer to this model as RNNsearch.

The ConvS2S model we trained has embeddings of dimension 256, a 16-layers encoder and 12-layers decoder. Each convolution uses  $3 \times 1$  filters and is followed by a gated linear unit with a total of  $2 \times 256$  channels. Residual connections link the input of a convolutional block to its output. We first trained the default architecture

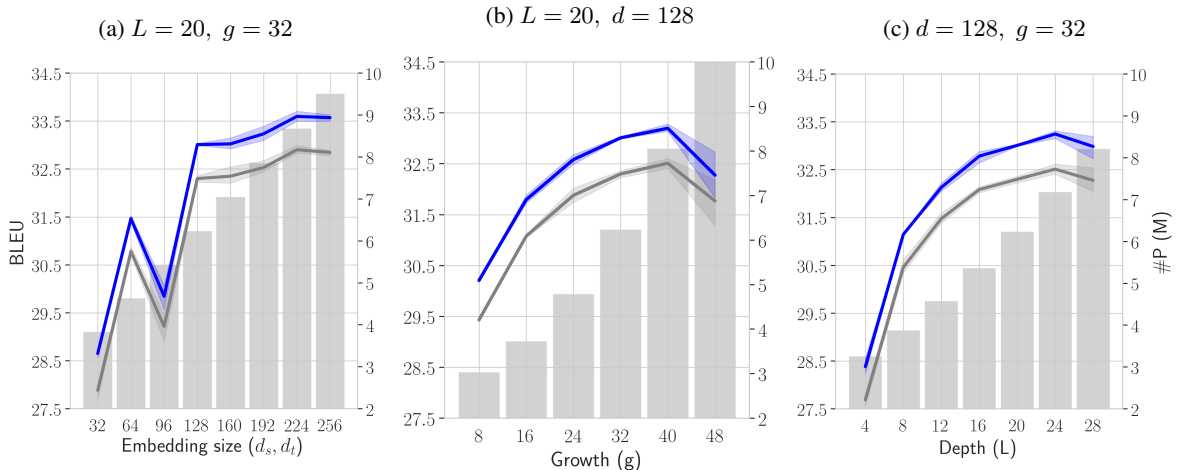


Figure 4: The impact of token embedding size, number of layers ( $L$ ), and growth rate ( $g$ ) on the validation set BLEU scores. In blue the results with beam search (width=5) and in gray with greedy decoding. The bars show the total number of parameters (in millions) for each setup.

for this dataset as suggested in FairSeq (Gehring et al., 2017b), which has only 4 layers in the encoder and 3 in the decoder, but achieved better results with the deeper version described above. The model is trained with label-smoothed cross-entropy ( $\epsilon = 0.1$ ) using Nesterov accelerated gradient with a momentum of 0.99 and an initial learning rate of 0.25 decaying by a factor of 0.1 every epoch. ConvS2S is also regularized with a dropout rate of 0.2.

For the transformer model, we use token embeddings of dimension 512, and the encoder and decoder have 6 layers and 4 attention heads. For the inner layer in the per-position feed-forward network we use  $d_{ff} = 1024$ . We optimize the label-smoothed ( $\epsilon = 0.1$ ) cross-entropy loss with Adam ( $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 1e^{-8}$ ) (Kingma and Ba, 2015). The learning rate starts from  $1e^{-7}$  and is increased during 4,000 warm-up steps. Afterwards, the learning rate is set to  $5e^{-4}$  and follows an inverse-square-root schedule (Vaswani et al., 2017). For the transformer we set the dropout to 0.3.

## 4.2 Experimental results

**Architecture evaluation.** In this section we explore the impact of several parameters of our model: the token embedding dimension, depth, growth rate and filter sizes. We also evaluate different aggregation mechanisms across the source dimension: max-pooling, average-pooling, and attention.

In each chosen setting, we train five models

Model	BLEU	Flops $\times 10^5$	#params
Average	$30.89 \pm 0.18$	3.63	7.18M
Max	$33.25 \pm 0.1$	3.44	7.18M
Attn	$31.55 \pm 0.11$	3.61	7.24M
Max, gated	$32.99 \pm 0.17$	3.49	9.64M
[Max, Attn]	$33.29 \pm 0.14$	3.51	7.24M

Table 1: BLEU scores of our model ( $L = 24, g = 32, d_s = d_t = 128$ ) on the validation set with different pooling operators and using gated convolutional units.

with different initializations and report the mean and standard deviation of the validation set BLEU scores. We also state the number of parameters of each model and the computational cost of training, estimated in a similar way as Vaswani et al. (2017), based on the wall clock time of training and the GPU single precision specs.

In Table 1 we see that using max-pooling instead average-pooling across the source dimension increases the performance with around 2.3 BLEU points. Scaling the average representation with  $\sqrt{|s|}$  Eq. (3) helped improving the performance but it is still largely outperformed by the max-pooling. Adding gated linear units on top of each convolutional layer does not improve the BLEU scores, but increases the variance due to the additional parameters. Stand-alone self-attention *i.e.* weighted average-pooling is slightly better than uniform average-pooling but it is still outperformed by max-pooling. Concatenating the max-pooled features (Eq. (2)) with the represen-

$k$	$L$	BLEU	Flops $\times 10^5$	#params
3	16	32.40 $\pm$ 0.08	2.47	4.32M
3	20	32.57 $\pm$ 0.23	3.03	4.92M
5	8	31.14 $\pm$ 0.04	0.63	3.88M
5	12	32.13 $\pm$ 0.11	2.61	4.59M
5	16	32.78 $\pm$ 0.16	3.55	5.37M
5	20	33.01 $\pm$ 0.03	3.01	6.23M
5	24	33.25 $\pm$ 0.1	3.44	7.18M
5	28	32.99 $\pm$ 0.3	5.35	8.21M
7	12	31.81 $\pm$ 0.2	2.76	5.76M
7	16	32.43 $\pm$ 0.36	3.13	6.94M

Table 2: Performance of our model ( $g = 32$ ,  $d_s = d_t = 128$ ) for different filter sizes  $k$  and depths  $L$  and filter sizes  $k$  on the validation set.

tation obtained with self-attention (Eq. (9)) leads to a small increase in performance, from 33.25 to 33.29. In the remainder of our experiments we only use max-pooling for simplicity, unless stated otherwise.

In Figure 4 we consider the effect of the token embedding size, the growth rate of the network, and its depth. The token embedding size together with the growth rate  $g$  control the dimension of the final feature used for estimating the emission probability. We generally use the same embedding dimension for both languages i.e.  $d = d_t = d_s$ , thus the final representation is of size  $f_L = 2d + gL$ . In Figure 4 we see that a minimal dimension is required, in this case  $d = 128$ , in order for the model to be complex enough and capture the training data statistics. For embedding sizes between 128 and 256, the BLEU score slowly increases from 33 to 33.6

The depth of the network is of similar impact. Training deeper networks (from 4 to 24 layers) increases the BLEU score by about 5 points. An argument similar to the one about the growth rate can be made in this case too for networks with more than 24 layers.

The receptive field of our model is controlled by its depth and the filter size. In Table 2, we note that narrower receptive fields are better than larger ones with less layers at equivalent complexities e.g. comparing  $(k = 3, L = 20)$  to  $(k = 5, L = 12)$ , and  $(k = 5, L = 16)$  with  $(k = 7, L = 12)$ .

**Comparison to the state of the art.** We compare our results to the state of the art in Table 3 for both directions German-English (De-En) and English-German (En-De).

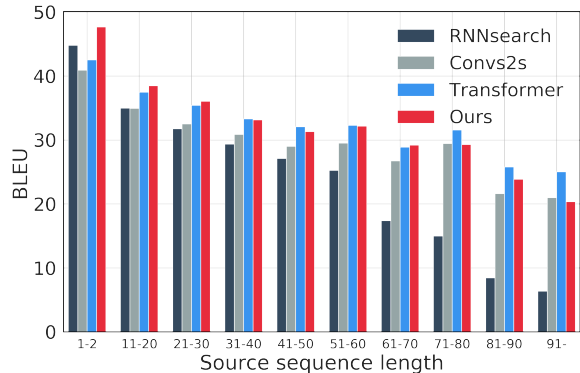


Figure 5: BLEU scores across sentence lengths.

In this section, the parameters of our models are trained using label-smoothed cross-entropy ( $\epsilon = 0.1$ ) similarly to the *ConvS2S* and *Transformer* baselines. To successfully train our models with large embeddings ( $d = 512$ ) we increase the dropout ( $p = 0.4$ ) and normalize the initial 2D grid. For decoding we use a beam-search of width 5 enhanced with length and coverage penalties (Wu et al., 2016).

Our model has about the same number of parameters as RNNsearch (with V1 vocabularies), yet improves performance by 3.88 BLEU points. It is also better than the recent work of Deng et al. (2018) on recurrent architectures with variational attention.

Our model outperforms its 1D convolutional counterpart Gehring et al. (2017b) in both translation directions and is competitive with transformer (0.3 points behind) while having about 2 to 4 times fewer parameters.

**Performance across sequence lengths.** In Figure 5 we consider translation quality as a function of sentence length, and compare our model to RNNsearch, ConvS2S and Transformer. Our model gives the best results across all sentence lengths, except for the longest ones where ConvS2S and Transformer are better. Overall, our model combines the strong performance of RNNsearch on short sentences with good performance of ConvS2S and Transformer on longer ones.

**Implicit sentence alignments.** Following the method described in Section 3, we illustrate in Figure 6 the implicit sentence alignments the max-pooling operator produces in our model. For reference we also show the alignment produced by our model using self-attention. We see that with

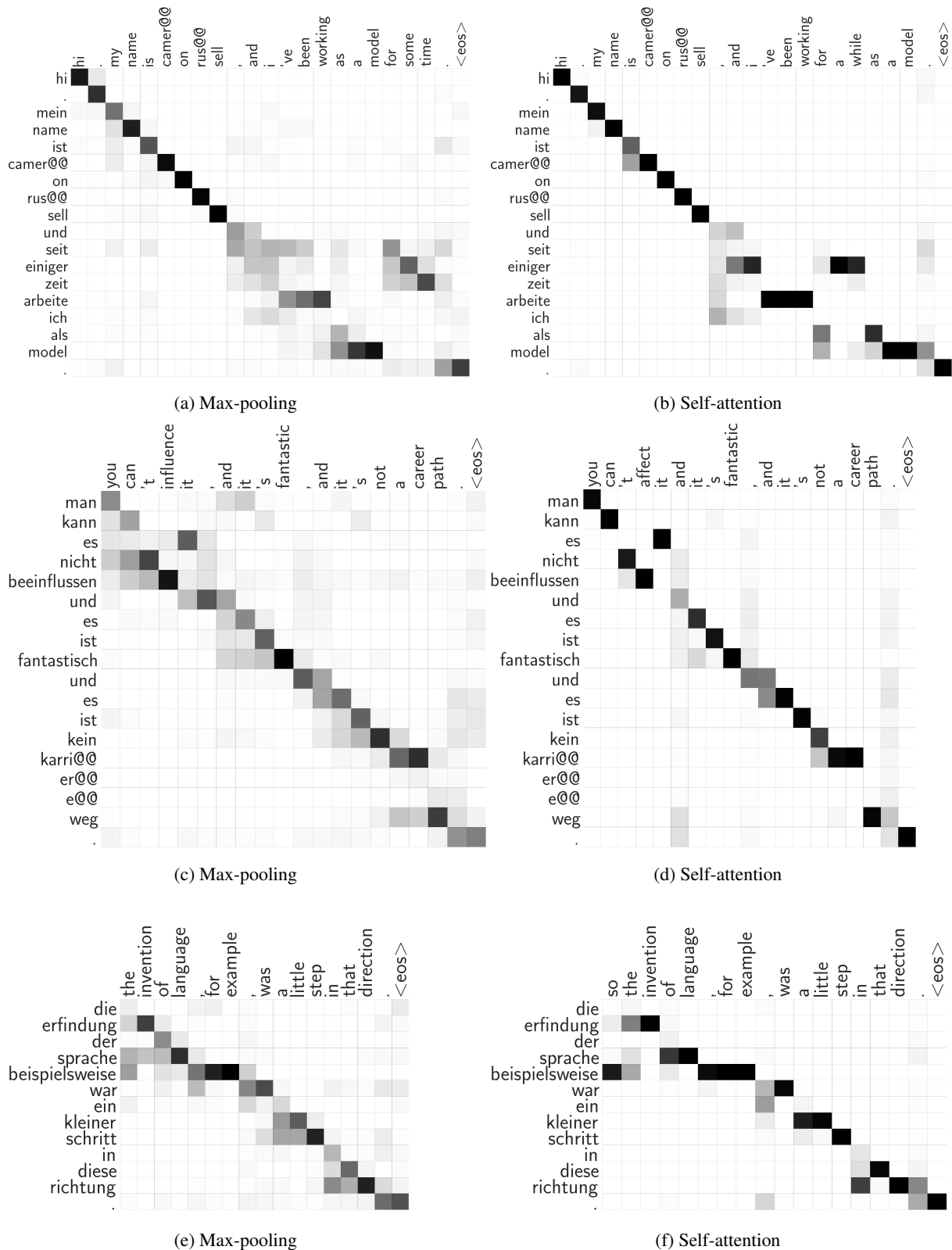


Figure 6: Implicit BPE token-level alignments produced by our Pervasive Attention model. For the max-pooling aggregation we visualize  $\alpha$  obtained with Eq. (7) and for self-attention the weights  $\rho$  of Eq. (8).



<b>Word-based</b>	De-En	Flops ( $\times 10^5$ )	# prms	En-De	# prms
Conv-LSTM (MLE) (Bahdanau et al., 2017)	27.56				
Bi-GRU (MLE+SLE) (Bahdanau et al., 2017)	28.53				
Conv-LSTM (deep+pos) (Gehring et al., 2017a)	30.4			25.36	
NPMT + language model (Huang et al., 2018)	30.08				
<b>BPE-based</b>					
ConvS2S (MLE+SLE) (Edunov et al., 2018)	32.84				
Variational attention (Deng et al., 2018)	33.10				
RNNsearch* (Bahdanau et al., 2015), V1	29.98	1.79	13M	25.04	15M
ConvS2S** (MLE) (Gehring et al., 2017b), V1	32.31	1.35	21M	26.73	22M
Transformer** (Vaswani et al., 2017), V1	34.42		46M	<b>28.23</b>	48M
Transformer** (Vaswani et al., 2017), V2	<b>34.44</b>		52M	28.07	52M
Pervasive Attention (this paper), V1	33.86		<b>11M</b>	27.21	<b>11M</b>
Pervasive Attention (this paper), V2	34.18		22M	27.99	22M

Table 3: Comparison to state-of-the art results on IWSLT German-English translation. (\*): results obtained using our implementation. (\*\*): results obtained using FairSeq (Gehring et al., 2017b).

both max-pooling and attention qualitatively similar implicit sentence alignments emerge.

Notice in the first example how the max-pool model, when writing *I've been working*, looks at *arbeite* but also at *seit* which indicates the past tense of the former. Also notice some cases of non-monotonic alignment. In the first example *for some time* occurs at the end of the English sentence, but *seit einiger zeit* appears earlier in the German source. For the second example there is non-monotonic alignment around the negation at the start of the sentence. The first example illustrates the ability of the model to translate proper names by breaking them down into BPE units. In the second example the German word *Karriereweg* is broken into the four BPE units *karri,er,e,weg*. The first and the fourth are mainly used to produce the English *a career*, while for the subsequent *path* the model looks at *weg*.

Finally, we can observe an interesting pattern in the alignment map for several phrases across the three examples. A rough lower triangular pattern is observed for the English phrases *for some time*, *and it's fantastic*, *and it's not*, *a little step*, and *in that direction*. In all these cases the phrase seems to be decoded as a unit, where features are first taken across the entire corresponding source phrase, and progressively from the part of the source phrase that remains to be decoded.

## 5 Conclusion

We presented a novel neural machine translation architecture that departs from the encoder-decoder paradigm. Our model jointly encodes the source and target sequence into a deep feature hierarchy in which the source tokens are embedded in the context of a partial target sequence. Max-pooling over this joint-encoding along the source dimension is used to map the features to a prediction for the next target token. The model is implemented as 2D CNN based on DenseNet, with masked convolutions to ensure a proper autoregressive factorization of the conditional probabilities.

Since each layer of our model re-encodes the input tokens in the context of the target sequence generated so far, the model has attention-like properties in every layer of the network by construction. Adding an explicit self-attention module therefore has a very limited, but positive, effect. Nevertheless, the max-pooling operator in our model generates implicit sentence alignments that are qualitatively similar to the ones generated by attention mechanisms. We evaluate our model on the IWSLT'14 dataset, translation German to English and vice-versa. We obtain excellent BLEU scores that compare favorably with the state of the art, while using a conceptually simpler model with fewer parameters.

We hope that our alternative joint source-target encoding sparks interest in other alternatives to the encoder-decoder model. In the future, we plan to explore hybrid approaches in which the input to

our joint encoding model is not provided by token-embedding vectors, but the output of 1D source and target embedding networks, *e.g.* (bi-)LSTM or 1D convolutional. We also want to explore how our model can be used to translate across multiple language pairs.

Our PyTorch-based implementation is available at <https://github.com/elbayadm/attn2d>.

**Acknowledgment.** This work has been partially supported by the grant ANR-16-CE23-0006 “Deep in France” and ANR-11-LABX-0025-01 “LabEx PERSYVAL”.

## References

- Parnia Bahar, Christopher Brix, and Hermann Ney. 2018. Towards two-dimensional sequence to sequence model in neural machine translation. In *EMNLP*.
- D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. 2017. An actor-critic algorithm for sequence prediction. In *ICLR*.
- D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.
- M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *IWSLT*.
- K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.
- R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*.
- Y. Dauphin, A. Fan, M. Auli, and D. Grangier. 2017. Language modeling with gated convolutional networks. In *ICML*.
- Y. Deng, Y. Kim, J. Chiu, D. Guo, and A. Rush. 2018. Latent alignment and variational attention. *arXiv preprint arXiv:1807.03756*.
- S. Edunov, M. Ott, M. Auli, D. Grangier, and M. Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *NAACL*.
- J. Gehring, M. Auli, D. Grangier, and Y. Dauphin. 2017a. A convolutional encoder model for neural machine translation. In *ACL*.
- J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. Dauphin. 2017b. Convolutional sequence to sequence learning. In *ICML*.
- A. Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.
- Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *NAACL*, pages 937–948.
- S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, pages 2042–2050.
- G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger. 2017. Densely connected convolutional networks. In *CVPR*.
- P. Huang, C. Wang, S. Huang, D. Zhou, and L. Deng. 2018. Towards neural phrase-based machine translation. In *ICLR*.
- S. Ioffe and C. Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*.
- S. Jean, K. Cho, R. Memisevic, and Y. Bengio. 2015. On using very large target vocabulary for neural machine translation. In *ACL*.
- N. Kalchbrenner and P. Blunsom. 2013. Recurrent continuous translation models. In *ACL*.
- N. Kalchbrenner, I. Danihelka, and A. Graves. 2016a. Grid long short-term memory. In *ICLR*.
- N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu. 2016b. Neural machine translation in linear time. *arXiv, arXiv:1610.10099*.
- N. Kalchbrenner, E. Grefenstette, and P. Blunsom. 2014. A convolutional neural network for modelling sentences. In *ACL*.
- Y. Kim. 2014. Convolutional neural networks for sentence classification. In *ACL*.
- D. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
- P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*.
- Y. LeCun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature*, 52:436–444.

- D. Levy and L. Wolf. 2017. Learning to align the source code to the compiled object code. In *ICML*.
- Z. Lin, M. Feng, C. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. 2017. A structured self-attentive sentence embedding. In *ICLR*.
- T. Luong, H. Pham, and C. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.
- F. Meng, Z. Lu, M. Wang, H. Li, W. Jiang, and Q. Liu. 2015. Encoding source language with convolutional neural network for machine translation. In *ACL*.
- V. Nair and G. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *ICML*.
- A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. 2016a. Wavenet: a generative model for raw audio. In *ISCA Speech Synthesis Workshop*.
- A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. 2016b. Pixel recurrent neural networks. In *ICML*.
- A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. 2016c. Conditional image generation with PixelCNN decoders. In *NIPS*.
- K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *ACL*.
- A. Parikh, O. Täckström, D. Das, and J. Uszkoreit. 2016. A decomposable attention model for natural language inference. In *EMNLP*.
- A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. 2017. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*.
- Martin Raison, Pierre-Emmanuel Mazaré, Rajarshi Das, and Antoine Bordes. 2018. Weaver: Deep co-encoding of questions and documents for machine reading. *arXiv preprint arXiv:1804.10490*.
- M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. 2016. Sequence level training with recurrent neural networks. In *ICLR*.
- S. Reed, A. van den Oord, N. Kalchbrenner, S. Gómez Colmenarejo, Z. Wang, D. Belov, and N. de Freitas. 2017. Parallel multiscale autoregressive density estimation. In *ICML*.
- T. Salimans, A. Karpathy, X. Chen, and D. Kingma. 2017. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. In *ICLR*.
- M. Schuster and K. Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing*, 45(11):2673–2681.
- R. Sennrich, B. Haddow, and A. Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.
- I. Sutskever, O. Vinyals, and Q. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. In *NIPS*.
- Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. 2016. A deep architecture for semantic matching with multiple positional sentence representations. In *AAAI*, volume 16, pages 2835–2841.
- C. Wang, Y. Wang, P.-S. Huang, A. Mohamed, D. Zhou, and L. Deng. 2017. Sequence modeling via segmentations. In *ICML*.
- L. Wu, Y. Xia, L. Zhao, F. Tian, T. Qin, J. Lai, and T.-Y. Liu. 2017. Adversarial neural machine translation. *arXiv*, arXiv:1704.06933.
- 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*.
- K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.