Chapter 7

Alternate Architectures

Most of neural network research has focused on the use of recurrent neural networks with attention. But this is by no means the only architecture for neural networks. Arguable, a disadvantage of using recurrent neural networks on the input side is that it requires a long sequential process that consumes each input word in one step. This also prohibits the ability to parallelize the processing of all words at once, thus limiting the use of the capabilities of GPUs.

There have been a few alternate suggestions for the architecture of neural machine translation models. We will briefly present some of them in this section. It remains to be seen, if they are a curiosity or conquer the field.

7.1 Convolutional Neural Networks

The first end-to-end neural machine translation model of the modern era (Kalchbrenner and Blunsom, 2013) was actually not based on recurrent neural networks, but based on **convolutional neural networks**. These had been shown to be very successful in image processing, thus looking for other applications was a natural next step.

See Figure 7.1 for an illustration of a convolutional network that encodes an input sentence. The basic building block of these networks is a convolution. It merges the representation of

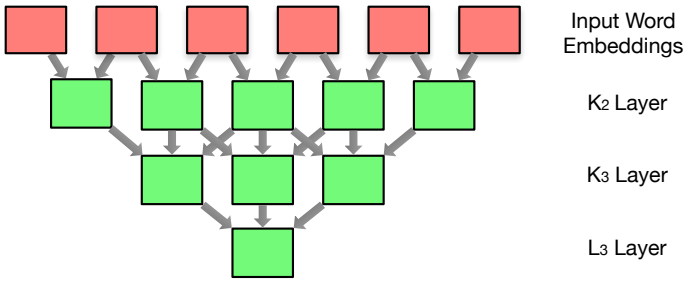

Figure 7.1: Encoding a sentence with a convolutional neural network. By always using two convolutional layers, the size of the convolutions differ (here K_2 and K_3). Decoding reverses this process.

i input words into a single representation by using a matrix K_i. Applying the convolution to every sequence of input words reduces the length of the sentence representation by $i - 1$. Repeating this process leads to a sentence representation in a single vector.

The illustration shows an architecture with two convolutional K_i layers, followed by a final L_i layer that merges the sequence of phrasal representations into a single sentence representation. The size of the convolutional kernels K_i and L_i depends on the length of the sentences. The example shows a 6-word sentence and a sequence of K_2, K_3, and L_3 layers. For longer sentences, bigger kernels are needed.

The hierarchical process of building up a sentence representation bottom-up is well grounded in linguistic insight in the recursive nature of language. It is similar to chart parsing, except that we are not committing to a single hierarchical structure. On the other hand, we are asking an awful lot from the resulting sentence embedding to represents the meaning of an entire sentence of arbitrary length.

Generating the output sentence translation reverses the bottom-up process. One problem for the decoder is to decide the length of the output sentence. One option to address this problem is to add a model that predicts output length from input length. This then leads to the selection of the size of the reverse convolution matrices.

See Figure 7.2 for an illustration of a variation of this idea. The shown architecture always uses a K_2 and a K_3 convolutional layer, resulting in a sequence of phrasal representations, not a single sentence embedding. There is an explicit mapping step from phrasal representations of input words to phrasal representations of output words, called transfer layer.

The decoder of the model includes a recurrent neural network on the output side. Sneaking in a recurrent neural network here does undermine a bit the argument about better parallelization. However, the claim still holds true for encoding the input, and a sequential language model is just a too powerful tool to disregard.

While the just-described convolutional neural machine translation model helped to set the scene for neural network approaches for machine translation, it could not be demonstrated to achieve competitive results compared to traditional approaches. The compression of the sentence representation into a single vector is especially a problem for long sentences. However, the model was used successfully in reranking candidate translations generated by traditional statistical machine translation systems.

7.2 Convolutional Neural Networks With Attention

Gehring et al. (2017) propose an architecture for neural networks that combines the ideas of convolutional neural networks and the attention mechanism. It is essentially the sequence-to-sequence attention that we described as the canonical neural machine translation approach, but with the recurrent neural networks replaced by convolutional layers.

We introduced convolutions in the previous section. The idea is to combine a short sequence of neighboring words into a single representation. To look at it in another way, a convolution encodes a word with its left and right context, in a limited window. Let us now describe in more detail what this means for the encoder and the decoder in the neural model.

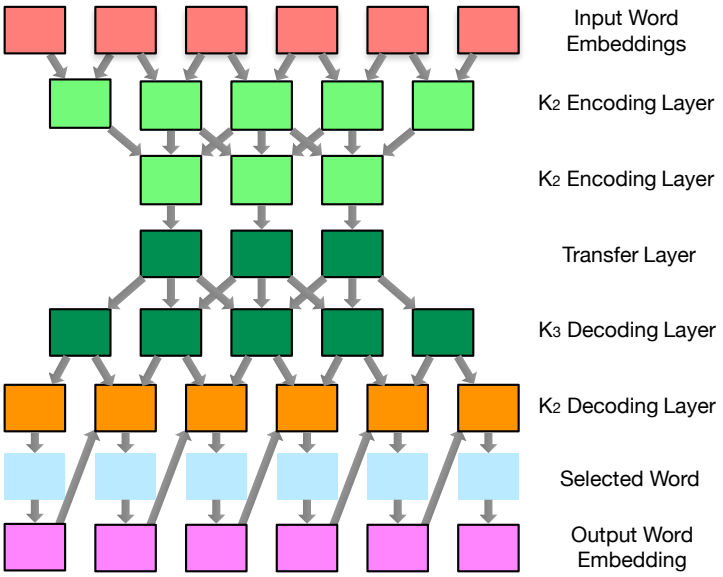

Figure 7.2: Refinement of the convolutional neural network model. Convolutions do not result in a single sentence embedding but a sequence. The encoder is also informed by a recurrent neural network (connections from output word embeddings to final decoding layer.

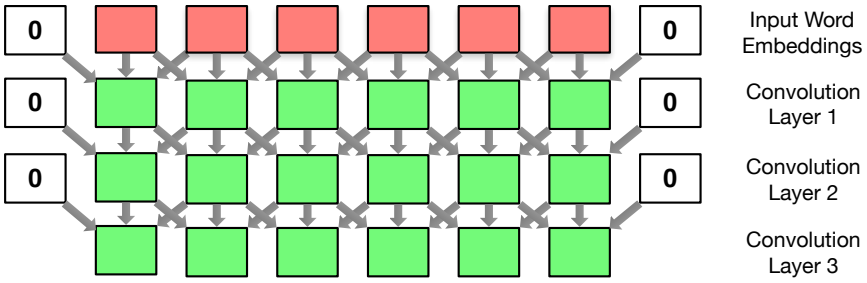

Figure 7.3: Encoder using stacked convolutional layers. Any number of layers may be used.

7.2.1 Encoder

See Figure 7.3 for an illustration of the convolutional layers used in the encoder. For each input word, the state at each layer is informed by the corresponding state in the previous layer and its two neighbors. Note that these convolutional layers do not shorten the sequence, because we have a convolution centered around each word, using padding (vectors with zero values) for word positions that are out of bounds.

Mathematically, we start with the input word embeddings Ex_j and progress through a sequence of layer encodings $h_{d,j}$ at different depth d until a maximum depth D.

$$\begin{aligned} h_{0,j} &= E\ x_j \\ h_{d,j} &= f(h_{d-1,j-k}, ..., h_{d-1,j+k}) \quad \text{for } d > 0, d \leq D \end{aligned} \tag{7.1}$$

The function f is a feed-forward layer, with a residual connection from the corresponding previous layer state $h_{d-1,j}$.

Note that even with a few convolutional layers, the final representation of a word $h_{D,j}$ may only be informed by partial sentence context — in contrast to the bi-directional recurrent neural networks in the canonical model. However, relevant context words in the input sentence that help with disambiguation may be outside this window.

On the other hand, there are significant computational advantages to this idea. All words at one depth can be processed in parallel, even combined into one massive tensor operation that can be efficiently parallelized on a GPU.

7.2.2 Decoder

The decoder in the canonical model also has at its core a recurrent neural network. Recall its state progression defined in Equation 5.3 on page 55:

$$s_i = f(s_{i-1},\ Ey_{i-1}, c_i) \tag{7.2}$$

where s_i is the encoder state, Ey_{i-1} the embedding of the previous output word, and c_i the input context.

The convolutional version of this does not have recurrent decoder states, i.e., the computation does not depend on the previous state s_{i-1}, but is conditioned on the sequence of the κ most recent previous words.

$$s_i = f(Ey_{i-\kappa}, ..., Ey_{i-1}, c_i) \tag{7.3}$$

Furthermore, these decoder convolutions may be stacked, just as the encoder convolutional layers.

$$\begin{aligned} s_{1,i} &= f(Ey_{i-\kappa}, ..., Ey_{i-1}, c_i) \\ s_{d,i} &= f(s_{d-1,i-\kappa-1}, ..., s_{d-1,i}, c_i) \quad \text{for } d > 0, d \leq \hat{D} \end{aligned} \tag{7.4}$$

See Figure 7.4 for an illustration of these equations. The main difference between the canonical neural machine translation model and this architecture is the conditioning of the states of the decoder. They are computed in a sequence of convolutional layers, and also always the input context.

7.2.3 Attention

The attention mechanism is essentially unchanged from the canonical neural translation model. Recall that is is based on an association $a(s_{i-1}, h_j)$ between the word representations computed by the encoder h_j and the previous state of the decoder s_{i-1} (refer back to Equation 5.6 on page 57).

Since we still have such encoder and decoder states ($h_{D,j}$ and $s_{\hat{D},i-1}$), we use the same here. These association scores are normalized and used to compute a weighted sum of the input word embeddings (i.e., the encoder states $h_{D,j}$). A refinement is that the encoder state

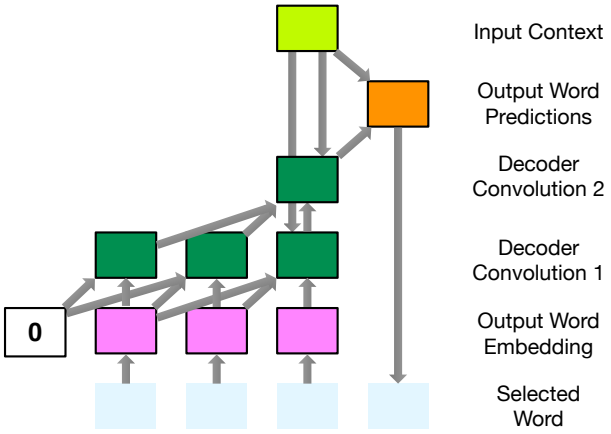

Figure 7.4: Decoder in convolutional neural network with attention. The decoder state is computed as a sequence of convolutional layers (here: 2) over the already predicted output words. Each convolutional state is also informed by the input context computed from the input sentence and attention.

$h_{D,j}$ and the input word embedding x_j is combined via addition when computing the context vector. This is the usual trick of using residual connections to assist training with deep neural networks.

7.3 Self-Attention

The critique of the use of recurrent neural networks is that they require a lengthy walk-through, word by word, of the entire input sentence, which is time-consuming and limits parallelization. The previous sections replaced the recurrent neural networks in our canonical model with convolutions. However, these have a limited context window to enrich representations of words. What we would like is some architectural component that allows us to use wide context and can be highly parallelized. What could that be?

In fact, we already encountered it: the attention mechanism. It considers associations between every input word and any output word, and uses it to build a vector representation of the entire input sequence. The idea behind **self-attention** is to extend this idea to the encoder. Instead of computing the association between an input and an output word, self-attention computes the association between any input word and any other input word. One way to view it is that this mechanism refines the representation of each input word by enriching it with context words that help to disambiguate it.

7.3.1 Computing Self-Attention

Vaswani et al. (2017) define self attention for a sequence of vectors h_j (of size $|h|$), packed into a matrix H, as

$$\text{self-attention}(H) = \text{softmax}\Big(\frac{HH^T}{\sqrt{|h|}}\Big)H \tag{7.5}$$

Let us look at this equation in detail. The association between every word representation h_j any other context word h_k is done via the dot product between the packed matrix H and its transpose H^T, resulting in a vector of *raw association* values HH^T. The values in this vector are first scaled by the size of the word representation vectors $|h|$, and then by the softmax, so that their values add up to 1. The resulting vector of *normalized association* values is then used to weigh the context words.

Another way to put Equation 7.5 without the matrix H notation but using word representation vectors h_j:

$$\begin{aligned} a_{jk} &= \frac{1}{|h|} h_j h_k^T && \text{raw association } \left(\frac{HH^T}{\sqrt{|h|}}\right) \\ \alpha_{jk} &= \frac{\exp(a_{jk})}{\sum_\kappa \exp(a_{j\kappa})} && \text{normalized association (softmax)} \\ \text{self-attention}(h_j) &= \sum_k \alpha_{j\kappa} h_k && \text{weighted sum} \end{aligned} \tag{7.6}$$

7.3.2 Self-Attention Layer

The self-attention step described above is only one step in the self-attention layer used to encode the input sentence. There are four more steps that follow it.

- We combine self-attention with residual connections that pass the word representation through directly

$$\text{self-attention}(h_j) + h_j \tag{7.7}$$

- Next up is a layer normalization step (described in Section 2.6.6 on page 25).

$$\hat{h}_j = \text{layer-normalization}(\text{self-attention}(h_j) + h_j) \tag{7.8}$$

- A standard feed-forward step with ReLU activation function is applied.

$$\text{relu}(W\hat{h}_j + b) \tag{7.9}$$

- This is also augmented with residual connections and layer normalization.

$$\text{layer-normalization}(\text{relu}(W\hat{h}_j + b) + \hat{h}_j) \tag{7.10}$$

Taking a page from deep models, we now stack several such layers (say, $D = 6$) on top of each other.

$$\begin{aligned} h_{0,j} &= Ex_j && \text{start with input word embedding} \\ h_{d,j} &= \text{self-attention-layer}(h_{d-1,j}) && \text{for } d > 0, d \leq D \end{aligned} \tag{7.11}$$

The deep modeling is the reason behind the residual connections in the self-attention layer — such residual connections help with training since they allow a shortcut to the input which may be utilized in early stages of training, before it can take advantage of the more complex interdependencies that deep models enable. The layer normalization step is one standard training trick that also helps especially with deep models.

7.3.3 Attention in the Decoder

Self-attention is also used in the decoder, now between output words. The decoder also has more traditional attention. In total there are 3 sub layers.

- *Self attention:* Output words are initially encoded by word embeddings $s_i = Ey_i$. We perform exactly the same self-attention computation as described in Equation 7.5. However, the association of a word s_i is limited to words s_k with $k \leq i$, i.e., just the previously produced output words. Let us denote the result of this sub layer for output word i as $\tilde{s}_i$

- *Attention:* The attention mechanism in this model follows very closely self-attention. The only difference is that, previously, we compute self attention between the hidden states H and themselves. Now, we compute attention between the decoder states $\tilde{S}$ and the final encoder states H.

$$\text{attention}(\tilde{S}, H) = \text{softmax}\Big(\frac{\tilde{S}H^T}{\sqrt{|h|}}\Big)H \tag{7.12}$$

  Using the same more detailed exposition as above for self-attention:

$$\begin{aligned} a_{ik} &= \frac{1}{|h|}\tilde{s}_i h_k^T && \text{raw association } \Big(\frac{\tilde{S}H^T}{\sqrt{|h|}}\Big) \\ \alpha_{ik} &= \frac{\exp(a_{ik})}{\sum_\kappa \exp(a_{i\kappa})} && \text{normalized association (softmax)} \\ \text{attention}(\tilde{s}_i) &= \sum_k \alpha_{j\kappa} h_k && \text{weighted sum} \end{aligned} \tag{7.13}$$

  This attention computation is augmented by adding in residual connections, layer normalization, and an additional ReLU layer, just like the self-attention layer described above.

  It is worth noting that, the output of the attention computation is a weighted sum over input word representations $\sum_k \alpha_{j\kappa} h_k$. To this, we add the (self-attended) representation of the decoder state $\tilde{s}_i$ via a residual connection. This allows skipping over the deep layers, thus speeding up training.

- *Feed-forward layer:* This sub layer is identical to the encoder, i.e., $\text{relu}(W_s\hat{s}_i + b_s)$

Each of the sub-layers is followed by the add-and-norm step of first using residual connections and then layer normalization (as noted in the description of the attention sub layer).

The entire model is shown in Figure 7.5,

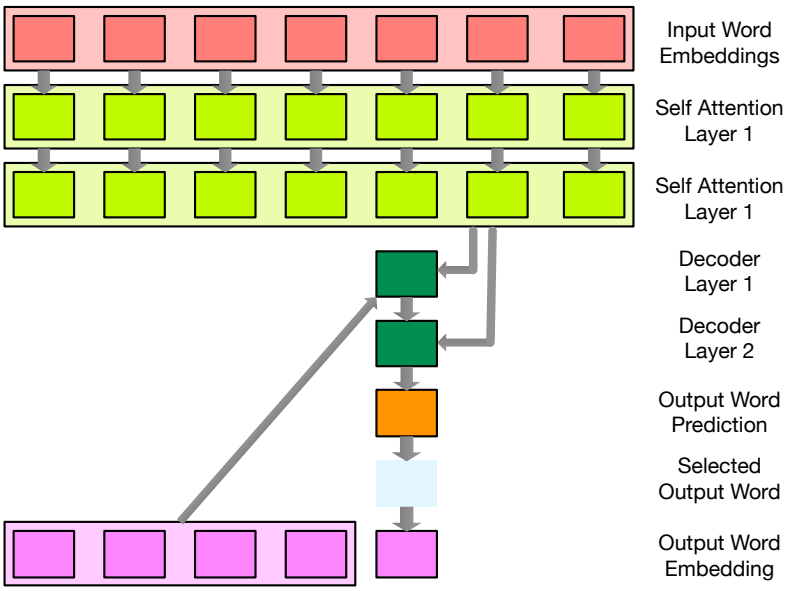

Figure 7.5: Attention-based machine translation model: the input is encoded with several layers of self-attention. The decoder computes attention-based representations of the input in several layers, initialized with the previous word embeddings.

Further Readings Kalchbrenner and Blunsom (2013) build a comprehensive machine translation model by first encoding the source sentence with a convolutional neural network, and then generate the target sentence by reversing the process. A refinement of this was proposed by Gehring et al. (2017) who use multiple convolutional layers in the encoder and the decoder that do not reduce the length of the encoded sequence but incorporate wider context with each layer.

Vaswani et al. (2017) replace the recurrent neural networks used in attentional sequence-to-sequence models with multiple self-attention layers, both for the encoder as well as the decoder. There are a number of additional refinements of this model: so-called multi-head attention, encoding of sentence positions of words, etc.