8 Neural MT 2: Attentional Neural MT

In the past chapter, we described a simple model for neural machine translation, which uses an encoder to encode sentences as a fixed-length vector. However, in some ways, this view is overly simplified, and by the introduction of a powerful mechanism called **attention**, we can overcome these difficulties. This section describes the problems with the encoder-decoder architecture and what attention does to fix these problems.

8.1 Problems of Representation in Encoder-Decoders

Theoretically, a sufficiently large and well-trained encoder-decoder model should be able to perform machine translation perfectly. As mentioned in Section 5.2, neural networks are universal function approximators, meaning that they can express any function that we wish to model, including a function that accurately predicts our predictive probability for the next word $P(e_t|F, e_1^{t-1})$. However, in practice, it is necessary to learn these functions from limited data, and when we do so, it is important to have a proper **inductive bias** – an appropriate model structure that allows the network to learn to model accurately with a reasonable amount of data.

There are two things that are worrying about the standard encoder-decoder architecture. The first was described in the previous section: there are long-distance dependencies between words that need to be translated into each other. In the previous section, this was alleviated to some extent by reversing the direction of the encoder to bootstrap training, but still, a large number of long-distance dependencies remain, and it is hard to guarantee that we will learn to handle these properly.

The second, and perhaps more, worrying aspect of the encoder-decoder is that it attempts to store sentences of any arbitrary length in a hidden vector of fixed size. In other words, even if our machine translation system is expected to translate sentences of lengths from 1 word to 100 words, it will still use the same intermediate representation to store all of the information about the input sentence. If our network is too small, it will not be able to encode all of the information in the longer sentences that we will be expected to translate. On the other hand, even if we make the network large enough to handle the largest sentences in our inputs, when processing shorter sentences, this may be overkill, using needlessly large amounts of memory and computation time. In addition, because these networks will have large numbers of parameters, it will be more difficult to learn them in the face of limited data without encountering problems such as overfitting.

The remainder of this section discusses a more natural way to solve the translation problem with neural networks: attention.

8.2 Attention

The basic idea of attention is that instead of attempting to learn a single vector representation for each sentence, we instead keep around vectors for every word in the input sentence, and reference these vectors at each decoding step. Because the number of vectors available to reference is equivalent to the number of words in the input sentence, long sentences will have many vectors and short sentences will have few vectors. As a result, we can express input sentences in a much more efficient way, avoiding the problems of inefficient representations for encoder-decoders mentioned in the previous section.

First we create a set of vectors that we will be using as this variably-lengthed representation. To do so, we calculate a vector for every word in the source sentence by running an RNN in both directions:

$$\overrightarrow{\boldsymbol{h}}_j^{(f)} = \text{RNN}(\text{embed}(f_j), \overrightarrow{\boldsymbol{h}}_{j-1}^{(f)})$$
$$\overleftarrow{\boldsymbol{h}}_j^{(f)} = \text{RNN}(\text{embed}(f_j), \overleftarrow{\boldsymbol{h}}_{j+1}^{(f)}).$$

Then we concatenate the two vectors $\overrightarrow{\boldsymbol{h}}_j^{(f)}$ and $\overleftarrow{\boldsymbol{h}}_j^{(f)}$ into a bidirectional representation $\boldsymbol{h}_j^{(f)}$

$$\boldsymbol{h}_j^{(f)} = [\overleftarrow{\boldsymbol{h}}_j^{(f)}; \overrightarrow{\boldsymbol{h}}_j^{(f)}].$$

We can further concatenate these vectors into a matrix:

$$H^{(f)} = \text{concat\_col}(\boldsymbol{h}_1^{(f)}, \ldots, \boldsymbol{h}_{|F|}^{(f)}).$$

This will give us a matrix where every column corresponds to one word in the input sentence.

However, we are now faced with a difficulty. We have a matrix $H^{(f)}$ with a variable number of rows depending on the length of the source sentence, but would like to use this to compute, for example, the probabilities over the output vocabulary, which we only know how to do (directly) for the case where we have a vector of input. The key insight of attention is that we calculate a vector $\boldsymbol{\alpha}_t$ that can be used to combine together the columns of H into a vector $\boldsymbol{c}_t$

$$\boldsymbol{c}_t = H^{(f)}\boldsymbol{\alpha}_t. \tag{73}$$

$\boldsymbol{\alpha}_t$ is called the **attention vector**, and is generally assumed to have elements that are between zero and one and add to one.

The basic idea behind the attention vector is that it is telling us how much we are "focusing" on a particular source word at a particular time step. The larger the value in $\boldsymbol{\alpha}_t$, the more impact a word will have when predicting the next word in the output sentence. An example of how this attention plays out in an actual translation example is shown in Figure 27, and as we can see the values in the alignment vectors generally align with our intuition.

8.3 Calculating Attention Scores

The next question then becomes, from where do we get this $\boldsymbol{\alpha}_t$? The answer to this lies in the *decoder* RNN, which we use to track our state while we are generating output. As before, the decoder's hidden state $\boldsymbol{h}_t^{(e)}$ is a fixed-length continuous vector representing the previous target words e_1^{t-1}, initialized as $\boldsymbol{h}_0^{(e)} = \boldsymbol{h}_{|F|+1}^{(f)}$. This is used to calculate a context vector $\boldsymbol{c}_t$ that is used to summarize the source attentional context used in choosing target word e_t, and initialized as $\boldsymbol{c}_0 = \boldsymbol{0}$.

First, we update the hidden state to $\boldsymbol{h}_t^{(e)}$ based on the word representation and context vectors from the previous target time step

$$\boldsymbol{h}_t^{(e)} = \text{enc}([\text{embed}(e_{t-1}); \boldsymbol{c}_{t-1}], \boldsymbol{h}_{t-1}^{(e)}). \tag{74}$$

Based on this $\boldsymbol{h}_t^{(e)}$, we calculate an **attention score** $\boldsymbol{\alpha}_t$, with each element equal to

$$a_{t,j} = \text{attn\_score}(\boldsymbol{h}_j^{(f)}, \boldsymbol{h}_t^{(e)}). \tag{75}$$

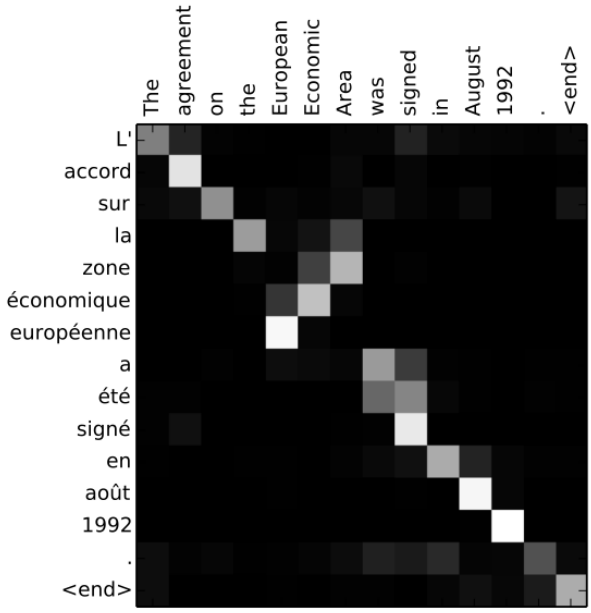

Figure 27: An example of attention [2], English is the source, French is the target, and a higher attention weight when generating a particular target word is indicated by a lighter color in the matrix.

attn_score$(\cdot)$ can be an arbitrary function that takes two vectors as input and outputs a score about how much we should focus on this particular input word encoding $\boldsymbol{h}_j^{(f)}$ at the time step $\boldsymbol{h}_t^{(e)}$. We describe some examples at a later point in Section 8.4.

We then normalize this into the actual attention vector itself by taking a softmax over the scores:

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{a}_t). \tag{76}$$

This attention vector is then used to weight the encoded representation H^f to create a context vector $\boldsymbol{c}_t$ for the current time step, as mentioned in Equation 73.

We now have a context vector $\boldsymbol{c}_t$ and hidden state $\boldsymbol{h}_t^{(e)}$ for time step t, which we can pass on down to downstream tasks. For example, we can concatenate both of these together when calculating the softmax distribution over the next words:

$$\boldsymbol{p}_t^{(e)} = \text{softmax}(W_{hs}[\boldsymbol{h}_t^{(e)}; \boldsymbol{c}_t] + b_s). \tag{77}$$

It is worth noting that this means that the encoding of each source word $\boldsymbol{h}_j^{(f)}$ is considered much more directly in the calculation of output probabilities. In contrast to the encoder-decoder, where the encoder-decoder will only be able to access information about the first encoded word in the source by passing it over $|F|$ time steps, here the source encoding is accessed (in a weighted manner) through the context vector Equation 73.

This whole, rather involved, process is shown in Figure 28.

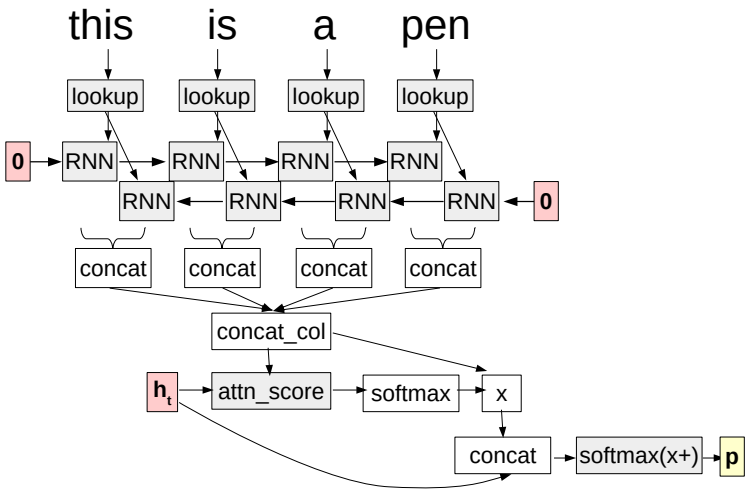

Figure 28: A computation graph for attention.

8.4 Ways of Calculating Attention Scores

As mentioned in Equation 75, the final missing piece to the puzzle is how to calculate the attention score $a_{t,j}$.

[8] test three different attention functions, all of which have their own merits:

Dot product: This is the simplest of the functions, as it simply calculates the similarity between $\boldsymbol{h}_t^{(e)}$ and $\boldsymbol{h}_j^{(f)}$ as measured by the dot product:

$$\text{attn\_score}(\boldsymbol{h}_j^{(f)}, \boldsymbol{h}_t^{(e)}) := \boldsymbol{h}_j^{(f)\intercal} \boldsymbol{h}_t^{(e)}. \tag{78}$$

This model has the advantage that it adds no additional parameters to the model. However, it also has the intuitive disadvantage that it forces the input and output encodings to be in the same space (because similar $\boldsymbol{h}_t^{(e)}$ and $\boldsymbol{h}_j^{(f)}$ must be close in space in order for their dot product to be high). It should also be noted that the dot product can be calculated efficiently for every word in the source sentence by instead defining the attention score over the concatenated matrix $H^{(f)}$ as follows:

$$\text{attn\_score}(H^{(f)}, \boldsymbol{h}_t^{(e)}) := H_j^{(f)\intercal} \boldsymbol{h}_t^{(e)}. \tag{79}$$

Combining the many attention operations into one can be useful for efficient impementation, especially on GPUs. The following attention functions can also be calculated like this similarly.[26]

Bilinear functions: One slight modification to the dot product that is more expressive is the **bilinear function**. This function helps relax the restriction that the source and target embeddings must be in the same space by performing a linear transform parameterized by W_a before taking the dot product:

$$\text{attn\_score}(\boldsymbol{h}_j^{(f)}, \boldsymbol{h}_t^{(e)}) := \boldsymbol{h}_j^{(f)\intercal} W_a \boldsymbol{h}_t^{(e)}. \tag{80}$$

[26] *Question: What do the equations look like for the combined versions of the following functions?*

This has the advantage that if W_a is not a square matrix, it is possible for the two vectors to be of different sizes, so it is possible for the encoder and decoder to have different dimensions. However, it does introduce quite a few parameters to the model ($|\boldsymbol{h}^{(f)}| \times |\boldsymbol{h}^{(e)}|$), which may be difficult to train properly.

Multi-layer perceptrons: Finally, it is also possible to calculate the attention score using a multi-layer perceptron, which was the method employed by [2] in their original implementation of attention:

$$\text{attn\_score}(\boldsymbol{h}_t^{(e)}, \boldsymbol{h}_j^{(f)}) := \boldsymbol{w}_{a2}^\intercal \tanh(W_{a1}[\boldsymbol{h}_t^{(e)}; \boldsymbol{h}_j^{(f)}]), \tag{81}$$

where W_{a1} and $\boldsymbol{w}_{a2}$ are the weight matrix and vector of the first and second layers of the MLP respectively. This is more flexible than the dot product method, usually has fewer parameters than the bilinear method, and generally provides good results.

In addition to these methods above, which are essentially the defacto-standard, there are a few more sophisticated methods for calculating attention as well. For example, it is possible to use recurrent neural networks [14], tree-structured networks based on document structure [15], convolutional neural networks [1], or structured models [5] to calculate attention.

8.5 Copying and Unknown Word Replacement

One pleasant side-effect of attention is that it not only increases translation accuracy, but also makes it easier to tell which words are translated into which words in the output. One obvious consequence of this is that we can draw intuitive graphs such as the one shown in Figure 27, which aid error analysis.

Another advantage is that it also becomes possible to handle unknown words in a more elegant way, performing **unknown word replacement** [7]. The idea of this method is simple, every time our decoder chooses the unknown word token $\langle \text{unk} \rangle$ in the output, we look up the source word with the highest attention weight at this time step, and output that word instead of the unknown token $\langle \text{unk} \rangle$. If we do so, at least the user can see which words have been left untranslated, which is better than seeing them disappear altogether or be replaced by a placeholder.

It is also common to use alignment models such as those in Section 11 to obtain a translation dictionary, then use this to aid unknown word replacement even further. Specifically, instead of copying the word as-is into the output, if the chosen source word is f, we output the word with the highest translation probability $P_{dict}(e|f)$. This allows words that are included in the dictionary to be mapped into their most-frequent counterpart in the target language.

8.6 Intuitive Priors on Attention

Because of the importance of attention in modern NMT systems, there have also been a number of proposals to improve accuracy of estimating the attention itself through the introduction of intuitively motivated prior probabilities. [3] propose several methods to incorporate biases into the training of the model to ensure that the attention weights match our belief of what alignments between languages look like.

These take several forms, and are heavily inspired by the alignment models that will be explained in Section 11. These models can be briefly summarized as:

Position Bias: If two languages have similar word order, then it is more likely that alignments should fall along the diagonal. This is demonstrated strongly in Figure 27. It is possible to encourage this behavior by adding a prior probability over attention that makes it easier for things near the diagonal to be aligned.

Markov Condition: In most languages, we can assume that most of the time if two words in the target are contiguous, the aligned words in the source will also be contiguous. For example, in Figure 27, this is true for all contiguous pairs of English words except "the, European" and "Area, was". To take advantage of this property, it is possible to impose a prior that discourages large jumps and encourages local steps in attention. A model that is similar in motivation, but different in implementation, is the **local attention** model [8], which selects which part of the source sentence to focus on using the neural network itself.

Fertility: We can assume that some words will be translated into a certain number words in the other langauge. For example, the English word "cats" will be translated into two words "les chats" in French. Priors on fertility takes advantage of this fact by giving the model a penalty when particular words are not attended too much, or attended to too much. In fact one of the major problems with poorly trained neural MT systems is that they repeat the same word over and over, or drop words, a violation of this fertility constraint. Because of this, several other methods have been proposed to incorporate coverage in the model itself [11, 9], or as a constraint during the decoding process [13].

Bilingual Symmetry: Finally, we expect that words that are aligned when performing translation from F to E should also be aligned when performing translation from E to F. This can be enforced by training two models in parallel, and enforcing constraints that the alignment matrices look similar in both directions.

[3] experiment extensively with these approaches, and find that the bilingual symmetry constraint is particularly effective among the various methods.

8.7 Further Reading

This section outlines some further directions for reading more about improvements to attention:

Hard Attention: As shown in Equation 76, standard attention uses a soft combination of various contents. There are also methods for hard attention that make a hard binary decision about whether to focus on a particular context, with motivations ranging from learning explainable models [6], to processing text incrementally [16, 4].

Supervised Training of Attention: In addition, sometimes we have hand-annotated data showing us true alignments for a particular language pair. It is possible to train attentional models using this data by defining a loss function that penalizes the model when it does not predict these alignments correctly [10].

Other Ways of Memorizing Input: Finally, there are other ways of accessing relevant information other than attention. [12] propose a method using **memory networks**, which have a separate set of memory that can be written to or read from as the processing continues.

8.8 Exercise

In the exercise for this chapter, we will create code to train and generate translations with an attentional neural MT model.

Writing the program will entail extending your encoder-decoder code to add attention. You can then generate translations and compare them to others.

- Extend your encoder-decoder code to add attention.
- On the training set, write code to calculate the loss function and perform training.
- On the development set, generate translations using greedy search. Look at them to see if they look good or not, and compare with those generated by the encoder-decoder.

It is also highly recommended, but not necessary, that you attempt to implement unknown word replacement.

Potential improvements to the model include implementing any of the improvement to attention mentioned in Section 8.6 or Section 8.7.

References

[1] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A convolutional attention network for extreme summarization of source code. *arXiv preprint arXiv:1602.03001*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[3] Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 876–885, 2016.

[4] Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor OK Li. Learning to translate in real-time with neural machine translation. 2017.

[5] Y. Kim, C. Denton, L. Hoang, and A. M. Rush. Structured Attention Networks. *ArXiv e-prints*, February 2017.

[6] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 107–117, 2016.

[7] Minh-Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 11–19, 2015.

[8] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, 2015.

[9] Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 955–960, 2016.

[10] Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. Supervised attentions for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2283–2288, 2016.

[11] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 76–85, 2016.

[12] Mingxuan Wang, Zhengdong Lu, Hang Li, and Qun Liu. Memory-enhanced decoder for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 278–286, 2016.

[13] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[14] Zichao Yang, Zhiting Hu, Yuntian Deng, Chris Dyer, and Alex Smola. Neural machine translation with recurrent attention modeling. *arXiv preprint arXiv:1607.05108*, 2016.

[15] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[16] Lei Yu, Jan Buys, and Phil Blunsom. Online segment to segment neural transduction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1307–1316, 2016.