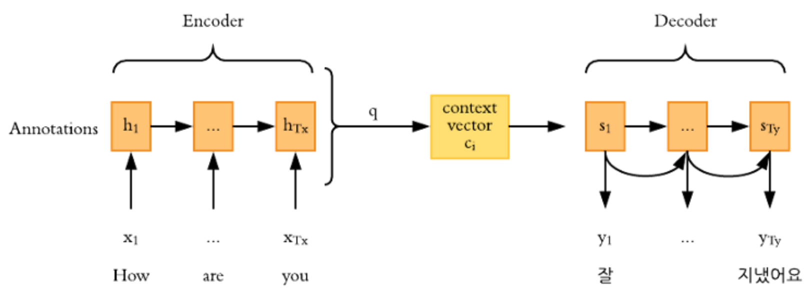
Kate Loginova Follow
Jun 22, 2018 · 13 min read

In this post, I will describe recent work on attention in deep learning models for natural language processing. I'll start with the attention mechanism as it was introduced by Bahdanau. Then, we will go through self-attention, two-way attention, key-value-predict models and hierarchical attention.

In many tasks, such as machine translation or dialogue generation, we have a sequence of words as an input (e.g., an original text in English) and would like to generate another sequence of words as an output (e.g., a translation to Korean). Neural networks, especially recurrent ones (RNN), are well suited for solving such a task. I assume that you are familiar with RNNs and LSTMs. Otherwise, I recommend to check out an explanation in a famous blog post by Christopher Olah.

The "sequence-to-sequence" neural network models are widely used for NLP. A popular type of these models is an "encoder-decoder". There, one part of the network—encoder—encodes the input sequence into a fixed-length context vector. This vector is an internal representation of the text. This context vector is then decoded into the output sequence by the decoder. See an example:



An encoder-decoder neural network architecture. An example on machine translation: an input sequence is an English sentence "How are you" and the reply of the system would be a Korean translation: "잘 지냈어요".

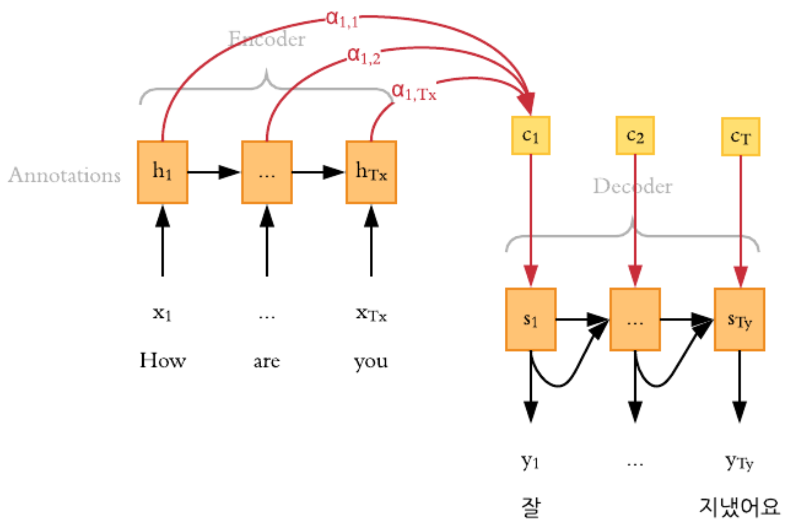
Here *h* denotes hidden states of the encoder and *s* of the decoder. *Tx* and *Ty* are the lengths of the input and output word sequences respectively. *q* is a function which generates the context vector out of the encoder's hidden states. It can be, for example, just q({h_i}) = h_T. So, we take the last hidden state as an internal representation of the entire sentence.

You can easily experiment with these models, as most deep learning libraries have general purpose encoder-decoder frameworks. To name a few, see Google's implementation for Tensorflow and IBM's one for PyTorch.

However, there is a catch with the common encoder-decoder approach: a neural network compresses all the information of an input source sentence into a fixed-length vector. It has been shown that this leads to a decline in performance when dealing with long sentences. The attention mechanism was introduced by Bahdanau in "Neural Machine Translation by Jointly Learning to Align and Translate" to alleviate this problem.

Attention

The basic idea: each time the model predicts an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence. In other words, it only pays attention to some input words. Let's investigate how this is implemented.



An illustration of the attention mechanism (RNNSearch) proposed by [Bahdanau, 2014]. Instead of converting the entire input sequence into a single context vector, we create a separate context vector for each output (target) word. These vectors consist of the weighted sums of encoder's hidden states.

Encoder works as usual, and the difference is only on the decoder's part. As you can see from a picture, the decoder's hidden state is computed with a context vector, the previous output and the previous hidden state. But now we use not a single context vector c, but a separate context vector c_i for each target word.

These context vectors are computed as a weighted sum of *annotations* generated by the encoder. In Bahdanau's paper, they use a Bidirectional LSTM, so these annotations are concatenations of hidden states in forward and backward directions.

The weight of each annotation is computed by an *alignment model* which scores how well the inputs and the output match. An alignment model is a feedforward neural network, for instance. In general, it can be any other model as well.

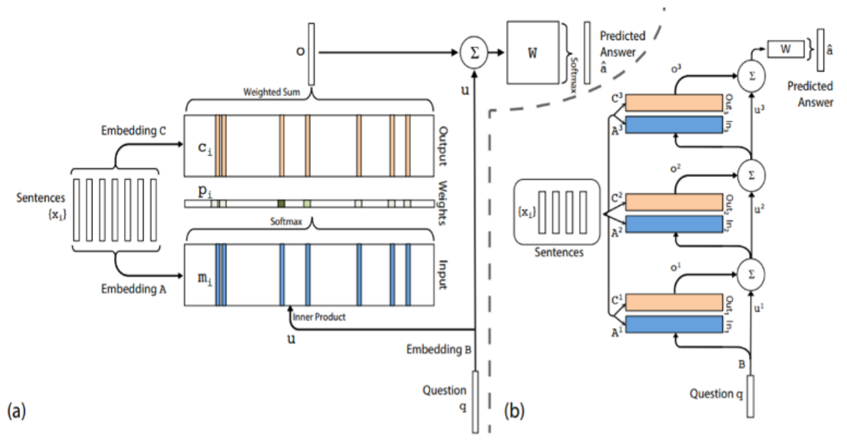
As a result, the alphas—the weights of hidden states when computing a context vector—show how important a given annotation is in deciding the next state and generating the output word. These are the attention scores.

If you want to read a bit more about the intuition behind this, visit WildML's blog post. You can also enjoy an interactive visualization in the Distill blog. In the meantime, let's move on to a bit more advanced attention mechanisms.

Memory networks

One group of attention mechanisms repeats the computation of an attention vector between the query and the context through multiple layers. It is referred to as multi-hop. They are mainly variants of end-to-end memory networks, which we will discuss now.

[Sukhbaatar, 2015] argues that the attention mechanism implemented by Bahdanau can be seen as a form of memory. They extend this mechanism to a multi-hop setting. It means that the network reads the same input sequence multiple times before producing an output, and updates the memory contents at each step. Another modification is that the model works with multiple source sentences instead of a single one.



End-to-End Memory Networks.

Let's take a look at the inner workings. First, let me describe the single layer case (a). It implements a single memory hop operation. The entire input set of sentences is converted into memory vectors m. The query q is also embedded to obtain an internal state u. We compute the match between u and each memory by taking the inner product followed by a softmax. This way we obtain a probability vector p over the inputs (this is the attention part). Each input also has a corresponding output vector. We use the weights p to weigh a sum of these output vectors. This sum is a response vector o from the memory. Now we have an output vector o and the input embedding u. We sum them, multiply by a weight matrix W and apply a softmax to predict a label.

Now, we can extend the model to handle K hop operations (b). The memory layers are stacked so that the input to layers k + 1 is the sum of the output and the input from layer k. Each layer has its own embedding matrices for the inputs.

When the input and output embeddings are the same across different layers, the memory is identical to the attention mechanism of Bahdanau. The difference is that it makes multiple hops over the memory (because it tries to integrate information from multiple sentences).

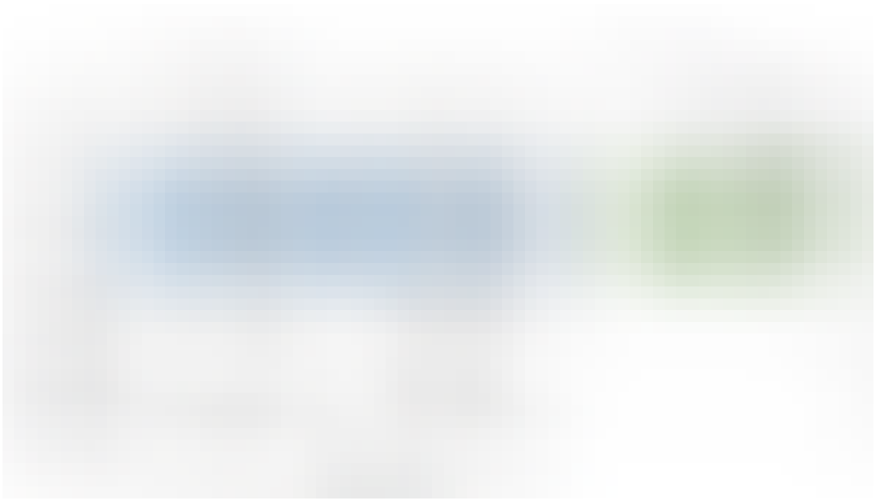
A fine-grained extension of this method is an Attentive Reader introduced by [Hermann, 2015].

. . .

Variations of attention

[Luong, 2015] introduces the difference between **global** and **local** attention. The idea of a global attention is to use all the hidden states of the encoder when computing each context vector. The downside of a global attention model is that it has to attend to all words on the source side for each target word, which is computationally costly. To overcome this, the local attention first chooses a position in the source sentence. This position will determine a window of words that the model attends to. The authors also experimented with different alignment functions and simplified the computation path compared to Bahdanau's work.

Attention Sum Reader [Kadlec, 2016] uses **attention as a pointer** over discrete tokens in the text. The task is to select an answer to a given question from the context paragraph. The difference with other methods is that the model selects the answer from the context directly using the computed attention instead of using the attention scores to weigh the sum of hidden vectors.



Attention Sum Reader.

As an example, let us consider the question-context pair. Let the context be "A UFO was observed above our city in January and again in March." and the question be "An observer has spotted a UFO in … ." January and March are equally good candidates, so the previous models will assign equal attention scores. They would then compute a vector between the representations of these two words and propose the word with the closest word embedding as the answer. At the same time, Attention Sum Reader would correctly propose January or March, because it chooses words directly from the passage.

· · ·

Two-way Attention & Coattention

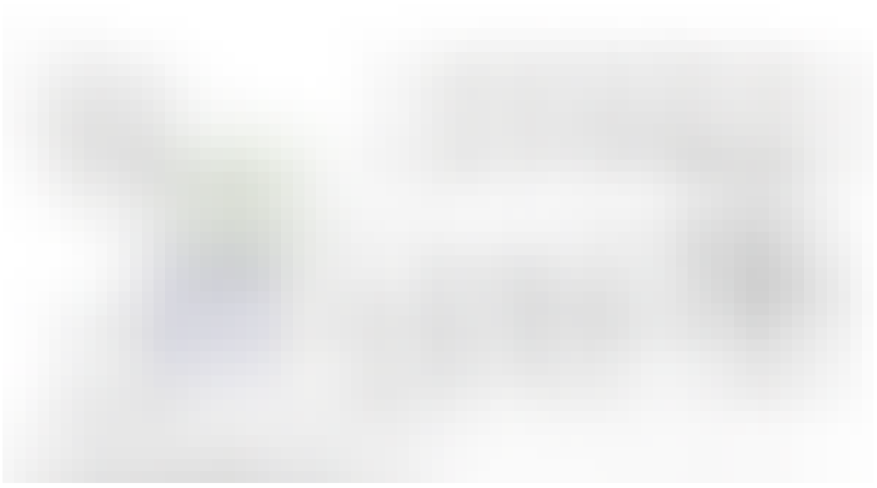
As you might have noticed, in the previous model we pay attention from source to target. It makes sense in translation, but what about other fields? For example, consider textual entailment. We have a premise *"If you help the needy, God will reward you"* and a hypothesis "*Giving money to a poor man has good consequences".* Our task is to understand whether the premise entails the hypothesis (in this case, it does). It would be useful to pay attention not only from the hypothesis to the text but also the other way around.

This brings the concept of **two-way attention** [Rocktäschel, 2015]. The idea is to use the same model to attend over the premise, as well as over the hypothesis. In the simplest form, you can simply swap the two sequences. This produces two attended representations which can be concatenated.

Top: model from [Rocktäschel, 2015]. Bottom: MatchLSTM from [Wang, Jiang, 2016]. h vectors in the first model are weighted versions of the premise only, while in the second model they "represent the matching between the premise and the hypothesis up to position k."

However, such a model will not let you emphasize more important matching results. For instance, alignment between stop words is less important than between the content words. In addition, the model still uses a single vector to represent the premise. To overcome these limitations, [Wang, Jiang, 2016] developed MatchLSTM. To deal with the importance of the matching, they add a special LSTM that will remember important matching results, while forgetting the others. This additional LSTM is also used to increase the granularity level. We will now multiply attention weights with each hidden state. It performed well in question answering and textual entailment tasks.

The question answering task gave rise to even more advanced ways to combine both sides. Bahdanau's model, that we have seen in the beginning, uses a summary vector of the query to attend to the context. In contrast to it, the **coattention** is computed as an alignment matrix on *all* pairs of context and query words. As an example of this approach, let's examine Dynamic Coattention Networks [Xiong, 2016].



Dynamic Coattention Networks [Xiong, 2016].

Let's walk through what is going on in the picture. First, we compute the affinity matrix of all pairs of document and question words. Then we get the attention weights AQ across the document for each word in the question and AD—the other way around. Next, the summary or attention context of the document in light of each word in the question is computed. In the same way, we can compute it for the question in light of each word in the document. Finally, we compute the summaries of the previous attention contexts given each word in the document. The resulting vectors are concatenated into a co-dependent representation of the question and the document. This is called the *coattention context*.

. . .

Self-attention

Another problem is that recurrent network can only memorize limited passage contexts in practice despite its theoretical capabilities. For example, in question answering, one answer candidate is often unaware of the clues in other parts of the paragraph. [Cheng, 2016] proposed a self-attention, sometimes called intra-attention. It is a mechanism relating different positions of a *single* sequence to compute its internal representation.

Self-attention has been used successfully in a variety of tasks. One of the use cases is sentiment analysis. For tasks like this, standard attention is not directly applicable, because there is no extra information: the model is only given one sentence as input. A common approach is to use the final hidden state or pooling. However, it is hard to preserve semantics this way. In Question Answering, there are

models used in the SQuAD competition: r-net [Wang, 2017] and Reinforced Mnemonic Reader [Hu, 2017]. In Natural Language Inference—Decomposable Attention Model [Parikh, 2016].
In machine translation, self-attention also contributes to impressive results. For example, recently a model, named Transformer, was introduced in a paper with a rather bold title "Attention Is All You Need" [Vaswani, 2017]. As you can guess, this model relies only on self-attention without the use of RNNs. As a result, it is highly parallelizable and requires less time to train, while establishing state-of-the-art results on WMT2014.

But perhaps the most exciting property for linguists would be that self-attention seems to learn sophisticated syntactic patterns. Take a look at the example of the networks learning how to resolve anaphora in the sentence:

Syntactic patterns learnt by the Transformer [Vaswani, 2017] using solely self-attention.

· · ·

Key-Value(-Predict) attention

In the discussed algorithms, we require output vectors to simultaneously store information for predicting the next word, computing the attention, and encode content relevant to future steps. Such an overloaded use of output representations can make training unnecessarily difficult. A key-value(-predict) attention [Daniluk, 2017] has been proposed to combat the problem. In a key-value part, we separate output vectors into

- keys to calculate the attention, and
- values to encode the next-word distribution and context representation.

However, we still use the value part for two goals at once. So, the authors split it again and in the end, the model outputs three vectors at each time step. The first is used to encode the next-word distribution, the second serves as a key to compute the attention vector, and the third as value for an attention mechanism.

Key-value(-predict) attention. We separate output vectors into key, value (and predict) parts to lower the burden of storing information for three different goals in the same vector.

• • •

Hierarchical & Nested attention

Texts tend to have a hierarchical structure and the importance of words and sentences are highly context dependent. To include this insight, hierarchical model proposed by [Yang, 2016] uses two levels of attention—one at the word level and one at the sentence level. Such architecture also allows for a better visualization—the highly informative components of a document are highlighted. A similar idea was presented in the literature for word- and character-levels as well and adapted to a multilingual setting by [Pappas, 2017].

Another take on this is Attention-over-Attention [Cui, 2016]. The idea is to place another attention over the primary attentions, to indicate the "importance" of each attentions.

Attention-over-Attention [Cui, 2016].

Attention-over-Attention is motivated by the Attention Sum Reader we have seen before. They first calculate a pair-wise matching matrix M by multiplying contextual embeddings produced by an LSTM for each pair of words. Then, they apply a column-wise softmax to get the query-to-document attention weights (alphas). The weights represent an alignment between the entire document and a single query word. After that, document-to-query attention weights are obtained by applying softmax row-wise. It is again a form of two-way attention. But now we calculate the dot-product of these weights to get attention-over-attention.

. . .

Attention flow

The last of the more established concepts I would like to mention is an attention flow network. In such a network the attention model is decoupled from the RNN. An idea was introduced in the BiDAF model [Seo, 2016]. The attention is computed for every time step, and the attended vector, along with the representations from previous layers, is allowed to flow through to the next modelling layer. The goal is to reduce the information loss caused by early summarization. A multi-hop extension is the Ruminating Reader [Gong, 2017].

BiDirectional Attention Flow Model [Seo, 2016].

On the image, we can see the attention flow layer. It is responsible for linking and fusing information from the context and the query words by usin two-way attention. The inputs to the layer are contextual vector representations of the context and the query. The outputs of the layer are both the query-aware representations of the context words and the contextual embeddings. Note that we do not produce a single vector, but rather let the attention information flow to the next layer at each time step.

. . .

Classification of attention mechanisms

There have been several attempts to classify the existing attention models. I'll outline the main differences mentioned in different literature overviews.

[Huang, 2017] distinguish between several types of attention fusion processes: (1) Word-level fusion. (2) High-level fusion. (2′) High-level fusion (Alternative). (3) Self-boosted fusion. (3′) Self-boosted fusion (Alternative). See the paper for more details.

1. attention weights computation—see [Seo, 2016]

- a dynamic attention mechanism—the attention weights are updated dynamically given the query and the context as well as the previous attention: RNNSearch, Attentive Reader, MatchLSTM …
- computes the attention weights once, which are then fed into an output layer for final prediction: Attention Sum Reader
- repeats computing an attention vector between the query and the context through multiple layers: End-to-End Memory Networks

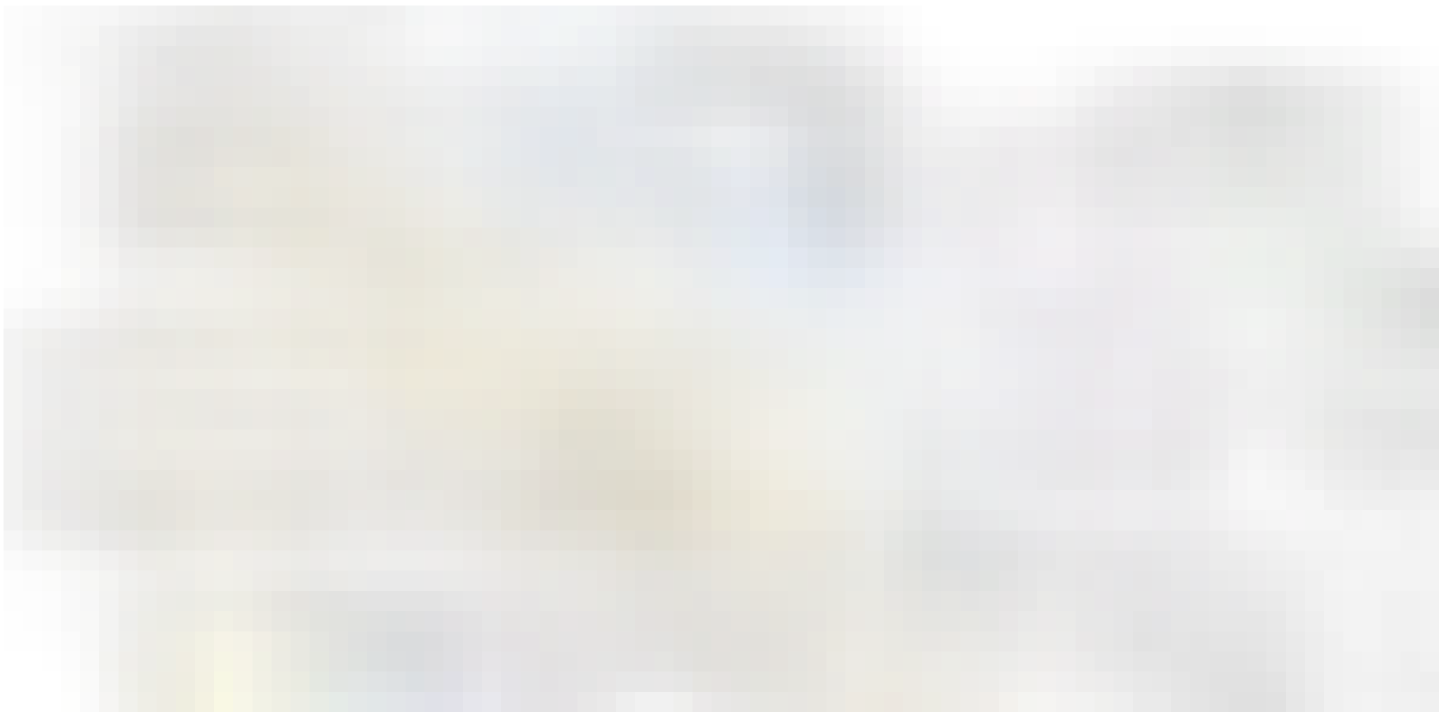
2. multi-pass & single-pass—see [Gong, 2017]

3. one-dimensional attention & two-dimensional attention

In one-dimensional attention, the whole source text is represented by one embedding vector. On the contrary, every word in the source has its own embedding vector in the situation of two-dimensional attention.

4. one-way & two-way attention (especially relevant for textual entailment and question answering)

5. different matching and fusion functions


Some of the most important papers on attention in NLP, organised by year and references to each other.

I have tried to organize the papers mentioned in this post (and some other related to them) into a hierarchy of citations. You can find a clickable version here as a draw.io diagram. The colors of the cells correspond to the NLP task.

Remaining issues

Standard attention architecture does not directly model any structural dependencies that may exist among the source elements. Instead, it relies completely on the hidden layers of the neural network. However, modelling such structural dependencies has been shown to be important in many deep learning applications. It can significantly improve the results and there is quite some recent research into this:

- Tree-to-Sequence Attentional Neural Machine Translation [Eriguchi, 2016]
- Structured Attention Networks [Kim, 2017]
- A Structured Self-attentive Sentence Embedding [Lin, 2017]

Another thing to look out for is attentive weights that are biased towards the words in the end of the sentence (in case of uni-directional RNNs) [Wang, 2016].

. . .

Implementations

Finally, here are some implementations of the attention mechanisms available for Python deep learning frameworks: