Attention

Eilif Solberg

September 19, 2018

Contents

1 Introduction

We are all familiar with the importance of *attention* through experiences in our own lives. Having found yourself in a crowded room with multiple simultaneous conversations, you are still able to interpret what your conversational partner is saying. Trying to listen in on someone else's conversation, however, you suddenly find yourself embarrassed having to ask your partner to repeat him or herself. This common situation is a powerful illustration of both the promise and disadvantages of attention mechanisms.

In the same spirit humans are in general not great *multitaskers*. We are able to walk and chew gum at the same time, but performing two cognitively demanding tasks simultaneously is near impossible without severe degradation in performance. We thus see that we have some cognitive bottlenecks for both *processing of input* as well as in our *ability to perform actions*.

Humans seem to both be able to change the *focus* of their mental effort as well as their *level of attention*. The second part refers to the *total mental effort* we put into a situation, while the first refers to the *distribution* of the total mental effort.

Thus, when someone tells us that "we should pay attention", our failure may lie in either our *level* or our *distribution* of attention. Humans have the ability to dynamically change the level of attention depending on the need of the task. This is useful, as we can save energy when possible, and use all our mental capacity when needed. We will not aim for something as ambitious as this. Instead we will focus on how we *distribute* our attention. There are at least a couple of arguments that can be made for introducing attention into our models

- **Ignoring irrelevant part of the input**. There are cases where some of the input data provides little or no additional information for the task at hand. We saw this clearly in the introductory example, other peoples conversations are unlikely to help if you are trying to hear what your partner is saying. Part of the input data serves only as a distractor, and ignoring it sooner rather than later seems like a good idea.
- **Using computational resources wisely**. If we apply the same processing to all the input data, irregardless of their usefulness in the accomplishment of a certain task, it is clear that we will waste a lot of computational resources. In general computation is a scarce resource, we should treat is as such.

We will now take a closer look at ways we can make models that uses attention. We have already seen an example of this on the notes on extensions to recurrent neural networks, where we increased the effective memory of RNNs by allowing them to attend to previous states. We have also previously seen different ways of addressing, location and content-based, in our discussions of external memory. We will see that there are in fact very close relations between attention and the addressing mechanism described before. We will make some remarks on some of the similarities and differences at the end of this chapter.

2 Content-based attention

One very common way of attention is *content-based* attention. Here we have basically the same setup as for content-based *addressing*. We don't specify the exact location where we want to attend, we set up a query and retrieve a *fixed-sized* vector according to the matching scores. This approach is taken in e.g. [1, 2]. For both of these papers we basically have an *encoder-decoder* setup. For [1] we "translate" an image into a sentence. The encoder

is a convolutional neural network that takes an image as input, and outputs a higher-level representation of this. The decoder is an RNN. Before attention-mechanisms gained traction, the common approach was to use the *penultimate* layer of a image classifier (typically trained on ImageNet) as a fixed size feature vector. A disadvantage of this is that we now have lost most of the spatial structure of the image. As the goal of the paper is to create a caption to an image, such spatial information may be useful. In addition, having a fixed size representation may become a bottleneck for larger images. The contribution of [1] was to, instead of using the penultimate layer, use the output of an earlier *convolutional* layer as input. The input is then a $h \times w \times c$ feature map which could be of variable size (though in the paper they had $h = w = 14$ in all cases). The idea is that over the course of the caption generation we may want to focus on different parts of the image. This is accomplished by generating a query at each time step, where the query is a function of the state at the current time. You may have noticed, but the model we have just presented is only a simplified version of the RNN with external memory presented earlier. The memory matrix M has $h \times w$ cells, where each of the cells are c-dimensional. The model for the read mechansim needs no change, while the write mechansim can be removed as we treat the memory as read-only. In this example we don't have any additional input data, our input is already given to us through the "memory".

The case in [2] is similar, here we are dealing with a translation model, in the more traditional interpretation of the word. Before attention was introduced in this context it was common to use an RNN as encoder and then to feed the final state of the encoder as input to the decoder, which was often an RNN as well. As this vector was of fixed length however it became a bottleneck when encoding long sentences. So the idea of [2] was then to feed *all* the states of the encoder as input to the decoder. For each new word generated in the target sentence the decoder attends to a different part of the state sequence of the encoder through content-based attention. Feeding the whole sequence as input to the decoder also allowed them to use a bidirectional RNN as encoder. The argument to show that this is also a special case of the RNN with external memory is similar as before. A "memory" cell is now one of the state vectors of the encoder RNN (or the concatenation of the forward and reverse RNNs). There are T such cells in total, where T is the length of the input sentence.

We will now look at an *extension* of the content-based attention scheme to something we shall for the time being call *self-attention*. Up until now we have thought about the query coming from a source *different* from the data

we are querying. It's like we have had a computational brain which have decided what things to attend to. Furthermore our discussions have been concerned with a single query, or at least a fixed number of queries being performed, at any time. Neither of these assumptions will be true in the self-attention case. Let's get into it.

Assume we have data $X^1, X^2, \ldots, X^L \in \mathbb{R}^n$. Each X^i could e.g. be an element of a sequence. We again have a key function K and a matching function g. So far nothing is new. Now comes the twist. Instead of having a fixed number of queries based on e.g. the current state in the RNN case, each of the data points X^i will perform their own queries, all in parallel! Every cell may in general query any other cell. It's like we have moved from centralized control to a distributed system! Another way to put it is that we have moved from *top-down* to *bottom-up* attention.

We will first describe the scenario where each cell performs exactly *one* query. We define a query function $Q\colon \mathbb{R}^n \to \mathbb{R}^d$, and get attention scores

$$\alpha_{i,j} = g(Q(X^i), K(X^j)) \tag{1}$$

We then normalize the score to get probabilities by applying e.g. the softmax function. For each $i, j \in \{1, \ldots L\}$ we have

$$p_{i,j} = \frac{e^{\alpha_{i,j}}}{\sum_k e^{\alpha_{i,k}}} \tag{2}$$

These coefficients now define L different probability distributions $\pi^1, \ldots, \pi^L$ over $\{X^1, \ldots, X^L\}$. As before we now have the choice between hard and soft attention. In the hard attention case, for each $i \in \{1, \ldots, L\}$ we retrieve one of the X^j by drawing from π^i, i.e.

$$Z^i \sim \pi^i(\{X^1, \ldots, X^L\}) \tag{3}$$

In the soft-attention case we again take a weighted average

$$Z^{\mathrm{i}} = \sum_{\mathrm{j}=1}^{\mathrm{L}} p_{\mathrm{i,\,j}} X^{\mathrm{j}} \tag{4}$$

which may be seen as an expectation over the distribution induced by π^i.

We are not quite done yet, however. We shall also introduce a *value* function V. The role of the value function is to decide what properties of X^j to return given that our query matches the key for X^j. With a value

function we have that (3) and (4) transforms into

$$Z^i \sim \pi^i(\{V(X^1), \ldots, V(X^L)\}) \tag{5}$$

$$Z^i = \sum_{j=1}^{L} p_{i,j} V(X^j) \tag{6}$$

We have just described the case where each cell performs exactly one query. Why stop there? Our query might contain one interesting property, but there may be many others. In addition several queries will allow us to focus narrowly on several different place. We call this *multi-head* attention, instead of having a single Q, K and V we now have M (query, key, value) triplets $(Q_1, K_1, V_1), \ldots, (Q_M, K_M, V_M)$ which each will give rise to a series of data points $Z_m^1, \ldots, Z_m^L$. For further processing we may e.g. combine the results by concatenation, i.e. $Z^i = (Z_1^i, \ldots Z_M^i)$.

The setup we have just described was introduced in [3], where they had great success applying this to natural language processing tasks. For the translation task they used the common encoder-decoder setup, but instead of using RNNs for the encoder and decoder they implemented modules heavily based on the attention mechanism described above.

The self-attention mechanism described appears to be quite different from what is traditionally thought of as attention mechanisms. Although it seems that the self-attention mechanism could be used as a way of e.g. judging the relevance of other input data, it is not clear that this will actually happen. It might be views as some, perhaps more general, data-dependent processing mechanism that we don't yet have a good understanding of.

3 Location-based attention

We will now look at an example of how we can apply *location-based* attention. For concreteness we will take image data as an example and we will base our discussions on [4]. Though in principle the methods we will discuss here could be applied to any set of data points X^1, ..., X^L some of the techniques discussed will make the most sense for data that has some sort of grid structure, e.g. data that is ordered as a sequence or has some kind of spatial structure.

Assume we have an input image X and we would like to classify a single object that is present in the image. Instead of feeding the whole image into e.g. a convolutional neural network, we will have an RNN that at each time step takes only part of the image as input. We shall call each such subimage

a *glimpse*. It will be the RNN itself that decides what part of the image to feed in at each time step. We can imagine that applying attention in this case is particularly useful if the object only occupies only a small part of the image, and there is a lot of background clutter. Attention will let us focus on the part of the image that contains the object. Note that this attention mechanism potentially has both of the benefits that we mentioned in the introduction, (1) better performance by ignoring possibly distracting parts of the input data and (2) computational savings as we use most of our computational resources on the input data that are relevant to the task at hand.

For simplicity assume we have a fixed number of glimpses, say τ. Each glimpse will be a fixed $h \times w$ crop of the image, which we shall assume to be much smaller than the height and width of the image, centered around a location l^t. At the first time step the network may e.g. get a glimpse located at the center of the image, or at a random location. All subsequent glimpses will be decided by the network. In addition to our $h \times w$ crop, we will extract at least one other crop, say of size $4h \times 4w$ which we will immediatly resized to $h \times w$. This *lower-resolution* crop, sharing the same center, gives us some context and may provide us with useful cues on where to look next. Note that if we do not have any lower-resolution patches, we might have to go through a large portion of the image before even locating the object. Having lower-resolution crops will help us guide our search quickly towards the areas in the image which we would like to process at full resolution.

For all but the final step we output the next location at which to extract the glimpse with a function f_l dependent on our current state, i.e.

$$l^t = f_l(s^t) \tag{7}$$

For the final step we output either the predicted class, or a probability vector distributed over the classes

$$c = f_c(s^\tau) \tag{8}$$

The update equation of our RNN looks like

$$s^t = h(x(l^{t-1}), s^{t-1}, l^{t-1}) \tag{9}$$

where $x(l^{t-1})$ is the image crop in addition to lower resolution crops, centered at l^{t-1}.

We now discuss a couple of ways of how to encode l^t, our next center of attention. The approach taken in [4] is to encode l^t as a vector (x, y),

where $x, y \in [-1, 1]$. Thus if we output $l^t = (-1, -1)$ our next glimpse will be centered on the top-left part of the image. If we output $l^t = (0, 0)$ we will extract a glimpse at the center of the image and so on. In principle we have infinitely many possible location centers, though many will of course have highly overlapping fields of view. In [4] the function f_l outputs a location center $\mu^t = (\mu_x^t, \mu_y^t)$ and we draw the location from a normal distribution with mean μ^t and fixed variance σ^2.

$$l^t \sim \pi_l(s^t) = (N(f_l(s^t)_x, \sigma^2), N(f_l(s^t)_y, \sigma^2)) \tag{10}$$

Another approach would be to divide each image into a $h \times w$ grid, where h and w are fixed, independent of image size. We would then first have a function π_l that outputs a probability distribution over the $h \times w$ cells and then draw from this distribution.

$$l^t \sim \pi_l(s^t) = softmax(f_l(s^t)) \tag{11}$$

Note that applying *soft* attention in this case would counteract our goal of reducing the amount of computations used for processing. There is one more thing that we need to discuss. How do we actually train our location policy f_l? As we don't really know where the best places to look are, we can't train this in a supervised fashion. Instead we will apply a technique from from reinforcement learning called *policy gradient*. A rigorous explantation will have to wait, but we will provide some intution about how it works. If we ultimately made the correct classification we probably looked at a lot of the right places, and we then *increase* the probability of looking at those places. If we at the end of the episode misclassified the object we might have focused on the wrong parts of the image. We then update the location policy π to *decrease* the probability for those areas, so that next time we are more likely to look somewhere else. Of course this signal is far from perfect as we might focus on exactly the right locations and still get the class wrong, but as long as we get the class right *more often* when we look at relevant areas our updates will *on average* go in the right direction. Exactly how we change these probabilities depends on what kind of encoding we choose for l^t and our parametrization of π_l.

We will end this section on location-based attention by mentioning another location-based attention mechnism we have already encountered, namely the LSTM *output gate*. It was not introduced as an attention mechanism per se, but as we argued before it may be viewed as one (though over the state

vector and not the input data). We will briefly revisit the form of it here. If you recall, the LSTM has a hidden state vector s^t that is not directly visible to the outside world, but instead presents a vector $\bar{s}^t$. We have

$$\bar{s}^t = o^t \odot a(s^t) \tag{12}$$

$$o^t = \sigma(U_o x^t + V_o \bar{s}^{t-1} + W_o y^{t-1} + b_o) \tag{13}$$

As we argued earlier, the output gate may try to mask out data from s^t that are currently not useful, which is exactly one of the purposes of attention. It does not save us any computations, but this is not a strict requirement for an attention mechanism.

4 Addressing external memory vs attention

We have seen that addressing external memory and attention mechanisms share many common properties. Especially content-based addressing and attention is used so similarly that we in this chapter could apply some of the same models we introduced earlier. This section, however, contains some thoughts on possible *differences* both in motivation and implementation between the mechanisms. They should rather be treated starting points for discussions than as concluding remarks.

- Addressing is about *retrieving* some information. The information is not already at our fingertips. Attention is about *suppressing* or *ignoring* information. The information is easily accessible, but we see ourselves better off without it.
- With external memory we often assume a *limited bandwidth* connection to the *external* memory. This may not always be the case with attention. In some sense the default is to get *all* the information.
- External memory is often assumed to be *variable-sized* and *unbounded*. This could be the case with attention mechanisms also, but it is not uncommon to have fixed size inputs either.
- Difference in where data is coming from, how we control the content. Memory more flexible in that we control the content, may allow for other kinds of addressing?
- With attention, maybe less critical exactly which cells we retrieve? E.g. if we are doing attention over convnet feature layer, the neighbouring cells probably contain a lot of the same information...

- The reason that location-based attention works is not because we know what we *wrote*, rather that we have read it before (though possibly a downsampled version as in our example with visual attention).

5 Bibliography

References

[1] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

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

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[4] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.