WILDML
Artificial Intelligence, Deep Learning, and NLP

JANUARY 3, 2016 BY DENNY BRITZ

Attention and Memory in Deep Learning and NLP

A recent trend in Deep Learning are Attention Mechanisms. In an interview, Ilya Sutskever, now the research director of OpenAI, mentioned that Attention Mechanisms are one of the most exciting advancements, and that they are here to stay. That sounds exciting. But what are Attention Mechanisms?

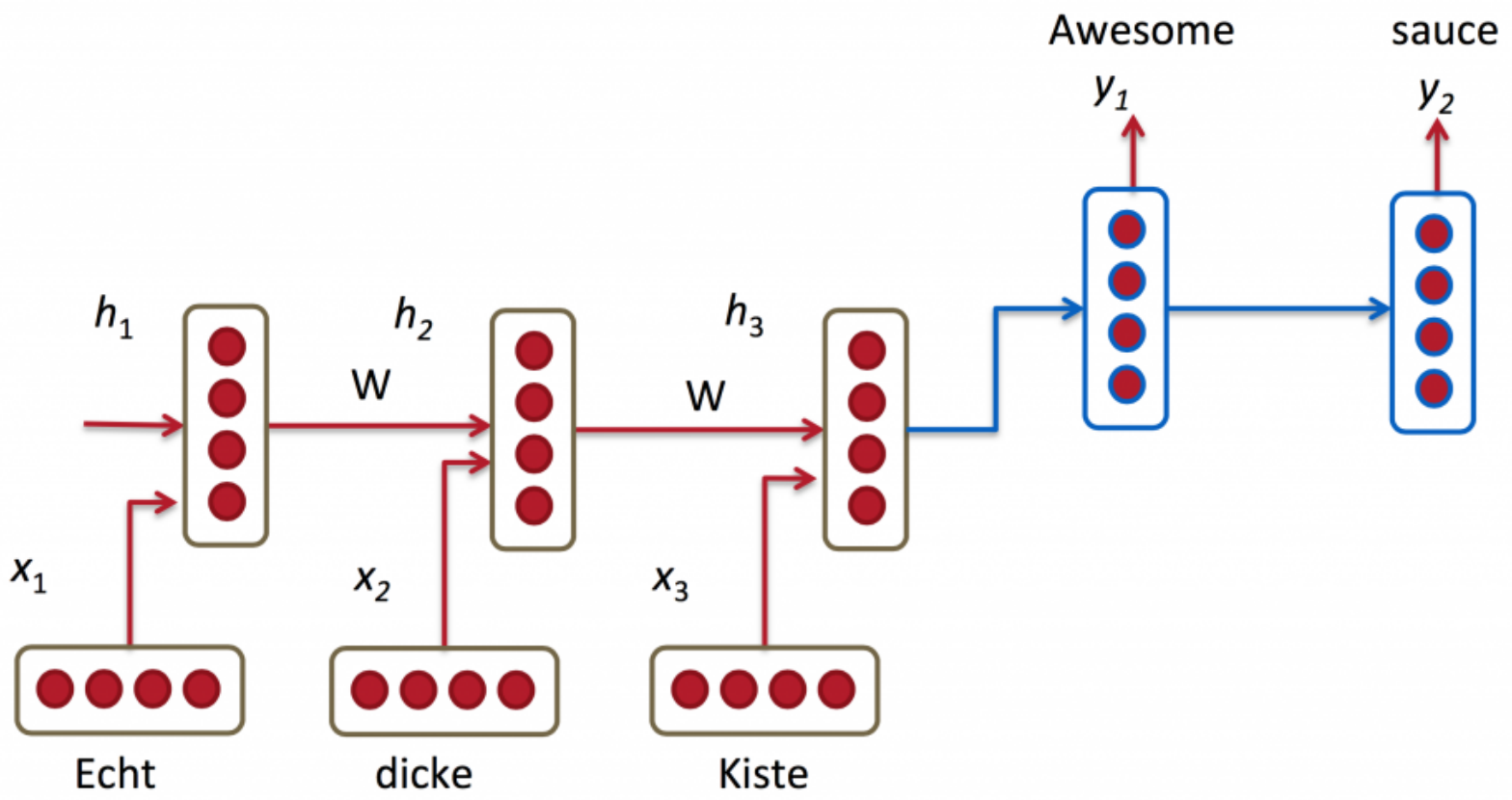
Attention Mechanisms in Neural Networks are (very) loosely based on the visual attention mechanism found in humans. Human visual attention is well-studied and while there exist different models, all of them essentially come down to being able to focus on a certain region of an image with "high resolution" while perceiving the surrounding image in "low resolution", and then adjusting the focal point over time.

Attention in Neural Networks has a long history, particularly in image recognition. Examples include Learning to combine foveal glimpses with a third-order Boltzmann machine or Learning where to Attend with Deep Architectures for Image Tracking. But only recently have attention mechanisms made their way into recurrent neural networks architectures that are typically used in NLP (and increasingly also in vision). That's what we'll focus on in this post.

What problem does Attention solve?

To understand what attention can do for us, let's use Neural Machine Translation (NMT) as an example. Traditional Machine Translation systems typically rely on sophisticated feature engineering based on the statistical properties of text. In short, these systems are complex, and a lot of engineering effort goes into building them. Neural Machine Translation systems work a bit differently. In NMT, we map the meaning of a sentence into a fixed-length vector representation and then generate a translation based on that vector. By not relying on things like n-gram counts and instead trying to capture the higher-level meaning of a text, NMT systems generalize to new sentences better than many other approaches. Perhaps more importantly, NTM systems are much easier to build and train, and they don't require any manual feature engineering. In fact, a simple implementation in Tensorflow is no more than a few hundred lines of code.

Most NMT systems work by *encoding* the source sentence (e.g. a German sentence) into a vector using a Recurrent Neural Network, and then *decoding* an English sentence based on that vector, also using a RNN.



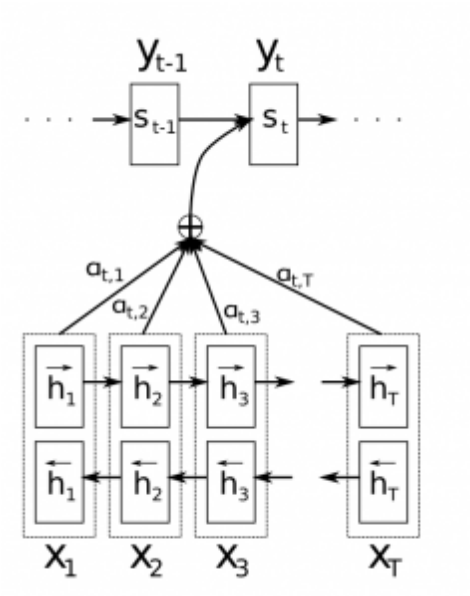
In the picture above, “Echt”, “Dicke” and “Kiste” words are fed into an encoder, and after a special signal (not shown) the decoder starts producing a translated sentence. The decoder keeps generating words until a special end of sentence token is produced. Here, the h vectors represent the internal state of the encoder.

If you look closely, you can see that the decoder is supposed to generate a translation solely based on the last hidden state (h_3 above) from the encoder. This $h3$ vector must encode everything we need to know about the source sentence. It must fully capture its meaning. In more technical terms, that vector is a sentence *embedding*. In fact, if you plot the embeddings of different sentences in a low dimensional space using PCA or t-SNE for dimensionality reduction, you can see that semantically similar phrases end up close to each other. That’s pretty amazing.

Still, it seems somewhat unreasonable to assume that we can encode all information about a potentially very long sentence into a single vector and then have the decoder produce a good translation based on only that. Let’s say your source sentence is 50 words long. The first word of the English translation is probably highly correlated with the first word of the source sentence. But that means decoder has to consider information from 50 steps ago, and that information needs to be somehow encoded in the vector. Recurrent Neural Networks are known to have problems dealing with such long-range dependencies. In theory, architectures like LSTMs should be able to deal with this, but in practice long-range dependencies are still problematic. For example, researchers have found that reversing the source sequence (feeding it backwards into the encoder) produces significantly better results because it shortens the path from the decoder to the relevant parts of the encoder. Similarly, feeding an input sequence twice also seems to help a network to better memorize things.

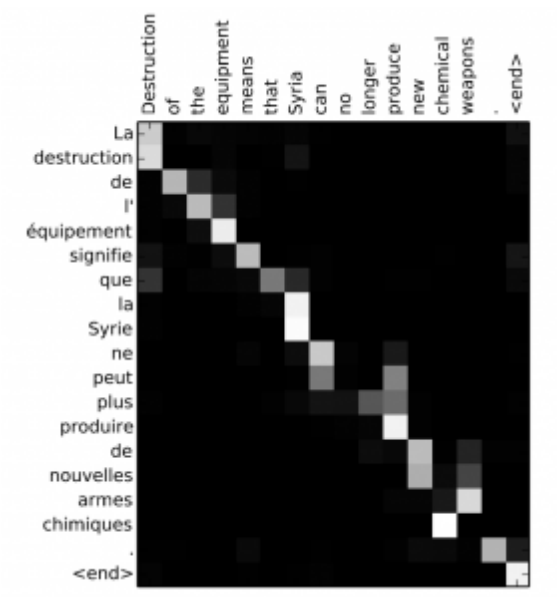
I consider the approach of reversing a sentence a “hack”. It makes things work better in practice, but it’s not a principled solution. Most translation benchmarks are done on languages like French and German, which are quite similar to English (even Chinese word order is quite similar to English). But there are languages (like Japanese) where the last word of a sentence could be highly predictive of the first word in an English translation. In that case, reversing the input would make things worse. So, what’s an alternative? Attention Mechanisms.

With an attention mechanism we no longer try encode the full source sentence into a fixed-length vector. Rather, we allow the decoder to “attend” to different parts of the source sentence at each step of the output generation. Importantly, we let the model **learn** what to attend to based on the input sentence and what it has produced so far. So, in languages that are pretty well aligned (like English and German) the decoder would probably choose to attend to things sequentially. Attending to the first word when producing the first English word, and so on. That’s what was done in Neural Machine Translation by Jointly Learning to Align and Translate and look as follows:



Here, The y's are our translated words produced by the decoder, and the x's are our source sentence words. The above illustration uses a bidirectional recurrent network, but that's not important and you can just ignore the inverse direction. The important part is that each decoder output word y_t now depends on a **weighted combination of all the input states**, not just the last state. The a's are weights that define in how much of each input state should be considered for each output. So, if $a_{3,2}$ is a large number, this would mean that the decoder pays a lot of attention to the second state in the source sentence while producing the third word of the target sentence. The $a's$ are typically normalized to sum to 1 (so they are a distribution over the input states).

A big advantage of attention is that it gives us the ability to interpret and visualize what the model is doing. For example, by visualizing the attention weight matrix a when a sentence is translated, we can understand how the model is translating:



Here we see that while translating from French to English, the network attends sequentially to each input state, but sometimes it attends to two words at time while producing an output, as in translation "la Syrie" to "Syria" for example.

The Cost of Attention

If we look a bit more look closely at the equation for attention we can see that attention comes at a cost. We need to calculate an attention value for each combination of input and output word. If you have a 50-word input sequence and generate a 50-word output sequence that would be 2500 attention values. That's not too bad, but if you do character-level computations and deal with sequences consisting of hundreds of tokens the above attention mechanisms can become prohibitively expensive.

Actually, that's quite counterintuitive. Human attention is something that's supposed to **save** computational resources. By focusing on one thing, we can neglect many other things. But that's not really what we're doing in the above model. We're essentially looking at everything in detail before deciding what to focus on. Intuitively that's equivalent outputting a translated word, and then going back through *all* of your internal memory of the text in order to decide which word to produce next. That seems like a waste, and not at all what humans are doing. In fact, it's more akin to memory access, not attention, which in my opinion is somewhat of a misnomer (more on that below). Still, that hasn't stopped attention

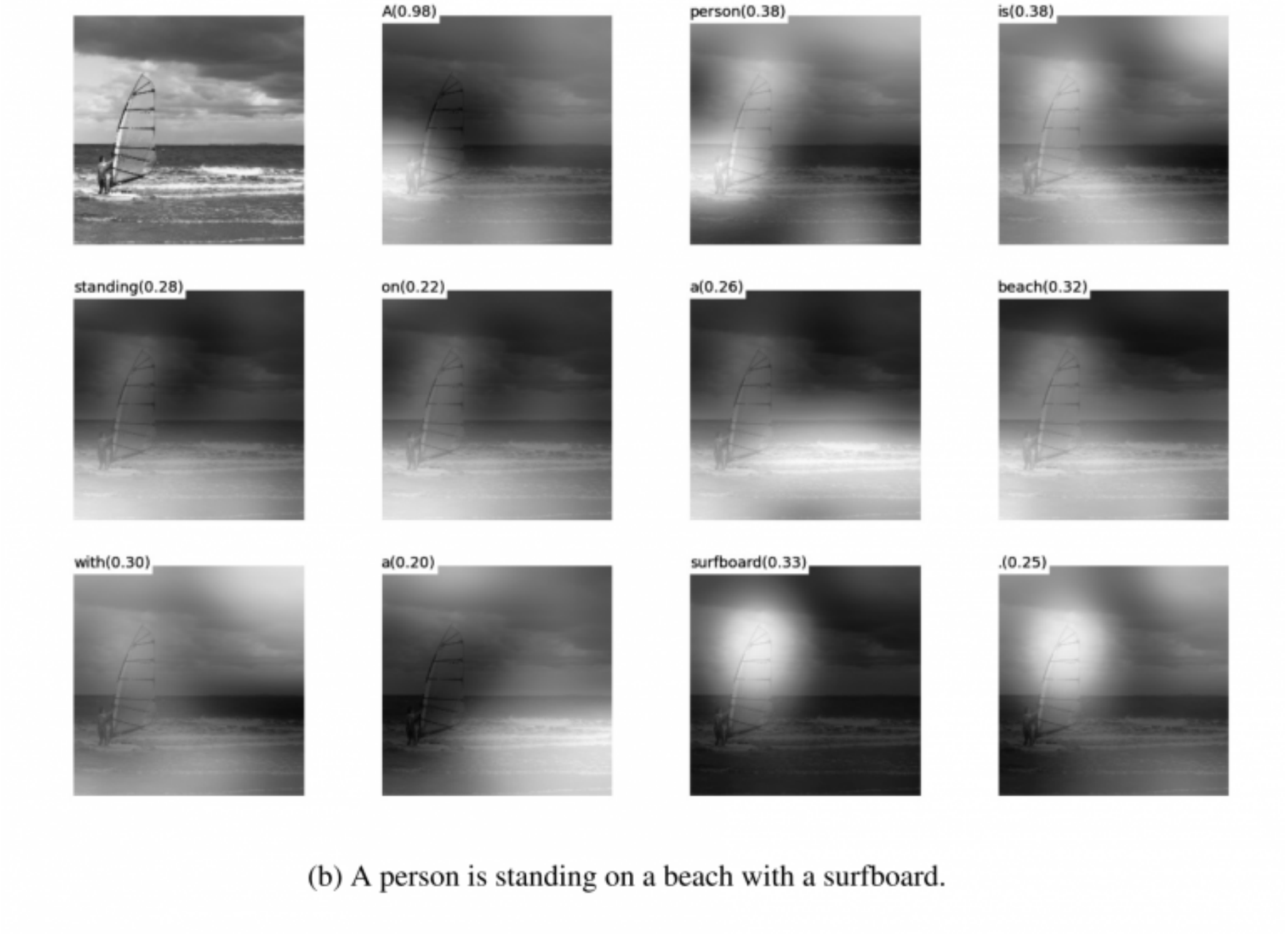
mechanisms from becoming quite popular and performing well on many tasks.

An alternative approach to attention is to use Reinforcement Learning to predict an approximate location to focus to. That sounds a lot more like human attention, and that's what's done in Recurrent Models of Visual Attention.

Attention beyond Machine Translation

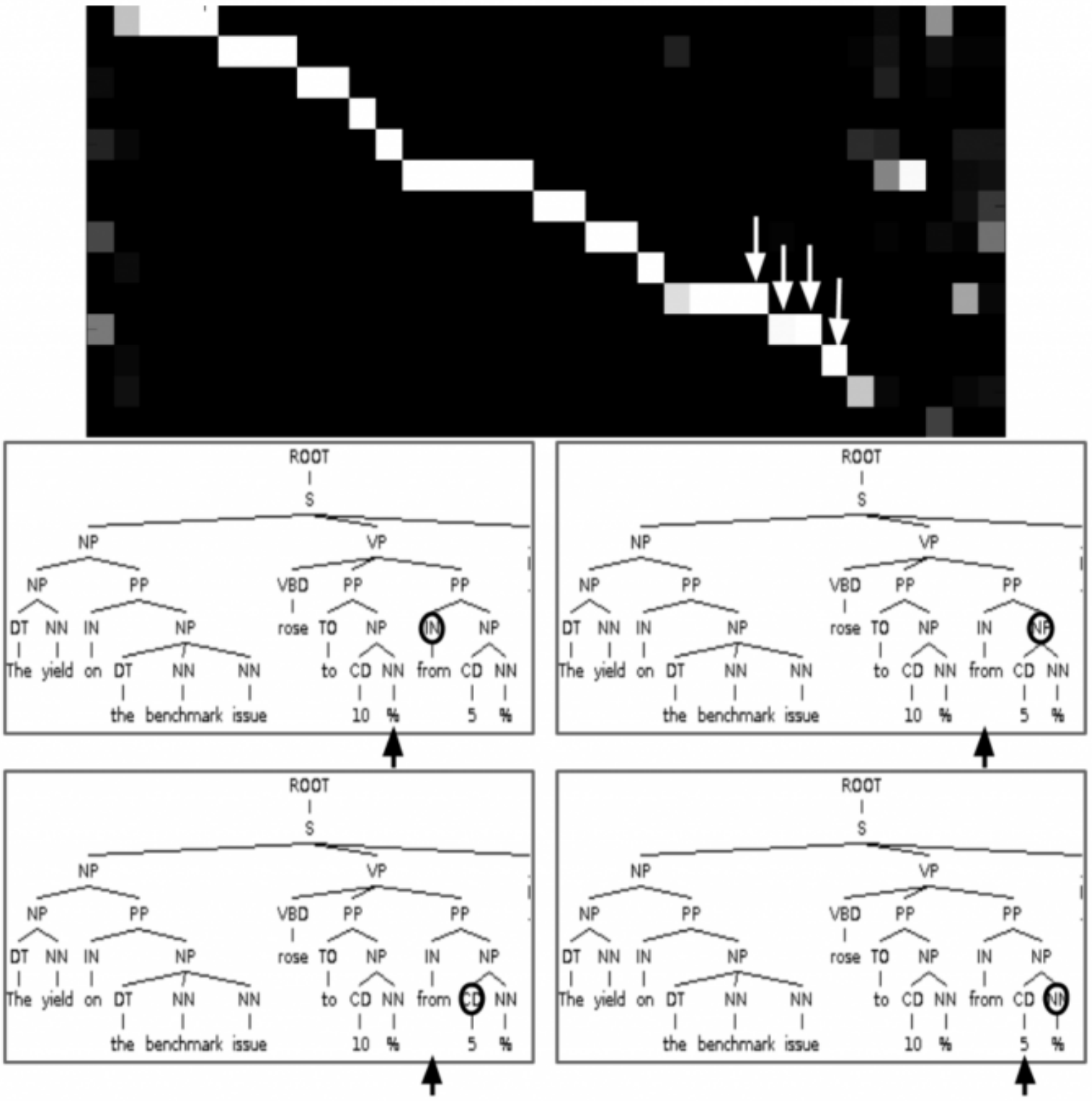
So far we've looked at attention applied to Machine Translation. But the same attention mechanism from above can be applied to any recurrent model. So let's look at a few more examples.

In Show, Attend and Tell the authors apply attention mechanisms to the problem of generating image descriptions. They use a Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanisms to generate a description. By visualizing the attention weights (just like in the translation example), we interpret what the model is looking at while generating a word:

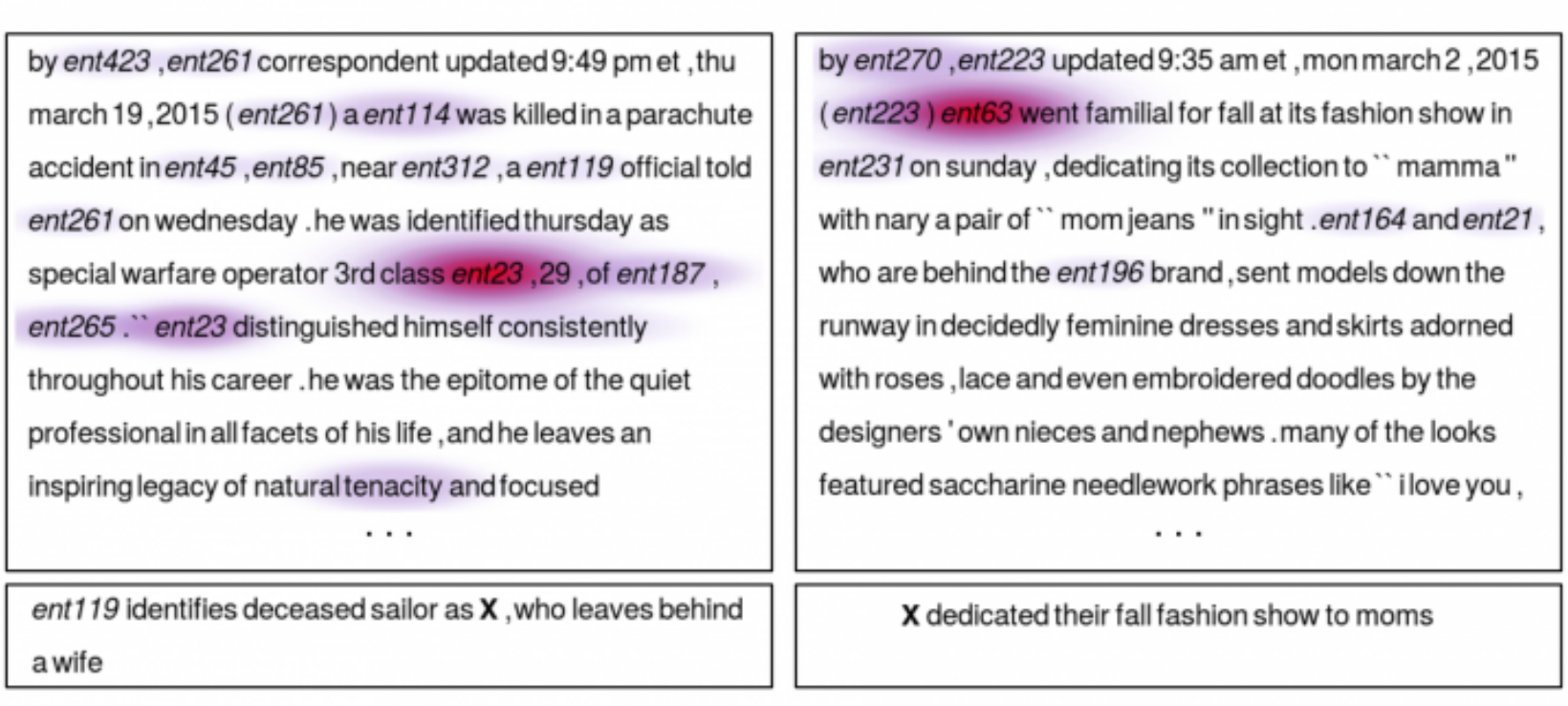


(b) A person is standing on a beach with a surfboard.

In Grammar as a Foreign Language, the authors use a Recurrent Neural Network with attention mechanisk to generate sentence parse trees. The visualized attention matrix gives insight into how the network generates those trees:



In Teaching Machines to Read and Comprehend, the authors use a RNN to read a text, read a (synthetically generated) question, and then produce an answer. By visualizing the attention matrix we can see where the networks "looks" while it tries to find the answer to the question:



Attention = (Fuzzy) Memory?

The basic problem that the attention mechanism solves is that it allows the network to refer back to the input sequence, instead of forcing it to encode all information into one fixed-length vector. As I mentioned above, I think that attention is somewhat of a misnomer. Interpreted another way, the attention mechanism is simply giving the network access to its internal memory, which is the hidden state of the encoder. In this interpretation, instead of choosing what to "attend" to, the network chooses what to retrieve from memory. Unlike typical memory, the memory access mechanism here is soft, which means that the network retrieves a weighted combination of all memory locations, not a value from a single discrete location. Making the memory access soft has the benefit that we can easily train the network end-to-end using backpropagation (though there have been non-fuzzy approaches where the gradients are calculated using sampling methods instead of backpropagation).

Memory Mechanisms themselves have a much longer history. The hidden state of a standard Recurrent Neural Network is itself a type of internal memory. RNNs suffer from the vanishing gradient problem that prevents them from learning long-range dependencies. LSTMs improved upon this by using a gating mechanism that allows for explicit memory deletes and updates.

The trend towards more complex memory structures is now continuing. End-to-End Memory Networks allow the network to read same input sequence multiple times before making an output, updating the memory contents at each step. For example, answering a question by making multiple reasoning steps over an input story. However, when the networks parameter weights are tied in a certain way, the memory mechanism inEnd-to-End Memory Networks identical to the attention mechanism presented here, only that it makes multiple hops over the memory (because it tries to integrate information from multiple sentences).

Neural Turing Machines use a similar form of memory mechanism, but with a more sophisticated type of addressing that using both content-based (like here) and location-based addressing, allowing the network to learn addressing pattern to execute simple computer programs, like sorting algorithms.

It's likely that in the future we will see a clearer distinction between memory and attention mechanisms, perhaps along the lines of Reinforcement Learning Neural Turing Machines, which try to learn access patterns to deal with external interfaces.

DEEP LEARNING, LANGUAGE MODELING, MEMORY, NEURAL NETWORKS, NLP