CS 182/282A: Designing, Visualizing and Understanding Deep Neural Networks **Spring 2019**

Section 7: Attention Mechanisms, Transformers and RNN Decoding

Notes by: Forrest Huang

7.1 Course Logistics

- You should have met with the GSI to check-in with your project last week. Please complete the milestone as agreed with your GSI during the check-in (typically gathering all required data) this week.

7.2 Attention Mechanisms and Transformer Network

For many NLP and visual tasks we train our deep models on, features appear on the input text/visual data often contributes unevenly to the output task. For example, in a translation task, not the entirety of the input sentence will be useful (and may even be confusing) for the model to generate a certain output word, or not the entirety of the image contributes to a certain sentence generated in the caption. While some RNN architectures we previously covered possess the capability to maintain a memory of the previous inputs/outputs, to compute output and to modify the memory accordingly, these memory states need to encompass information of many previous states, which can be difficult especially when performing tasks with long-term dependencies.

As such, researchers developed attention mechanisms to improve the network's capability of orienting perception onto parts of the data, and to allow random access to the memory of processing previous inputs. In the context of RNNs, attention mechanisms allow networks to not only utilize the current hidden state, but also the hidden states of the network computed in previous time steps as shown in Figure 7.1. In this Section, we will explore multiple attention mechanisms that ultimately lead to the attention-only Transformer network [4].

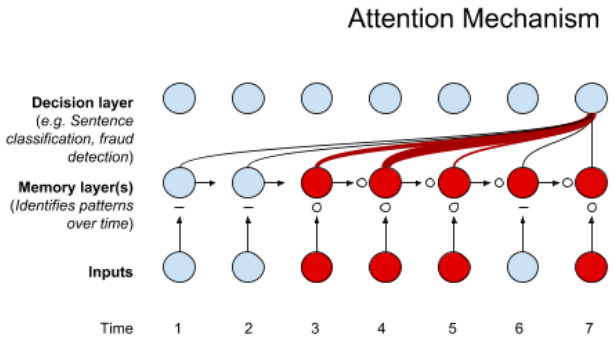

Figure 7.1: Attention Mechanism Illustrated

7.2.1 Luong Attention

The Luong attention [2] is one of the commonly used attention mechanism in Neural Machine Translation, which is trained on the task of translating text from source to target language. At each time-step of decoding, the Luong Attention computes a set of alignment(attention) weights a_t based on alignment scores and use this to augment the computation of output probabilities in the translation task.

As shown in Figure 7.2, we have our encoding states (RNN hidden states when passed a source language into the model first) h_s. At decoding time, we have the original hidden states h_t initially computed with the previous output token and hidden states.

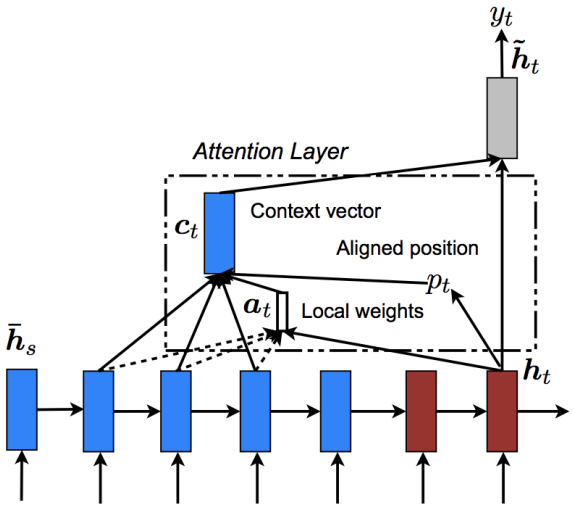

Figure 7.2: Luong Attention

With h_t and each h_s we compute an alignment score (Note: The original paper proposed multiple alignment scores, the general one they propose is $h_t^T W_a h_s$, another simple 'location' based attention is just computed using the decoding hidden state at time t: $W_a h_t$), we can take a Softmax function to obtain attention weights from the alignment scores.

With these attention weights, we compute a context vector c_t, which is a weighted sum of h_s. As we get c_t, we compute a transformation $\tilde{h}_t = \tanh(W_f[h_t; c_t])$ applied on the concatenation of the context vector and the original hidden state. This is then used to compute the final output.

The motivation for this kind of machine translation system was to treat the encoder (the blue part) as like a memory which we can access later during the decoding process (colored red). Most of the early neural machine translation systems ran the encoder with the input sentence to get a hidden state, and then that hidden state was the input to the decoder which needed to generate the resulting sentence. With this model, we are able to use the other hidden state outputs from the encoder, not just the last one.

7.2.2 Scaled Dot-Product/Key-Query-Value Attention in Transformer Networks

Scaled Dot-Product attention is an attention mechanism introduced in the Transformer architecture which undergoes similar procedures as the Luong attention. We illustrate the self-attention variant of Scaled Dot-

Product Attention. The first step of the attention is to compute Q, K, V using different transformations from the original input embedding as shown in Figure 7.3.

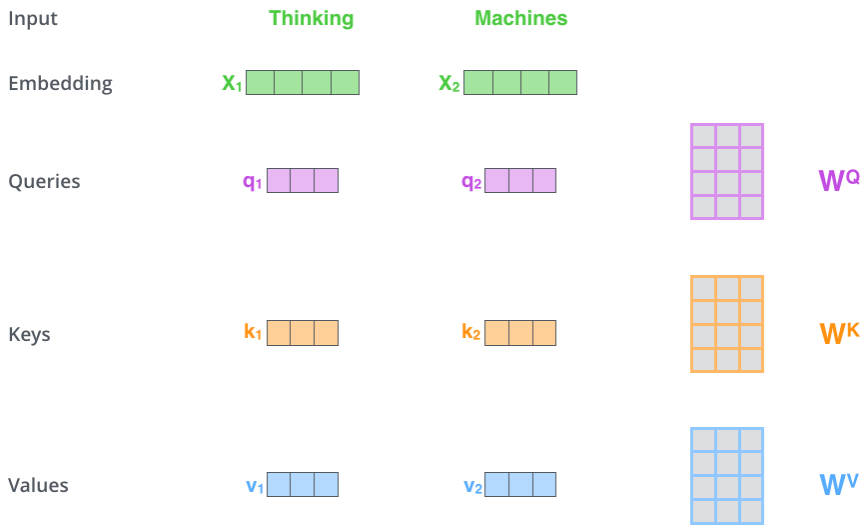

Figure 7.3: Computing K, Q, V from input embeddings in a Transformer Network. Figure taken from [3].

Then, using Q and K, we can compute a dot product as the 'score' of K for Q as shown in Figure 7.4. Intuitively, Q is the querying term that you would like to find it's relations for each corresponding K and V pairs (key-value) pairs, can be computed using the key. Note that this dot product is computed across various time steps by matrix multiplication. So we get a score for each K for each Q. We then use a Softmax function to get our attention weights.

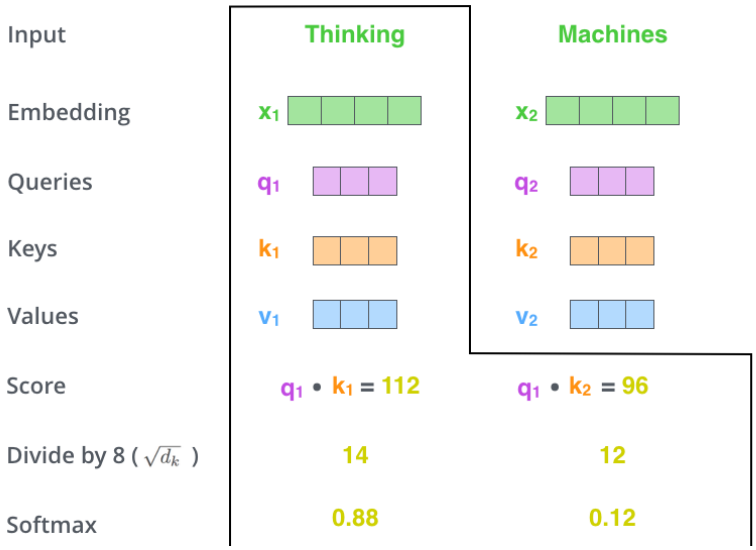

Figure 7.4: Computing Attention Scores from K, Q, V. Figure taken from [3].

Finally, using these weights, we can compute our weighted sum by multiplying the weights with the values. Comparing to the Luong attention, query is analogous to the original h_t, key and query are analogous to the original h_s. The general architecture is shown in Figure 7.5.

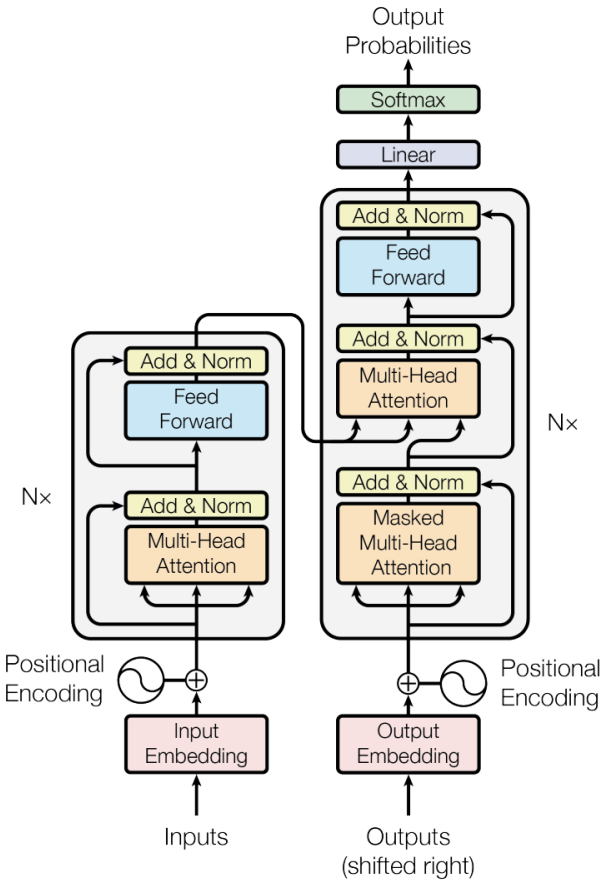

Figure 7.5: Full Transformer Architecture

While we described the self-attention variant of a single attention head above, this can be generalized to cross-attention where query is the transformed output word embedding, and key and value is the transformed input word embedding. Moreover, the transformer network also concatenates multiple sets of these weights as multiple *heads* in the attention mechanism. This forms the most important building-block of the Transformer Architecture. For a more detailed discussion of the Transformer Network, please refer to the original paper [4] and the Illustrated Transformer [3].

7.2.3 Tutorials on Attention Networks and Eager Execution on Tensorflow

Here is a tutorial that we can run through in class for implementing attention in RNNs with `tf.eager` and `keras`: `https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/nmt_with_attention/nmt_with_attention.ipynb#scrollTo=ddefjBMa3jF0`

7.2.4 Interpretability of Attention Mechanisms

One big advantage of the attention mechanism is that the attention weights are potentially human-interpretable, and we can determine the importance of various parts of the data perceived by the networks.

We show several examples of these attention weights in practical tasks. Figure 7.6 shows the attention maps of an image-captioning model attending to relevant parts of the image during the decoding of the corresponding word in the caption. Figure 7.7 shows that networks in a self-driving task attends to cars and pedestrians on the road. Figure 7.8 shows the corresponding words between two languages during the translation of a German sentence to English.

Figure 7.6: Attention Weights on a Captioning Network [5].

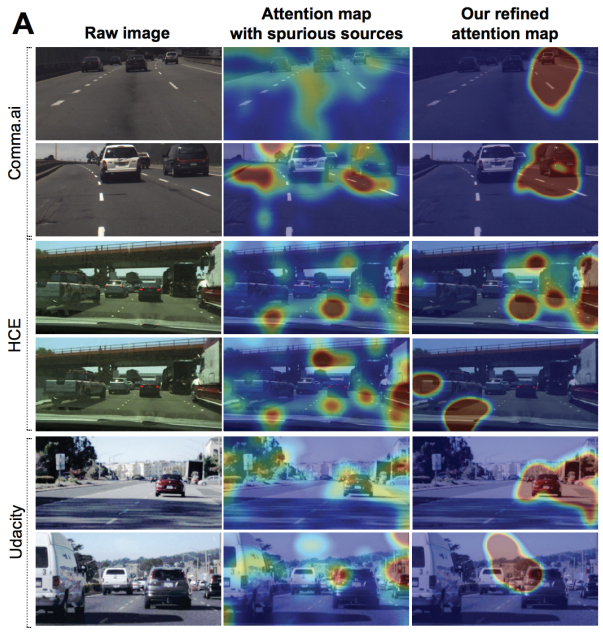

Figure 7.7: Attention Weights for Autonomous Driving Tasks [1].

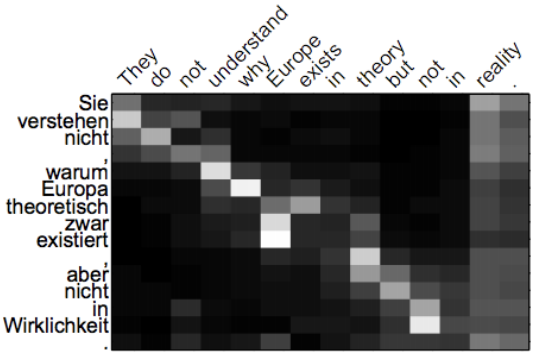

Figure 7.8: Attention Weights of each source/target pairs during translation from German to English [2]

7.2.5 Notes on Hard Attention

We described multiple attention method above which are all instances of *soft* attention. Researchers have also developed *hard* attention mechanisms historically, which instead of computing a weighted average of the values of interest, simply sample the entirety one of the values while dropping the others. *Hard* attention is non-differentiable and is typically trained using reinforcement learning, for example using vanilla policy gradients (called "REINFORCE" in the literature).

7.3 RNNs in practice

In many cases, an RNN will be used to predict the next outputs in a sequence, and using these outputs as the inputs at the next step. When training an RNN for these tasks, we typically don't take perform this step during training, but instead feed the ground truth inputs at every time steps into the network since that is available during training time. *(Notes to GSIs: Draw RNNs during training on board)*

This is however, not possible during inference (e.g. generation of text), since we don't have ground-truth data available at every time-steps. As such, we need to sample an output at each time-step from the RNN, and use it to feed into the network as the input in the next step. For example, for a text generation problem with the RNN outputing the probability of each tokens at each time-step, the simplest strategy is to take the token of the highest probability. This is called the *Greedy* strategy, and what you implemented in the homeworks for image captioning.

This, however, might result in sentences far from the optimal generation. Thus, another strategy commonly use is beam-search, with the joint-probability of the tokens generated so far as the heuristic. With beam-search, we retain the sequences with top k (beam width) probabilities at each-time steps while continuously pruning the rest. In practice, k is usually set to 5-10 for machine translation, and may differ for other tasks.

References

[1] Jinkyu Kim and John Canny. "Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.

[2] Minh-Thang Luong, Hieu Pham, and Chris Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Conference on Empirical Methods for Natural Language Processing (EMNLP)*. 2015.

[3] *The Illustrated Transformer*. `http://jalammar.github.io/illustrated-transformer/`. Accessed: 2019-03-17.

[4] Ashish Vaswani et al. "Attention Is All You Need". In: *Neural Information Processing Systems (NIPS)*. 2017.

[5] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning (ICML)*. 2015.