Notes on Attention Models for Neural Networks

David Meyer

dmm613@gmail.com

Last update: June 9, 2022

1 Introduction

Statistical language models are probability distributions which have many applications, including speech recognition, machine translation, part-of-speech tagging, parsing, handwriting recognition, and information retrieval. Given a sequence of length n, a language model assigns a probability $p(x_1, \ldots, x_n)$ to the whole sequence. Here x_i is the i^{th} word or symbol in the sequence. A model that computes either $p(x_1, \ldots, x_m)$ or $p(x_n|x_1, x_2, \cdots, x_{n-1})$ is called a language model. Using the chain rule for conditional probabilities [9], we can see that

$$\begin{aligned}
p(x_1, x_2, \ldots, x_n) &= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_n|x_1, \ldots, x_{n-1}) && (1) \\
&= \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_{i-1}) && (2)
\end{aligned}$$

Observe that since our language model is sequential[1], the value of n in Equation (2) is in some sense how far back in time we need to look to predict the next word or symbol (I'll just say word from here on out). A language model that looks at the n previous words in a sequence is called an *n-gram*, and is defined using the chain rule as shown in Equation (1). Interesting n-gram models include

$$\begin{aligned}
p(x_1, x_2, \ldots, x_n) &\approx \prod_{i=1}^{n} p(x_i) && \text{\# the unigram model} \\
p(x_i|x_1, x_2, \ldots, x_{i-1}) &\approx p(x_i|x_{i-1}) && \text{\# the bigram model}
\end{aligned}$$

The unigram model assumes we can predict the i^{th} word, x_i, independently of $x_1, x_2, \ldots, x_{i-1}$ (the words that came before it). The bigram model assumes that the future is independent of the past given the present, i.e., the Markov assumption.[2]

Note that we can estimate the n-gram probabilities in a straightforward way. Consider the bigram case. Here the Maximum Likelihood Estimate (MLE) is simply is

$$p(x_i|x_{i-1}) \quad = \quad \frac{\text{count}(x_{i-i}, x_i)}{\text{count}(x_{i-1})} \qquad \text{\# bigram MLE estimate}$$

[1] x_i occurs before x_j $\forall i, j$ $1 \leq i < j$

[2] Sometimes called a first-order Markov assumption.

Here we are simply counting how many times x_i appeared in the context x_{i-1} and normalizing by all observations of x_{i-1}.

There are a few problems with *n-gram* models. First, for any reasonable n the *n-gram* is likely to be an insufficient model of the language due to the long-range dependencies typically found in natural language. This forces larger values of n. The second is that *n-grams* suffer from sparse data distributions; as n grows the space of all possible sequences grows rapidly and the probability of most sequences or next words is tends towards zero. In addition, the number of possible parameters grows exponentially with n. As a result, there will be never enough of the training data to estimate parameters of high-order *n-gram* models. That said, there are many cases in which we can get away with *n-grams*.

2 Encoder-Decoder Architecture

Recurrent Neural Networks (RNNs) take a different approach to machine translation. Rather than keeping counts, a RNN summarizes what it has seen previous steps in its current hidden state; you can think of the hidden state h_t as the memory of the network. Thus h_t captures information about what happened in the previous $t-1$ time steps. This means that the RNN must be able to summarize all the information from the $t-1$ previous steps in a *fixed length* vector (h_t). This property will turn out to be one of the limitations of RNNs that motivated the development of attention mechanisms. Finally, note that the output distribution at time t, y_t, s calculated solely based on h_t.

As an aside, while a vanilla RNNs can in principle capture dependencies over some period of time in past, in practice they have trouble with long-range dependencies. As a result, these networks are typically outfitted with some kind of memory, such as in the Long Short-Term Memory (LSTM) or the gated units described in [3]. Note here: Neural Turing Machines [5] and Differentiable Neural Computers [6] use a more explicit and function memory; however, as pointed out in [4] the hidden state matrix is just a memory matrix of the form $[h_{t-L} \dots h_{t-1}] \in \mathbb{R}^{n \times L}$, where n is the output dimension of the RNN cells and L is a sliding window.

Recent state of the art performance on translation tasks has been achieved using the Encoder-Decoder framework as described in Sutskever et al. [8] and Cho et al. [3]. The following description of the Encoder-Decoder framework follows the notation found in [1].

In the Encoder-Decoder framework, an encoder reads the input sentence, which is a sequence of vectors $\mathbf{x} = (x_1, \cdots, x_{T_x})$. The input sequence is taken from a vocabulary V_x, with $|V_x| = K_x$. Here each x_i is a column vector of length K_x such that $x_i \in \mathbb{R}^{K_x \times 1}$ (usually written $x_i \in \mathbb{R}^{K_x}$) is the one-hot encoding for the word at position i. That is, the i^{th} input word looks like

$$x_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{K_x i} \end{bmatrix}$$

so that

$$\mathbf{x} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1T_x} \\ x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2T_x} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{K_x1} & x_{K_x2} & \cdots & x_{K_xi} & \cdots & x_{K_xT_x} \end{bmatrix}$$

The encoder RNN calculates its hidden state[3] at time t, $h_t \in \mathbb{R}^n$ as

$$h_t = f(x_t, h_{t-1})$$

and also produces a context vector c from the hidden states: $c = q(\{h_1, h_2, \ldots, h_{T_x}\})$. As an example, [8] used an LSTM for f and defined $q(\{h_1, h_2, \ldots, h_T\}) = h_T$.

The decoder is usually trained to predict the next word y_t given the context vector c and all the previously predicted words $\{y_1, \ldots, y_{t-1}\}$; the decoder defines a probability over $\mathbf{y}$ (the translation) by decomposing it into its joint probabilities, conditioned on the previously translated words and the context vector.

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, y_2, \ldots, y_{T_y}\}, c)$$

where $\mathbf{y} = (y_1, \ldots, y_{T_y})$. With an RNN, the conditional probability of each translation is modeled as

$$p(y_t | \{y_1, y_2, \ldots, y_{T_y}\}, c) = g(y_{t-1}, s_t, c)$$

where g is a nonlinear, potentially multi-layered, function that outputs the probability of y_t, and s_t is the hidden state of the RNN.

Recurrent Neural Networks (RNNs) take a different approach to machine translation. From a probabilistic perspective, the machine translation task is to find a target sentence $\mathbf{y}$ that maximizes the conditional probability of $\mathbf{y}$ given a source sentence $\mathbf{x}$, that is, $\operatorname{argmax} p(\mathbf{y}|\mathbf{x})$ (where the argmax is over $\mathbf{y}$). In neural machine translation, a parameterized model is fit which maximizes the conditional probability of sentence pairs using a parallel training corpus.

[3] In general the hidden state of the encoder is called h_t and the hidden state of the decoder is called s_t.

2.1 Preliminaries

We consider an input sequence over a vocabulary V_x, with $|V_x| = K_x$, where the input is a sequence of vectors $\mathbf{x} = (x_1, \cdots, x_{T_x})$. Here each x_i is a column vector of length K_x such that $x_i \in \mathbb{R}^{K_x \times 1}$ (usually written $x_i \in \mathbb{R}^{K_x}$) is the one-hot encoding for the word at position i. That is, the i^{th} input word looks like

$$x_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{K_x i} \end{bmatrix}$$

so that

$$\mathbf{x} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1T_x} \\ x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2T_x} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{K_x 1} & x_{K_x 2} & \cdots & x_{K_x i} & \cdots & x_{K_x T_x} \end{bmatrix}$$

3 RNN Encode-Decoder Architecture

4 Appendix A: The Evolution of Attention Models

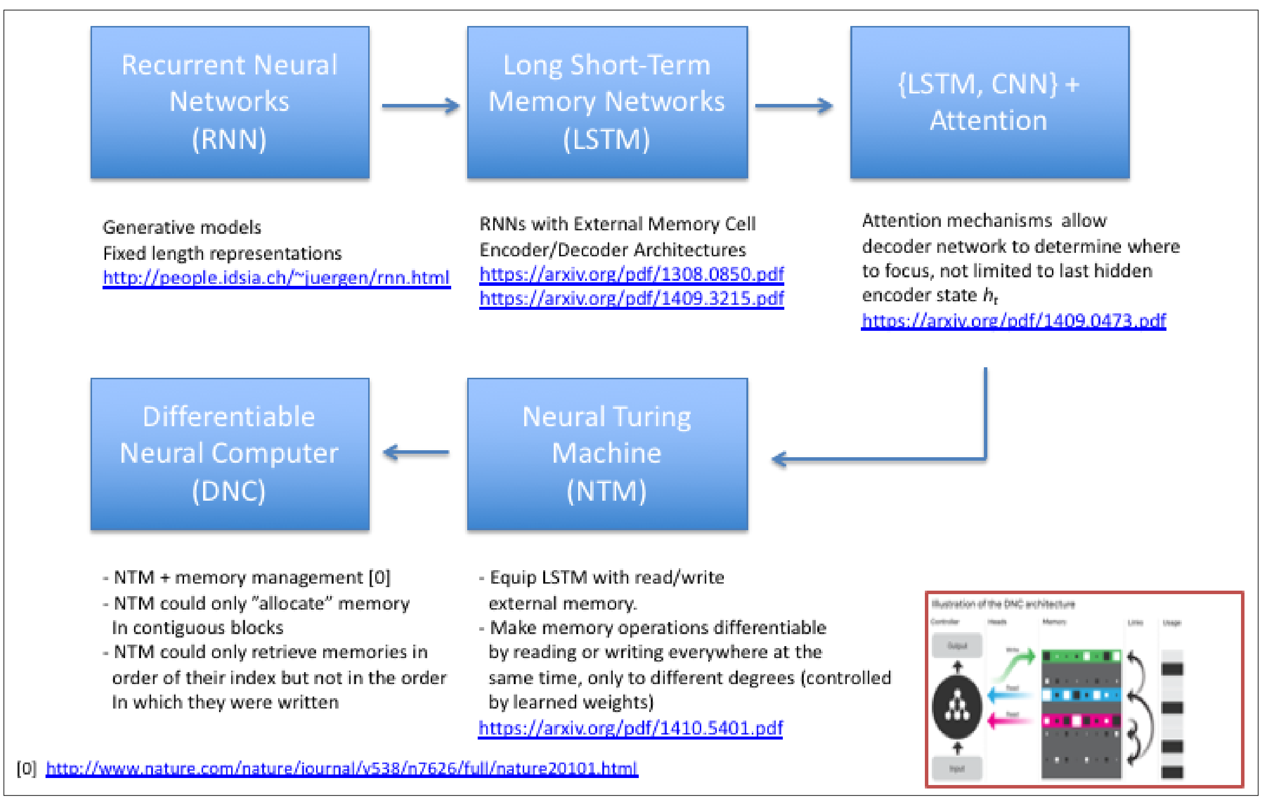

Figure 1: The Evolution of Attention Models

Acknowledgements

Thanks to Stephen Wolff for pointing out Shannon's work in this area [2, 7].

LaTeX Source

`https://www.overleaf.com/read/gshqdkhqdnmm`

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. September 2014.

[2] C.E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, July 1948.

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. June 2014.

[4] Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. February 2017.

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. October 2014.

[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwiska, Sergio Gmez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrinech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016.

[7] Mike Passwall. Notes on Computer-Generated Text. `http://mike.passwall.com/class/public/ethernet/shannon1/`. Accessed: Wed Jun 8 07:36:00 PDT 2022.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[9] Lisa Yan. Conditional Probability. `https://web.stanford.edu/class/archive/cs/cs109/cs109.1208/lectureNotes/LN04_cond_bayes.pdf`, April 2020. Accessed: Thu Jun 2 10:31:13 PDT 2022.