ifpdf[2019/10/25 v3.4 ifpdf legacy package. Use iftex instead.] iftex

Modern Topics on Statistical Learning Theory Spring 2023

Lecture 13 — Transformer

Prof. Qi Lei *Scribe: Seok Hoan Choi, Lev Paciorkowski, Miao Zhang*

1 Announcements

- Please fill out course evaluations as soon as possible.
- Project final score submission due on Apr 28. Project final report due on May 5.

2 Introduction

Images can be considered as **static** information. However, for NLP tasks, it becomes important to consider the ordering of sentences. Accordingly, we need different architectures to model this sequential information.

Types of Problems:

1. 'one-to-one': Inputs and outputs are of **fixed** dimension. Many of the classic image-related tasks fall into this category.
2. 'many-to-one': Arbitrary input dimension, but fixed output dimension. Example: sentiment analysis of text.
3. 'many-to-many': Inputs and outputs are **both** of arbitrary dimension. Example: machine translation.
4. 'one-to-many': Fixed input dimension, but arbitrary output dimension. Example: image captioning.

This lecture will focus on the architectures needed to work on problem types 2-4.

3 Recurrent Neural Network (RNN)

RNNs are specifically designed for many-to-many or many-to-one type problems. Supposing we have our input as:

$$x_1, x_2, ..., x_T$$

Where each x_i represents a token. For example, for the phrase 'she is eating an apple', we could have x_1 = 'she'; x_2 = 'is'; x_3 = 'eating'; x_4 = 'an'; x_5 = 'apple'.

The hidden state of an RNN at index t is then defined as:

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

Where x_t is the input, h_{t-1} is the previous hidden state, and the function f_W is the same for each hidden state. As an example, we could have the following functional form for an RNN:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Then the predictions are made as $y_t = W_{hy}h_t$.

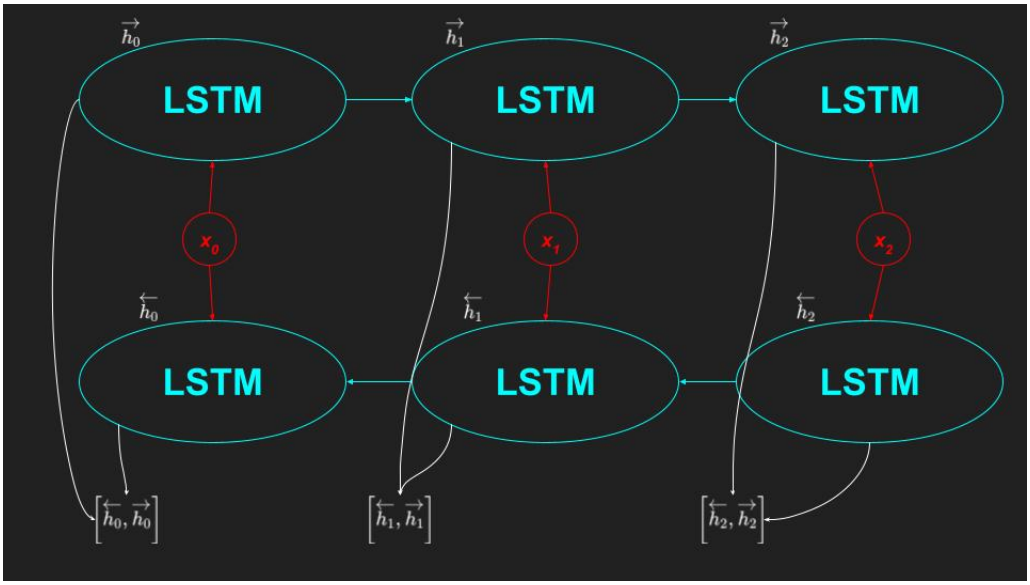
RNNs have demonstrated their utility in a variety of applications, although there is still a potential problem due to gradient exploding and vanishing. In particular, the vanishing gradient problem is harder to solve here. Exploding gradients can be adequately solved by gradient clipping.

The long short-term memory (LSTM) architecture was developed as an improvement on RNNs, with the intention to mitigate the vanishing gradient problem. The key addition of an LSTM is the so-called 'forget gate'. This has successfully reduced the gradient exploding/vanishing problems, but has also introduced another problem known as 'catastrophic forgetting'. For example, in a long sample of text, the model might forget context information from the beginning, and this could lead to worse output.

An additional problem is that these architectures can only model dependency on previous words. This can be unsuitable for text completion tasks. For example, if the model sees 'she is ____', its output y_3 might be 'happy', even though the full sentence might finish 'eating an apple'. As discussed in the following section, bidirectional neural networks attempt to solve this issue.

4 Bidirectional Recurrent Neural Network

The architecture of a bidirectional RNN is depicted below:



The key change is that there are two sets of hidden states, each going in opposite directions. For each index, the two hidden states are concatenated together and combined with a final weight matrix to generate predictions. This type of architecture can reduce the problem of the model depending *only* on previous words, but it does not solve catastrophic forgetting. Architectures discussed in the following sections are intended to remediate this last problem.

5 Attention/Self-attention: Motivation

If the input dense is arbitrarily long, then the previous architecture would be not effective. For instance, for the Machine Translation task, the previous many-one architectures only have 1-fixed dimension output has severe limitations on this machine translation task. In order to address this limitation, let's discuss encoder- decoder seq to seq model.

5.1 Encoder-decoder seq to seq model

In this seq-seq architecture, the encoder part compresses the information to something machine friendly format in contextualized encoded vector. For instance, the input sentence would be just get condensed into a vector, which represents the meaning and the story to be preserved for the later part to translate. After transforming this meaning and information into a form of representation(a vector in this case), the decoder network retrieves information from the context vector by unrolling the information into different languages. For example, "She is eating an apple" (from the example figure above) is translated into Chinese. In general, this architecture takes encoded information as its input, transforms it to a compressed representation, and decodes the compressed representation into different languages.

However, it is very clear to us that the depence level between the first word from the english sentence "She" and the last word from the translated sentence is much weaker than other words. Indeed, it is not clear which of the words within the sentences are semantically closer to other words.

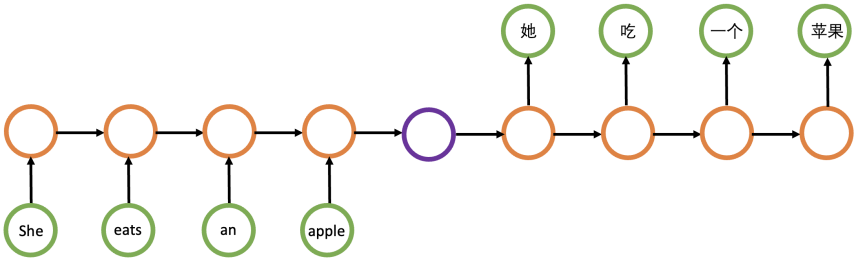


Figure 1: Machine Translation: Seq to Vec to Seq. Starting with the sequence of green shown in this figure, we compress all the information into the final h_t shown in purple above. We denote this purple circle as "context" vector. The decoder network untangles this context vector into Chinese

The motivation of the attention/self-attention is not just to memorize the long-source information, but also to avoid weaken relationship structure between early words in input sentencs. Thus, in attention, we train the weight matrix in order to address the semantic relationship between all the pairs in the input and output sentences. Utilizing this method, we can compute the importance of every single words within in the input sentences, which avoids the vanishing infomation for early words previosly.

5.2 Concept of Attention Mechanism

Consider a set of input x at time t such that

$$\{x\}_{i=1,...,t}^{t} = \{x_1, x_2, ..., x_t\}$$

where each x_i is the n-dimension input vector. Through the encoder layers, we would have the hidden representation h_i from given input vector x_i. With the attention score a_i, the context vector c_i is the linear combination of the a_i and h_i:

$$c_i = \sum_{j=1}^{T} a_{ij} h_j$$

With the context vector c_i, we utilize decoder modules for the machine translation task and softmax for regression problem; the remaining decision for the later module purely depends on the type of task we choose to do. Again this attention mechanism enables to capture the information in various positions, not only in information nearby each token of words.

5.3 Computing of Attention Matrix

An attention matrix $A \in \mathbb{R}^{T \times d_v}$ is defined as follows:

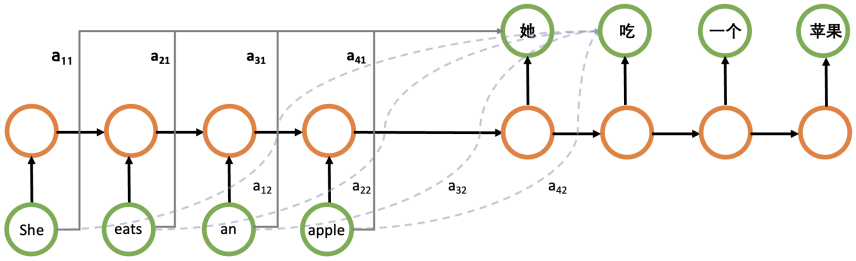


Figure 2: Machine Translation: Attention

$$A(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{1}$$

where Q, K, V denote as Query matrix, Key matrix, and Value matrix, repectively.

Also, each Q, K, V are defined as follows:

$$Q = X \cdot W^q \in \mathbb{R}^{T \times d_k}$$
$$K = X \cdot W^k \in \mathbb{R}^{T \times d_k}$$
$$V = X \cdot W^v \in \mathbb{R}^{T \times d_v}$$

where $W^q, W^k \in \mathbb{R}^{d \times d_k}, W^v \in \mathbb{R}^{d \times d_k}$, and the input matrix $X \in \mathbb{R}^{T \times d}$ where T is the length of the sequence, and d is the dimension for the embedding.

Each weight matrix describes the connection relationship of the input weight matrix. Notice that V has a different dimension $V \in \mathbb{R}^{T \times d_v}$ where d_v is the embedding dimension for the value matrix.

The name for each matrix might be arbitrary, but they are used to find the relationship between each word in the input matrix X. For example, suppose we want to find the meaning of the word "diffeomorphism" from a dictionary book. In this case, the word "diffeomorphism" is the query, and we check the query against **all the words** in the dictionary book–this is the key. The attention mechanism computes how aligned the query is against the keys by utilizing dot product between Q and K^T; the value for the dot product indicates the level of similarity. With the softmax function, we can represent it in the probabilistic density form, which is shown the softmax term from the equation (1). The attention matrix A is then simply the weighted sum of value vectors.

We also denote a_{ij} as the individual coordinate of the matrix A at ith row and jth column:

$$a_{ij} = \text{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right) = \frac{exp(q_i \cdot k_j)}{\sqrt{d_k} \cdot \sum_{r \in S_i} exp(q_i \cdot k_r)}$$

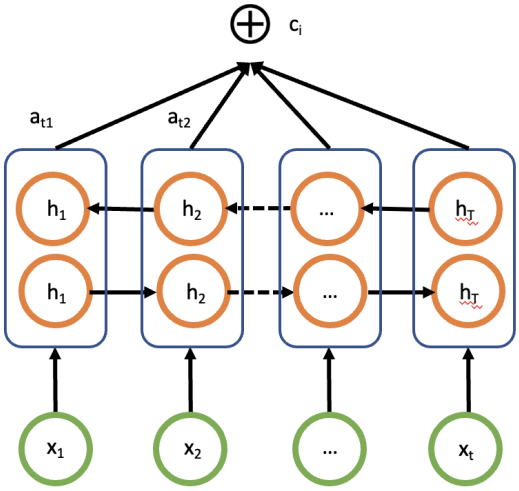


Figure 3: Attention scores from Bidirectional RNN

where q_i is the ith column vector of the Q which is the vector multiplication between ith row of input matrix X and jth column of the weight matrix W^q. S_i is the set of the import positions of the key positions for ith query to attend to.

6 Attention is all you need

What is introduced above is the self-attention mechanism. In the original paper in which attention mechanism is introduced [2], "Attention is all you need", attention is a broader concept, which could be the case that query is from the input but key is from other sets of reference points. However, self-attention is now more used in different architectures.

The $Attention(Q, K, V)$ defined above is a single attention. A multi-head attention can be used to make the model more expressive, defined as:

$$MultiheadAttention(X_q, X_k, X_v) = [head_1, ..., head_h]w^o. \tag{2}$$

, the concatenation of different heads times the output weight matrix, and

$$head_i = Attention(x_q W_i^q, x_k W_i^k, x_v w_i^v). \tag{3}$$

The multi-head attention architecture can be seen in Figure 4. For every round, the query, key, value pass the "Linear" module which is to time with weight w_i. The "scale" in "Scaled Dot-Product Attention"" indicates the softmax operation we have defined in attention matrix. Then, the h

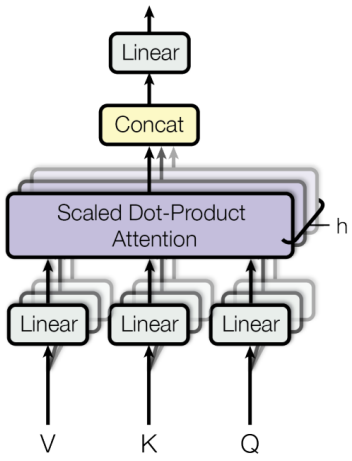


Figure 4: Multi-Head Attention [2].

different heads are concatenated and times with w^o.

The reason why the multi-head attention is expressive is that there are different weights for different heads, thus can approximate a broader class of functions. It is a key part in Transformer.

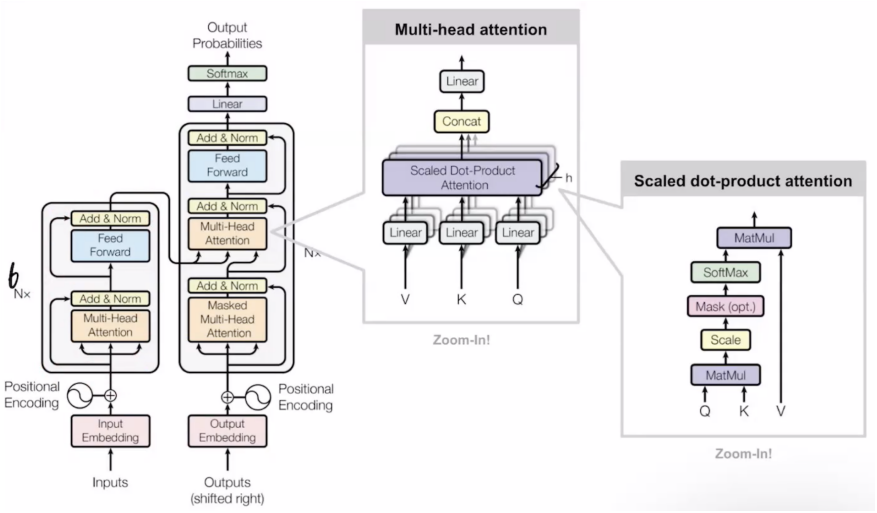


Figure 5: The transformer model architecture [2].

6.1 Transformer

The original transformer is designed for machine translation task so it is in encoder-decoder architecture. Later there are variations including encoder-only BERT, decoder-only GPT.

The transformer model can be seen in Figure 5. The input will firstly be mapped to an embedding space using a pre-trained matrix. Then it goes through multi-head attention modules (6 commonly)

with a feed-forward architecture. Then the output layer function depends on the target task, for example, probabilities for classification tasks and decoder for machine translation tasks.

The skip-connection in the model is important to train stable deep networks, which originally comes from the residual learning method in ResNet models.

The use of position encoder. The input positions in the current attention modules do not affect results, in other words, the attention is permutation-invariant.

$$head_i = softmax\left(\frac{Qk^T}{\sqrt{d_k}}\right)v = softmax\left(\frac{X_q w^q (X_k w^k)^T}{\sqrt{d_k}}\right)X_v w^v. \tag{4}$$

Here if the ordering of X_k and X_v change together, it will not change the result.

To hand-code the position information, we use an encoder matrix:

$$\mathbb{P} \in \mathbb{R}^{T\times d} \tag{5}$$

, where T is the sequence length, and d is the same as the embedding dimension of each word. It is additive to the original embedding $X \in \mathbb{R}^{T\times d}$. The resulting new embedding has ording information. $\mathbb{P}$ can be learnt in the context for specific task or it can be prescribed, for example, sinusoidal positional encoding.

References

[1] Thore Husfeldt. "Single run of Karger's Mincut algorithm." *Wikipedia*, N.p., 12 September 2012. Web. `https://fr.wikipedia.org/wiki/Algorithme_de_Karger#/media/File:Single_run_of_Karger%E2%80%99s_Mincut_algorithm.svg`

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).