From Neural Network to GPT

YouTube Videos

Andrej Karpathy - Let's build GPT: from scratch, in code, spelled out.

Umar Jamil - Attention is all you need (Transformer) - Model explanation (including math), Inference and Training

StatQuest with Josh Starmer - Transformer Neural Networks, ChatGPT's foundation, Clearly Explained!!!

Pascal Poupart - CS480/680 Lecture 19: Attention and Transformer Networks

The A.I. Hacker - Michael Phi - Illustrated Guide to Transformers Neural Network: A step-by-step explanation

How I Learn

Once I had read half of the book "Neural Networks and Deep Learning", I began to replicate the neural network example of recognizing handwritten digits. I created a repository on GitHub, https://github.com/lzwjava/neural-networks-and-zhiwei-learning.

That's the real hard part. If one can write it from scratch without copying any code, one understands very well.

My replicate code still lacks the implementation of update_mini_batch and backprop. However, by carefully observing the variables in the phase of loading data, feed forwarding, and evaluating, I got a much better understanding of the vector, dimensionality, matrix, and shape of the objects.

And I began to learn the implementation of the GPT and transformer. By word embedding and positional encoding, the text changes to the numbers. Then, in essence, it has no difference to the simple neural network to recognize hand-written digits.

Andrej Karpathy's lecture "Let's build GPT" is very good. He explains things well.

The first reason is that it is really from scratch. We first see how to generate the text. It is kind of fuzzy and random. The second reason is that Andrej could say things very intuitively. Andrej did the project nanoGPT for several months.

I just had a new idea to judge the quality of the lecture. Can the author really write these codes? Why I don't understand and which topic does the author miss? Besides these elegant diagrams and animations, what are their shortcomings and defects?

Back to the machine learning topic itself. As Andrej mentions, the dropout, the residual connection, the Self-Attention, the Multi-Head Attention, the Masked Attention.

By watching more above videos, I began to understand a bit.

By positional encoding with sin and cos functions, we get some weights. By word embedding, we change the words to numbers.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

The pizza came out of the oven and it tasted good.

In this sentence, how does the algorithm know whether it refers to pizza or oven? How do we calculate the similarities for every word in the sentence?

We want a set of weights. If we use the transformer network to do the task of translation, every time we input a sentence, it can output the corresponding sentence of another language.

About the dot product here. One reason we use the dot product here is that the dot product will consider every number in the vector. What if we use the squared dot product? We first calculate the square of the numbers, then let them do the dot product. What if we do some reversed dot product?

About the masked here, we change the numbers of half of the matrix to the negative infinity. And then we use softmax to make the values range from 0 to 1. How about we change the left-bottom numbers to the negative infinity?

Plan

Continue to read code and papers and watch videos. Just have fun and follow my curiosity.

https://github.com/karpathy/nanoGPT

https://github.com/jadore801120/attention-is-all-you-need-pytorch