Hamzeh Hamdan

Math 157

Final Project

Neural Networks and ChatGPT: How Does It Work?

Introduction

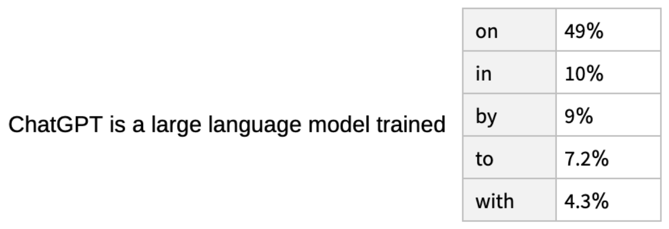
On November 30, 2022, OpenAI released ChatGPT-3, an artificial intelligence language model capable of generating natural language responses with high degrees of accuracy. While interacting with ChatGPT-3, users are allowed to enter a prompt and ChatGPT-3 responds coherently, often contextualizing its response with previous parts of the conversation; when prompted to "describe ChatGPT in one sentence," it responds "ChatGPT is a large language model trained on a diverse range of data, capable of generating human-like text and engaging in conversations on a wide variety of topics." Seemingly more impressive, ChatGPT generates a unique response every time it is prompted, even if the prompt remains the same. Within just five days, ChatGPT surpassed 1 million users, and within five months, it has surpassed 100 million users.

With many users misattributing its accurate human-like responses with an "understanding" of natural language, it is important to dive beneath the technology itself and understand what ChatGPT was created to do and how it does it. In this paper, I will dive into the math behind ChatGPT, starting with feedforward neural networks and backpropagation. I will then explore recurrent neural networks and explain the vanishing gradient problem and how it is solved by long short-term memory networks. My main purpose is to show the math behind neural networks and show their potential. Note that my analysis will not be unique to ChatGPT --

in fact, I will not be diving into transformer architecture at all. Instead, my explanations will be generalized on explaining how the general technology worked prior to the adoption of transformer architecture.

ChatGPT: Background

In the simplest terms, ChatGPT is always trying to find the most "reasonable" continuation of the text it currently has – where "reasonable" is simply how representative a generated text is to the billions of web pages ChatGPT is trained on. Whenever it has generated some text, like "ChatGPT is a large language model trained", it searches for the best next word and produces a vector of possible words and their probabilities. Simulating this with the ChatGPT2 model using Mathematica, the most probable next word is "on" with a 49% probability of occurring.



ChatGPT is a large language model trained

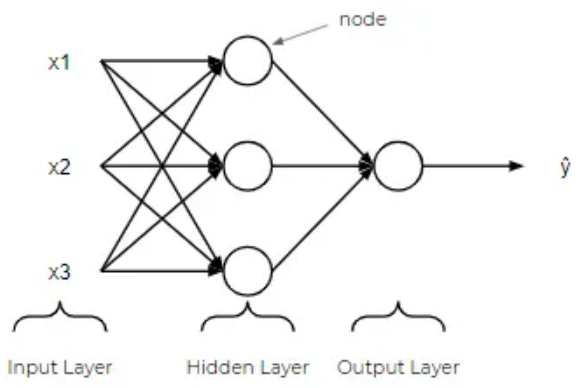
on	49%
in	10%
by	9%
to	7.2%
with	4.3%

Following the most probable word would have given us a rather confusing, and not quite accurate, explanation of ChatGPT: "ChatGPT is a large language model trained on the use of the GPT language." For a reason we do not quite understand, ChatGPT, amongst other natural language models, work much better when we, at random, select less probable words. For essay generation, this rate, called the *temperature*, seems to be best around 0.8. Note that, in reality, ChatGPT doesn't necessarily always look for the next best *word*, but rather the next best *token*, which can represent a part of a word. For now, this technicality is irrelevant. In fact, in completing the text "ChatGPT is a large language model trained on the use of the GPT language", the model returned "G" as the most probable following text (and then later returned "PT") – this is why it can sometimes "generate" new words.

So how exactly does ChatGPT find the most probable next word? To better understand what word should come next, ChatGPT is trained on billions of words of text. When assessing what the next best word is, a possible strategy is to simply find the frequency of the possible words, given the most recent word. This strategy tends to create incoherent text and is incapable of maintaining context or consistency within a sentence or a conversation. In theory, if we were to take enough words into consideration, we could find the best next word with better and better accuracy – however, with 40,000 commonly used words in the English language, the number of possible four-word combinations is over 2 quintillion, which is more than 20,000 times the estimated number of words on the internet. Thus, another strategy is required for estimating what the next best word is. In recent years, the most popular and successful method has been deep learning, a process that uses models based on neural networks.

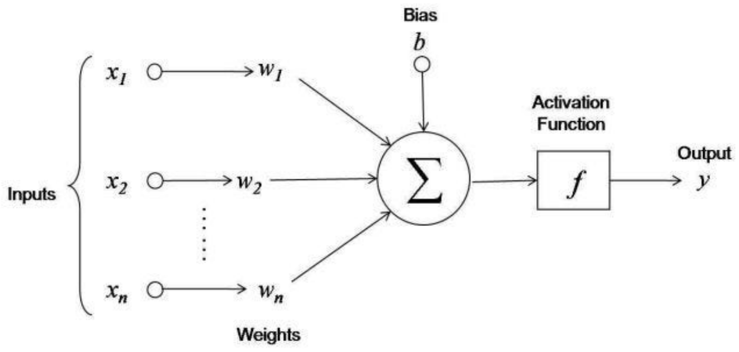
A Better Strategy: Neural Networks

Modeled after our understanding of neurological neural networks, artificial neural networks (which we will simply call neural networks moving forward) is a model that uses interconnected nodes in a layered structure that resembles the brain. These nodes, called *neurons*, are divided into different layers, with each neuron from a layer interacting with every neuron in the previous and subsequent layers, with each interaction having a certain *weight* to it. Neural networks typically have an input layer, at least one hidden layer, and an output layer. The following example has three inputs ($x1$, $x2$, $x3$) and one output (ŷ). A real world example of this could be a model that predicts output "House Price" based on inputs "Lot Size", "Number of Beds", and "Average Family Income" – while we do not completely understand to what degree these factors impact each other and the overall price, a neural network is able to find the best prediction algorithm by constantly updating the weights of the interactions to produce the best results through a process called backpropagation. Note that each node has a value, influenced by the values and weights of previous nodes, and that nodes in the hidden and output layers have an *activation function* that transforms the input signals from previous nodes into an output signal that is passed onto the next layer in the network.



In the real world, many more nodes are needed to create an accurate model and accuracy tends to increase with the number of nodes used. The example provided above has only four nodes – the ChatGPT3 model has 175 billion nodes and is thus able to discover much more complex trends in text. In the remaining part of this section, I will go over the mathematical processes behind backpropagation and network optimization.

Suppose we have a neural network, shown below, with a singular input vector $x^T = [x_1 \quad x_2 \quad x_3]$, weight vector $w = [w_{11} \quad w_{21} \quad w_{31}]$ with w_{xy} representing the weight from neuron x to neuron y, node value $a = x^T \cdot w + b$, bias b to control the baseline activation of the neuron, true value y, and loss $L = (y - a)^2$. We start by randomly assigning a certain set of weights to the neural interactions. We can now, and at any point in the future, do two things: we can use this untrained model as prediction method (forward pass) or train it (backpropagation). A forward pass involves simply calculating the output variable using the equation for *a*.



The training process, in which the weights are optimized, includes a subprocess called gradient descent. We update w in the following way: $w' = w - \lambda \cdot \nabla L(w)$, where w' is the new weight; λ is the *learning rate*, which is normally chosen to be around 0.05, and $\nabla L(w) = \left(\frac{\partial L}{\partial w}\right)^T$ is the gradient of the loss with respect to the weights. We also update the bias in a similar fashion. Applying the vector chain rule, we calculate $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial w} = 2(a - y) \cdot x^T$ and $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial b} = 2(a - y)$. We calculate the new weights and bias as the following:

$$w' = w - \lambda \cdot \nabla L(w) = w - \lambda \cdot \left(\frac{\partial L}{\partial w}\right)^T = w - \lambda \cdot 2(a - y) \cdot x$$

$$b' = b - \lambda \cdot \nabla L(b) = b - \lambda \cdot \left(\frac{\partial L}{\partial b}\right)^T = b - \lambda \cdot 2(a - y)$$

Suppose now that each input dataset was comprised of three values – more formally, $x_1 = [x_{11} \;\; x_{12} \;\; x_{13}]$; $x_2 = [x_{21} \;\; x_{22} \;\; x_{23}]$; and, in general, $x_i = [x_{i1} \;\; x_{i2} \;\; x_{i3}]$. We can rewrite the input vector as $x = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \end{bmatrix}$ and adjust the loss function to $L = \frac{1}{m}\sum_i (y_i - a_i)^2$, where *m* is the number of datapoints used. The activation vector, *a*, can be found with

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \dots \end{bmatrix} = \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \\ \dots & \dots & \dots \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + \begin{bmatrix} b \\ b \\ \dots \end{bmatrix}.$$

Solving for $\frac{\partial L}{\partial w}$, and $\frac{\partial L}{\partial b}$, we get:

$$\frac{\partial L}{\partial a} = \begin{bmatrix} \frac{\partial L}{\partial a_1} & \frac{\partial L}{\partial a_2} & \dots \end{bmatrix} = \begin{bmatrix} \frac{\partial \frac{1}{m}\sum_i (y_i - a_i)^2}{\partial y_1} & \frac{\partial \frac{1}{m}\sum_i (y_i - a_i)^2}{\partial y_2} & \dots \end{bmatrix} = \begin{bmatrix} \frac{2(a_1 - y_1)}{m} & \frac{2(a_2 - y_2)}{m} & \dots \end{bmatrix} = \frac{2(a-y)^T}{m}$$

$$\frac{\partial a}{\partial w} = \frac{\partial}{\partial w}(x \cdot w + b) = x$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial w} = \left(\frac{2(a-y)^T}{m}\right) \cdot x$$

$$\frac{\partial a}{\partial b} = \begin{bmatrix} \frac{\partial a_1}{\partial b} \\ \frac{\partial a_2}{\partial b} \\ \dots \end{bmatrix} = \frac{1}{\partial b}\left(x \cdot w + \begin{bmatrix} b \\ b \\ \dots \end{bmatrix}\right) = \frac{1}{\partial b}\begin{bmatrix} b \\ b \\ \dots \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \dots \end{bmatrix}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial b} = \left(\frac{2(a-y)^T}{m}\right) \cdot \begin{bmatrix} 1 \\ 1 \\ \dots \end{bmatrix} = \frac{2}{m}\sum_{i=1}^{m}(a_i - y_i)$$

Putting it together, the new values for *w* and *b* are:

$$\mathrm{w}' = w - \lambda \cdot \nabla L(w) = w - \lambda \left(\frac{\partial L}{\partial w}\right)^T = w - \lambda \cdot \left(\frac{2(a-y)^T}{m}\right) \cdot x^T$$

$$b' = b - \lambda \cdot \nabla L(b) = b - \lambda \left(\frac{\partial L}{\partial b}\right)^T = b - \lambda \cdot \frac{2}{m}\sum_{i=1}^{m}(a_i - y_i)$$

As I mentioned earlier, nodes in neural networks typically have an *activation function* that determines the output value of the node. Activation functions are vital for understanding complex relationships in data as they introduce nonlinearity into the algorithm – without activation functions, the output value can be written as a sum of a scalar product of the input layer nodes, and thus would not be able to detect information that is compounded in the data. We will now show the forward and backward pass of a similar neural network with an activation function. We will consider three activation functions: tanh, sigmoid, and rectified linear unit; the functions and derivatives of these functions are given:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \frac{\partial \tanh(x)}{\partial x} = 1 - (\tanh(x))^2$$

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \frac{\partial \sigma(x)}{\partial x} = \sigma(\mathrm{x})(1 - \sigma(\mathrm{x}))$$

$$\mathrm{relu(x)} = \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0 \end{cases}$$

Before calculating the forward pass, we note that we are now using the sigmoid activation function, and, as such, $\mathrm{a} = \sigma(\mathrm{xw} + \mathrm{b})$. For notational and computational simplicity, we will denote $z = \mathrm{xw} + \mathrm{b}$ and $a = \sigma(\mathrm{z})$. Note that the loss function remains $L = \frac{1}{m}\sum_i (y_i - a_i)^2$.

$$\begin{bmatrix} z_1 \\ z_2 \\ \dots \end{bmatrix} = \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \\ \dots & \dots & \dots \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + \begin{bmatrix} b \\ b \\ \dots \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \dots \end{bmatrix} = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \dots \end{bmatrix}$$

Moving onto backpropagation, we first apply the vector chain rule to get $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$ and $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}$. Calculating the derivatives:

$$\frac{\partial L}{\partial a} = \frac{2(a-y)^T}{m}$$

$$\frac{\partial L}{\partial b} = \begin{bmatrix} \frac{\partial \sigma(z_1)}{\partial z_1} & \frac{\partial \sigma(z_1)}{\partial z_2} & \dots \\ \frac{\partial \sigma(z_2)}{\partial z_1} & \frac{\partial \sigma(z_2)}{\partial 2} & \dots \\ \dots & \dots & \dots \end{bmatrix} = \begin{bmatrix} \sigma(z_1)(1 - \sigma(z_1)) & 0 & \dots \\ 0 & \sigma(z_2)(1 - \sigma(z_2)) & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

We can now calculate the new values of $\frac{\partial L}{\partial w}$, and $\frac{\partial L}{\partial b}$, we get:

$$\frac{\partial z}{\partial w} = \frac{\partial}{\partial w}(xw + b) = x \cdot \frac{\partial}{\partial w} w = x$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} = \left(\frac{2(\hat{y}-y)^T}{m}\right) \cdot \begin{bmatrix} \sigma(z_1)(1-\sigma(z_1)) & 0 & \dots \\ 0 & \sigma(z_2)(1-\sigma(z_2)) & \dots \\ \dots & \dots & \dots \end{bmatrix} \cdot x$$

$$\frac{\partial z}{\partial b} = \begin{bmatrix} \frac{\partial z_1}{\partial b} \\ \frac{\partial z_2}{\partial b} \\ \dots \end{bmatrix} = \frac{1}{\partial b}\left(w^T \cdot x + \begin{bmatrix} b \\ b \\ \dots \end{bmatrix}\right) = \frac{1}{\partial b}\begin{bmatrix} b \\ b \\ \dots \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \dots \end{bmatrix}$$

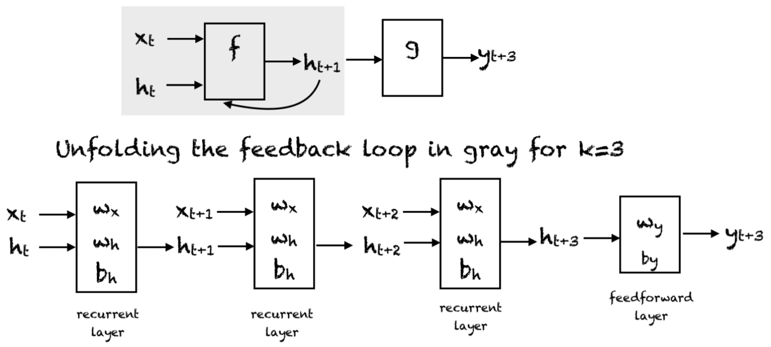
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b} = \sum \frac{2}{m}(\hat{y}-y)^T \cdot \begin{bmatrix} \sigma(z_1)(1-\sigma(z_1)) & 0 & \dots \\ 0 & \sigma(z_2)(1-\sigma(z_2)) & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Note that while these derivatives can be used to compute the new values of *w* and b, I will not be showing this computation for notational simplicity.

While feedforward neural networks like these are great at finding patterns in data, they lack the ability to maintain some "memory" between their run times. Suppose ChatGPT was to use one of these models, there would be no way to maintain the hidden state of the model over time. For more sequential data, like natural language, we need something called recurrent neural networks.

An Even Better Strategy: Recurrent Neural Networks

Unlike the feedforward neural networks we've discussed thus far, recurrent neural networks incorporate a concept of sequential data and time. In the recurrent neural network below, x_t represents the input at time *t*, y_t is the output generated by the model at time *t*, h_t represents the state of the hidden layer at time *t* (with $h_0 = 0$), w_x are weights associated with inputs in the recurrent layers, w_h are weights associated with hidden units in the recurrent layers, w_y are weights associated with hidden units to output units, b_h is the bias associated with the recurrent layer, and b_y is the bias associated with the feedforward layer.



In this model, to get an output at time $t + 1$, we must run the model t times. For some activation function f, the hidden layer at time $t + 1$ is $h_{t+1} = f(w_x x_t + w_h h_t + b_h)$ and the output y at time t is $y_t = f(w_y h_t + b_y)$. In a simplified manner, we might think of this as a model ChatGPT might run with a new word from the prompt every time; when prompted to "describe ChatGPT in one sentence", it would run the model with $x_0 =$ describe, $x_1 =$ ChatGPT, $x_2 =$ in, $x_3 =$ one, $x_4 =$ sentence. Note that in reality, the inputs are not words, and this analogy is not rigorous to the underlying processes of encoding and tokenizing text.

With many other similarities to feedforward neural networks, the training process for recurrent neural networks uses gradient descent. Recall that when updating the weights in feedforward neural networks, we use $w' = w - \lambda \cdot \nabla L(w)$ with $\nabla L(w) = \left(\frac{\partial L}{\partial w}\right)^T$ and $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial w}$. While the first two formulas remain the same, the last formula using the vector chain rule can be adopted here by rewriting the gradient of the loss function as a product of the gradients associated with all the activation functions of the nodes with respect to their weights. As such, the updated weights of the nodes in recurrent neural networks relies on the gradients of the activation functions of each node. With more and more layers being used in complex models, the value of the product of the gradients decreases until a point in which the partial derivative of the loss function approaches a value close to zero and the partial derivative vanishes – this is called the vanishing gradient problem and can be very problematic for the model as it becomes incapable of optimizing its weights with gradient descent.

To solve this problem, ChatGPT uses a form of recurrent neural networks called long short-term memory networks that are capable of storing information that is valuable through both short-term and long-term memory. Through every iteration, long short-term memory networks go through three important processes: (1) forget irrelevant information from the previous timestamp, (2) update current memory to include things learned from the current input, and (3) pass on updates from the current timestamp onto the next. Long short-term memory networks use attention mechanisms to figure out what exactly it should retain and what it should disregard.

Conclusion

Neural networks are an active area of research and are constantly being improved upon. The extent to which they are able to detect patterns in data is truly fascinating – even more so when considering the simple math behind the base of these technologies. The models discussed in this paper have been around for over 20 years and pose several limitations, amongst them the inefficiency from their inability to process the entire input sequence in parallel and the use of memory cells. In 2017, a research paper, “Attention is All You Need”, was published by a group of Google scientists that introduced the transformer architecture, a neural network model that solves the two main limitations posed above. This technology was, and remains, revolutionary to the field and has been adopted by many of the leading natural language processing services in the world. OpenAI continues to expand on their current technology, and has released a ChatGPT4, a model with one trillion nodes.

Bibliography

Sanjeevi, Madhu. "Chapter 10: Deepnlp - Recurrent Neural Networks with Math." *Medium*, Deep Math Machine Learning.ai, 10 Jan. 2018, medium.com/deep-math-machine-learning-ai/chapter-10-deepnlp-recurrent-neural-networks-with-math-c4a6846a50a2.

"Deep Learning with Tensorflow." *Introduction to Machine Learning, Neural Networks and Deep Learning - Deep Learning with TensorFlow*, devseed.com/tensorflow-eo-training/docs/Lesson1a_Intro_ML_NN_DL.html.

Duarte, Fabio. "Number of CHATGPT Users (2023)." *Exploding Topics*, Exploding Topics, 30 Mar. 2023, explodingtopics.com/blog/chatgpt-users.

Kostadinov, Simeon. "How Recurrent Neural Networks Work." *Medium*, Towards Data Science, 10 Nov. 2019, towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

"Mathematics behind the Neural Network." *Study Machine Learning*, studymachinelearning.com/mathematics-behind-the-neural-network/#:~:text=Neural%20Network%20is%20a%20sophisticated,variable%20and%20learn%20the%20patterns.

Mueller, Vincent. "Backpropagation in Neural Networks." *Medium*, Towards Data Science, 28 Oct. 2021, towardsdatascience.com/backpropagation-in-neural-networks-6561e1268da8.

Saeed, Mehreen. "An Introduction to Recurrent Neural Networks and the Math That Powers Them." *MachineLearningMastery.com*, 5 Jan. 2023, machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/.

Shin, Terence. "A Beginner-Friendly Explanation of How Neural Networks Work." *Medium*, Towards Data Science, 3 June 2020, towardsdatascience.com/a-beginner-friendly-explanation-of-how-neural-networks-work-55064db60df4.

"What Is Chatgpt Doing ... and Why Does It Work?" *Stephen Wolfram Writings RSS*, 14 Feb. 2023, writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/.