mathwork

Adam Ibrahim

September 2025

1 Introduction

this is a project that I've started in my Junior year in High school. It's a fun project and would highly reccomend for any one to do. The project has pretty notable restrictions I've applied on my self and it's as follows.

1. No imports
2. Little to no outside help for coding
3. Make maintainable code

The 2nd and 3rd rule is pretty reasonable for a project. The first rule some may have some questions and I'll break it down. "No imports" means no outside library or code that I haven't written myself. While I've violated rule 2, and arguably rule 3, I have written atleast 95% of the code and know how the 5% works completely.

While this is a tough project it has led me to understand a whole lot of stuff that isn't traditionally taught in a book or in a standard library, and has led me into some serious rabbit holes, but this should be a successful project? I hope, because I didn't really intend to finish this as I'm just trying to learn.

2 MLOPS

Forward propagation seems to work by just executing the neural network with random weights and biases

$$Z = WX + b$$

Where Z is the neuron W is the weight matrix, X is the input vector, and b is the bias, for binary outputs eg (Right or Wrong) you use the sigmoid function to use as the activation function. ReLU is defined as

$$\text{ReLU}(x) = \max(0, x)$$

This is also the sigmoid function which you can use for activation. In the notes I'll be using it , but in practice I'll be using ReLU. Sigmoid is defined as

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

There's the tanh function which is short for tan hyperbolic.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Now this comes at the cost at computing 4 exponentials. ReLU seems to be the cheapest out of them all.

ReLU seems to be the cheapest and seems to get the job done so I'll go with that.

I wanted to know how to activate a neuron and with this I'll be using the sigmoid for the math but it doesn't matter that much

$$\mathbf{W} = \begin{pmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{pmatrix}$$

So im going to let W be the weights of our neural network of general size and I'll let I as in input be the input vector which should look like

$$I = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

with these computing a vector matrix product should be relative ease with the matrix ops file having that method. Now I'll let another vector with the biases

$$B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Now this is where the *magic* comes in the actual ML

$$a_0^{(1)} = \sigma(\mathbf{W}I + B)$$

$$a_0^{(1)} = \sigma(w_{0,0}a_1 + w_{0,1}a_2 + \cdots + w_{0,n}a_n + b_1)$$

this is for activating ONE neuron if we have hundreds of neurons in our hidden layers then it's going to redo this computation maybe thousands of times. Now we need a cost function or a boolean output

This is going to be in the form of right and wrong and what is the correct answer is. So let $C(x)$ be our cost function

$$C(x) = \Sigma(\text{ResultVector}_n - \text{AnswerVector}_n)^2$$

The result vector is the results you get from the output layer and the answervector should be what the output layer should be so it should look like a bunch of zeros then a 1 and the resultvector should look like a bunch of numbers from 0-1 ergo confidence in the output

Large values in $C(x)$ is BAD it means that the NN doesn't know anything and it's garbage and we gotta get it low as possible.

A big thing to know about is the weights initilization, because how you setup the weights is critical for the execuation of the neural network. We first define the std deviation of the function to be He initilization

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}}$$

And we would do this on a gauss distrubution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} = \frac{1}{\sigma\sqrt{2\pi} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2}}$$

$$\text{Where } \mu = 0$$

Now what we do is that we square the std deviation of He initlization. We can reduce computation by removing the square root of calculating our std deviation. Which can help us in the future for floating point precision.

$$\sigma = \frac{2}{n_{\text{in}}}$$

Now we can directly substituite it in the normal distrubution

$$\mathcal{N}(0, \sigma)$$

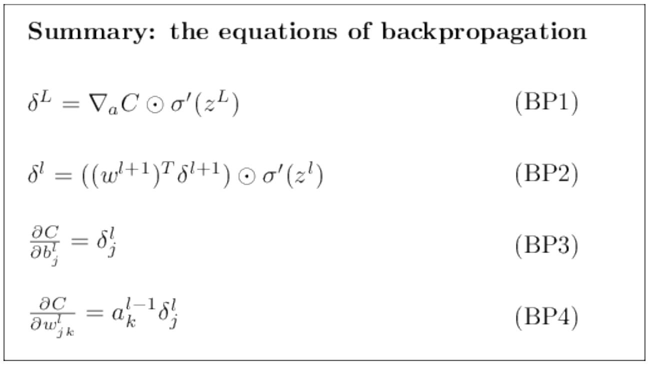
$$\text{Traditional } \mathcal{N}(0, \sigma^2)$$

There's another way to do this and thats with He Uniform where we draw it from a uniform distrubution.

$$U(-L, L)$$

$$\text{Where } L = \sqrt{\frac{6}{n_{\text{in}}}}$$

Now there's questions on what a uniform distrubution is and I won't address that.



Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad \text{(BP4)}$$

Figure 1: Backpropagation Diagram
[7]

2.1 Backpropagation

So backpropagation is the method of updating the weights and biases of the neural network. It's essentially what enables it to learn, it's nearly how you might reflect upon yourself when you get an incorrect answer.

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

While this equation uses symbols not conceived by any language known to man. It's important to know that mathematacians don't know how to program and that this equation can and will do it for any neural network. We'll start first with the cost function. In machine learning the cost and loss are essentially synonymous and I won't deal with semantics so I'll call it the cost function.

$$C = -\sum_{i=1} y_i \ln a_i$$

This is called the Categorical Cross-Entropy loss function. A metric ton of words but what it means is that if our model is correct but not confident we'll penalize it for low confidence. There are other cost functions but usually you need to differentiate those and write it in code, so it's important to know what problem you're facing and knowing which tools are fit for the job. Going back into the first equation we can break down $\nabla_a C \odot \sigma'(z^L)$

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^L_j)$$

Everything in the error formula is easily computable except for the partial. We can go into it.

$$\frac{\boxed{\partial C}}{\partial a^L_j} = \frac{\partial}{\partial a^L_j}\left(-\sum_{i=1} y_i \ln a_i\right) = p_i - y_i$$

This is the derivative of the cost function with respect to the activation of the output layer. This is important because it tells us how much we need to change the weights and biases to minimize the cost function. Now we can plug this back into the error formula.

3 MatrixOps

3.1 What is a matrix?

A matrix is a 2d array in cs or a 2d vector a matrix is denoted by uppercase letters so "A" is a matrix but "a" is not a matrix. You could also bold it to emphazise it but it's up to you.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

This is a 3x3 matrix, you can also have non-square matrices like

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

this is a 2x3 matrix, you can also have a mxn matrix where m is the number of rows and n is the number of columns. You can also have a 1xn matrix which is a row vector or a mx1 matrix which is a column vector. You can also have a 1x1 matrix which is just a scalar. With this you can relate this to a system of equations like

$$f(x) = \begin{cases} 2x + 3y = 6 \\ 4x + 5y = 10 \end{cases}$$

This can be represented as a matrix equation like

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

This is a coefficent matrix, where it's just the coefficents of the variables in the equations. You can also have an augmented matrix which is the coefficent matrix with the constants on the right side of the equations. This is called an augmented matrix because it's augmented with the constants.

$$\begin{pmatrix} 2 & 3 & | & 6 \\ 4 & 5 & | & 10 \end{pmatrix}$$

with these augmented matrices we can generalize any system of equations to an augmented or coefficent matrix. This is useful because we can use row operations to solve the system of equations. Row operations are just operations that we can do to the rows of the matrix to get a solution. You already used row operations in middle school when you did substitution and gaussian elimination

A way to describe a solution for a matrix is to put it in row echelon form or reduced row echelon form. Row echelon form is when the leading coefficient of each row is to the right of the leading coefficient of the previous row. The leading coefficient is the first non-zero number in a row. It looks like

$$\begin{pmatrix} 1 & 2 & 3 & | & 6 \\ 0 & 1 & 4 & | & 5 \end{pmatrix}$$

This is in row echelon form

$$\begin{pmatrix} 1 & 0 & 0 & | & -14 \\ 0 & 1 & 0 & | & 5 \end{pmatrix}$$

This is in reduced row echelon form

To get to row echelon form you can use the following row operations:

- Swap the positions of two rows (interchange)
- Multiply a row by a non-zero scalar (scaling)
- Add or subtract a multiple of one row to another row (replacement)

Matrix multiplication is multiplying two matrices (I know shocking)

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{pmatrix}$$

3.2 Singular Value Decomposition

Singular Value Decomposition is a method to decompose a matrix into a sum of rank 1 matrices. A Rank 1 matrix is where you can decompose it into the product of two vectors

$$\begin{pmatrix} 4 & 1 & 2 & 1 \\ 12 & 3 & 6 & 3 \\ 8 & 2 & 4 & 2 \end{pmatrix} = \begin{bmatrix} 2 \\ 6 \\ 4 \end{bmatrix} \begin{bmatrix} 2 & \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}$$

Now we can use this for compression because we went from 12 numbers to just 7

4 linear regression

MLR

$$(X^\top X)^{-1} X^\top Y$$

this is also the equation for a linear line

$$y = mx + b$$

$$m = \frac{n \sum y \sum x^2 - \sum y \sum xy}{n \sum x^2 - (\sum x)^2}$$

$$b = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

The sum for the first dimensional linear equation is actaully the result of plugging $y = mx + b$ in the MLR equation. So MLR serves as a one size fits all for any n dimensional

5 StatOps

In StatOps it features functions that I use for my project or for myself in my Statistics Class.

One notable thing about statistics is that you can find 10 billion different ways to sample or make a random number. The bedrock of statistics is the normal/gauss distrubution. Machine learning features weights and biases the problem is that how do you initilize the weights? Random numbers? If so what range?

This is where the box muller transform [2] makes that descision for us. The box muller transform is defined as

$$Z_0 = R\cos(\Theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2)$$

$$Z_1 = R\sin(\Theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2)$$

They're indepedent of eachother so it doesn't matter if you use sin or cos, at the end of the day it's the same stuff.

$$\Theta = 2\pi U_2$$

$$R^2 = -2 \cdot \ln U_1$$

$$\text{let } u = U_1, v = U_2, \text{ and } s = u^2 + v^2$$

utilising this you can rewrite the expression without trigonemetric functions. allowing you to avoid the computationally expensive sin and cos

6 Tensor

Off the bat Tensors are a very intimidating object to learn about, but the defination is a genereralization of a matrix. Matrixes are repeated vectors or vectors in vectors. Tensors are represented as N-dimensional matrices. We can represent it as a line, square or cube and we can say that each block in it has a number in it.

while it's simple in concept it's not that simple in coding it. First of all we need a way to access the data in the tensor as a coodinate system. The question is how to do that, to solve the sizing issue we can store all the data as an type array and we can use the coordinates to access the data. This leads to a problem since it's an array how do we translate the coordinates into an index?

To solve this we can use a simple formula

$$\text{index} = \sum_{i=0}^{n-1} \text{coordinates}[i] \cdot \text{dimensions}[i]$$

References

[1] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Chapter 4: Singular Value Decomposition (SVD). Cambridge: Cambridge University Press, 2020. DOI: `10.1017/9781108755528`. (Visited on 12/28/2025).

[2] *Box-Muller Transform*. URL: `https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform` (visited on 11/28/2025).

[3] *Computational complexity of mathematical operations*. URL: `https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations` (visited on 11/28/2025).

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`. URL: `https://arxiv.org/abs/1412.6980`.

[5] Sourish Kundu. *Who's Adam and What's He Optimizing? — Deep Dive into Optimizers for Machine Learning!* YouTube video. Channel: Sourish Kundu. Length: 23:20. Apr. 2024. URL: `https://www.youtube.com/watch?v=MD2fYip6QsQ` (visited on 12/14/2025).

[6] Hao Li et al. *Visualizing the Loss Landscape of Neural Nets*. 2018. arXiv: `1712.09913 [cs.LG]`. URL: `https://arxiv.org/abs/1712.09913`.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: `http://neuralnetworksanddeeplearning.com/`.

[8] Bidyut Baran Chaudhuri Shiv Ram Dubey Satish Kumar Singh. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. URL: `https://arxiv.org/abs/2109.14545` (visited on 11/28/2025).