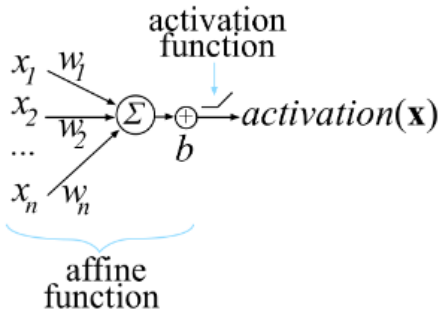
Using Machine Learning To Classify Data Clusters

Scope of Exploration

In the realm of boundless data, where patterns hide and insights reside, machine learning emerges as the key to unlocking the secrets within. Personally, I have been programming for around 4-5 years, from the start, one field of study intrigued me the most. Machine learning. It intrigued me even to this day how a machine can be trained to recognize and do things just through the use of data and math. I remember stumbling upon a video on how to create one of these using a popular programming language called Python. This was before the boom in AI where large language models such as ChatGPT came out, the field was still relatively unexplored. My first model was made to recognize handwritten digits, made using a pre-existing dataset, after seeing it work, I was hooked. My second model involved myself taking the time to manually label hundreds of images of tanks with bounding boxes, and using the dataset I had created myself. This was before the time ChatGPT and the boom in AI started, and to this day, in terms of programming, it is one of my strongest areas in terms of programming.

Background information

The term “machine learning” and “artificial intelligence” is rather broad in a sense. Both buzz words don’t truly captivate what is truly under the hood. Before looking into the math that is truly behind this work of wonders, first, it is important to understand the theory behind machine learning. Essentially, the end goal here is: Using a dataset, design a neural network and teach it to classify *x*. First things first, a run through of the steps is required to understand the math as a whole. One phrase that will be heard throughout this exploration is the term *Neural Network*, the basis of machine learning. One way to think of it is a big function consisting of numerous smaller ones. The end goal here is to minimize what's called the *Cost Function*, which is the function computed for how bad the neural network is at what it does, however, it is impossible to know at once what the local extrema of the cost function are. Think of a person blindfolded and he has been given the task of finding the lowest point in a hill (local minimum), he would only know what is directly beside him. This analogy can be applied here as, the *big function* itself consists of hundreds (or even millions) of parameters at a time, and is extremely difficult for even super computers to compute. One term to call each of these smaller functions is *node* or *neuron*, depicted by the diagram below.



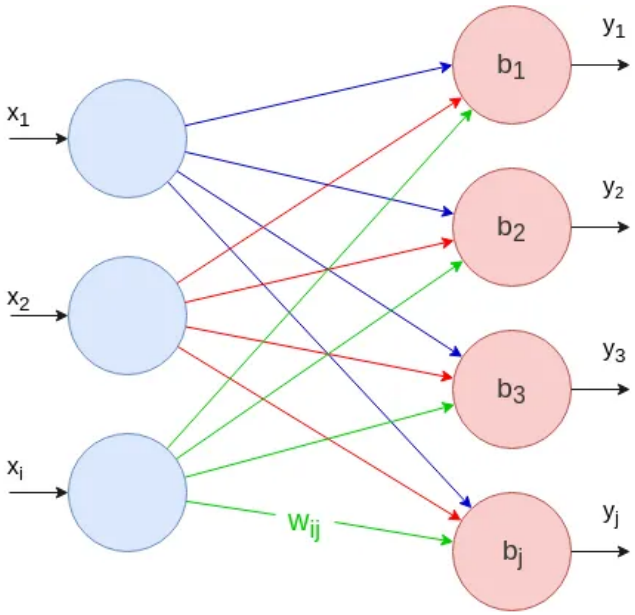
Where x_n are the activations (output number) of the previous layer, each multiplied by a unique weight, all of which is summed together and added on to a bias. The number then goes through what’s called an activation function. The final output is called an activation. The parameters of this singular small function are the weight and the bias (this applies to all neurons).

Introducing dataset

Defining the problem

https://www.kaggle.com/datasets/samuelcortinhas/2d-clustering-data/data

The bigger picture of what's shown above is the neural network which it makes up of, an example would be the diagram below.



Essentially, given a dataset, each weight and bias (or parameter) of the network can be tuned so that it is able to model the dataset itself and when given a new value, it will be able to *extrapolate* from what it has "learned" previously. In this case, the dataset contains clusters of data points, each cluster is a different color. Using the dataset, the neural network can eventually learn which region of the coordinate plane corresponds to which color.

Math

Notation used to express a 'node' in terms of a math equation

$$a = \sigma(\sum_{i} w_{ij}^{\ l} \times a_{i}^{\ l-1} + b_{b}^{\ l})$$

Passing the summation of all the weights of the current layer multiplied by the activations of the previous layer adding on a bias through an activation function denoted by σ, in this case, tanh(x)

tanh is an activation function in the context of neural networks, used to introduce non linearity, as most datasets do not have a linear correlation and often have more than two variables.

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

Deriving tanh'(x) using quotient rule (this will be of use later)

$$\begin{aligned} tanh'(x) &= \frac{(e^{x} + e^{-x})(e^{x} + e^{-x}) - (e^{x} - e^{-x})(e^{x} - e^{-x})}{(e^{x} + e^{-x})^{2}} \\ &= 1 - \frac{(e^{x} - e^{-x})(e^{x} - e^{-x})}{(e^{x} - e^{-x})(e^{x} - e^{-x})} \\ &= 1 - \frac{(e^{x} - e^{-x})^{2}}{(e^{x} - e^{-x})^{2}} \\ &= 1 - tanh^{2}(x) \end{aligned}$$

As mentioned previously, the cost function is used to compute how accurate the neural network is when computing an output from a given input. Here, the mean squared error (MSE) formula is used to calculate the cost.

$$C = MSE = \frac{1}{n} \sum_{i=1}^{n} (y_{i} - \hat{y}_{i})^{2}$$

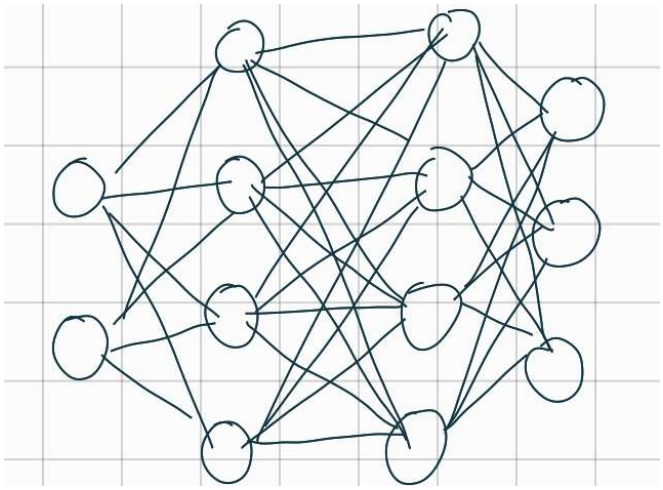
Defining the problem

Dataset used: https://www.kaggle.com/datasets/martininf1n1ty/simple-classification-playground

Based on the data set, using only math, determine the final color of a point based on its location in the 2D plane. Although this might seem simple, it is only for demonstration purposes, as the wider the range of the data, the larger the neural network.

Since coordinates consist of two values (x and y) the first layer (or input layer) consists of two neurons. The last layer (or output layer) in this case, has three nodes as it is evident from the dataset that there are a total of three colors, hence each output node corresponds to one. Both layers in the middle each consist of four neurons each as based on previous experience, a problem like this shouldn't require a big amount.

Figure 1



Each parameter (weights and biases) should be initialized at random within a certain range. Logically, one would assume that they should start with the value of "0," however, this makes it almost impossible to train as all nodes are the same as each other, making it borderline impossible to find its derivative with respect to the cost. To ensure everything is completely randomized, a Python script will be used, generating a value with a range of $[-\ 0.5,\ 0.5]$.

```
import numpy as np

weights = np.random.rand(input_size, output_size) - 0.5
bias = np.random.rand(1, output_size) - 0.5
```

Based on figure one, the each vector here shows the weights and bias of a specific node

```
weight: [[ 0.20270676  0.35069129  0.18092389 -0.24843021]
 [ 0.14773044  0.17711229 -0.28161897 -0.26025532]]
bias: [[ 0.12217286  0.17984042 -0.18992356  0.26243351]]
weight: [[-0.41405969 -0.22312742  0.18738278  0.28292904]
 [-0.23892646 -0.36766588 -0.07462869  0.32137498]
 [-0.25472814 -0.00329202 -0.36066244  0.01228788]
 [-0.16431604  0.20086948 -0.27679755 -0.01047083]]
bias: [[0.16524822 0.00556481 0.33936819 0.11608472]]
weight: [[ 0.19339201 -0.02565732 -0.03878757]
 [ 0.19880628  0.32640826 -0.37321342]
 [ 0.05394662 -0.0768317   0.34648421]
 [-0.16086509 -0.15207403  0.25049519]]
bias: [[ 0.13924307  0.48198453 -0.23055401]]
```

Now the neural network is complete, training is required in order to make it able to predict outcomes input data. Running a value through the neural network is called forward propagation. Training begins and the first step is to forward propagate the first value in the training dataset linked earlier, this gives yield outputs which, as mentioned earlier, can be used to compute the cost of the function.

First value of dataset:

x-value	y-value	color
7.0	6.0	1

Based on the dataset given earlier, it is evident that there are 3 colors, red, green and blue, each represented by the values r, g, b respectively. However, the qualitative data needs to be transformed into quantitative one as neural networks are only able to take in a number. Hence, r, g, b will be processed into 0, 1, and 2 respectively.

Code:
https://github.com/stampixel/Math-IA