Lesson - 1: Introduction to Neural Networks

* **Feedforward**: The process neural networks use to turn input into an output.
* Deep learning is used for anything such as self-driving neural networks.
* Neural Networks are programs that determine a line to classify data. However, we may need a more complex neural network for a more complex data.
* In neural network we looks at the previous data.
* We don’t care about special cases in neural network as we are most looking to compute a line that will classify results correctly.
* Example: test: x1, grades: x2.
* Boundary line: 2x1 + x2 - 18 = 0. In this boundary line, anything above 0 is above the line and anything below 0 is below the lining, meaning failed.
* Generally, our boundary will be an equation of the form w1x1 + w2x2 + b = 0 or, wx + b = 0. W: vector(w1, w2), x: vector(x1, x2)
* Label: What we are trying to predict.
* Each point is in form x1x2y where y is 1 for the blue
* Prediction: y-hat, the neural network prediction. 1 if wx + b >= 0, 0 if wx + b < 0
* In 3 dimensions, the only change is that now instead of a line we will have a plan flying in 3d. For example: w1x1 + w2x2 + w3x3 + b = 0, and still wx + b = 0. So as n dimensions just the dimensions increase.
* Perception: fit the data inside a node, and there are also nodes for inputs and output.
* Alternative way of thinking about that is that there are weights for each input, and all of the inputs go through their weight and then face the bias.
* Step function: 1 if the output is positive, and 0 if the output is negative.
* Two ways to think about the nodes: bias inside the node and bias as an input
* This objects are called neural network because they look like neurons in the brain which decide whether they output.
* AND operators: Takes two inputs and returns an output. This can also be represented as a line, where negative area returns a 0 and positive area returns 1. In this perception, we need the bias to be opposite from the weights and slightly greater in magnitude but not greater than the sum of the two weights.
* Perception trick: Start random, look at how badly the line is doing and move it around to get it to work better and better. We look at each point and get information from them to improve the line.
* Mathematical Trick: We subtract the point value from the weights which we multiply by the learning rate.Or we add them if the point is in the other area.
* Error: Looking for small steps to descent and find a local minimum. We should use continuous function to know how much in percentage.In order to do this we will use sigmoid function which gives us number close to 1 and numbers close to 0, or 0.5. This kind of function will also provide us with a percentage output instead of a number output, allows us to maybe use multiple neural networks.
* Sigmoid: The continuous version of the step function, where there are small numbers instead of discrete numbers. For large numbers: close to 1, small: close to 0, 0: 0.5
* The softmax function is in charge of finding the probability of something and all the probabilities should add to 1. One way to do this is to take the score and divide it by the total and than we use the exponential function to turn the number into positive
* So what we actually do is square each number and than divide.
* One hot encoding: The process of instead of having one variable we can have n variables with one criteria where each represents either an 1 or a 0.
* Cross Entropy: Using the logarithm function to determine whether a mode is good or bad. Goal: minimize the cross entropy, meaning the sum of the log of all the points. What’s interesting is that the cross entropy uses probability to generate an error function using probability.
* **Cross Entropy algorithm**: the sum of yi \* log(pi) + (1-yi) \* log(1 - pi)
* Multi-Class: the negative of the summation of i\_equals\_one to n up to the summation from y\_equals\_j to m Yij \* log(Pij). YIJ = 1 makes sure that we are only adding the logarithm of the probabilities of the events that actually have occured.
  + M: number of classes
* Logistic Regression algorithm: Pick a random model for the data, calculate the error(cross entropy?), minimize the error and continue doing that.

Pseudo code:

1. If y = 1:
2. P(blue) = yhat
3. Error = -ln(yhat)
4. If y = 0:
5. P(red) = 1 - P(blue) = 1 - yhat
6. Error = -ln(1-yhat)
7. Error = -(1-y)(ln(1-yhat)) - yln(yhat)
8. Error function: The sum over all the error functions

* When we use the error function, we can move from E(W, b) to something like E prime w prime b, and since prime is a local minimum this is nice for us.
* Gradient Descent: Start with initial prediction y^ = sigmoid(Wx + b). Where the Gradient is precisely the vector formed by the partial derivative of the error function with respect to the weights and the bias. Note that the gradient is the error.

Pseudo code:

1. Start with random weights, w1...wn, b
2. For every point(x1….xn):
3. For i = 1….n
4. Update wi’ ← wi - a(y^-y)xi
5. Update b’ ← b - a(y^-y)
6. Repeat until the error is small.

* Doing arithmetic on model. A linear model gives us the probability for each point to be blue. We can combine the two by adding. We use the sigmoid function to get a new value, the probability. We can also have one model weight more by adding weights. Than we add the two probabilities. Than we apply the sigmoid function.
* We can take a linear combination of two models so the two models time a constant will give us a new model.
* Feedforward is the process neural networks use to turn input into output
* Backpropagation is the process of training neural networks, consists of:
  + Doing feedforward operation
  + Comparing the output of the model with the desired output(target)
  + Calculating the error
  + Running the feedforward operation backwards to spread the error to each of the weights
  + Use this to update the weights and get a better model
* In backpropagation we ask the point what it wants the region to do so that the model will classify it correctly, than we look over the two models and check which one is classifying the point and better and we listen to the model more, by increasing its weight.
* With backpropagation we ask each point what it wants the models to do in order for them to classify it correctly.
* Gradient: The vector formed by all the partial derivatives of the error function with respect to the weight wi up to wn.

Lab: Gradient Descent

Functions to implement:

1. Sigmoid activation function
2. Output\_formula: The formula for the prediction
3. Error\_formula: The formula for the error at a point
4. Update\_weights: The function that updates the parameters with one gradient descent step.

**Conclusion**: in this lab four function were implemented for the gradient descent algorithm.

1. The first function is the sigmoid function, which is used to change to make the value of the neural network output continues instead of discrete. Equation: 1/ (1 + e-x). I was thinking finding this function will be straightforward but I forgot parenthesis.
2. The second function is the prediction formula which I found by taking the dot product of the features(points?) and the weights and finally adding the bias.
3. The next function is the error formula which is used to tell how badly is the output function doing.
4. Finally the update weights function, I use the output formula from before to determine the error -(y - y^). Than I decrease the weights by the learn rate times the point times the error and I decrease the bias by the error times the learn rate.