A Simple Neural Network

This project is conceptually mostly math-intensive.

Part 1: Demonstrate how a neuron works

A neuron, the smallest block of a neural network, takes the input, performs mathematical operations on it, and returns an output. The operations performed are as follows: multiply each input by a weight, get the sum of the above and add a bias value of b, and finally input the sum into the activation function

Create a Python Module called SimpleNeuron.

- Define function as the Sigmoid function (activation function) f(x) = 1 / (1 + e^(-x) in order to transform any value x into a new value between (0,1)
- Define a class called SimpleNeuron which takes in two variables: weights and bias.
- Define another function (feedforward function) that estimates the scalar/dot product of inputs with weights, then adds the bias value b. This sum is then inputted into the sigmoid function, and we return the result

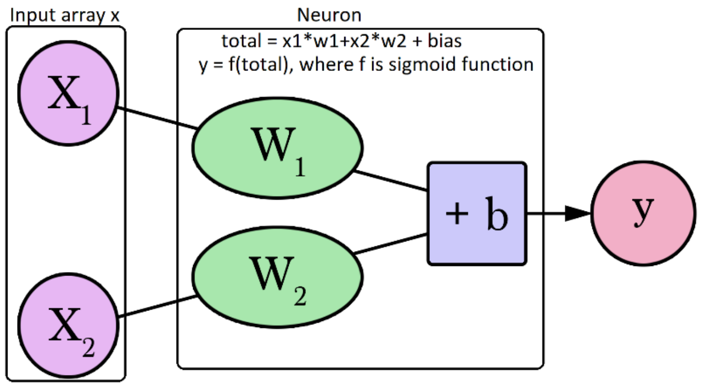
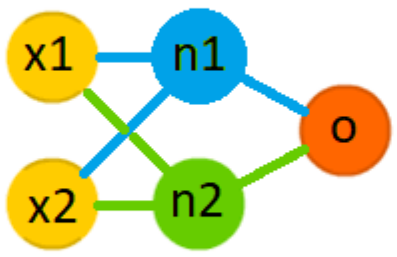

Input array x
Neuron
total = x1*w1+x2*w2 + bias
y = f(total), where f is sigmoid function
X1
X2
W1
W2
+ b
y

Figure 1. A neuron

Figure 2. Neural network of 3 Neurons



Part 2: Demonstrate how a group of 3 neurons work

A simple Neural Network consists of 2 inputs x1 and x2, 2 process layer neurons, and 1 output neuron. Assuming fixed values for weights and bias, this NN feeds array x to n1 and n2. The results from these 2 neurons are then fed into the output neuron. Each time an input is passed into a neuron, the mathematic operations

Create a Module called SimpleNeuralNetwork

- Define class Simple Neural Network of fixed weight and bias values
- Create 3 different neurons: neuron 1, neuron 2, output neuron.
- Define new feedforward function with input x: it takes x and performs the original feedforward function defined in Part 1 on neuron 1 and neuron 2. Then it takes the two results, and uses it as input to perform the feedforward function on the output neuron. Return result.

Part 3: demonstrate how to train our simple Neural Network

We can use the MSE (mean square error) function to measure how far the predicted result by the NN is from reality. To do so, find the average of the squared value of (predicted value minus true value). The larger this number, the bigger the loss. To reduce loss, we must tweak each weight. This can be done through the partial derivative: ∂*Loss/*∂*w*eight, which we calculate as the product of predicted value times (sigmoid derivative of previous outputs of neuron1 and 2) times (sigmoid derivative of values from real life). Based on the result, reduce or increase the value of the weight as needed, which decreases Loss.

New weight value = old weight value – (learning rate)*(∂*Loss/*∂*w*eight)

Create a Python Module called TrainedNeuralNetwork

```
- Define the MSE = mean((true value – predicted values)^2). We want to minimize this value.
- Define the derivative of the sigmoid: f'(x) = f(x) * (1 - f(x))
- Define new class TrainedNN
  input x is an array of 2 values, 2 neuron layer, 1 output neuron
- Initialize 6 weights as random numbers (2 weights connecting input to neuron 1, 2 weights
  connecting input array to neuron 2, 2 weights connecting the results from the first 2 neurons to
  the output neuron)
  3 bias values assigned as random numbers
- Define train function
  Declare variable learn rate (ie. 0.1)
  Declare variable e as number of loops to train the NN (adjust weights)

  Create loop that runs e times
        Use feedforward function on the input to get results predicted by current NN
        Compare predicted results with true results
        For each of the neurons, adjust weight = old weight value – (learning rate)*(
        ∂Loss/∂weight), where ∂Loss/∂weight is calculated utilizing the derivative of the sigmoid
        defined above
```

At the end of the loop, the weights will have adjusted and when we try inputting data, the neural network will make the best guess it can.