SOMETHING AWESOME

Jason Yu

UNSW HS1511

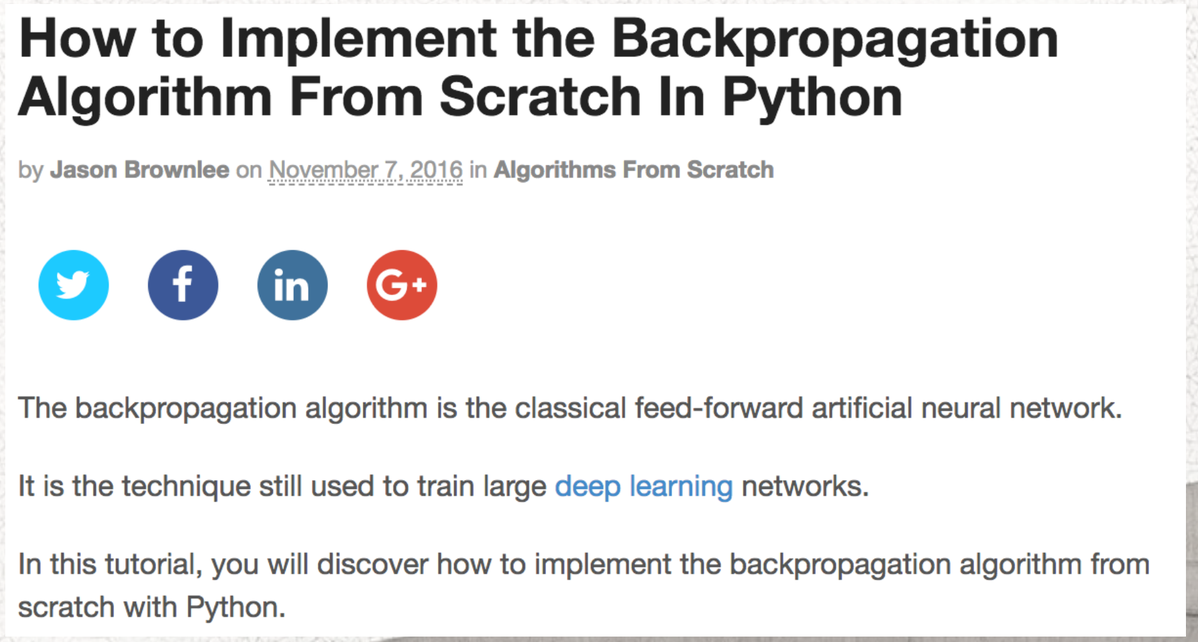
THE IDEA

To make a multilayer neural network from scratch without the use of outside libraries from basic Python

Admittidly a tthe start I was intimidated by the concept but after working tirelessly on the project I gained a much better understanding and passion for the field of machine learning with a working network.

THE PLAN

To get the needed knowledge for my project I did a lot of prior research watching many videos on youtube from channels such as 3b1b, Siraj, coding rainbow and many other channels. After gaining sufficient knowledge I followed an extensive tutorial on making one of these networks.



How to Implement the Backpropagation Algorithm From Scratch In Python

by **Jason Brownlee** on November 7, 2016 in **Algorithms From Scratch**

The backpropagation algorithm is the classical feed-forward artificial neural network.

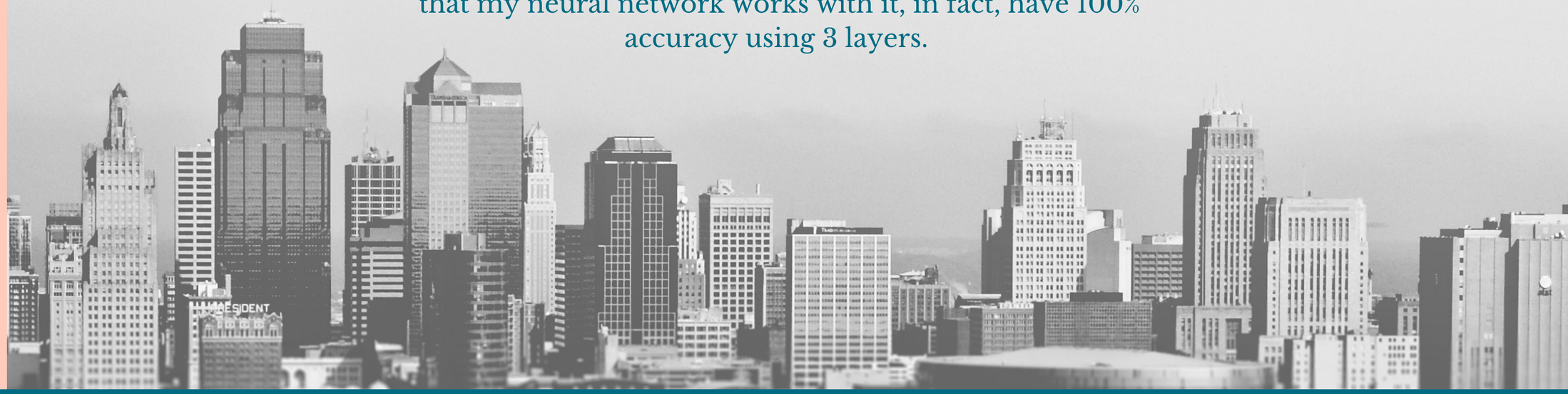
It is the technique still used to train large deep learning networks.

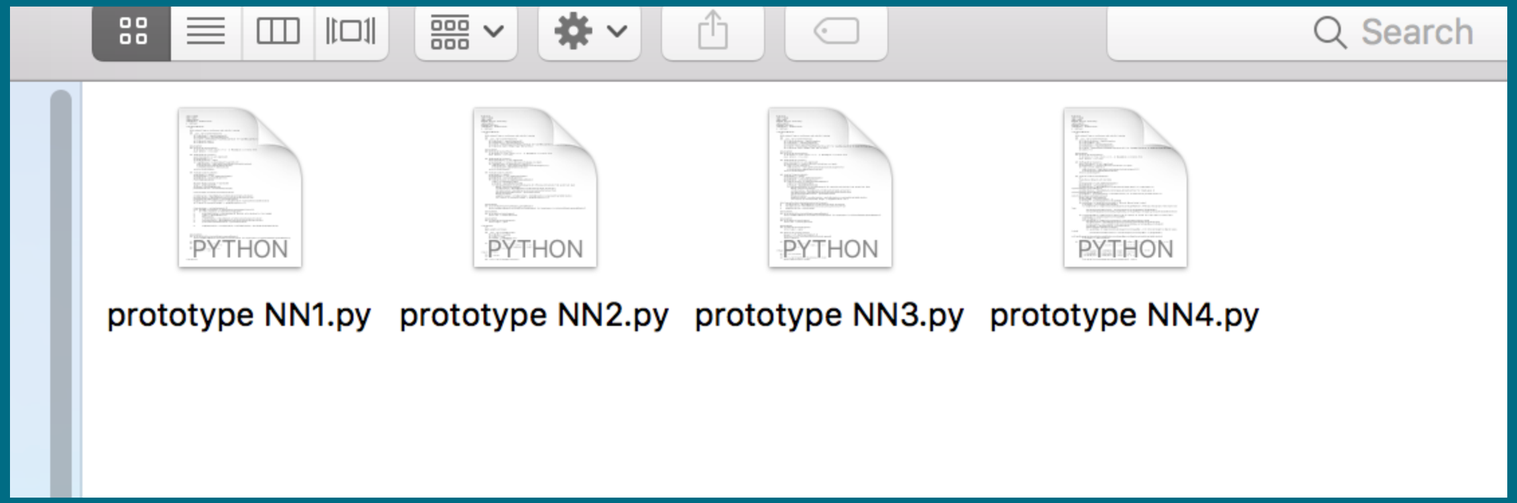
In this tutorial, you will discover how to implement the backpropagation algorithm from scratch with Python.

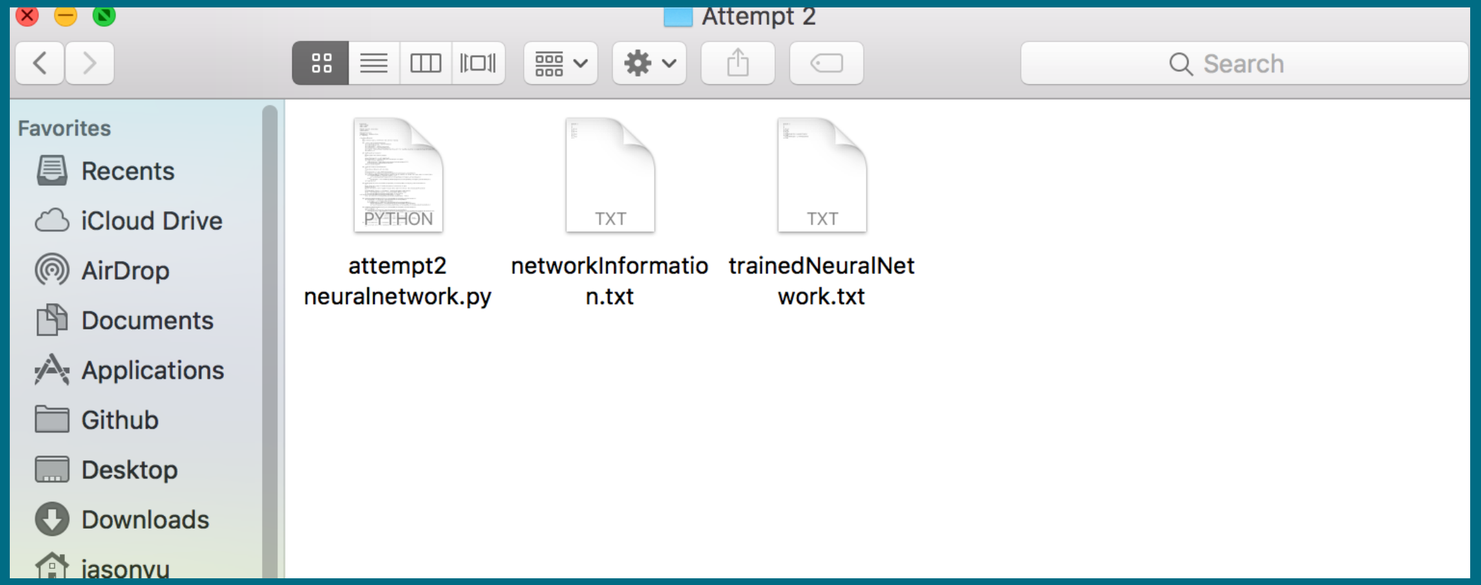
IMPLEMENTATOIN

The implementation took a while after doing a lot of experiments.

From the start, I had a basic idea of how these networks worked. After experimenting a great deal I gained a better understanding of these networks and how they used error to maximize results. Finally, I created a working neural network that could learn the XOR function. This XOR function proves that my neural network works with it, in fact, have 100% accuracy using 3 layers.







PAST ATTEMPTS AND DRAFTS

```
seed(1)

#Information
dataset = [[0,0,[0,1]],
           [0,1,[1,0]],
           [1,0,[1,0]],
           [1,1,[0,1]]]
outputMapping = {0:'True',1:'False'}
numInputs = len(dataset[0]) - 1
numOutputs = len(dataset[0][-1])
numEpochs = 100000
learningRate = 0.01

#Setup and Training
NN = NeuralNetwork(numInputs, 2, numOutputs,outputMapping=outputMapping)
NN.trainNetwork(dataset, learningRate, numEpochs)
accuracy = NN.accuracy(dataset)
NN.exportNN('XOR.txt')
```

```
>epoch=5000, lrate=0.01, error=2.00
>epoch=10000, lrate=0.01, error=2.00
>epoch=15000, lrate=0.01, error=2.00
>epoch=20000, lrate=0.01, error=2.00
>epoch=25000, lrate=0.01, error=1.99
>epoch=30000, lrate=0.01, error=1.94
>epoch=35000, lrate=0.01, error=1.71
>epoch=40000, lrate=0.01, error=1.39
>epoch=45000, lrate=0.01, error=0.68
>epoch=50000, lrate=0.01, error=0.25
>epoch=55000, lrate=0.01, error=0.14
>epoch=60000, lrate=0.01, error=0.09
>epoch=65000, lrate=0.01, error=0.07
>epoch=70000, lrate=0.01, error=0.05
>epoch=75000, lrate=0.01, error=0.04
>epoch=80000, lrate=0.01, error=0.04
>epoch=85000, lrate=0.01, error=0.03
>epoch=90000, lrate=0.01, error=0.03
>epoch=95000, lrate=0.01, error=0.02
Expected=False, Got=False For Inputs [0, 0]
Expected=True, Got=True For Inputs [0, 1]
Expected=True, Got=True For Inputs [1, 0]
Expected=False, Got=False For Inputs [1, 1]
Accuracy=100.00%
```

WORKING XOR FUNCTION

100% ACCURACY

```
    def errorPropogation(self,expected):
        for layerIndex in reversed(range(len(self.layers))):
            layer = self.layers[layerIndex]
            errors = []
            if layerIndex != (len(self.layers)-1):
                for neuronIndex,neuron in enumerate(layer.neurons):
                    currentError = 0
                    for nextLayerNeuron in self.layers[layerIndex+1].neurons:
                        weight = nextLayerNeuron.weights[neuronIndex]
                        errorSignal = nextLayerNeuron.errorSignal
                        currentError += weight * errorSignal
                    errors.append(currentError)
            else:
                for outputIndex,neuron in enumerate(layer.neurons):
                    expectedOutput = expected[outputIndex]
                    output = neuron.currentActivation
                    errors.append(expectedOutput - output)
            for neuronIndex,neuron in enumerate(layer.neurons):
                neuron.errorSignal = errors[neuronIndex] * sigmoid(neuron.currentActivation,derivative=True)

    def updateWeights(self,inputs,learningRate):
        for layer in self.layers:
            for neuron in layer.neurons:
                for weightIndex in range(layer.prevLayerSize):
                    inputVal = inputs[weightIndex]
                    neuron.weights[weightIndex] += learningRate * neuron.errorSignal * inputVal
                neuron.bias += learningRate * neuron.errorSignal
            inputs = [neuron.currentActivation for neuron in layer.neurons]

    def trainNetwork(self,dataset,learningRate,numEpoch):
        for epoch in range(numEpoch):
            sumError = 0
            for row in dataset:
                outputs = self.forwardPropogation(row)
                expected = row[-1]
                sumError += sum((expected[index]-outputs[index])**2 for index in range(numOutputs))
                self.errorPropogation(expected)
                self.updateWeights(row,learningRate)
            if epoch % int(numEpoch/20) == 0:
                print('>epoch={0}, lrate={1}, error={2:.2f}'.format(epoch, learningRate, sumError))
```

SNAPSHOT OF SOME CODE