Task 3 neural network

GitHub: <https://github.com/dardan-gashi01/DardanGashi_IN3063coursework>

**Introduction:**

In this task I will be creating a neural network from scratch using numpy for the neural network and pandas to load the dataset and split it into train and test data. I will be implementing a sigmoid and ReLU layers in my neural network with a forward and backwards pass. I will also be implementing a SoftMax output layer. I will also be implementing a stochastic gradient descent optimizer that has a stopping criterion also and will be fully parameterized so we can test what factors impact what in our model.

**How I built my model:**

So, to build my model I had to first understand the dataset I am working with. The dataset has 784 columns, so this means we have 784 inputs in the input layer. There are 10 different classes in this dataset T-shirt, Trouser, pullover, Dress, coat, sandal, shirt, sneaker, bag, and ankle boot which means we have 10 outputs in the output layer and 10 hidden layers. This allowed me to draw something like the diagram below to help me understand the work more:

Diagram

Description automatically generated

I then done the maths for this model, so I know what I am coding. And the maths I did is below (all the maths done is from lecture 6 slide 64 onwards):

Before we do forward and backward pass, I will generate weights and biases

Forward pass:

Z(1) = W(1)A(0) + B(1). A(0) being the input so in this case X and of size 784 x n\_samples

A(1) = σ(Z1) where σ = activation function in my case I will be using the ReLU function

Z(2) = W(2)A(1) + B(2)

A(2) = Softmax(Z(2))

The to work backwards using backwards proposition I need to find derivatives using chain rule and the answers I got were:

Backwards pass:

dZ(2) = A(2) – number of out nodes(Y)

dW(2) = (1/size of Y)\* dZ(2)A(1)

dB(2) = (1/size of Y)\* sum(dZ(2)

dZ(1) = W(2)dZ(2)A(1) \* derivative of ReLU(Z(1))

dW(1) =(1/size of Y) \*dZ(1)(size of X)

dB(1) = =(1/size of Y) \* sum(dZ(1))

This was the maths I worked out using the chain rule and reading from lecture 6.

When implemented fully we get something like

Text

Description automatically generated

As you can see in this code, we used the ReLU activation function for the forward and backward pass, in the forward pass we used the normal ReLU function so

Graphical user interface, text

Description automatically generated

I also have another version that uses sigmoid instead of ReLU and this function looks like

Text

Description automatically generated

For this function I was having a problem where I was getting an overflow error because the value was too large, so I had to find a function online that was a more stable version so considered if x is less that 0 or larger than 0 and I found this here[[1]](#footnote-1)

after implementing this it was working and not getting any errors anymore.

**Implementing SoftMax as the output layer:**

For the SoftMax output layer I implemented it using this functionA picture containing logo

Description automatically generated

This function was in the lab exercises, and I used it for my model. I implemented this for the output layer and took the input z2 which is the sum of the bias and the dot product of the weight and activation function ReLU.

**Implementing a fully parameterized neural network:**

I have made a fully parameterized neural network that takes the input nodes and the output nodes along with the number of hidden nodes and the learning rate.

Text

Description automatically generated

From the picture above it shows the class Neural network takes 4 inputs as stated above to create an object of that class and we create on by doing:

model = neuralNetwork (X\_train, Y\_train, 10 ,0.1)

so, the X\_train is the number of input nodes which will be 784 in this case, the Y\_train is the output which should be 10 because there are 10 classes that we will be getting. The 10 in the third parameter is the hidden nodes we have, and these will be used to adjust the weights and the last parameter in this case its 0.1 is the learning rate of the model.

**Implementing stochastic gradient descent:**

I decided to implement SGD as an optimizer instead of Adam. Adam converges faster which is an upside to using Adam however SGD generalises better than Adam which is why I decided to choose gradient descent instead of Adam.

To work out gradient descent we must use the formula

Wnew = wold − α[gradeint] where α is the learning rate and is the same as what I did with:

Text

Description automatically generated

In my gradient descent I did it for both weights so one in first layer and one in the second whilst also doing it to the bias values. The weights are defined earlier in the program along with the bias values and then I used the -= to update the value of the weights and biases to be learning rate multiplied by the derivative of the weights and the biases as that is equal to the gradient and I did data in the backwards propagation as shown above on the second page and I used chain rule to work that out for example:

self.w2\_delta = scaling \* self.z2\_delta.dot(self.a1.T)

when I do this, it then means that the weights will keep updating on every epoch and tries to find the local minimum to maximize accuracy in the model.

To visualise it we can use figure 1.1 on the last page and we can see how the weight decreases after every epoch and reaches the local minima at the bottom however we do not want to overfit the values and go past the minima to be inefficient so we add a stopping criteria which I will be talking about in the next section about how I implemented my stopping criterion and what I made the stopping criterion.

**Implementing a stopping criterion:**

I implemented a stopping criterion in my model for gradient descent. The criteria I gave it is when the accuracy only increases by less than 0.2% twice in a row. For example if have a list of accuracies and the last 3 show 50,50.1,50.2 the difference of these are 0.1 and 0.1 for both this means the accuracy is not increasing that much anymore and is flattening out so I want a stopping criteria for this and the way I implemented this was by creating the function below:

def difference(arr):

return [item-arr[i-1] for i, item in enumerate(arr) if i != 0]

this function I created will create a new list of the difference of the current value and the previous one so for example like above it would have a new list [0.1,0.1]. so in my training function I have got a condition with this function like this:

diff = difference(acc\_list[-3:])

if len(diff) > 1:

if(diff[-1] < 0.2 and diff[-2] < 0.2):

break

so this function first checks if the new array we made using the function is big enough to iterate. When it is greater than a size of 1 then we always take the last 2 values and check if they are both smaller than 0.2 then the program calls break so it stops the training loop and carries on the rest of the code.

An example of this happening in practise is:

Text

Description automatically generated

From this example we get an array of [0.23, 0.18,0.2]

**Train using back propagation:**

**How parameters affect the outcome of the accuracy:**

**Evaluation:**

Diagram

Description automatically generated

Figure 1.1

1. https://shaktiwadekar.medium.com/how-to-avoid-numerical-overflow-in-sigmoid-function-numerically-stable-sigmoid-function-5298b14720f6 [↑](#footnote-ref-1)