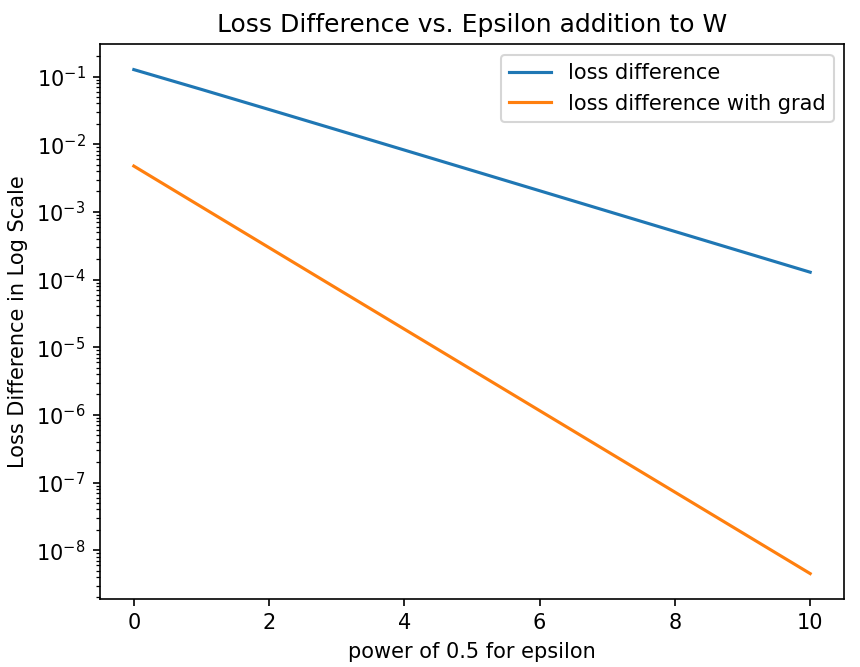
**Deep Learning**

**QS1:**

**2.1.1**

Results as we wanted to see. You can find the code in the testingGrad.py file in Qs2 directory

**A graph with a line and a number of numbers

Description automatically generated with medium confidence**

**A graph of loss difference between two different numbers

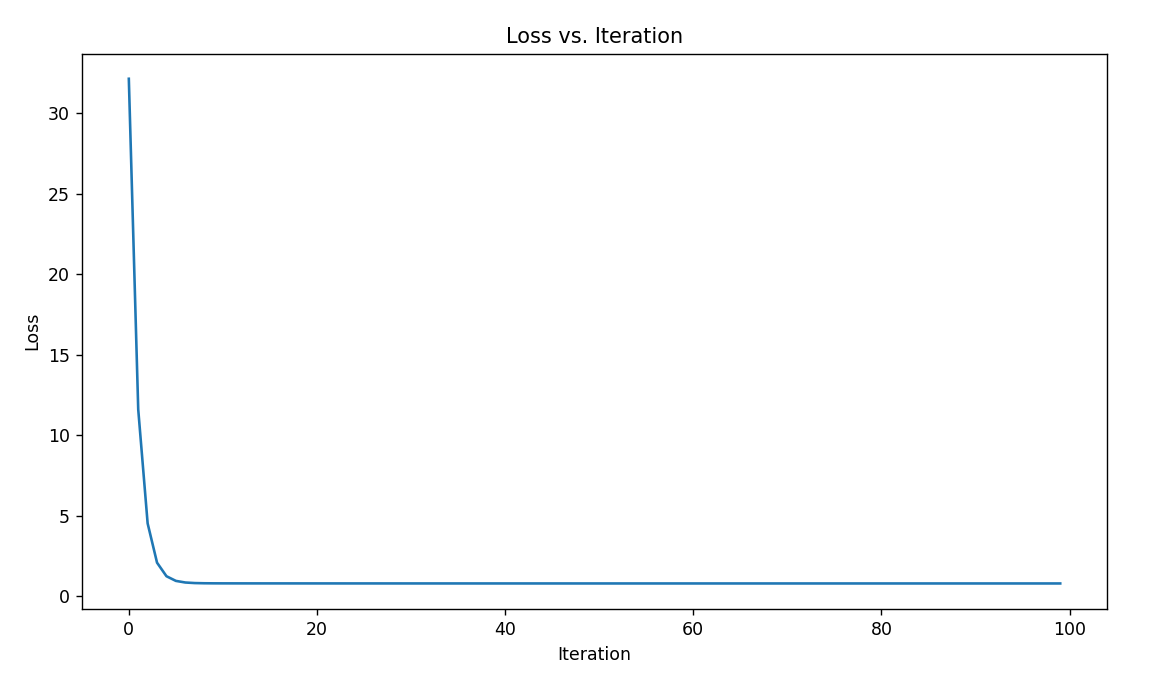
Description automatically generated**

**2.1.2**

We create our SGD in a way that it doesn’t use the softmax activation function on the output layer if there is only one output neoron in the least squared error example in for this case we don’t want the softmax or the sigmoid, because our network is targeted to predict the y value (a regression).

We then run our code for the MSE problem and get amazing results, even better then the closed form solution.

A line graph with blue dots

Description automatically generated\*It is worth mentioning that we used early stopping if loss doesn’t decrease for 10 iteration in a row, but it just kept on decreasing. It is hard to see that in the graph because the loss starts at a very high value.

In values, we get:

The closed form equation is: Our solution equation is:

Mean Squared Error (Closed-form): 6.1851108724937465

Mean Squared Error (SGD): 0.8065947869466693

I.e. our solution is better than the closed-form one.

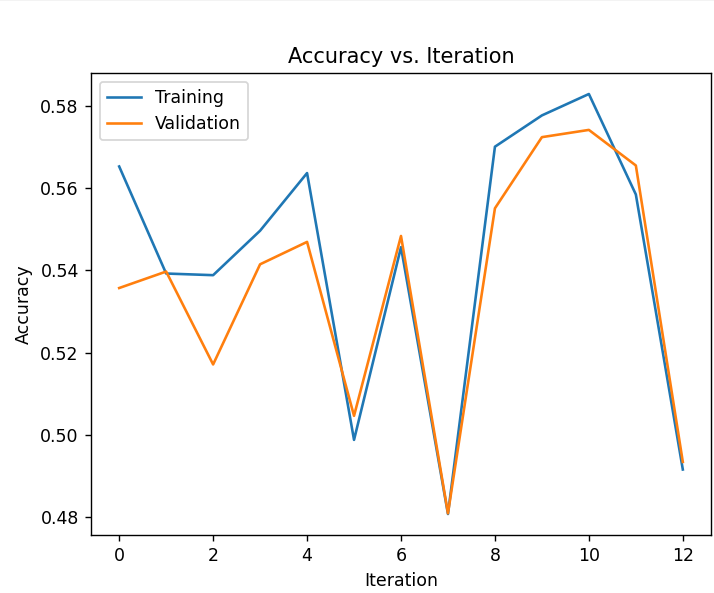
**2.1.3**

We test our minimization of the softmax function first using PeaksData dataset. Running our SGD on multiple combinations of learning rate and batch size. Before moving in, we have to hold in mind that one layer with a softmax activation function cant solve this well because the simple fact that having 0 inner layers means that the NN can learn only a linear separator, as it did well in the previous question. As this task isn’t really in the power of our “mini NN”, we have no high hopes for significant differences between the parameter combinations, and they can be highly dependent on the initialization of the random weights.  
Running all the possible combinations out of the options we can see in the code(Function Qs3), we find that the optimal combination to be: Learning rate: 0.1 and Batch size: 10.

We once again use early stopping when for 10 iterations the loss doesn’t decrease on the validation set.

The model stopes after 13 epochs(iterations), with a final accuracy on validation set: 0.55952.  
While this is not a got accuracy, given the fact that this can only learn a linear function, it is not that bad, it is better than random label (which is 0.2) but no learning seems to happen.

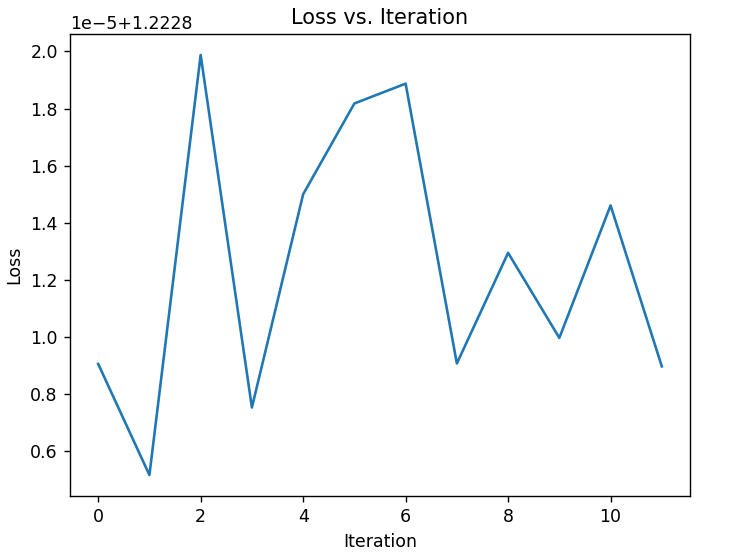
* In the graph, Iteration is equal to an epoch, meaning that it is an iteration over all the data(/batches)

A graph with blue lines

Description automatically generated

We then move on and run the same thing on the GMMData. Finding that the best combination is Learning rate: 0.01 Batch size: 50.

A graph with blue and orange lines

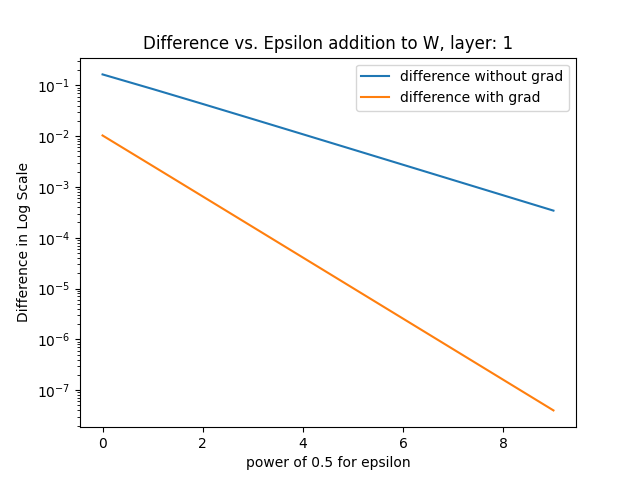
Description automatically generated

**QS2:**

**2.2.1**

After implementing our network we perform Jacobian tests.

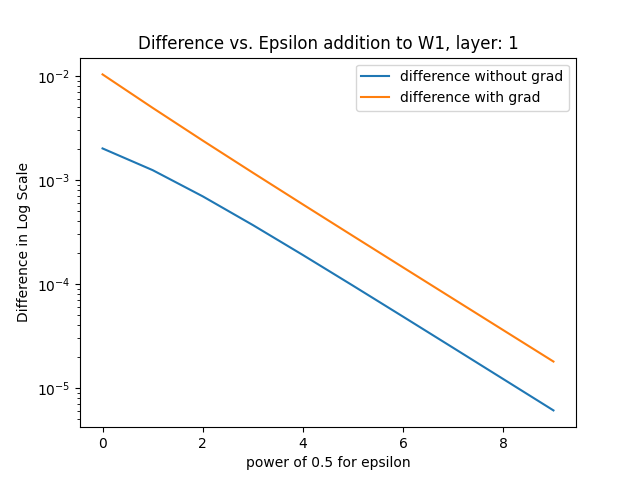
First on a regular network with 2 inner layers, can be found in the function “def runJacTest” in the NN.py file. The results can be found In the “Jacobian\_Test\_Regular\_NN” Directory, we will add a sample of them here.

 A graph with a line

Description automatically generated

**2.2.2**

We move and repeat this test for the residual network, again with 2 inner layers, using the function “def runResNetJacTest” in the NN.py file. The full results can be found in the “Jacobian\_Test\_ResNet” directory, we will add a sample of them here.

A graph of a difference between two different colored lines

Description automatically generated

**2.2.3**

We proceed to our grad test, again, with 2 inner layers, using the function “def run\_grad\_test” in the NN.py file. The full results can be found in the “Grad\_tests\_Regular\_NN” directory, we will add a sample of them here.(The x axis is “power of 0.5 for epsilon” as before)

A graph with a line

Description automatically generated A graph of different colored lines

Description automatically generated

We do the same to a resnet, and once again full results can be found In the “Grad\_tests\_ResNet” directory.

A graph with a line

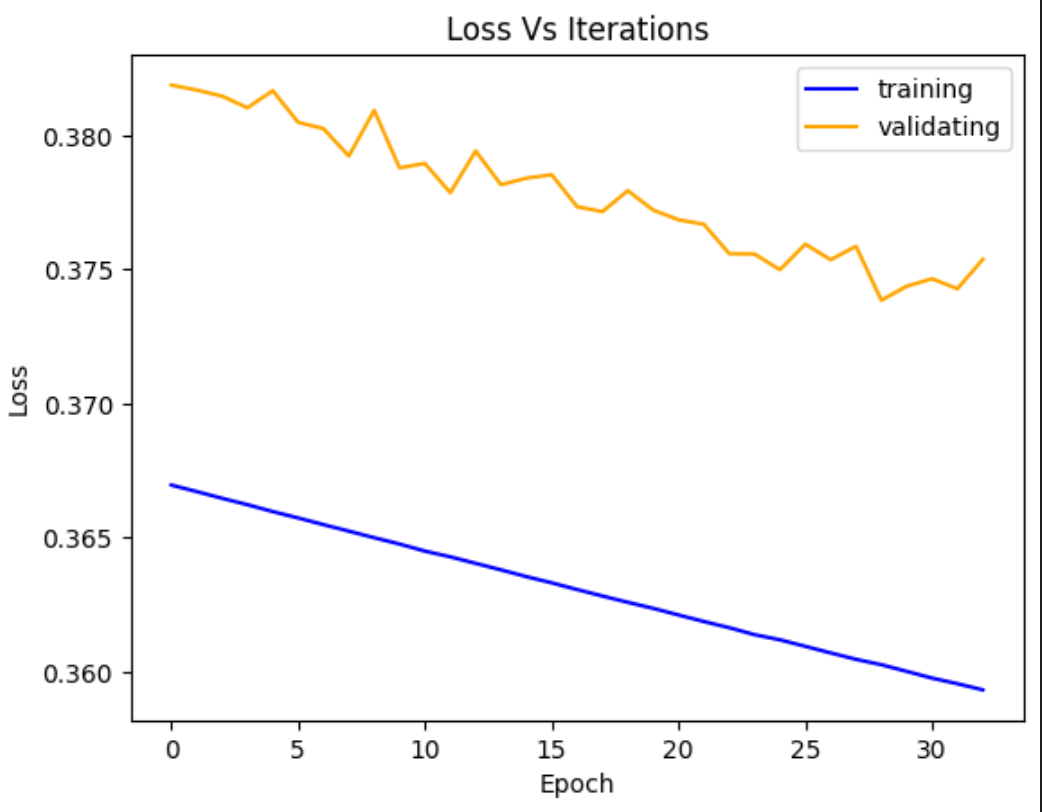
Description automatically generated A graph with a line and a line

Description automatically generated with medium confidence

**2.2.4**

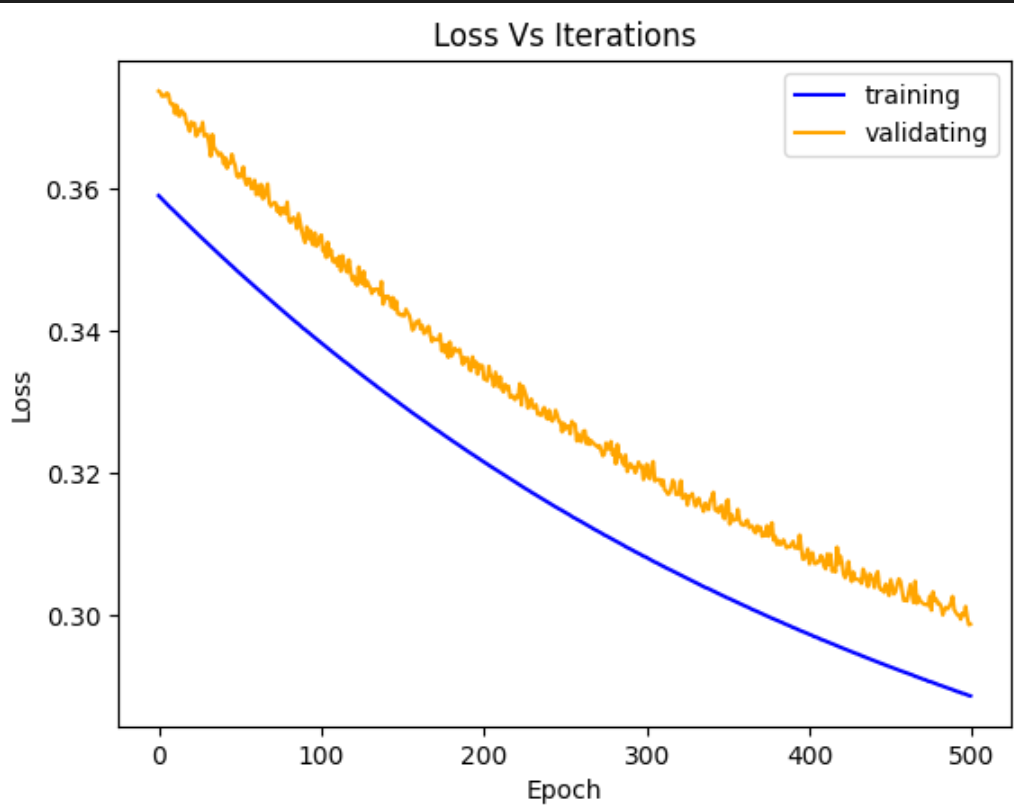
We proceed to train our NN on two data sets.

Starting with PeaksData we find that the best parameters are lr of 0.0001 and batch size of a 100. We once again use early stopping when for 10 iterations the loss doesn’t decrease on the validation set.   
The model we tried stops after 33 epochs returning the next results:

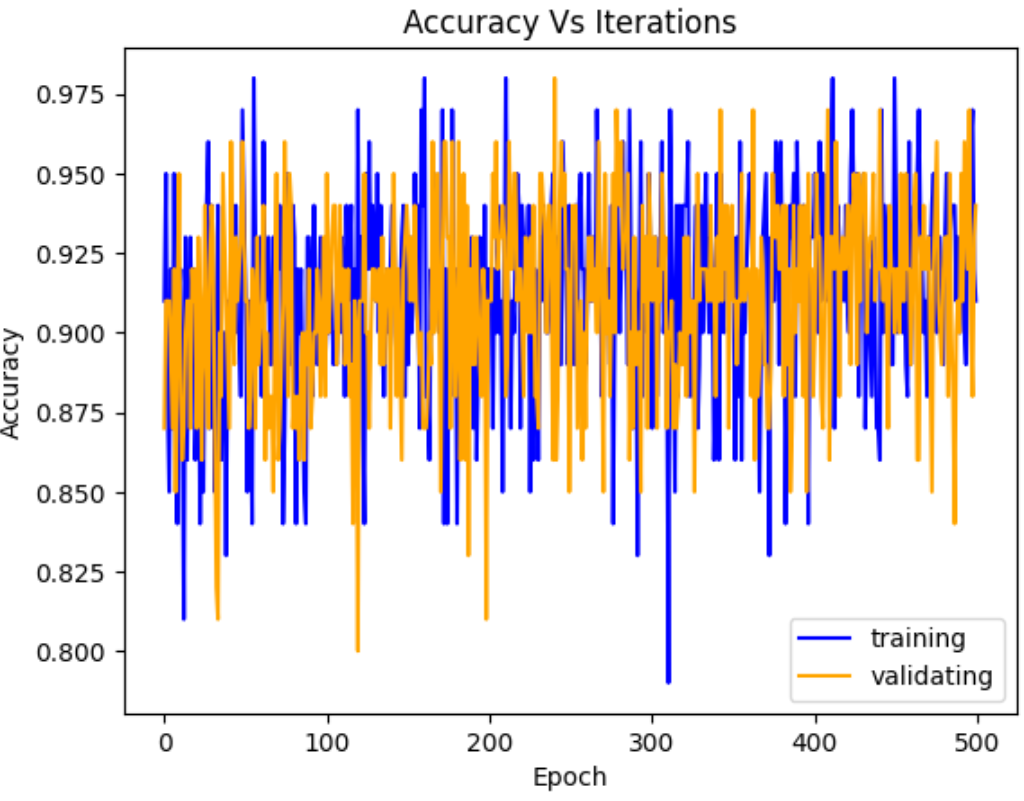


We can see that the model stops because although the train loss keeps decreasing, the validation loss increases. We suspect that the model can do better but the validation loss rises a bit because it doesn’t represent the data fully, as we use only part of it.

For this reason we decide to change the early stopping condition from 10 iterations to 100, this results much better results as we suspected. It is worth mentioning we max the number of epochs to 500. Increasinng it will result even better losses.



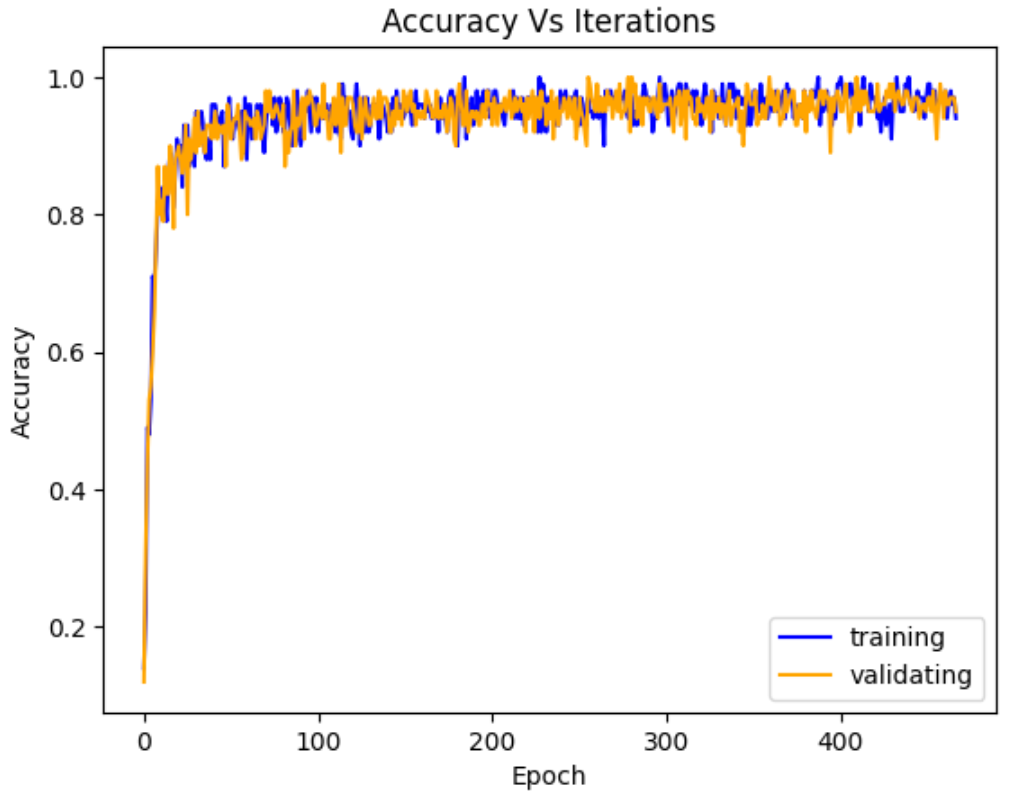
On the other hand, our accuracy graph is going crazy, but stays in high accuracies. We understand that this must be a bug as it doesn’t go together with the loss graph but we really can't find the problem.



We move on to the second data set(the GMMData). We find that the best parameters are lr: 0.0001 and batch size of 50.

Our model early stops after 466 epochs, resulting great results

A graph of loss and validation

Description automatically generated with medium confidence

**2.2.5**

Running the training on only 200 data points results as we suspected. While the training loss decreases similarly and even faster than when all given points were given, the validation loss doesn’t. We can even see a drastic decline close to the end and what starts to seem overfitting.   
This makes a lot of sense as this happens because:

1. Less data points, it is easier for the network to memorize more “edge points”
2. Less data points, a worse presentation of the validation set in the training data.

A graph of loss versus iteratives

Description automatically generated with medium confidence