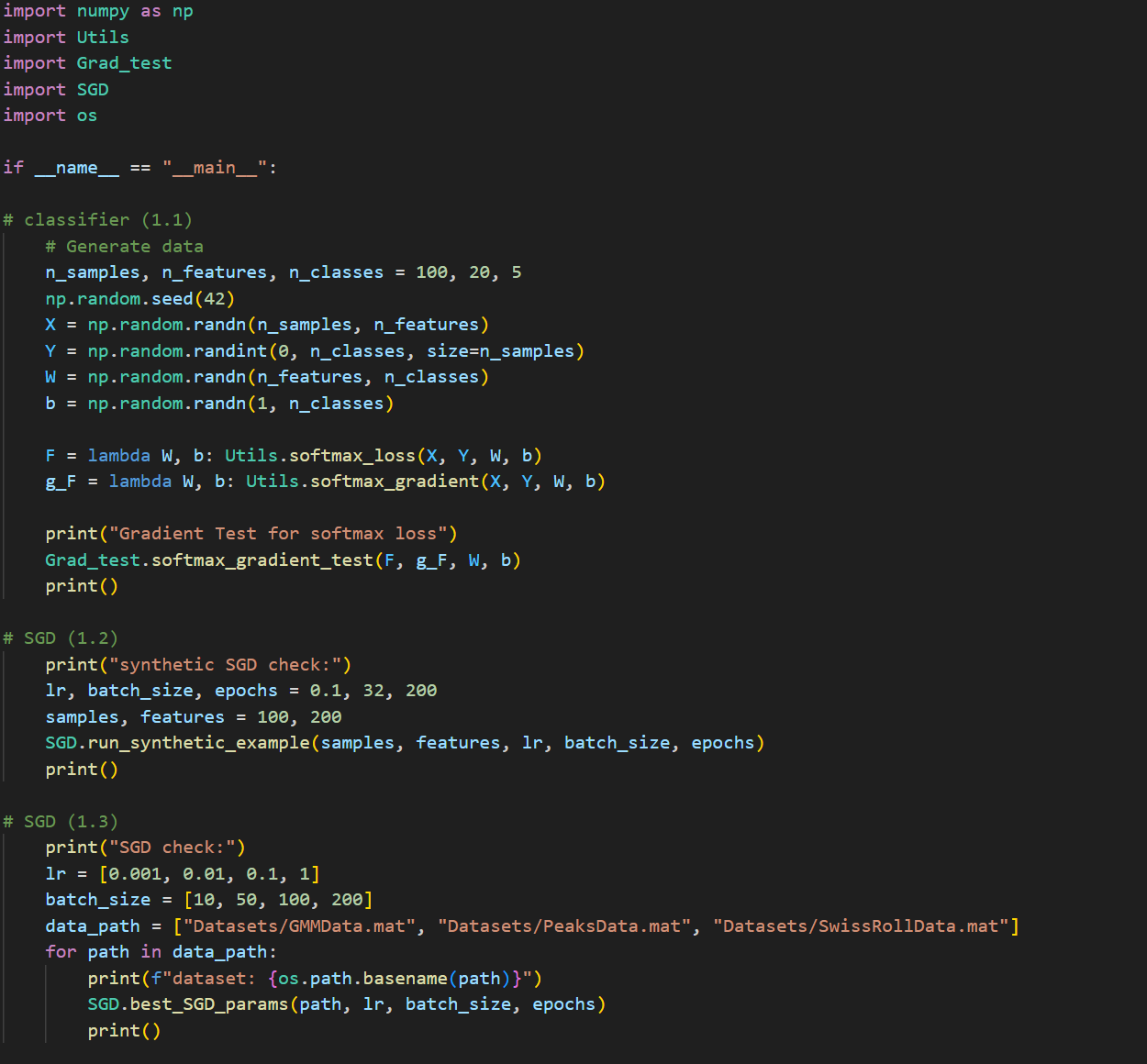
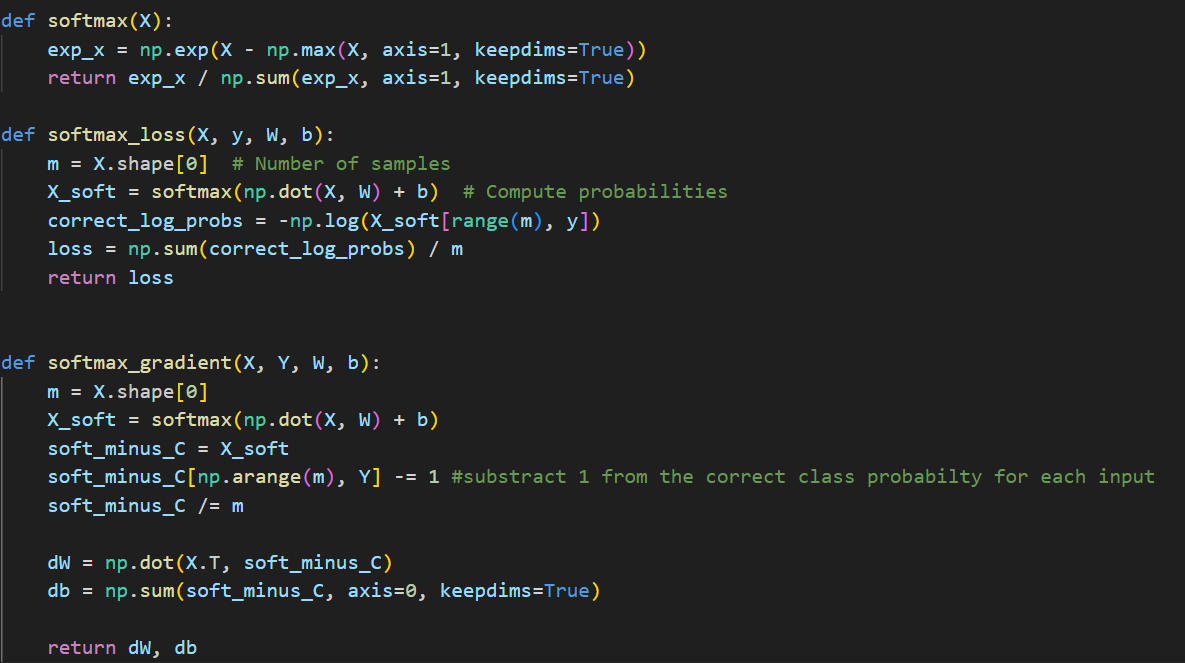
**Mini-project: Deep Learning from Scratch**

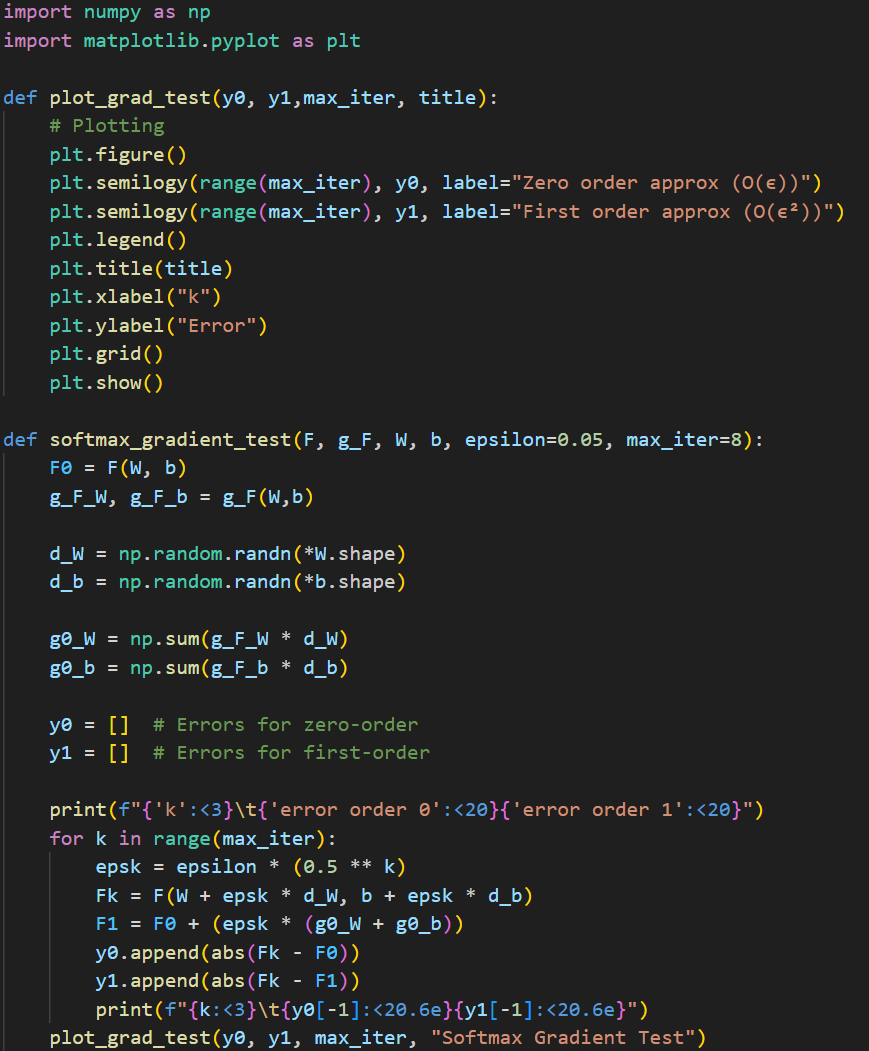
**Submitted by Lior Sharony and Eden Miran**

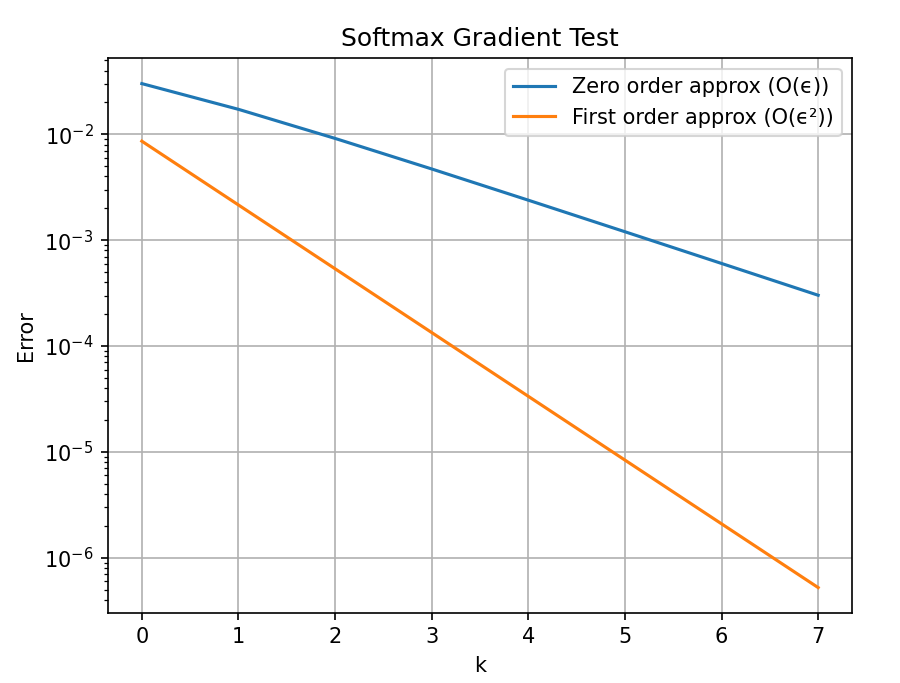
1. **Part I: the classifier and optimizer**

****

* 1. **loss function “soft-max regression” and its gradient**

We have tested the correctness of our soft-max regression gradient using the gradient test as shown in the class, with respect to the weights and biases, using 8 iteration and an epsilon value of 0.5 (shown in the code). 



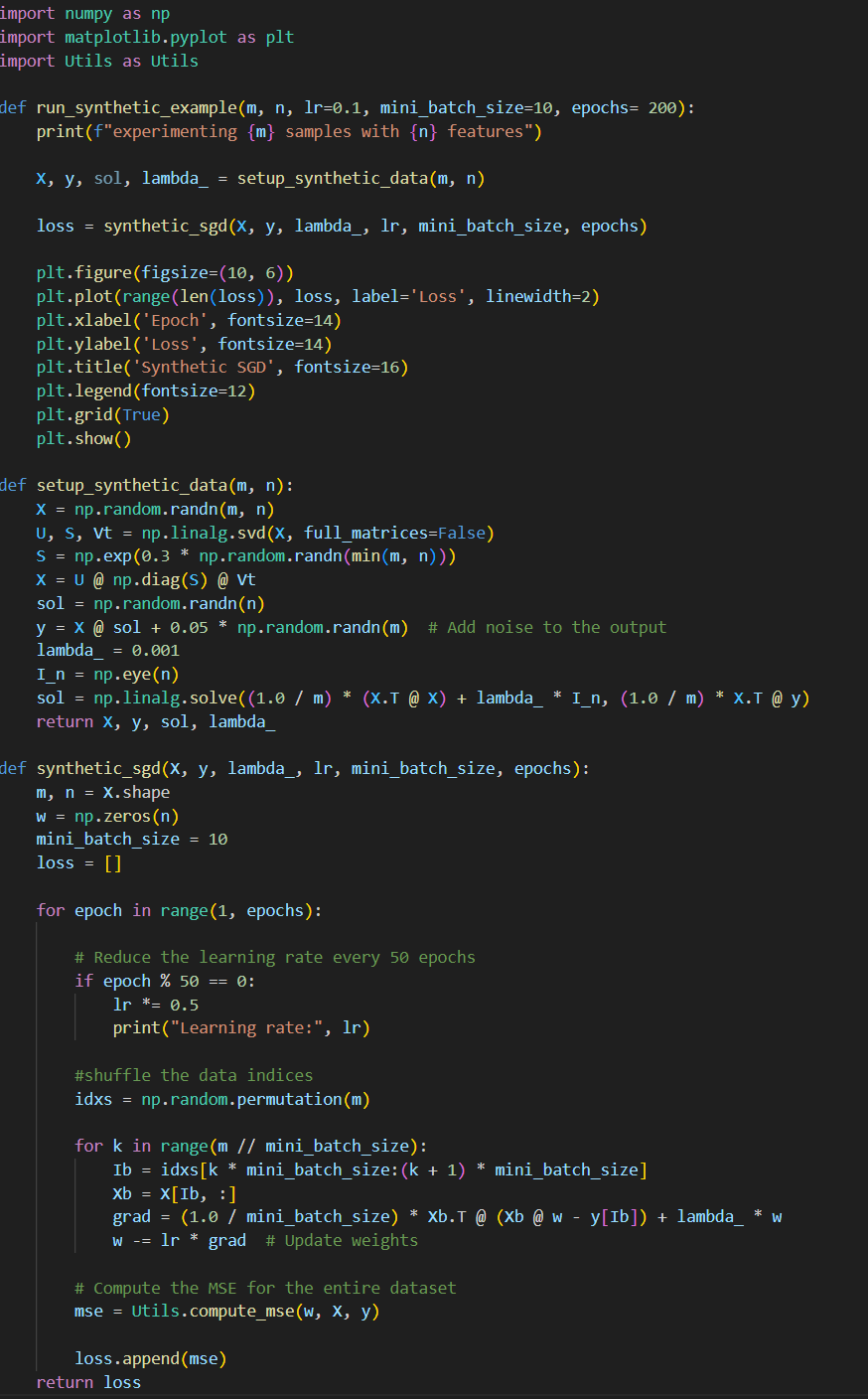


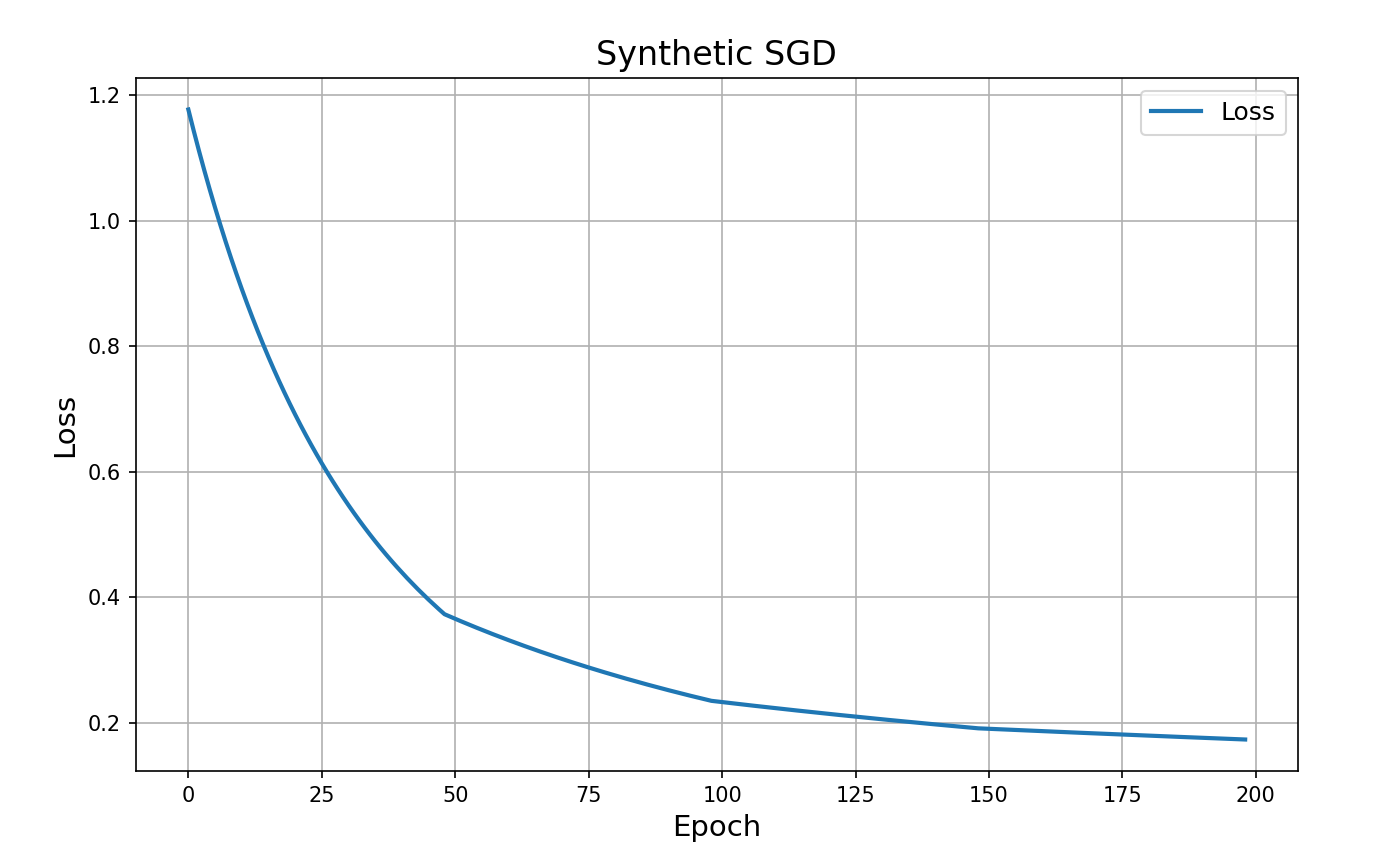
We can see in the result that the graph is linear with different slopes, so that the zero order decreases linearly (on a semilogarithmic scale) while the first order converges quadratically (on the same scale).

* 1. **Synthetic SGD**

We have implemented the SGD and tested it on a small synthetic data (as shown below), We implemented the data setup method as shown in the notes, the data we produced consisted of 100 samples with 200 features and the loss was calculated using mse.

The SGD learning rate was reduced by half every 50 epochs.





We can see that the SGD loss converges below around 0.17.

* 1. **Softmax SGD**

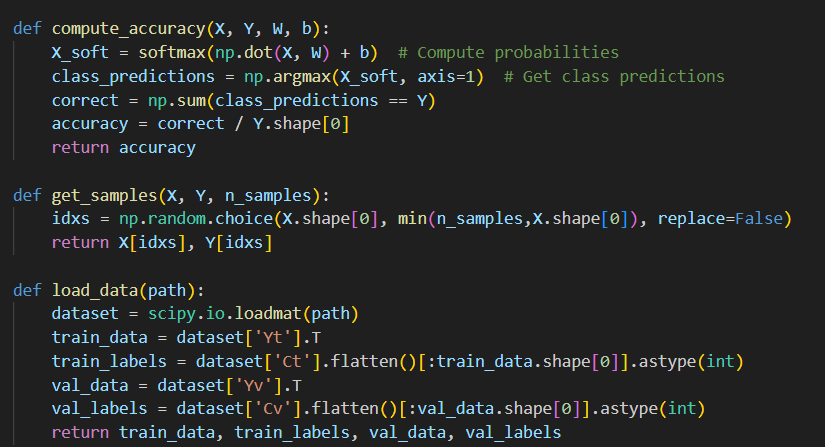
We've tried the following parameters:

* learning rates: [0.001, 0.01, 0.1, 1]
* mini-batch sizes: [10, 50, 100, 200]

For each data set we saved the best validation accuracy along with its parameters and the plot referring to that.

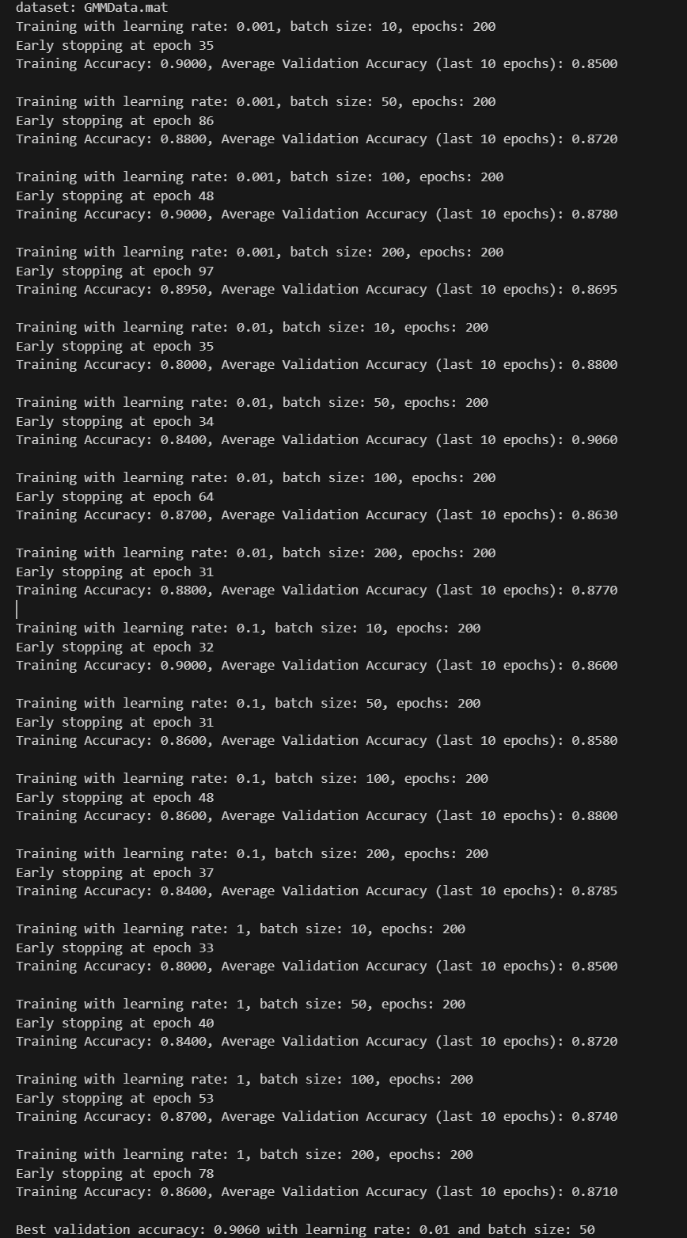
\*We have implemented in our code a mechanism which breaks the current SGD run whenever the validation accuracy failed to improve after 30 consecutive epochs.

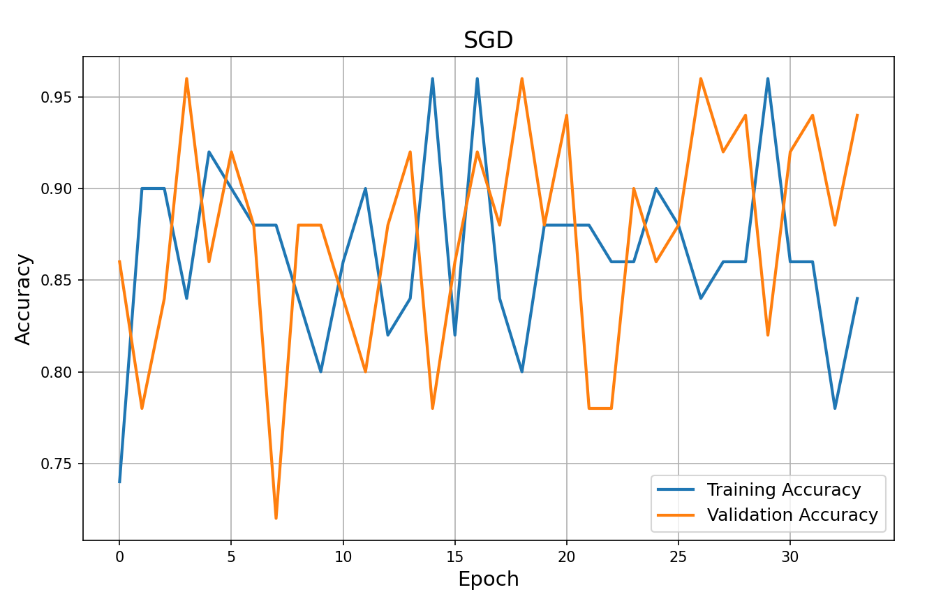




Results

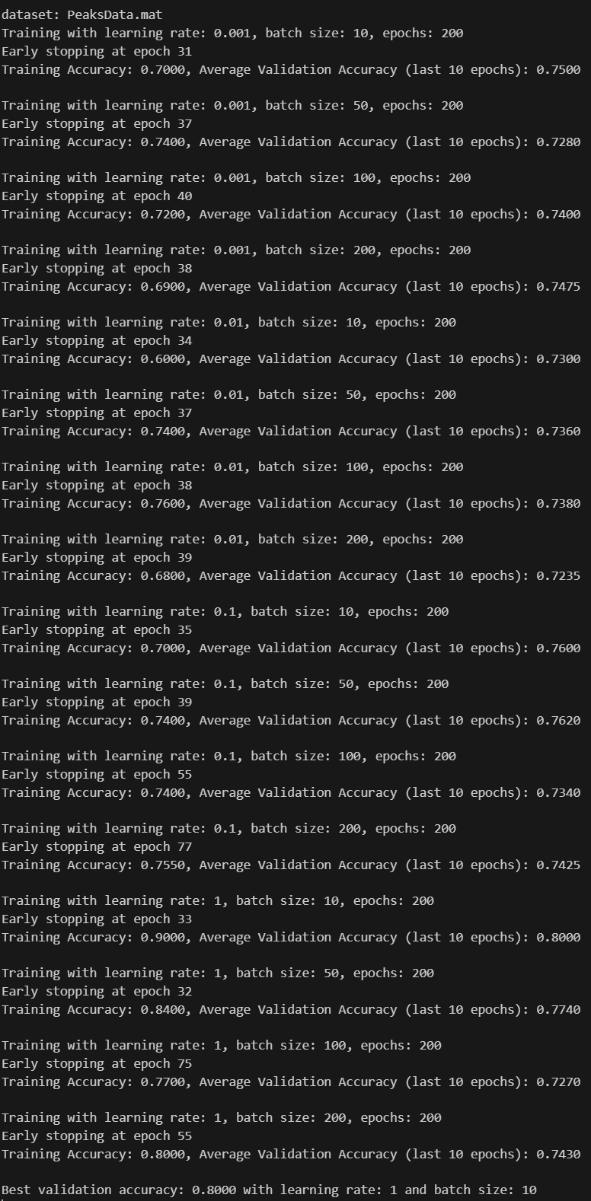
* GMMData
  + Best validation accuracy: 0.906
  + Best lr: 0.01
  + Best mini-batch size: 50

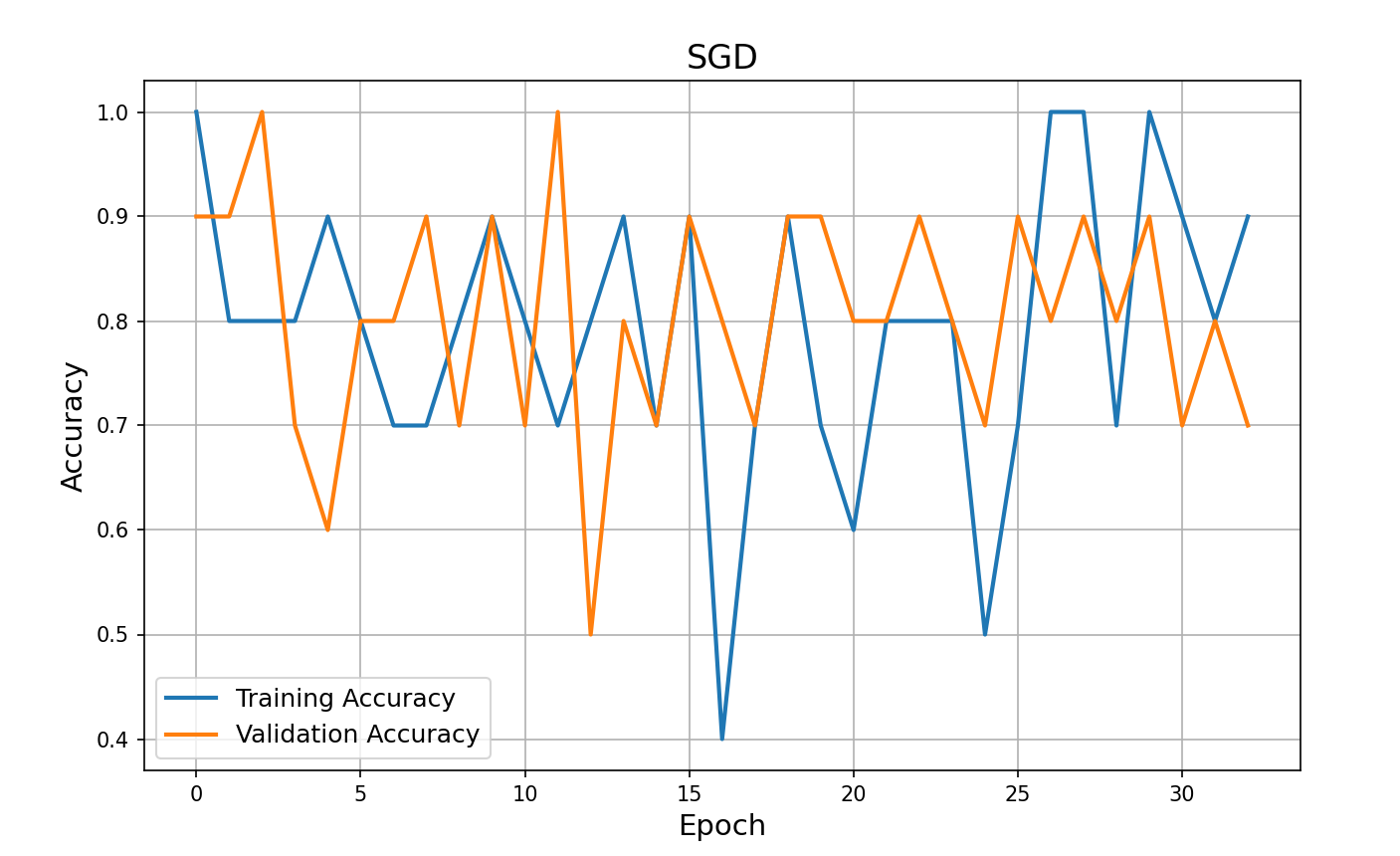




PeaksData

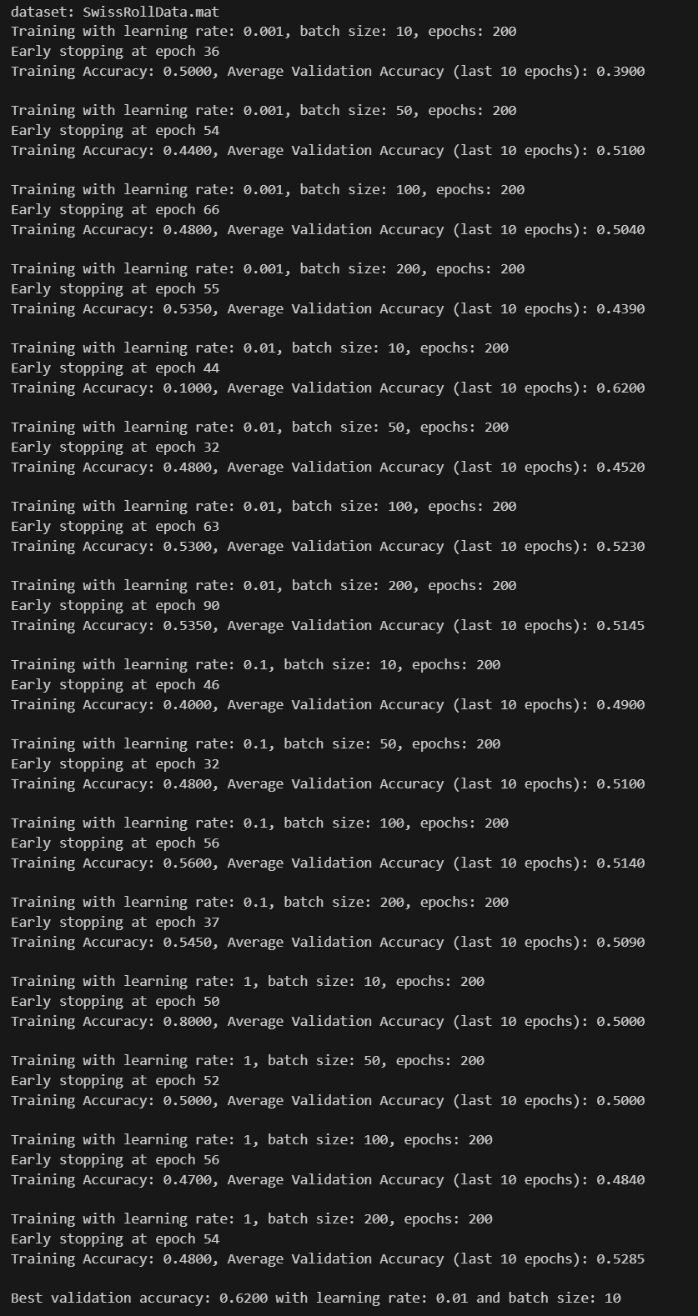
* + Best validation accuracy: 0.8
  + Best lr: 1
  + Best mini-batch size: 10

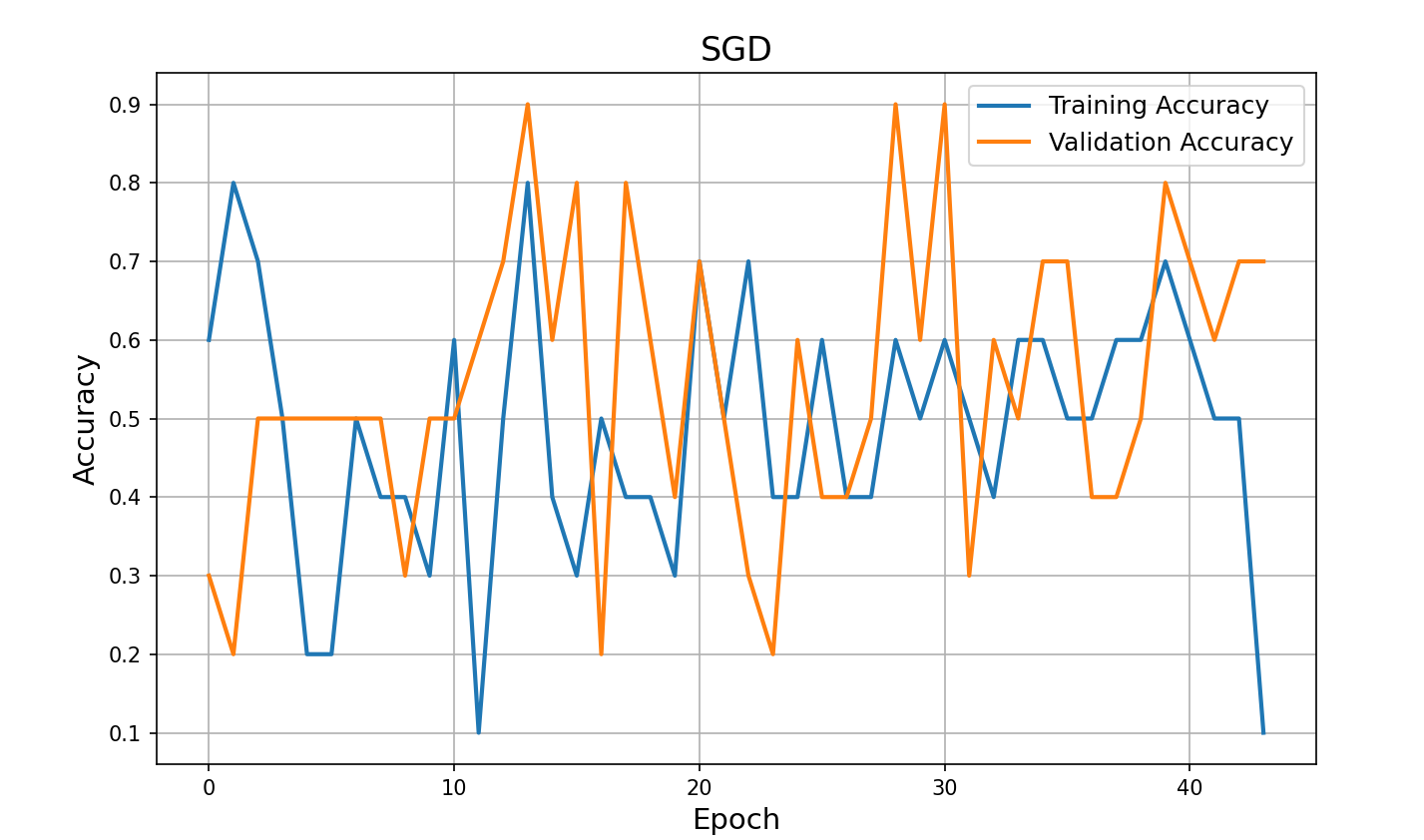




SwissRollData

* + Best validation accuracy: 0.62
  + Best lr: 0.01
  + Best mini-batch size: 10





1. **Part II: the neural network**
   1. **Neural network implementation**

Our neural network was implemented using a class called 'NeuralNetwork'.  
Our goal was to create one dynamic class which will hold all the methods and fields needed for creating, training and using a neural network model, therefor in its initialization the user can define its layers, the activation function (ReLU or TanH) and whether it is a resnet or not, the class initialize its weights and biases when created.  
In our implementation, we've implemented the weights and the biases as an array of the length of the layers provided as parameter.  
As seen in the SGD model, in the train method the model shuffles the indices to train on a different order of the data each epoch.

(NeuralNetwork Class)

Weve tested our backward and forward pass by using the jacobian test – we implemented the test by using the "direct Jacobian transposed test" as shown in the notes, we've tested 2 layers – the softmax layer and the regular model layer.

To use the "direct Jacobian transposed test" we've implemented the "grad test" as shown in class (with small changes of dimensions transformation).

(jac\_test\_layer, jac\_test\_softmax\_layer, grad\_test\_layer, plot\_grad\_test, plots)

* 1. **Resnet neural network**

As mentioned above, the implementation of the resnet model lays inside the NeuralNetwork class.

The resnet only accepts model which has atleast 2 hidden layers and the hidden layers are from the same size.

(resnet calculation only works between layers of the same size because the equation:   
Therefore the final summing must be of the same size.

Same as the "regular" model , we tested its backward and forward pass using the "direct Jacobian transposed test"

(NeuralNetwork Class, jac\_test\_resnet\_layer, plots)

* 1. **forward and backward pass of the whole network**

As shown in the plot the gradient test succeeded for testing the whole network, with 2 layers.

\*As described above, our weight and biases implementation are as a list of arrays

(plot)

* 1. **Running the network**

We tried the following parameters:

* Learning rates: [0.001, 0.01, 0.1]
* Mini-batch sizes: [32, 64, 128]
* Epochs: 500
* Activation function: ReLU
* Hidden layers: [[],[10], [10, 10, 10], [10, 10, 10, 10, 10], [50], [50,50,50]]

The result for each dataset is listed below:

* GMMData
  + Validation accuracy:
  + Lr:
  + Mini-batch size:
  + Hidden layers:
  + Runtime:
  + Resnet:

(plot)

* SwissRollData
  + Validation accuracy:
  + Lr:
  + Mini-batch size:
  + Hidden layers:
  + Runtime:
  + Resnet:

(plot)

s

In our conclusion we considered the validation accuracy, the loss, the model runtime and the architecture.

Conclusion