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

**Submitted by Lior Sharony, Eden Miran, Tomer Noama**

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

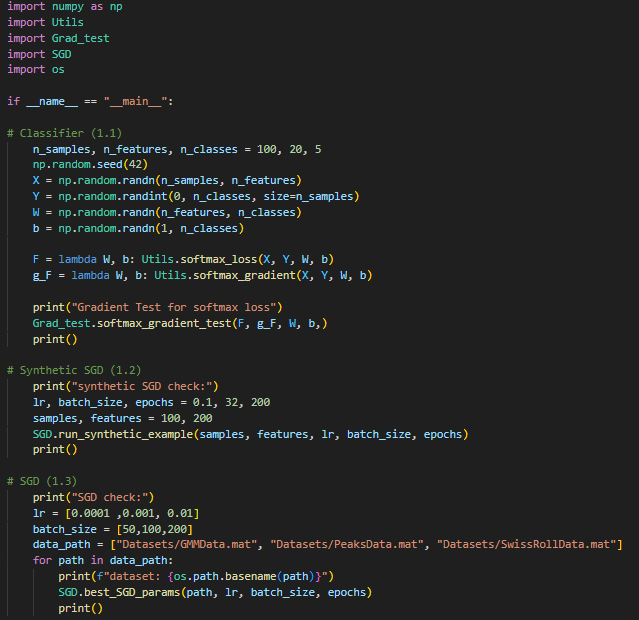
****

Fig 1: The main function of part 1

* 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).

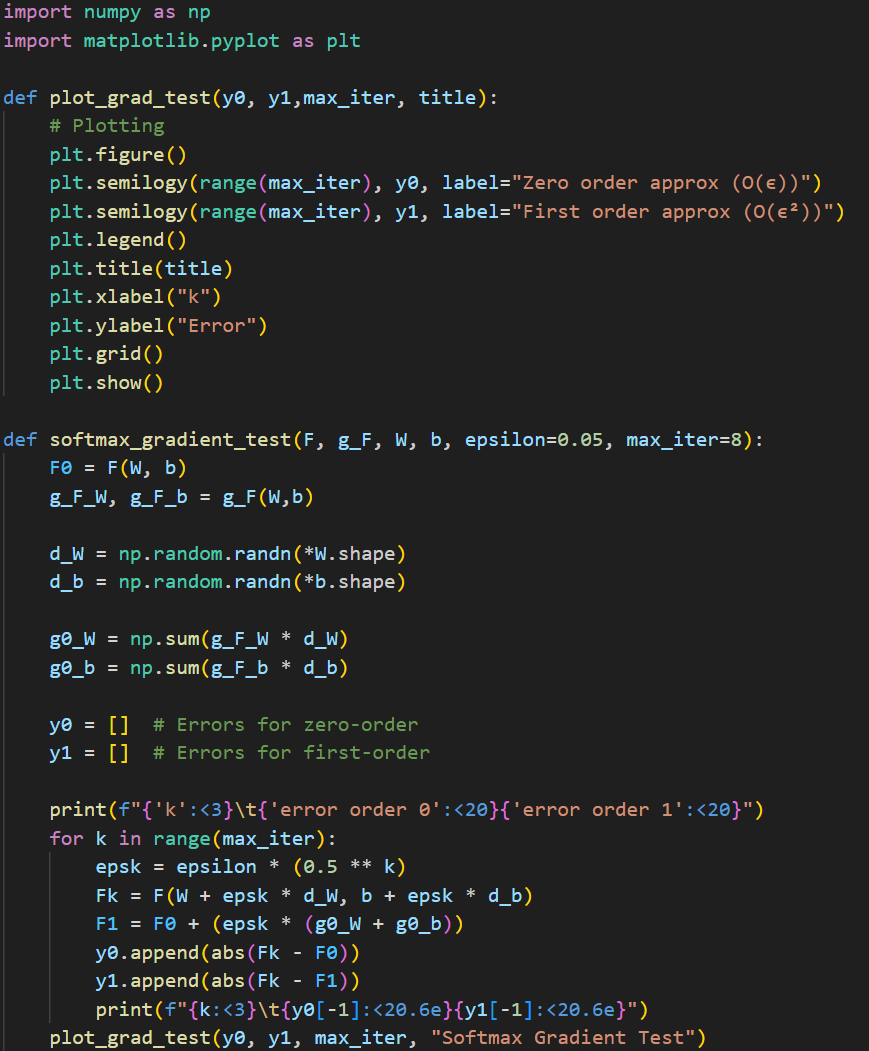


Fig 2: softmax function, loss and gradient

Fig 3: The gradient test code

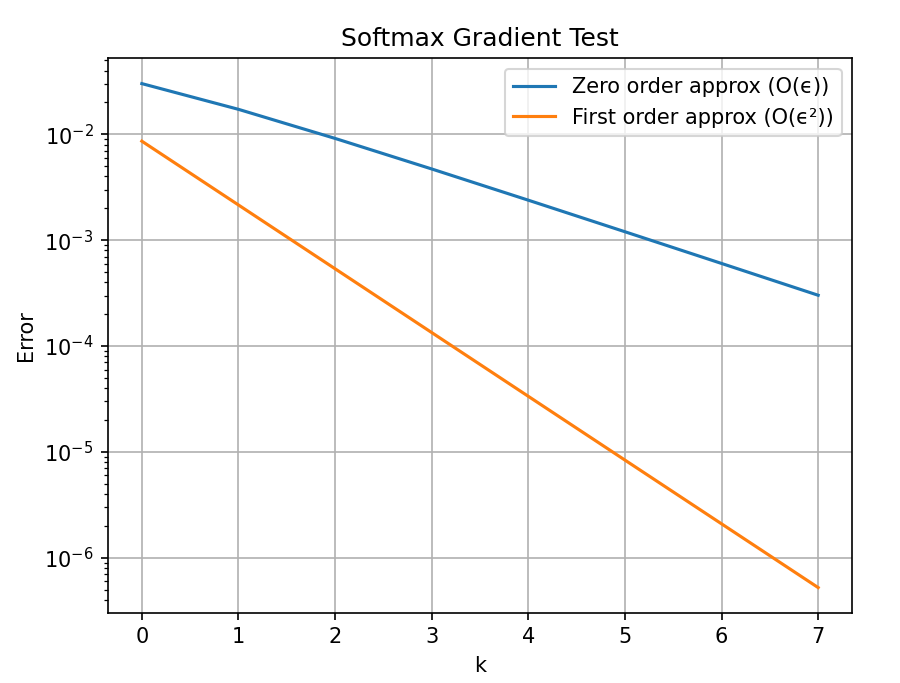


Fig 4: softmax gradient plot

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.

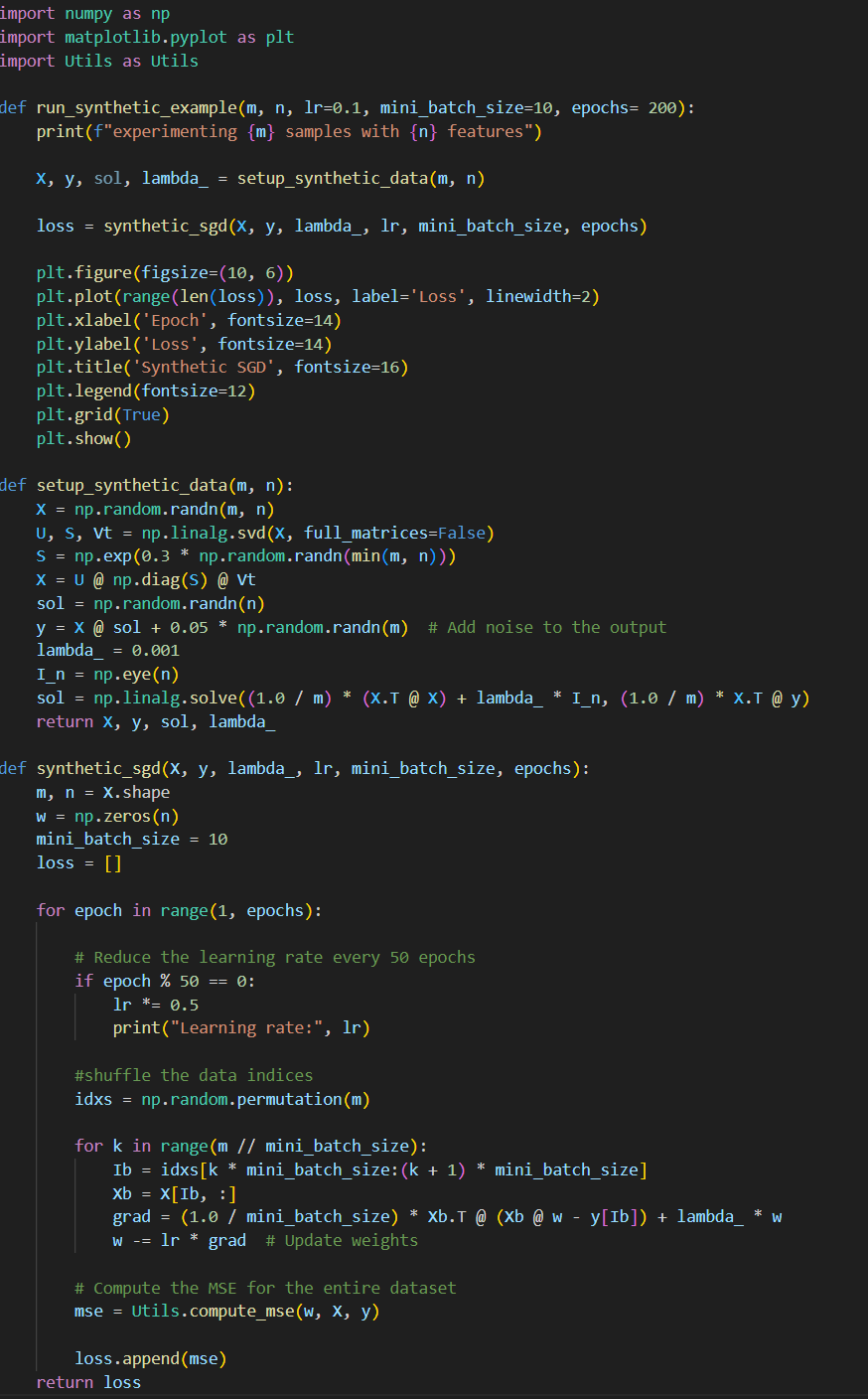


Fig 5: The synthetic SGD and the data setup

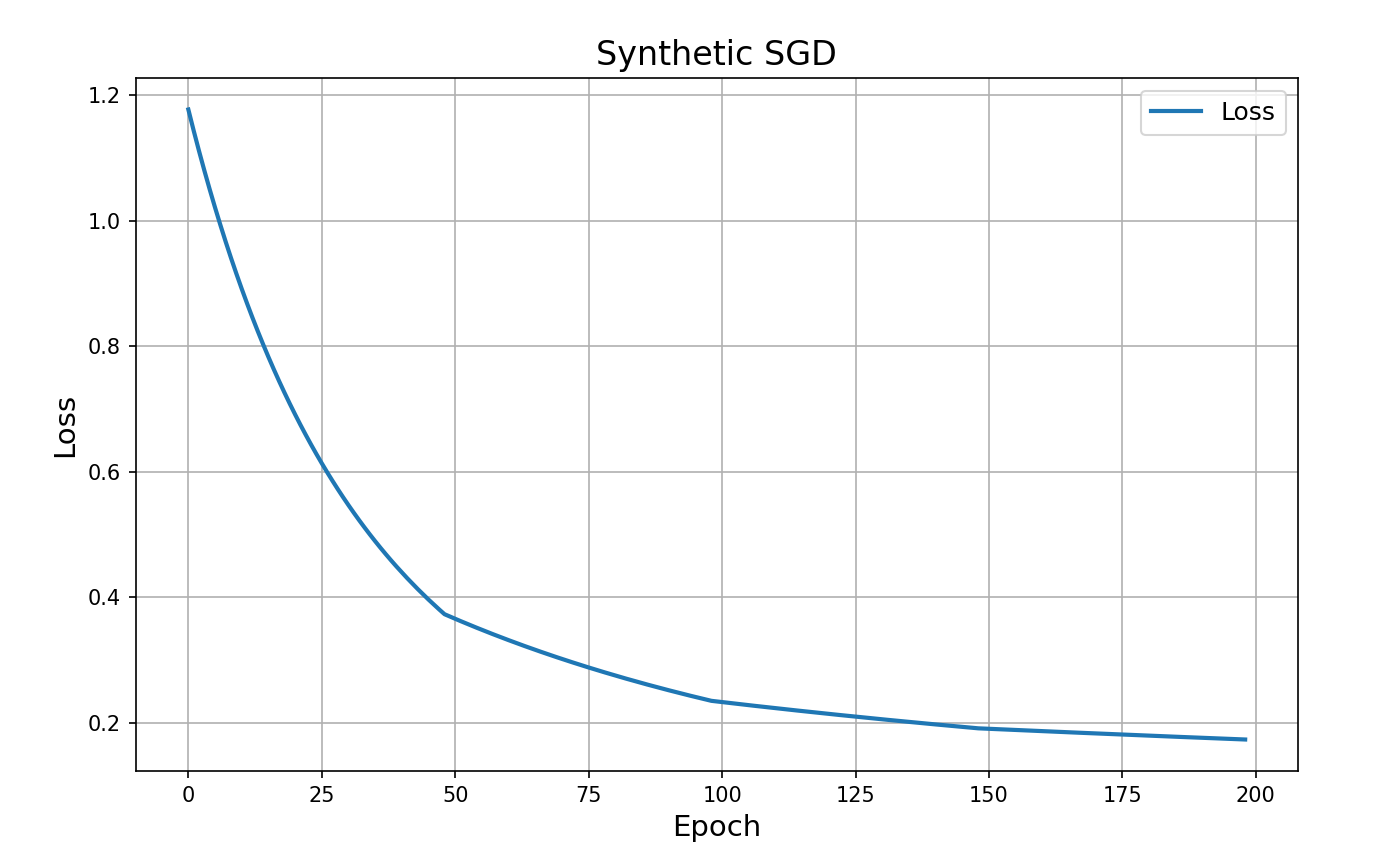


Fig 6: The Synthetic SGD loss plot

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

* 1. **Softmax SGD**

We've tried the following parameters:

* learning rates: [0.0001, 0.001, 0.01]
* mini-batch sizes: [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.

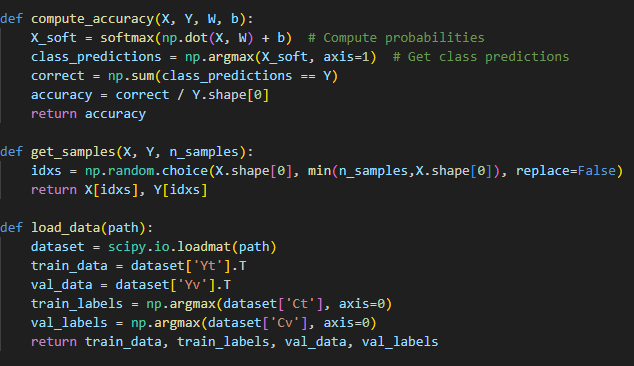


Fig 7: compute accuracy, get samples, load data methods

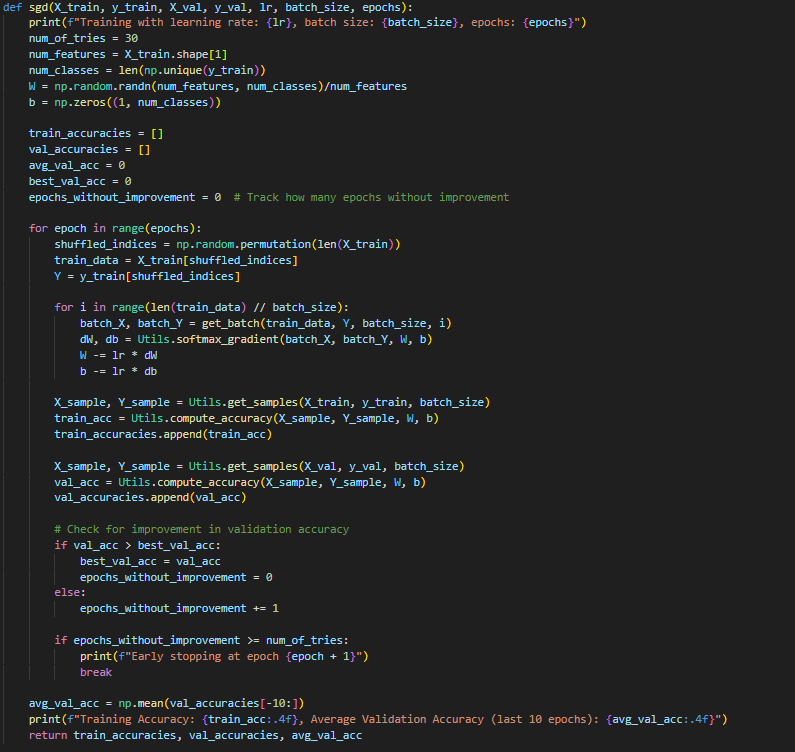


Fig 8: The SGD

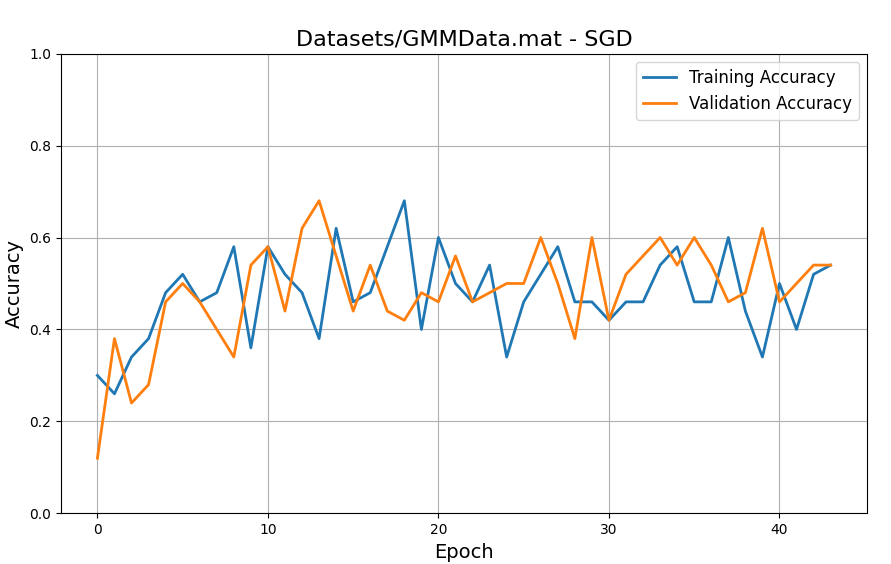
Results

* GMMData
  + Best validation accuracy: 0.5280
  + Best lr: 0.001
  + Best mini-batch size: 50

Fig 9: GMMData outputfor different parameters



Fig 10: GMMData SGD training and validation accuracy



PeaksData

* + Best validation accuracy: 0.5825
  + Best lr: 0.01
  + Best mini-batch size: 200

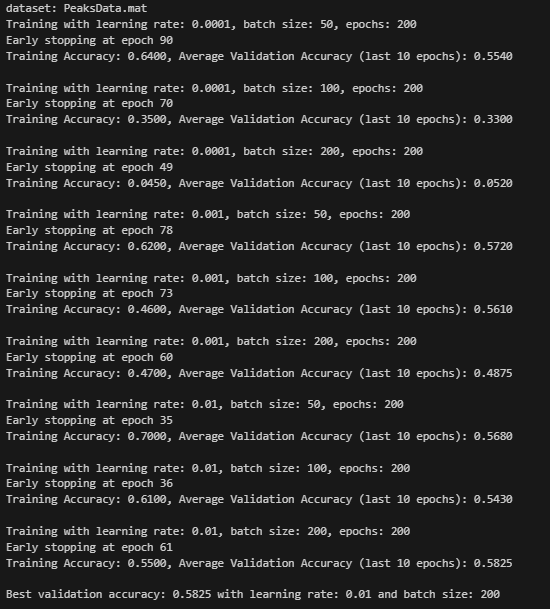


Fig 11: PeaksData outputfor different parameters

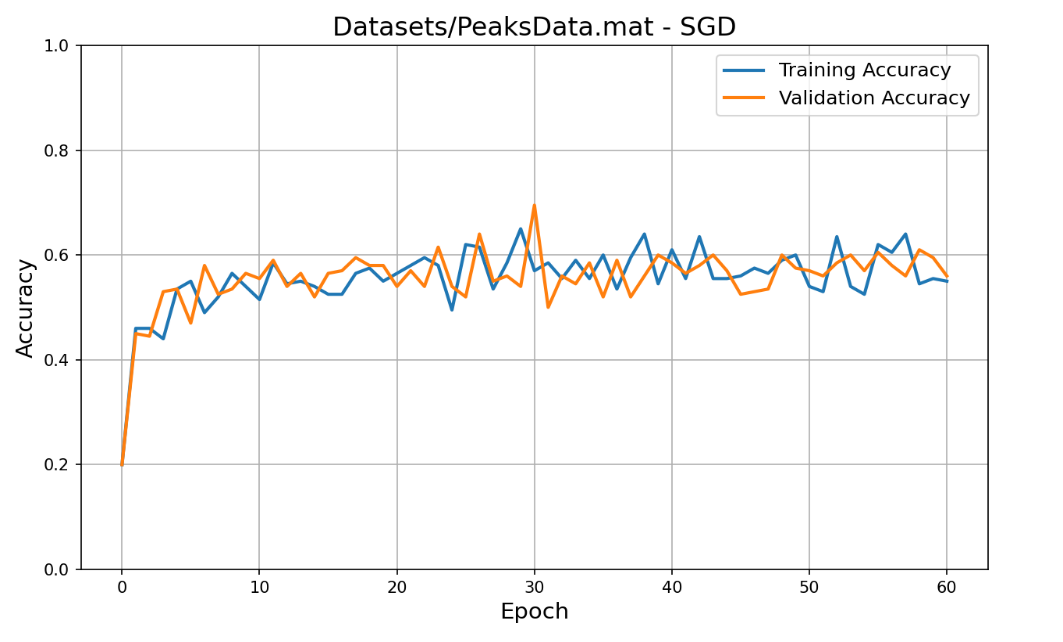


Fig 12: PeaksData SGD training and validation accuracy

SwissRollData

* + Best validation accuracy: 0.5345
  + Best lr: 0.01
  + Best mini-batch size: 200

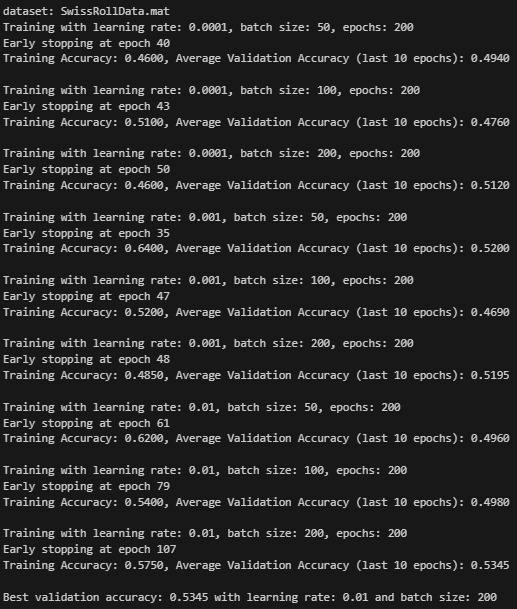


Fig 13: SwissRollData outputfor different parameters

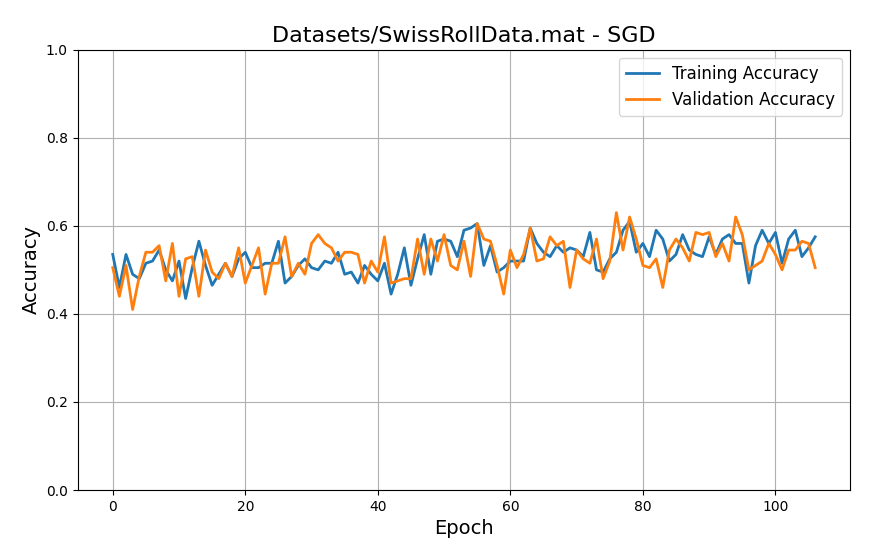


Fig 14: SwissRollData SGD training and validation accuracy

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

Our neural network implementation is encapsulated within a class called NeuralNetwork. The goal of this class is to provide a dynamic, flexible structure that integrates all the necessary methods and fields for creating, training, and using a neural network model.

**Key Features:**

1. **Initialization:**
   * The user specifies the network's architecture by providing the layer structure, the activation function (ReLU or TanH), and whether the network includes residual connections (ResNet).
   * During initialization, the class automatically sets up:
     + **Weights:** Initialized with random values and normalized to ensure ∣∣W∣∣=1, promoting numerical stability.
     + **Biases:** Initialized as zero vectors.
   * If ResNet is enabled, the initialization process includes additional validations:
     + All hidden layers must have the same size to ensure compatibility with residual connections.
     + The network must include at least two hidden layers. If either condition is not met, an error is raised.
2. **Dynamic Training:**
   * The train method implements stochastic gradient descent (SGD) and ensures data variety during training by shuffling the dataset indices at the start of each epoch. This prevents the model from overfitting to a fixed data order and improves generalization.

This unified design ensures that the NeuralNetwork class is highly reusable, allowing users to easily define, customize, and train their models with minimal setup.

תמונה שמכילה טקסט, צילום מסך

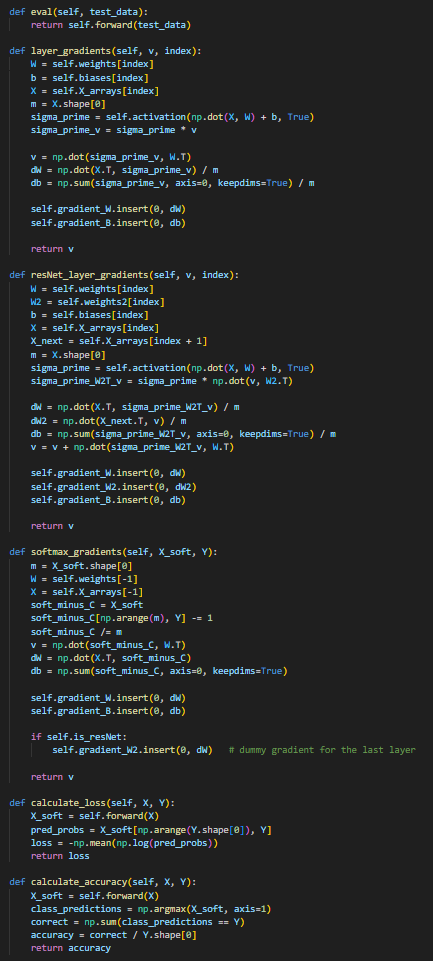
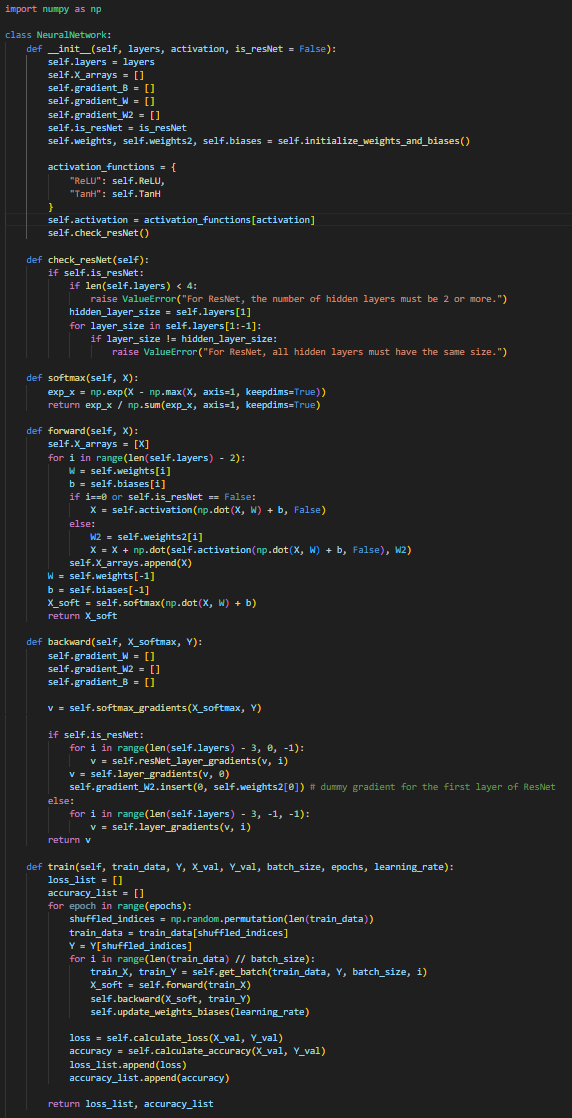
התיאור נוצר באופן אוטומטי

Fig 15-17: The neural network class

We validated the correctness of our forward and backward passes using the Jacobian test. Specifically, we implemented the **Direct Jacobian Transposed Test** as outlined in the course notes.

We applied this test to two components of our model: Softmax layer and Regular model layer.

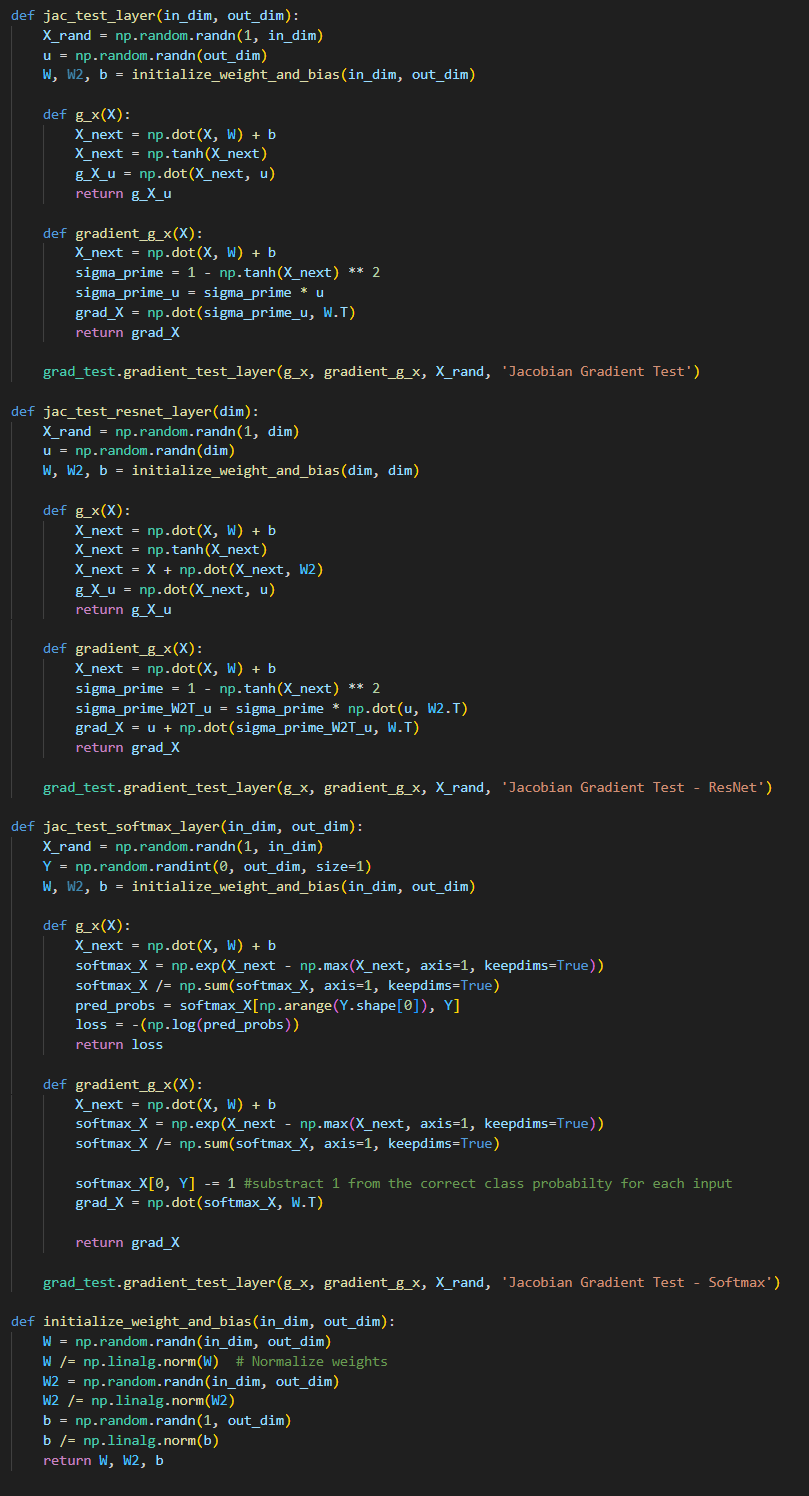


Fig 18: Jacobean test for different layers

To use the **Direct Jacobian Transposed Test** we implemented the "grad test" as shown in class (with small changes of dimensions transformation).

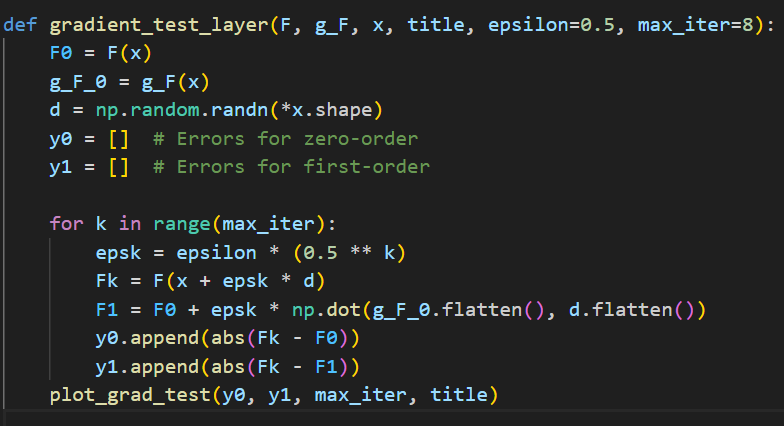


Fig 19: the gradient test

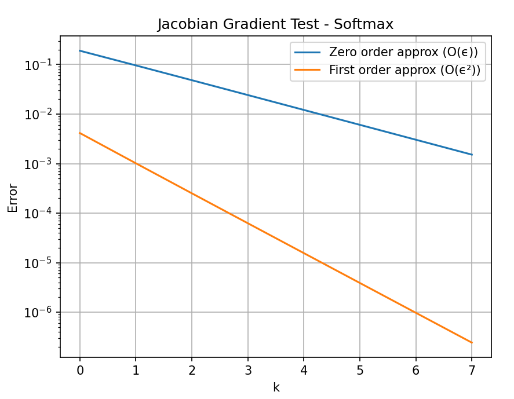
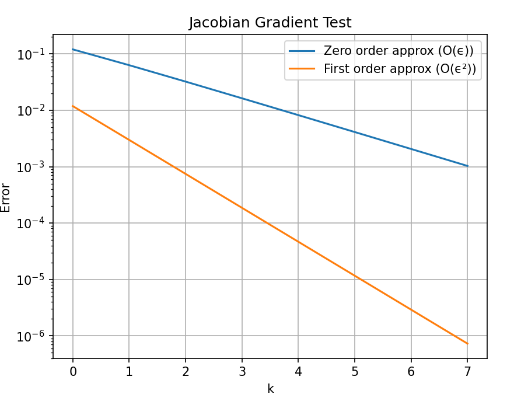
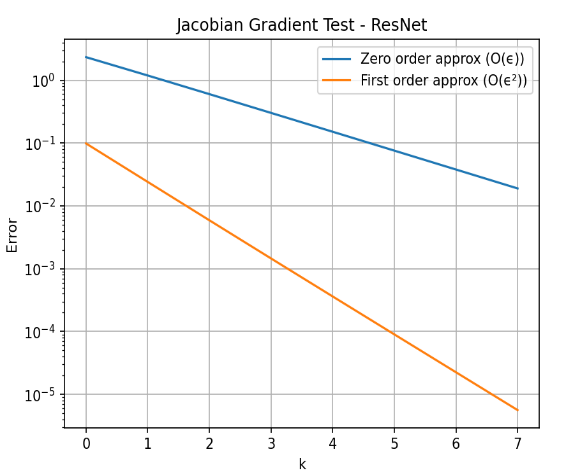


Fig 20-22: the gradient test plots for different layers

* 1. **Residual Neural Network**

As previously mentioned, the implementation of the ResNet model is integrated within the NeuralNetwork class. The ResNet architecture imposes specific constraints on the model's structure:

* The network must have at least **two hidden layers**.
* All hidden layers must be of the **same size** to ensure compatibility with residual connections.

Like the "regular" model, the ResNet implementation was rigorously tested for correctness in both forward and backward passes using the **Direct Jacobian Transposed Test**.

\* The ResNet architecture relies on the following equation for residual connections:

For the addition to be valid, the dimensions of and the output of must match. Hence, all hidden layers must maintain the same size to ensure dimensional consistency.

* 1. **Neural Network – Gradient Test**

We validated the entire network using the Gradient Test, where we defined the complete forward pass (including the loss function) as the function and the entire backward pass as its gradient.

The test was conducted on a network with two hidden layers, ensuring that both the forward and backward computations were correctly implemented.

As illustrated in the plot below, the gradient test confirmed the accuracy of the network's computations, demonstrating successful validation of the entire architecture.

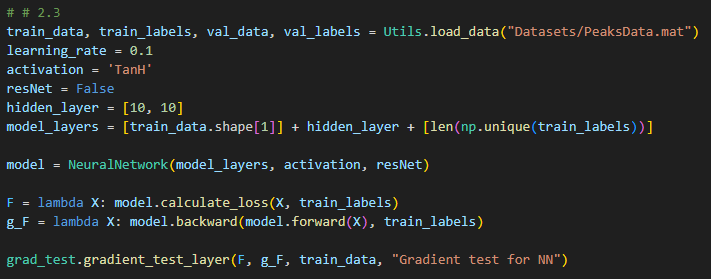


Fig 23: code implementation for gradient test on a full Neural Network

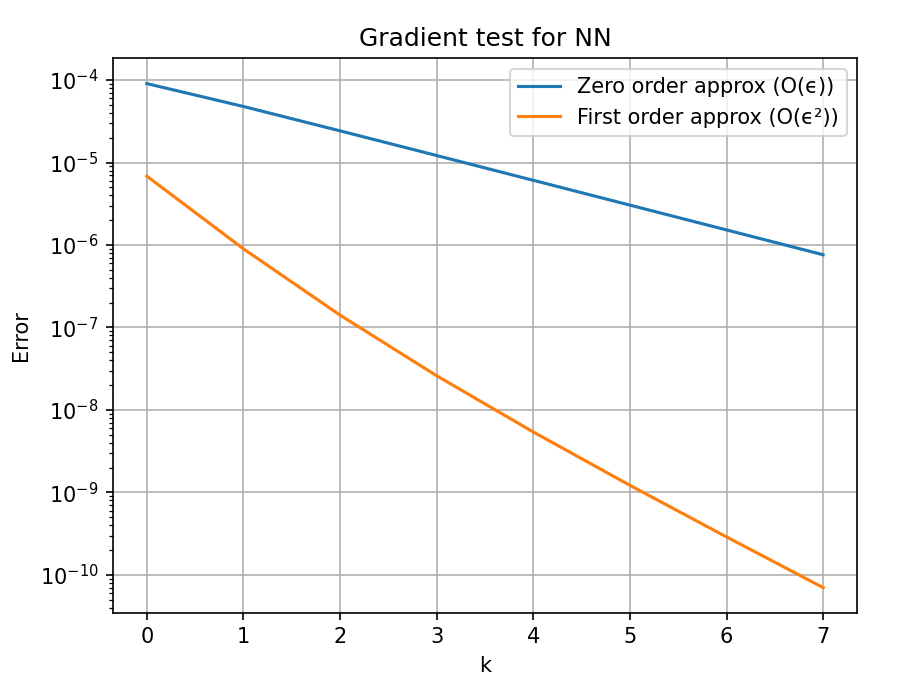


Fig 24: the gradient test for the Neural Network

* 1. **Neural Network experiments with different parameters**

We ran our NN with the datasets GMMData and PeaksData with the following parameters:

* Learning rates: [0.1, 0.01, 0.001]
* Mini-batch sizes: [32, 64, 128]
* Epochs: 200
* Activation function: ReLU
* Hidden layers: [[],[10], [10, 10, 10], [10, 10, 10, 10, 10], [50], [50,50,50]]

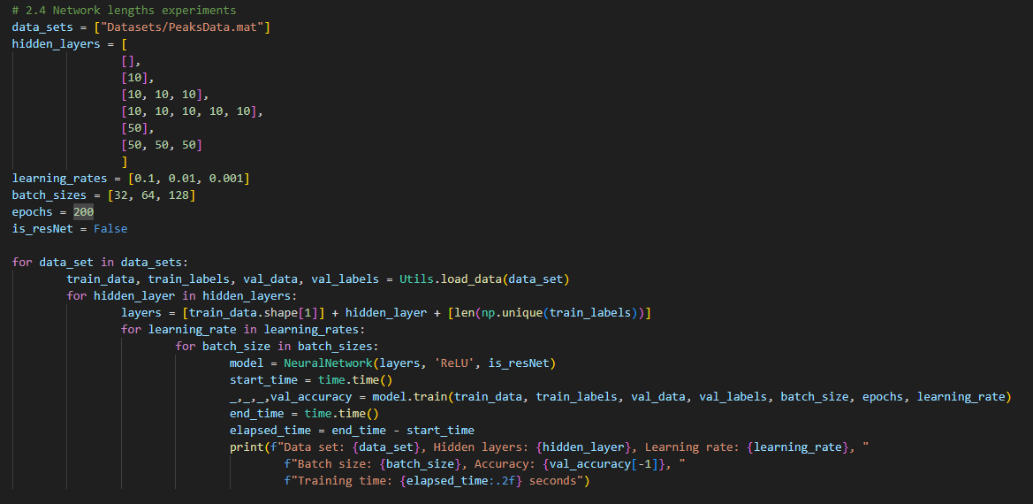


Fig 25: code implementation for running the NN with different parameters

The results given by running the code given above:

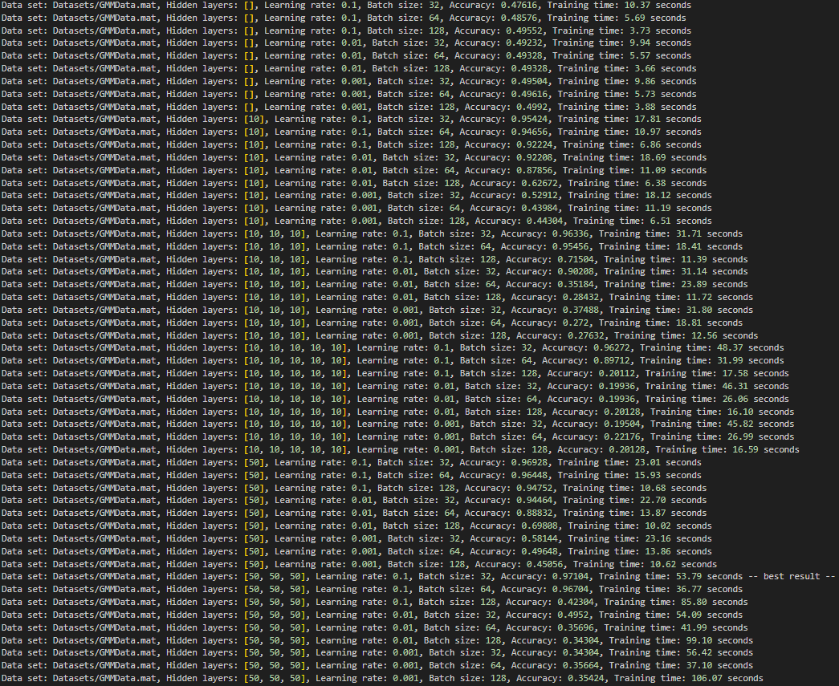


Fig 26: GMMData without ResNet

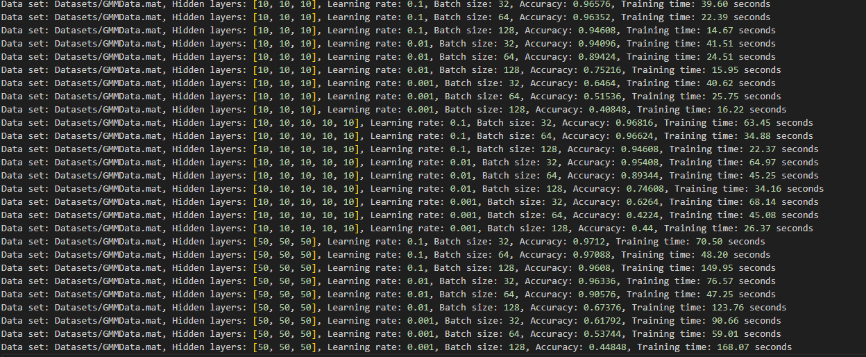
–

Fig 27: GMMData with ResNet

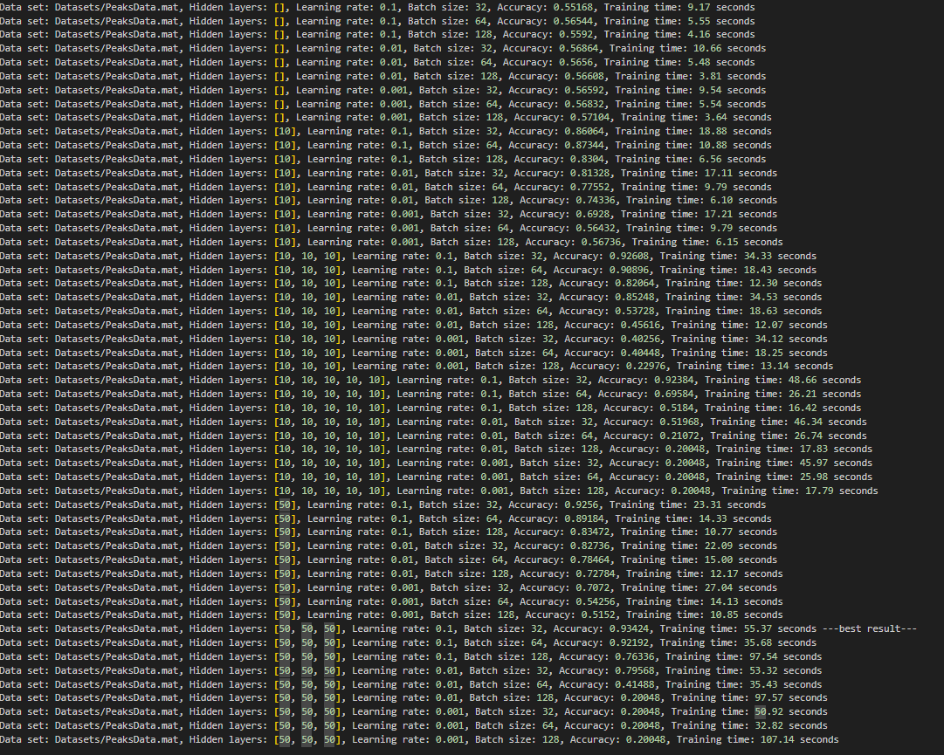


Fig 28: SwissRollData without ResNet

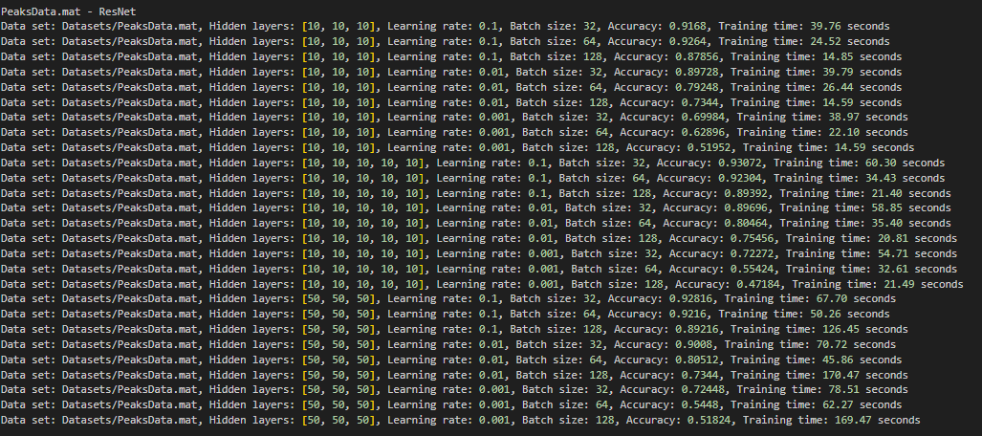


Fig 29: SwissRollData with ResNet

The **best** results achieved for each dataset are listed below:

* GMMData
  + Validation accuracy: 0. 97104
  + Lr: 0.1
  + Mini-batch size: 32
  + Hidden layers: [50, 50, 50]
  + Runtime: 53.79
  + Resnet: No

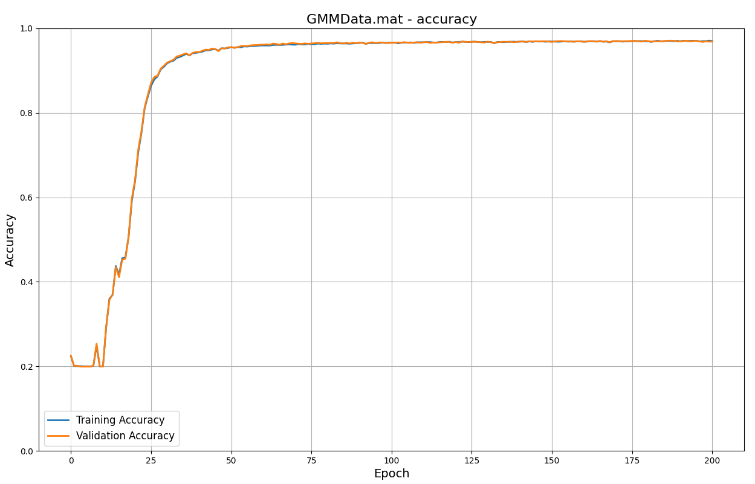


Fig 30: Training and Validation Accuracies

* PeaksData
  + Validation accuracy: 0. 93424
  + Lr: 0.1
  + Mini-batch size: 32
  + Hidden layers: [50, 50, 50]
  + Runtime: 30.73
  + Resnet: No

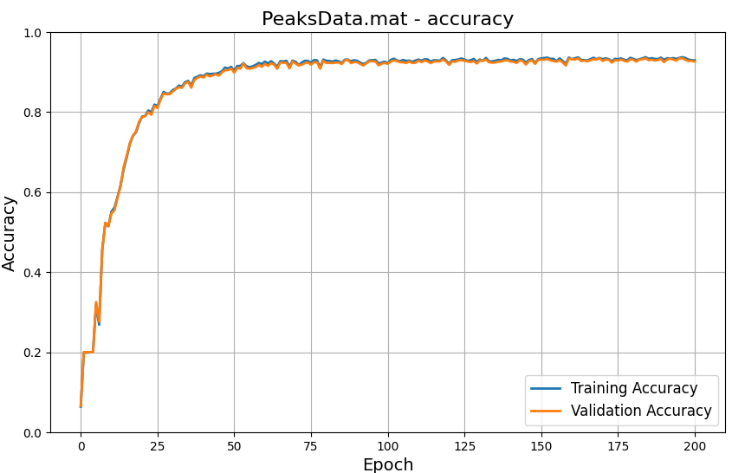


Fig 31: Training and Validation accuracy

**Conclusion**

**Learning Rate**: The best accuracy was consistently achieved with a learning rate of 0.1, as smaller learning rates tended to make smaller adjustments by the gradient, resulting in poorer performance.

**Batch Size**: Smaller batch sizes resulted in better accuracy, but this came at the cost of increased training time. While using smaller batches led to more refined updates and higher performance, the trade-off was a slower runtime.

**Hidden Layers**: Deeper and narrower networks generally outperformed shallower or wider ones, offering better accuracy. However, networks that were too deep encountered vanishing gradient issues, which negatively impacted their performance. Therefore, a balanced architecture with an optimal number of layers provided the best results.

**ResNet**: ResNet consistently provided excellent performance across all configurations, though it came with the trade-off of slower training times.

* For the GMMData and PeaksData dataset, both ResNet and non-ResNet neural networks delivered strong results, but the non-ResNet model achieved better runtime efficiency.
  1. **Neural Network with constrained parameters**

For computing the number of parameters of our NN we used a simple function which multiply the number of weights from each layer, and sums with the biases.

For a resnet NN, we can assume that all the hidden layers are from the same size, thus we power the size of the first hidden layer by 2, times the number of hidden layers.

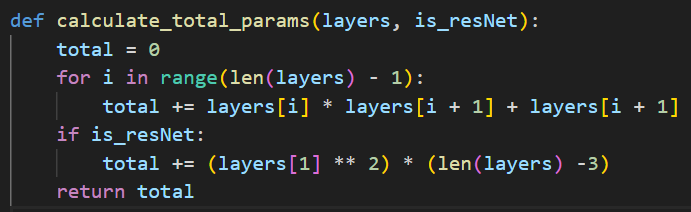


Fig 32: The function for calculating the total parameters

Our strategy was to start from a shallow and wide network, and in each test we narrowed it and made it deeper (in consideration of the parameters constraint –for both PeaksData and GMMData we had a maximum of 100\*5 = 500 parameters.

We conducted the test with the following parameters:

* Lr : 0.1
* Mini-batch size: 32
* Epochs: 200
* Activation function: ReLU

Architecture (500 parameters):

* Non resnet hidden layers:
  + [45]
  + [17,17]
  + [5,10,12,15]
  + [15,12,10,5]
  + [6,6,6,6,6,6,6,6,6,6,6]
* Resnet hidden layers:
  + [13,13]
  + [9,9,9]
  + [5,5,5,5,5,5,5,5,5]

\*The reason for the architecture of the 3rd and 4th layer options for non ResNet networks is to check if the order of the hidden layers effects the accuracy.

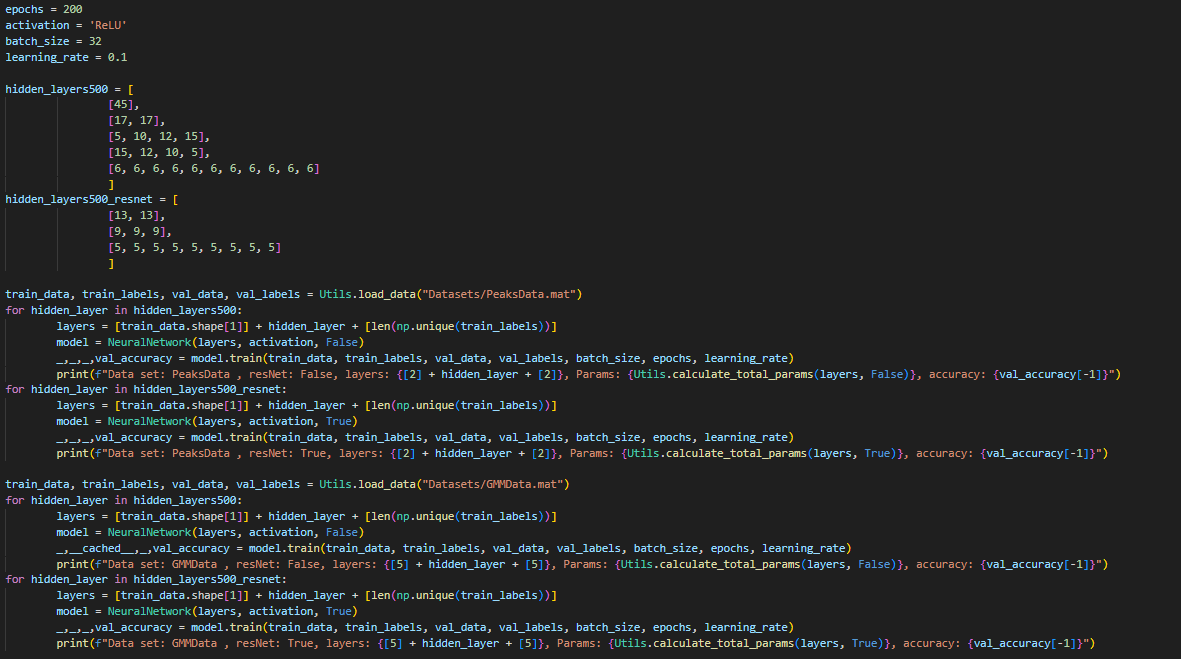
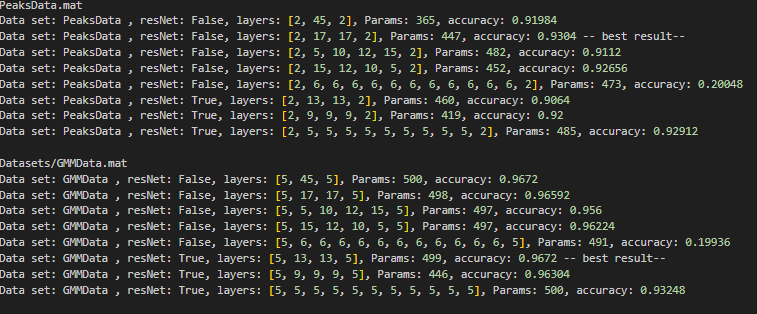


Fig 33: code implementation for running NN with parameters constraints

Fig 34: Results given by running Fig 33



Best Results:

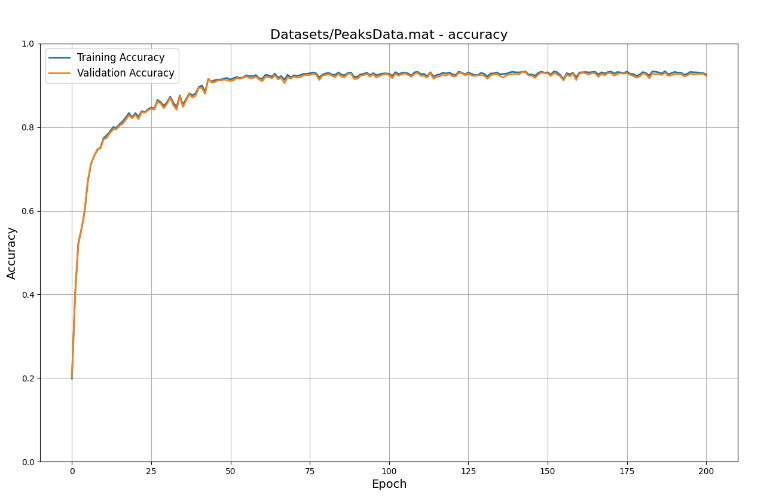
* PeaksData
  + Validation accuracy: 0. 9304
  + Hidden layers: [17, 17]
  + Params: 447
  + Resnet: No

Fig 35: Training and Validation Accuracies

* GMMData
  + Validation accuracy: 0. 9672
  + Hidden layers: [13, 13]
  + Params: 499
  + Resnet: Yes

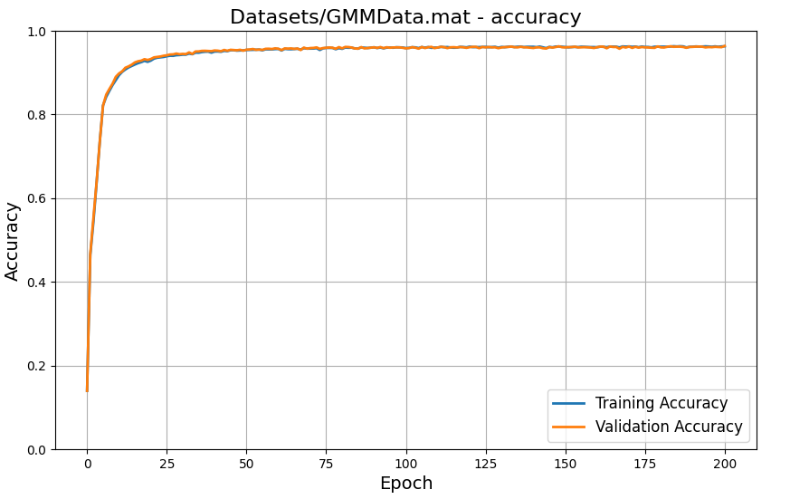


Fig 36: Training and Validation Accuracies

**Conclusion:**

Under parameter constraints:

* **Shallow and wide networks generally outperform deep and narrow networks**, as they are better at utilizing limited parameters to represent complex functions.
* **Both ResNet and non-ResNet architectures achieved same results,** though it is worth mentioning that when trained with deep and narrow network, due to the vanishing gradients problem, ResNet achieved much better results.