Practical Deep Learning
Assignment 1

Run as script:

```
usage: main.py [-h] [-task TASK] [-path PATH]

optional arguments:
  -h, --help  show this help message and exit
  -task TASK  Choose the task you want to run: task1 | task3 | task6 | task7
  -path PATH  Insert the absolute/relative prefix path for folder that the data is lying
```

Implementation details:

- Software engineering abstraction:
    - All the components in the network including the network, implement Module abstract class with 3 abstract methods:
        - forward - returns the output of the module.
        - backward - returns the output of the module in the backpropagation (the derivative).
        - parameters - returns the parameters of the module.

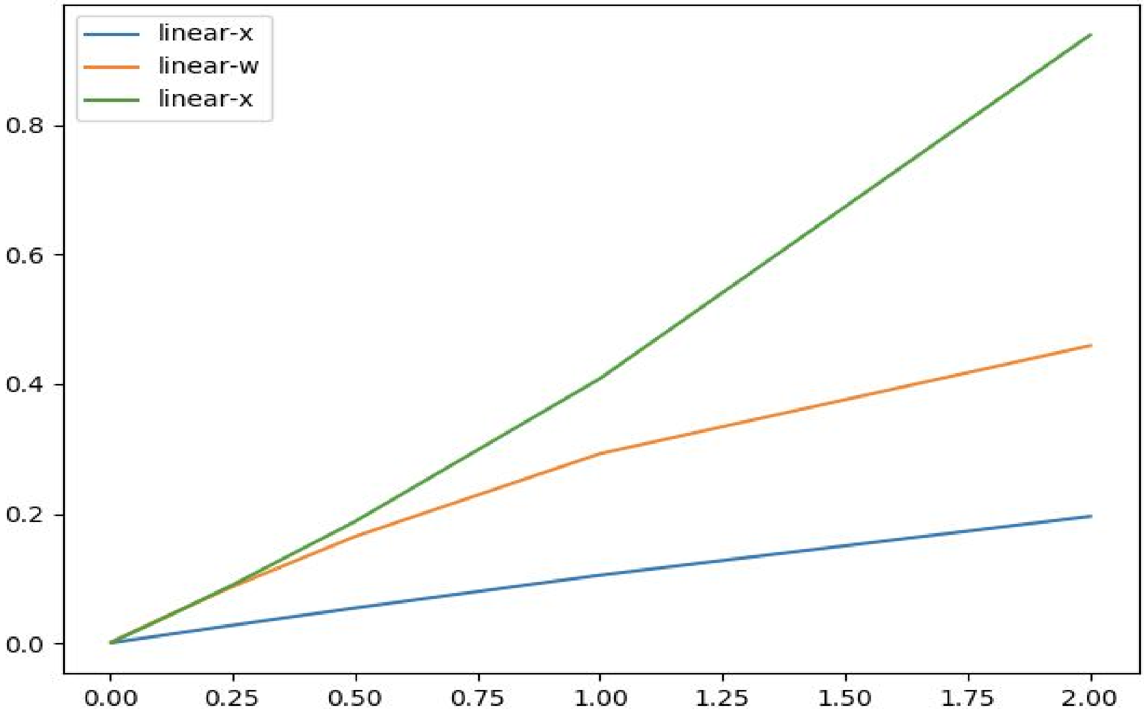
- Central classes:
    - Linear: implements Module,
        - constructor: randomly init the multiplication matrix and the bias vector.
        - forward: implements the forward pass of a linear layer following the non-linear activation function.
        - backward: computes the gradient of the layer multiplied by external v, which is the derivative of a forward layer.
    - NeuralNetwork: implements Module,
        - constructor: init a list of Linear layers as a default architecture of a neural network, possible to pass “use_arch=False” and use “add_layer” method.
        - forward: implements the forward pass of a the whole net.
        - backward: implements the backpropagation algorithm.
        - parameters: aggregates a reference for all the parameters in the network.
    - SGDOptimizer:
        - constructor: receive as an input the parameters to optimise.
        - update: update the parameters by a given learning rate and gradients.
        - optimize(f: NeuralNetwork, loss: Module, data, epochs=250, lr=1e-01, plot=False, dl: DataLoader = None, calc_acc=False):
          implements SGD optimisation method with a given neural network and a loss function.

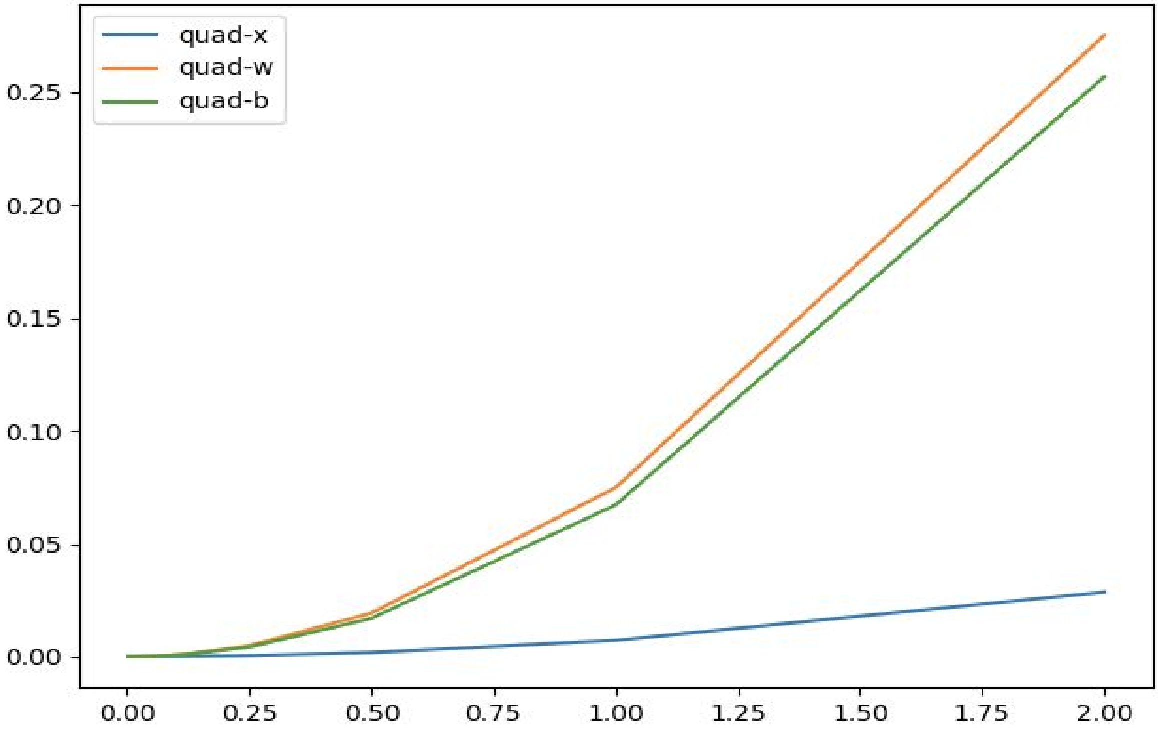
Task1:

We checked our gradient test for one example.

Details:

- Data: one example with 4 dimensions [1,2,3,4], label 0.



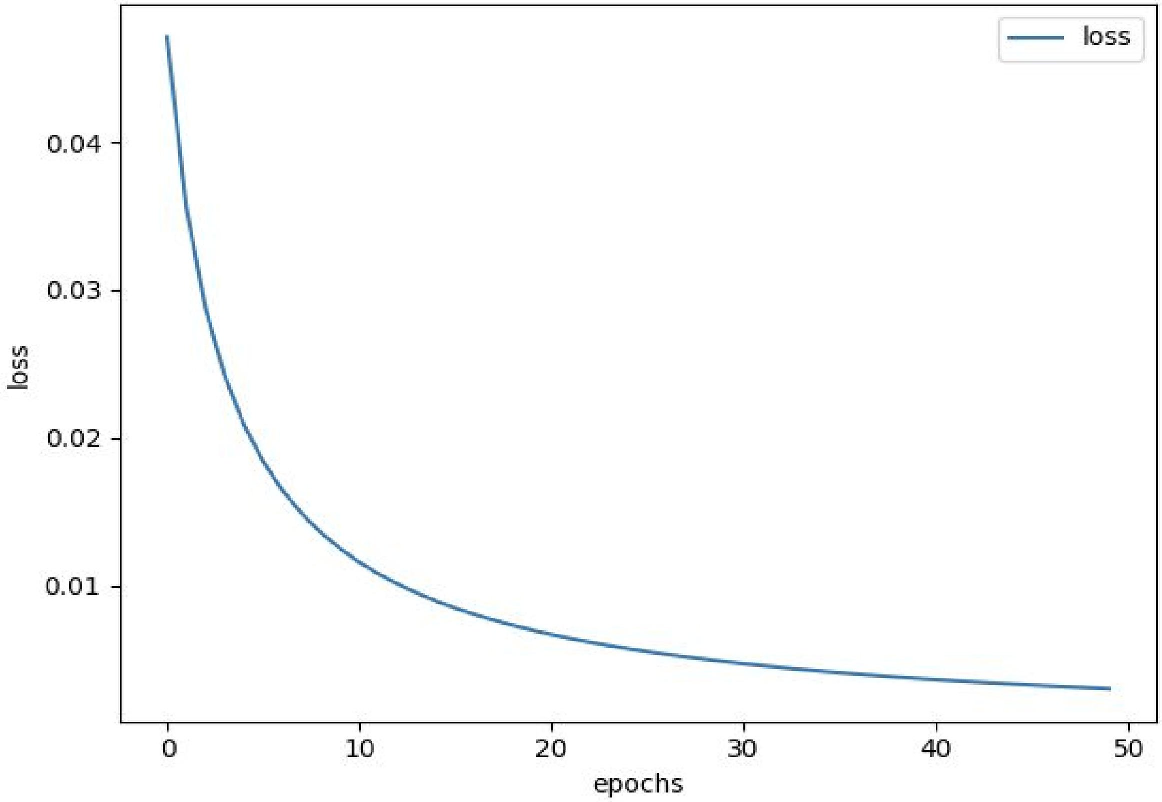


Task 3:

We first checked our SGD implementation on one example (which expected to overfit after a few epochs) with the softmax objective.

Details:

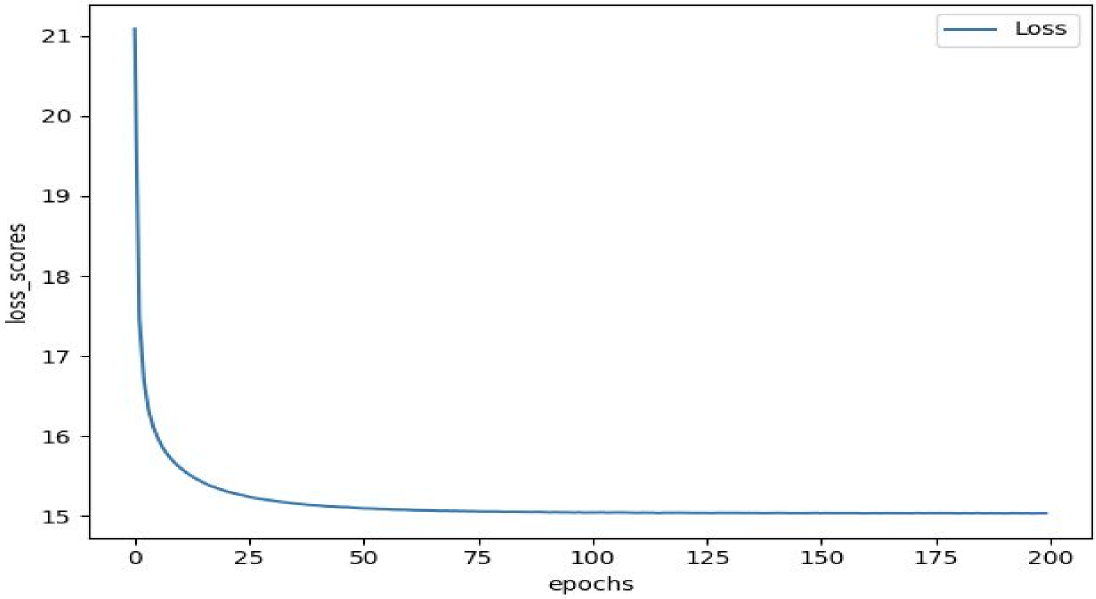
- Data: one example with 4 dimensions [1,2,3,4], label 0.
- Epochs: 50
- Learning rate: 0.1



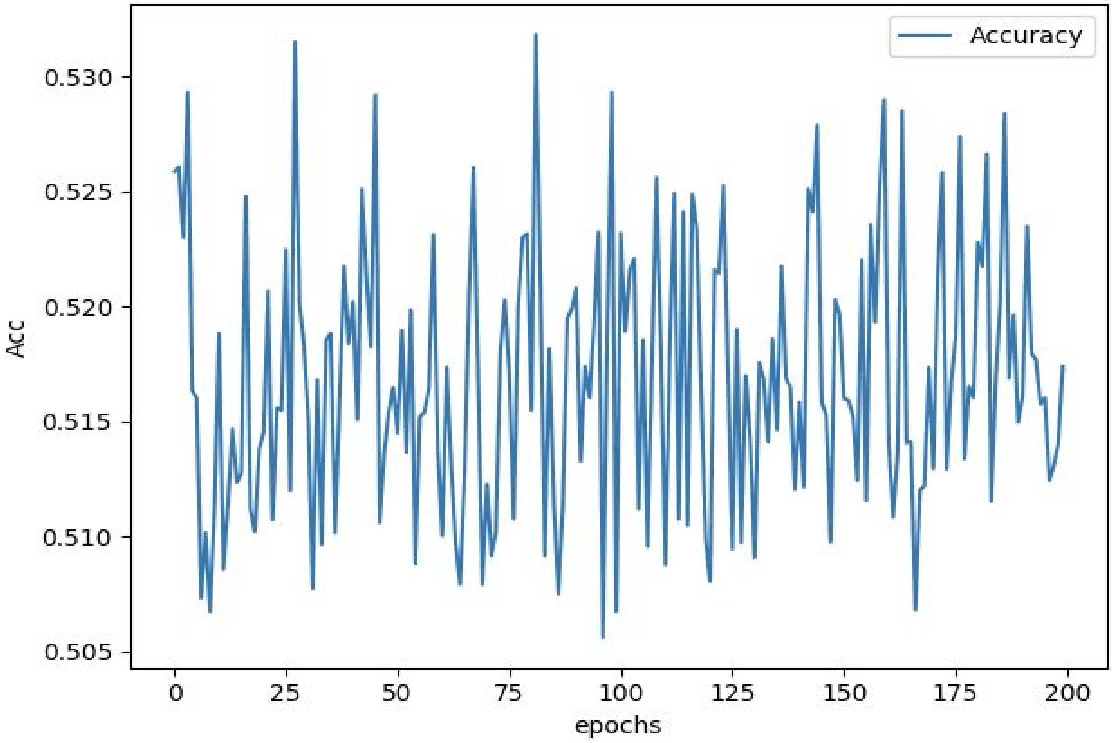
Next, we check this on real data:

Details:

- Data: GMM Data, 5 classes.
- Epochs: 200
- Learning rate: 0.1



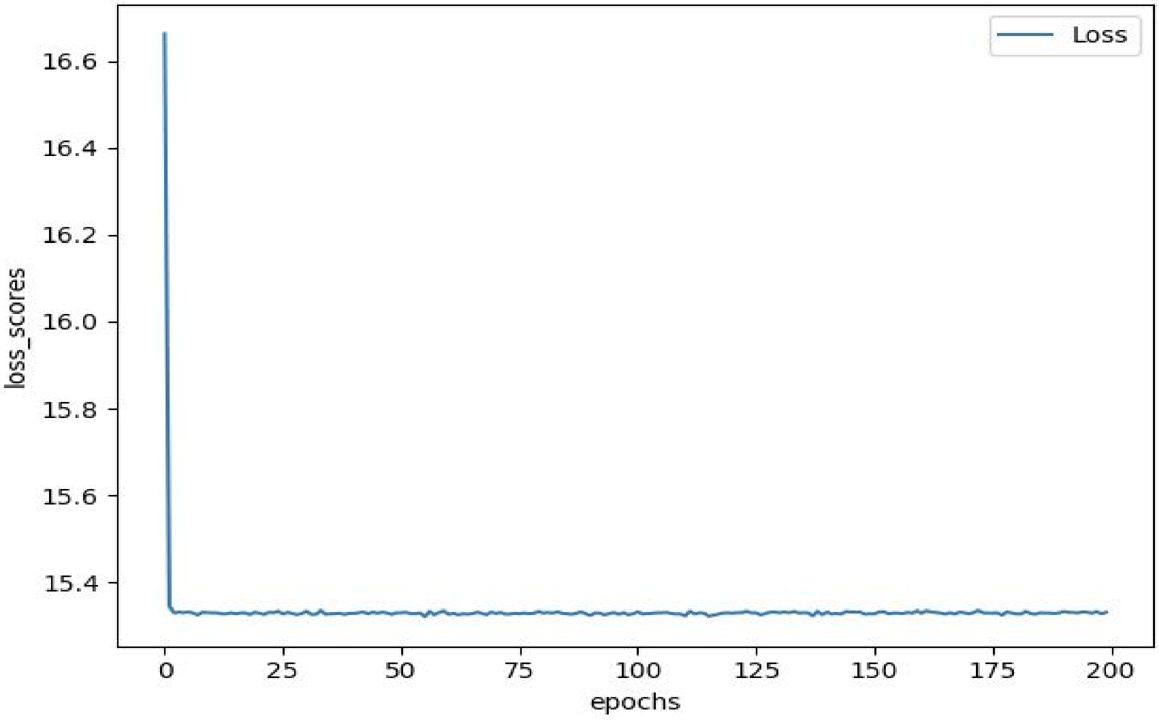
As expected, the softmax-objective didn't fit the data, because it has no non-linearity part.

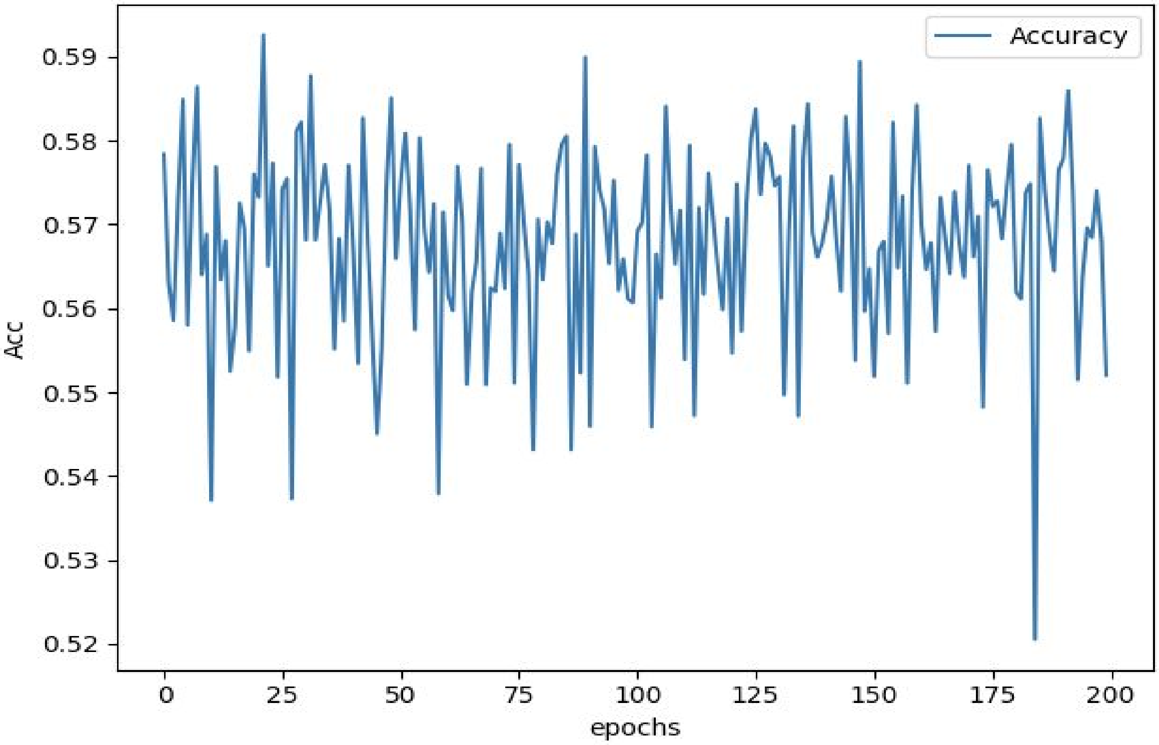


Next, we check this on real data:

Details:

- Data: PeaksData Data, 5 classes.
- Epochs: 200
- Learning rate: 0.1



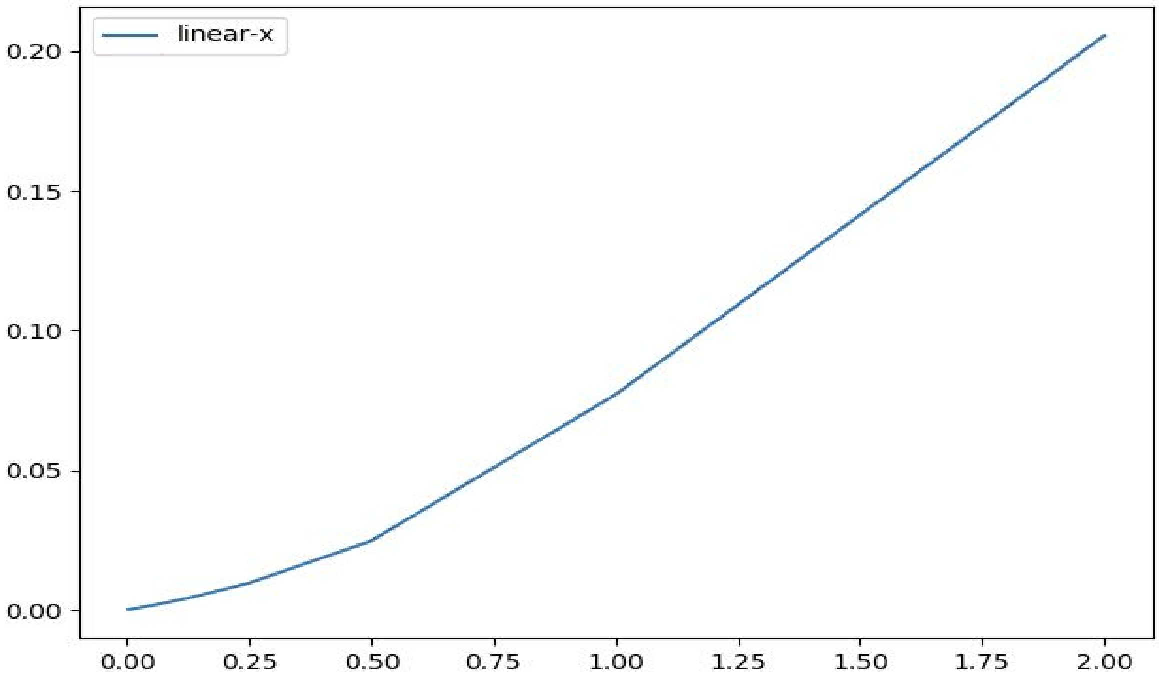


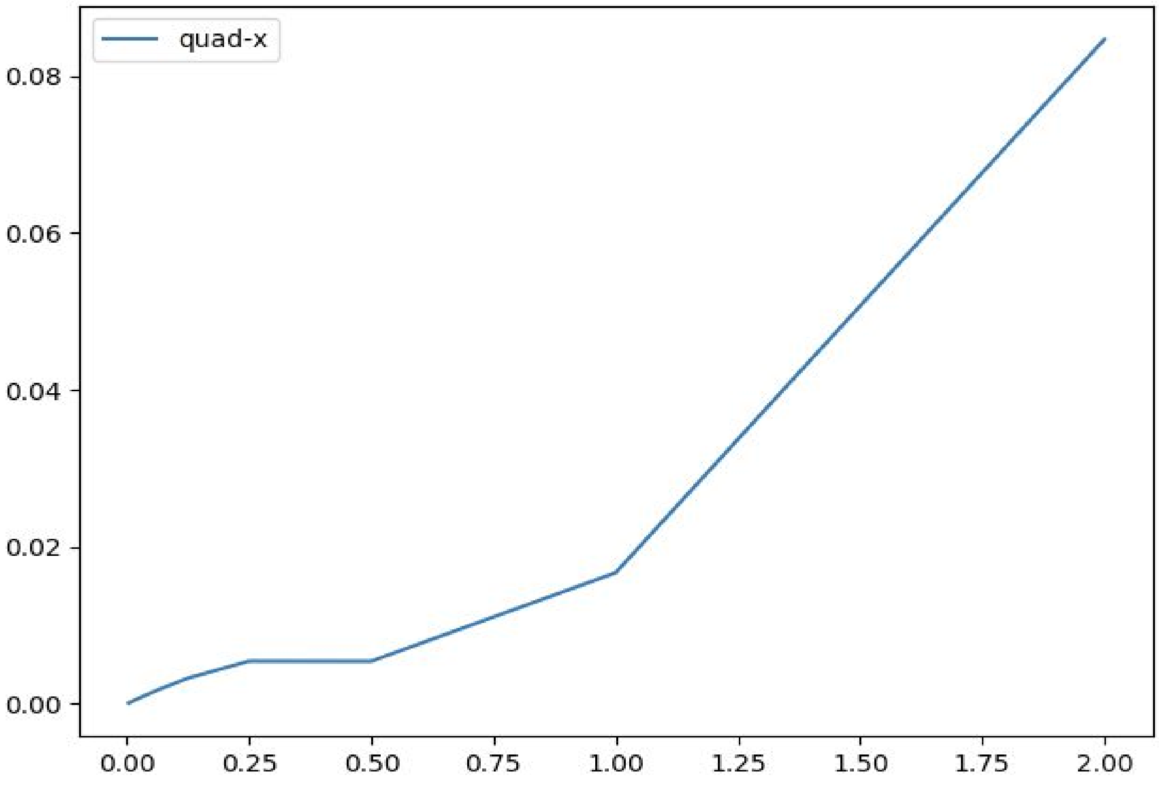
<u>Task6:</u>

In order to validate the Jacobian over network with 2 layers we checked our gradient test for one example.

Details:

- <u>Data</u>: one example with 4 dimensions [1,2,3,4], label 0.





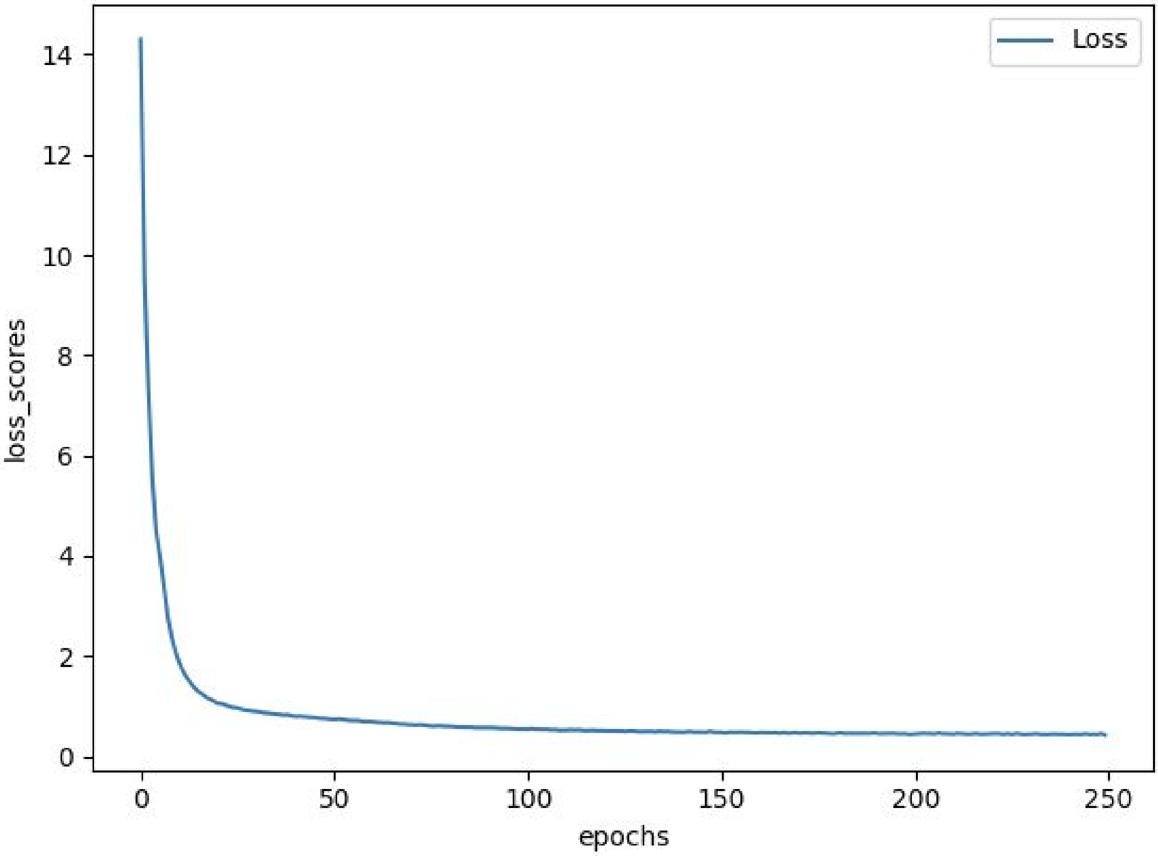
Task7:

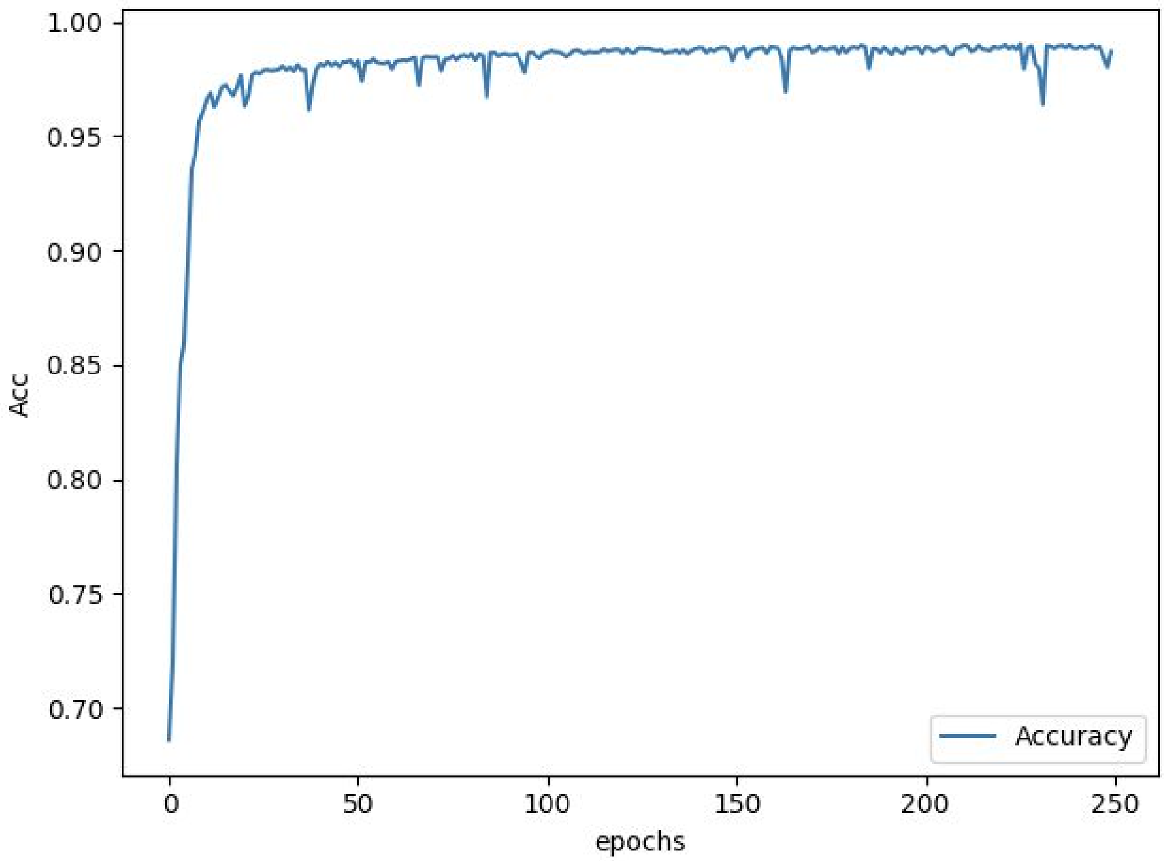
We trained the whole network, architecture:

```
[Linear(in_dim=input_dim, out_dim=10, activation=Tanh()),
 Linear(in_dim=10, out_dim=7, activation=Tanh()),
 Linear(in_dim=7, out_dim=num_of_classes, activation=None)]
Loss = SoftmaxCrossEntropyLoss()
```

Details:

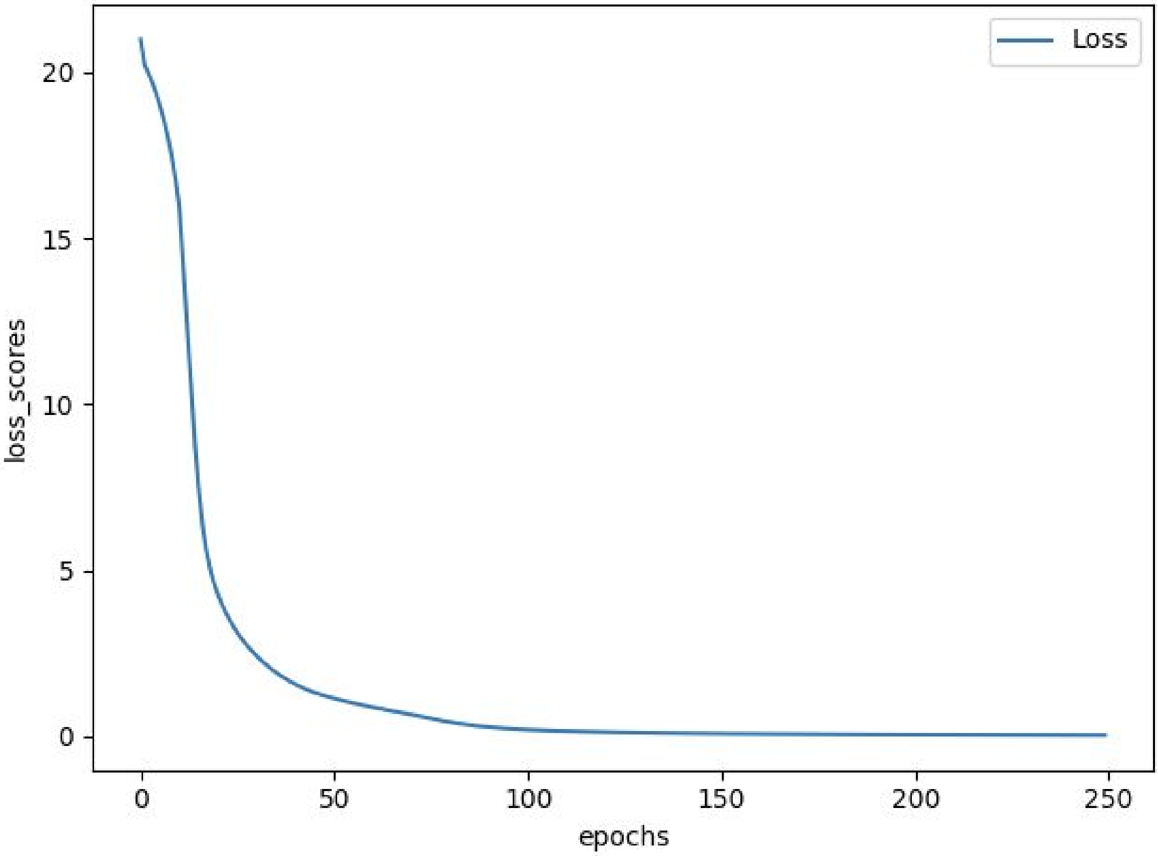
- Data: GMM Data, 5 classes.
- Batch-Size: 64
- Epochs: 250
- Learning rate: 0.1
- Final Accuracy on Test: 98.57%

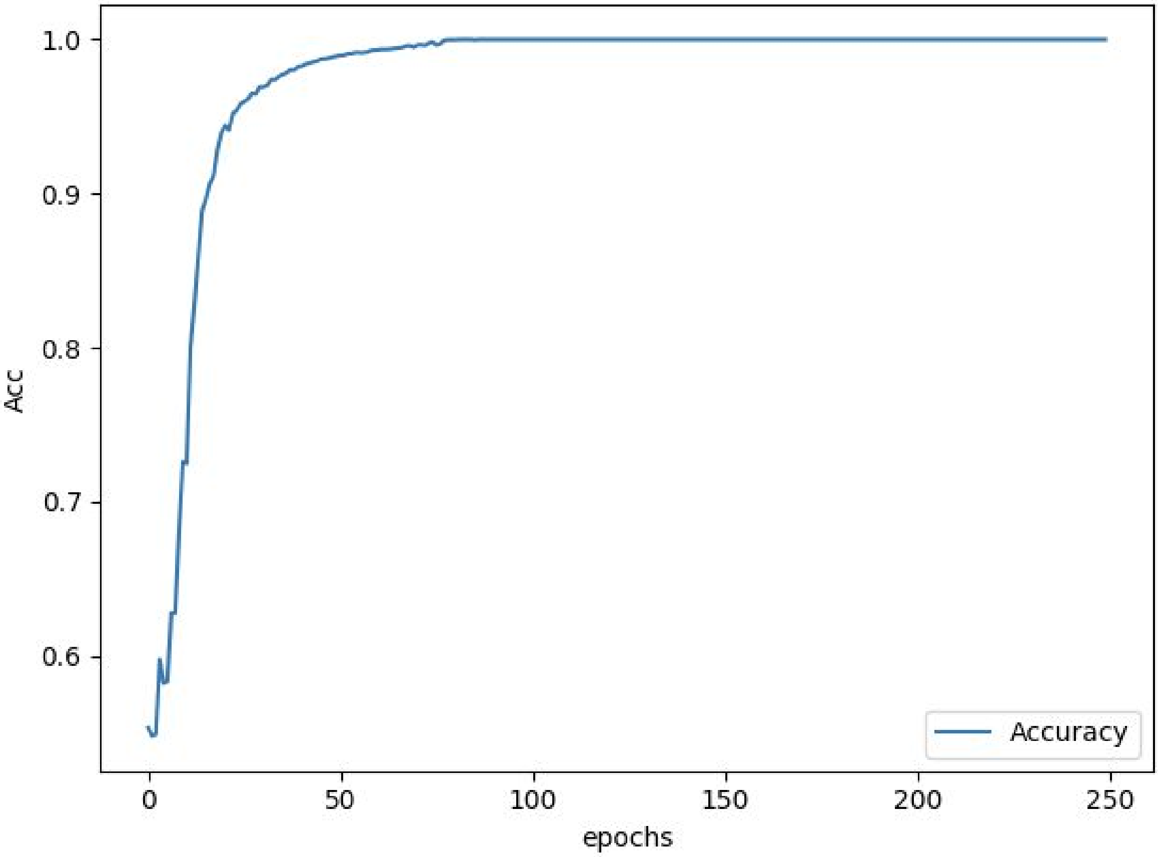




Details:

- Data: Swiss Roll Data, 2 classes.
- Batch-Size: 64
- Epochs: 250
- Learning rate: 0.1
- Final Accuracy on Test: 100%


Loss
loss_scores
20
15
10
5
0
0
50
100
150
200
250
epochs


Acc
1.0
0.9
0.8
0.7
0.6
Accuracy
0
50
100
150
200
250
epochs