Laboratory 5

Variant 2

Group 12

By Dimitrios Georgousis and Andrea Amato

Introduction

This mini-project aims to familiarize students with neural networks. Specifically, we were asked to implement a Multi-Layer Perceptron network for image classification. This model uses the mini-batch gradient descent method and, also, a back-propagation algorithm. The images the dataset provides us with are handwritten numbers and the aim of our algorithm is to be trained so as to identify the corresponding number shown.

Data Preparation

There are not many steps needed in data preparation. Our program loads the datasets and turns them into tensors which work better with most PyTorch tools. Our 'train()' function uses a data-loader in order to handle the mini-batch size and our gray scale images are flattened right before they are supplied to the neural network for training.

Data Split

The provided MNIST dataset has already been separated into training and test sets. We do not do anything extra.

Model Description

Our neural network model is quite heavily parameterized. We use a base configuration of those parameters and compare against it values for other parameters.

We are asked to produce results for at least 3 different numbers/values/types of:

- Learning rate
- Mini-batch size
- Number of hidden layers
- Width
- Optimizer type

If we used 3 values for each these parameters and evaluated all possible permutations we would have a rather large number of models to check. In order to avoid that but also be able to draw better conclusions from results we decided to always just vary one parameter of the base configuration and keep the rest constant.

Parameters which do not vary were just fixed to constant values/expressions. More specifically:

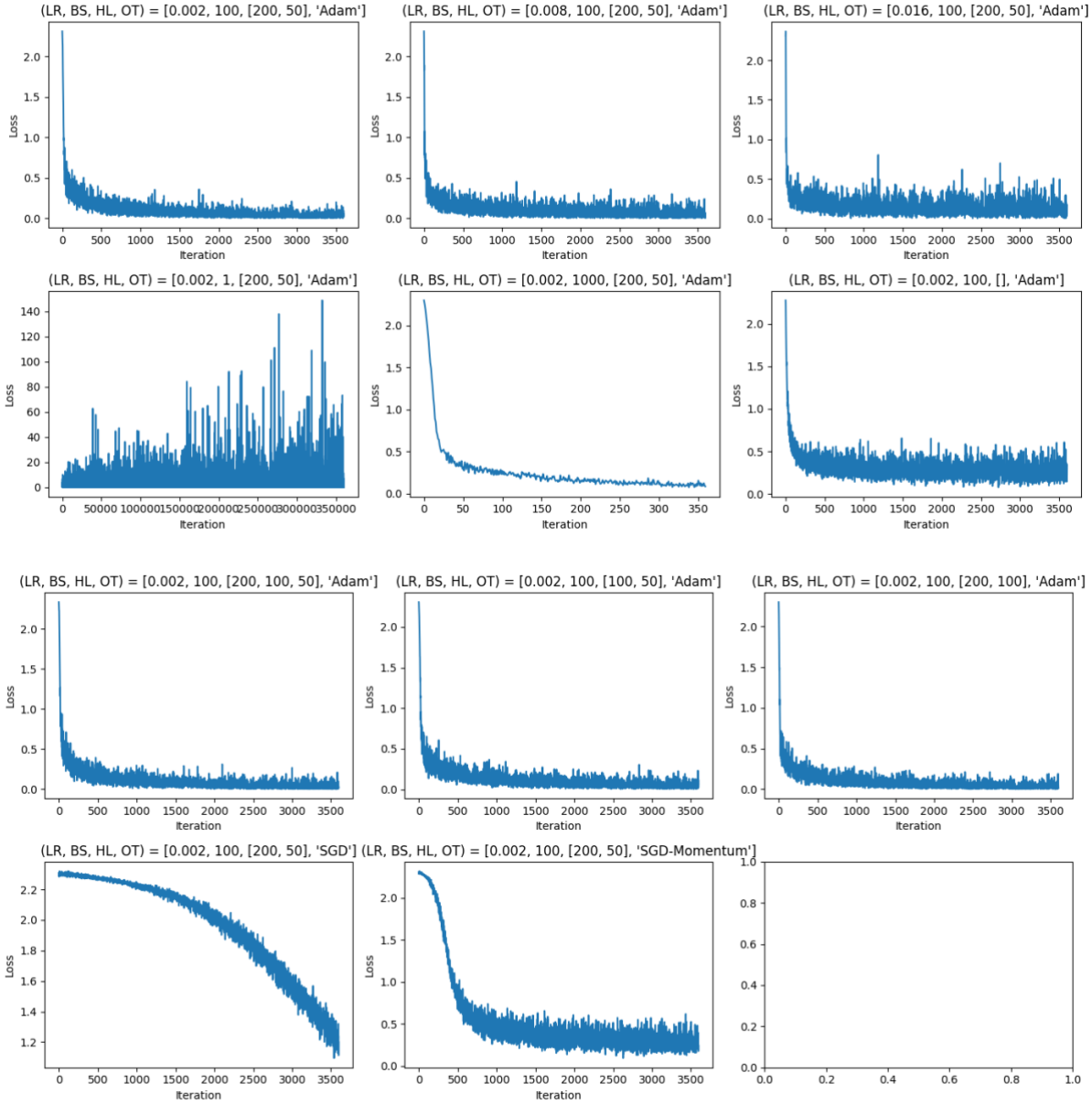
- Activation function: ReLU for all layers except the last one
- Activation function of last layer: softmax
- Loss function: cross entropy loss function
- Epoch count: 6

Lastly, every time we train a model we re-initialize the random seed to the same value, so as the have more comparable results.

Results

Our code produces a log of: mean (average) loss value, accuracy of model on training set and accuracy of model on test set after each epoch. We present this log at the end of the report as it is quite verbose and will not aid us greatly. First, we present the visualization of loss values for every learning step for each of our model configurations:

(LR, BS, HL, OT) = (learning rate, batch size, hidden layers, optimizer type)



Comments on loss value results

• **Learning rate (LR)**: The base value is 0.002, tested against 0.008 and 0.016. The learning rate is a hyperparameter that influences the magnitude of updates performed by the optimizer on the model parameters at each step. Theoretically, it is expected that low learning rates lead to slower convergence, but higher ones might overshoot and miss the optimal values for model parameters. In the above tests we see that larger learning rates lead to more noisy loss distributions, but, nonetheless, after enough iterations convergence seems to be achieved.

• **Mini-batch size (BS)**: Here, we compare a base size of 100 against extremes of 1 and 1000. The batch size is a hyperparameter that controls after how many items the algorithm will perform an update on its model parameters. Small values of batch size tend produce noisy and unstable results, while larger values might lead to convergence on suboptimal solutions. We can definitely observe that increasing the batch size removes a lot of the noise from the distribution. As for the quality of the resulting model in each case, we will comment on it when we discuss accuracy values.

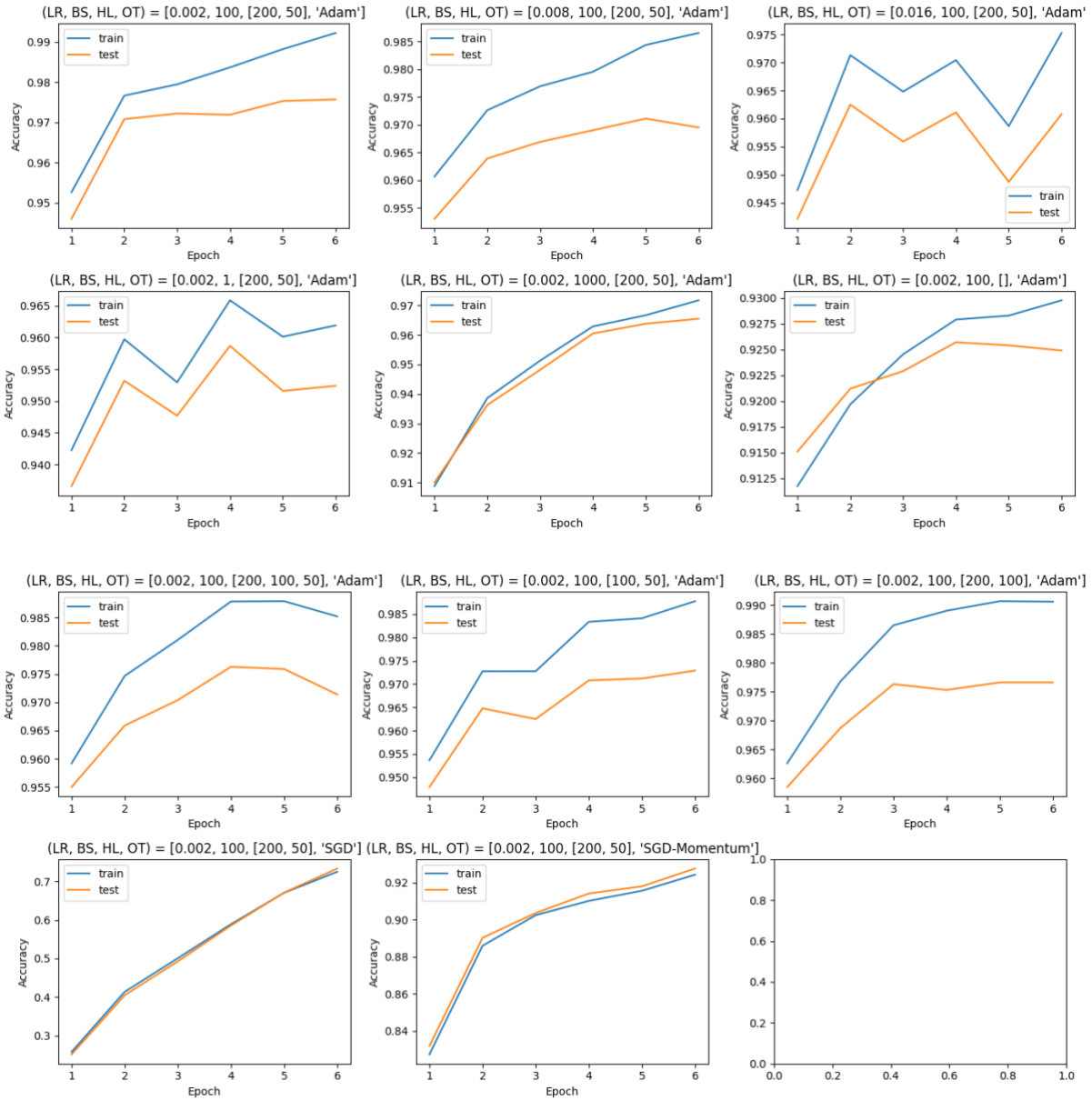
• **Number of hidden layers (HL)**: The base configuration uses two hidden layers ([200, 50]), compared to variants with none ([]) and three hidden layers ([200, 100, 50]). Having hidden layers that are connected with 'activation functions' is what allows multilayer perceptron neural networks to capture non linear distinctions or boundaries between groups in the data. The network with no hidden layers gives the noisiest loss distribution, while the other two do not have much of a distinction between them. We can observe that all distribution seem to lead to rather low values of loss by the last iterations. This can be attributed to the multiple epochs that the training includes, which allows for feedback and better optimization of the model parameters, as the above graphs include all iterations of all epochs.

• **Width (HL)**: Our baseline hidden layers configuration is [200, 50]. We tested variations [100, 50] for a narrower first layer, and [200, 100] for a wider second layer. Small hidden layer sizes are expected to be more prone in missing information of the data and thus not learning properly while larger ones are more prone to overfitting. In our case all three hidden layer sizes (smaller first layer or larger second layer) give similar loss values, which means they are all appropriate. To explain this we could either hypothesize that our algorithm is rather, otherwise, robust, the data we are trying to differentiate (numbers) is quite easy to do so or we did not use large enough differences in the sizes of the hidden layers to make an impact on the results.

• **Optimizer type (OT)**: The base optimizer 'Adam' is compared against 'SGD' and 'SGD with momentum' where 'momentum' is set to 0.9. Optimizers are the functions that perform updates to model parameters using some objective function. 'SGD' is a rather basic one and it performs updates based on the value of the gradient of the objective function with respect to the parameters. Its version with 'Momentum' incorporates an idea that past gradient values should also be taken into account when updates are performed. It is expected that

less oscillation will occur in the manner that the algorithm approaches an optimum and also the process will be faster. 'Momentum' values are between 0 and 1, where values closer to 1 mean that the influence of past gradients is better preserved. 'Adam' seems to give the 'cleanest' loss distribution, while the simple SGD version struggles quite a lot to reach low loss values. This is an expected result as 'Adam' is an algorithm which uses not only ideas similar to momentum but others which are advanced as well.

At this point, visualization of accuracy on training and test (validation) sets after each epoch is shown:



Comments on accuracy results

Comments on the different networks' abilities to converge, the speed of said convergence and the final accuracy achieved on the different sets and more general remarks follow.

• **Learning rate (LR)**: The base value is 0.002, tested against 0.008 and 0.016. The learning rate significantly impacts the rate of convergence, as visible in the accuracy plots. The base learning rate shows a consistent improvement in both training and test accuracies, indicating steady learning. In contrast, the higher learning rate of 0.016 leads to volatility, especially noticeable in the training set, suggesting that too large a learning rate may harm the model's ability to generalize due to potentially overshooting optimal parameters during training.

• **Mini-batch size (BS)**: Here, we compare a base size of 100 against extremes of 1 and 1000. The batch size of 1, which represents stochastic gradient descent, shows irregular accuracy on both sets. This implies high variance in the gradient estimates affecting the model's updates. Conversely, the larger batch size of 1000 results in smoother curves, with a much smaller gap between training and test accuracy compared to the base size of 100, suggesting that a larger batch can provide a better stability and generalization.

• **Number of hidden layers (HL)**: The base configuration uses two hidden layers, compared to variants with none and three hidden layers. The no hidden layer variant significantly underperforms, which is expected as it cannot capture the nonlinearity in the data. The configurations with two and three hidden layers perform comparably, with the two-layer variant showing a slightly higher accuracy on the test set, potentially indicating a better capacity for data representation. The three hidden layer model shows signs of deteriorating accuracy results in later epochs meaning that it (over)fits to information that is not actually present in the data set.

• **Width (HL)**: Our baseline hidden layers configuration is [200, 50]. We tested variations [100, 50] for a narrower first layer, and [200, 100] for a wider second layer. A smaller first hidden layer size theoretically reduces the network's capacity to learn complex representations, which is reflected in a slight dip in accuracy for both the training and test sets. Conversely, expanding the second layer to [200, 100] slightly improves the test set accuracy, and the convergence on the training set, but with an increased gap between training and test sets accuracy. The balanced configuration [200, 50] thus emerges as a well-tuned structure for the MNIST dataset, providing sufficient capacity to learn effectively while maintaining a good level of generalization to new data.

• **Optimizer type (OT)**: The base optimizer 'Adam' is compared against 'SGD' and 'SGD with momentum'. 'Adam' shows the best performance with higher accuracy for both training and test sets. Conversely, the 'SGD' variant shows the lowest accuracy, reflecting its basic nature without advanced optimization techniques, and 'SGD with momentum' stands between 'SGD' and 'Adam', showing the beneficial effect of considering past gradients but not matching the performance of 'Adam'. However, a closer inspection reveals that 'SGD', both with and without momentum, demonstrate a tighter convergence between training and

test accuracies potentially suggesting a better generalization and less overfitting than ‘Adam’.

In summary, the model's configuration with a learning rate of 0.002, batch size of 100, two hidden layers, and 'Adam' optimizer achieves a balance between learning efficiency and generalization capability. This is indicated by the smooth increase in accuracy and the convergence of training and test accuracies, suggesting good model performance without significant overfitting. The chosen parameters seem to work well for the MNIST dataset, known for being a relatively easier classification problem, which may account for the generally high accuracy across all tested configurations.

Conclusions

Throughout the course of this experiment, we scrutinized the effects of various hyperparameters on the performance of a Multi-Layer Perceptron when tasked with the classification of handwritten digits from the MNIST dataset. Our findings highlight the delicate balance between a model's capacity to learn and its ability to generalize. The learning rate, an essential hyperparameter, revealed that too high a value might underfit data and impede generalization, while too low a rate slows convergence. The investigation into batch sizes confirmed that while larger sizes smooth out the learning curve and contribute to stability, they must be chosen judiciously to avoid underfitting.

Examining the depth and width of the model illuminated the role of hidden layers in capturing the nuances of data; neither too few to miss complexity nor too many to overfit. It was observed that the baseline configurations of [200, 50] for hidden layers proved effective, striking a practical balance for this dataset.

A revelation came from comparing optimizer types; 'Adam' exhibited robust performance in training accuracy, yet 'SGD' and its momentum-enhanced variant displayed a more consistent alignment between the training and test set accuracies. This suggests that while advanced optimizers like 'Adam' can expedite the learning process, they don't necessarily lead to a better generalization, as seen in our experiment's results.

In conclusion, the experiment has underscored the significance of hyperparameter tuning in model performance. We learned that each choice carries implications for the model's training dynamics and its final efficacy. The knowledge gained from this experiment will serve as a guide for future endeavors in neural network optimization contributing to the ongoing dialogue bridging theoretical aspects with their practical applications in the field of machine learning.

Log of output values

```
Using cpu device

 --Information about the sets--

Dataset MNIST
    Number of datapoints: 60000
    Root location:
    Split: Train
    StandardTransform

Dataset MNIST
    Number of datapoints: 10000
    Root location:
    Split: Test
    StandardTransform
Transform: ToTensor()

 --Information about some image--

Image shape: torch.Size([1, 28, 28])
Label: 5
Configuration 1/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [200, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.3047710584724943
Accuracy Score On Train Set: 0.9526166666666667
Accuracy Score On Test Set: 0.946
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.11179180000908673
Accuracy Score On Train Set: 0.97665
Accuracy Score On Test Set: 0.9708
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.07570391378055016
Accuracy Score On Train Set: 0.97945
Accuracy Score On Test Set: 0.9722
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.057832007457812626
Accuracy Score On Train Set: 0.9836666666666667
Accuracy Score On Test Set: 0.9719
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.04466265098618654
Accuracy Score On Train Set: 0.9882
Accuracy Score On Test Set: 0.9753
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.03573032599970854
Accuracy Score On Train Set: 0.9922333333333333
Accuracy Score On Test Set: 0.9757
Configuration 2/11
{'lr': 0.008, 'batch_size': 100, 'hidden_layers': [200, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
```

```
Mean Loss: 0.2345979269159337
Accuracy Score On Train Set: 0.9606333333333333
Accuracy Score On Test Set: 0.953
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.11185759373707697
Accuracy Score On Train Set: 0.9726166666666667
Accuracy Score On Test Set: 0.9639
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.09140740850552295
Accuracy Score On Train Set: 0.9769333333333333
Accuracy Score On Test Set: 0.9669
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.07579658072286596
Accuracy Score On Train Set: 0.9795666666666667
Accuracy Score On Test Set: 0.969
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.06831432511176293
Accuracy Score On Train Set: 0.9844
Accuracy Score On Test Set: 0.9711
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.058875412250636146
Accuracy Score On Train Set: 0.9865666666666667
Accuracy Score On Test Set: 0.9695
Configuration 3/11
{'lr': 0.016, 'batch_size': 100, 'hidden_layers': [200, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.2540166000338892
Accuracy Score On Train Set: 0.9472166666666667
Accuracy Score On Test Set: 0.9421
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.15086289940169081
Accuracy Score On Train Set: 0.9713333333333334
Accuracy Score On Test Set: 0.9625
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.1361902533347408
Accuracy Score On Train Set: 0.9648166666666667
Accuracy Score On Test Set: 0.9559
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.1312009144667536
Accuracy Score On Train Set: 0.97045
Accuracy Score On Test Set: 0.9611
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.12695557989024867
Accuracy Score On Train Set: 0.9586666666666667
Accuracy Score On Test Set: 0.9487
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.11958169131462151
Accuracy Score On Train Set: 0.9753
Accuracy Score On Test Set: 0.9608
```

```
Configuration 4/11
{'lr': 0.002, 'batch_size': 1, 'hidden_layers': [200, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.29977978071267164
Accuracy Score On Train Set: 0.9422666666666667
Accuracy Score On Test Set: 0.9366
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.22928625499838085
Accuracy Score On Train Set: 0.9597333333333333
Accuracy Score On Test Set: 0.9532
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.2186331816862087
Accuracy Score On Train Set: 0.95295
Accuracy Score On Test Set: 0.9477
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.22682613642421928
Accuracy Score On Train Set: 0.9658666666666667
Accuracy Score On Test Set: 0.9587
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.22638985940418987
Accuracy Score On Train Set: 0.9601333333333333
Accuracy Score On Test Set: 0.9516
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.24699061963172717
Accuracy Score On Train Set: 0.9619166666666666
Accuracy Score On Test Set: 0.9524
Configuration 5/11
{'lr': 0.002, 'batch_size': 1000, 'hidden_layers': [200, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.7379049152135849
Accuracy Score On Train Set: 0.9088166666666667
Accuracy Score On Test Set: 0.9101
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.25995662560065586
Accuracy Score On Train Set: 0.9386333333333333
Accuracy Score On Test Set: 0.9363
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.1937031219402949
Accuracy Score On Train Set: 0.9513166666666667
Accuracy Score On Test Set: 0.9482
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.15119170074661573
Accuracy Score On Train Set: 0.9629
Accuracy Score On Test Set: 0.9605
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.12581526065866153
Accuracy Score On Train Set: 0.9666833333333333
Accuracy Score On Test Set: 0.9638
```

```
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.10360633060336114
Accuracy Score On Train Set: 0.9717333333333333
Accuracy Score On Test Set: 0.9655
Configuration 6/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [], 'optimizer_type':
'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.47791076434155305
Accuracy Score On Train Set: 0.91175
Accuracy Score On Test Set: 0.9151
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.30574265668789546
Accuracy Score On Train Set: 0.9196833333333333
Accuracy Score On Test Set: 0.9212
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.2836319160461426
Accuracy Score On Train Set: 0.9245333333333333
Accuracy Score On Test Set: 0.9229
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.27380436766892674
Accuracy Score On Train Set: 0.9279
Accuracy Score On Test Set: 0.9257
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.2664212550595403
Accuracy Score On Train Set: 0.9282833333333333
Accuracy Score On Test Set: 0.9254
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.26242648301025234
Accuracy Score On Train Set: 0.9297666666666666
Accuracy Score On Test Set: 0.9249
Configuration 7/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [200, 100, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.3124969141557813
Accuracy Score On Train Set: 0.9591833333333334
Accuracy Score On Test Set: 0.955
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.11749661256559193
Accuracy Score On Train Set: 0.9746833333333333
Accuracy Score On Test Set: 0.9659
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.07908374680827061
Accuracy Score On Train Set: 0.9810666666666666
Accuracy Score On Test Set: 0.9704
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.05835260510987913
Accuracy Score On Train Set: 0.9878666666666667
Accuracy Score On Test Set: 0.9763
```

```
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.046488063944852914
Accuracy Score On Train Set: 0.9879166666666667
Accuracy Score On Test Set: 0.9759
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.039004275237869784
Accuracy Score On Train Set: 0.9852166666666666
Accuracy Score On Test Set: 0.9714
Configuration 8/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [100, 50],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.34195859473198653
Accuracy Score On Train Set: 0.9536333333333333
Accuracy Score On Test Set: 0.9479
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.13729756186405817
Accuracy Score On Train Set: 0.97275
Accuracy Score On Test Set: 0.9648
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.09536262173050393
Accuracy Score On Train Set: 0.9727666666666667
Accuracy Score On Test Set: 0.9625
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.07490557943470776
Accuracy Score On Train Set: 0.9833666666666666
Accuracy Score On Test Set: 0.9708
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.05748977253601576
Accuracy Score On Train Set: 0.9841333333333333
Accuracy Score On Test Set: 0.9712
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.047513866435037924
Accuracy Score On Train Set: 0.9878
Accuracy Score On Test Set: 0.9729
Configuration 9/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [200, 100],
'optimizer_type': 'Adam'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 0.2746163241378963
Accuracy Score On Train Set: 0.9626166666666667
Accuracy Score On Test Set: 0.9585
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.10884826868462066
Accuracy Score On Train Set: 0.9767666666666667
Accuracy Score On Test Set: 0.9687
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.07374412871043508
Accuracy Score On Train Set: 0.9864833333333334
Accuracy Score On Test Set: 0.9763
```

```
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.054327107712160795
Accuracy Score On Train Set: 0.989
Accuracy Score On Test Set: 0.9753
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.04244225428556092
Accuracy Score On Train Set: 0.99065
Accuracy Score On Test Set: 0.9766
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.03517913646462451
Accuracy Score On Train Set: 0.9905666666666667
Accuracy Score On Test Set: 0.9766
Configuration 10/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [200, 50],
'optimizer_type': 'SGD'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 2.2874504697322844
Accuracy Score On Train Set: 0.25716666666666665
Accuracy Score On Test Set: 0.2508
Starting Epoch 2
--Epoch 2--
Mean Loss: 2.237277419169744
Accuracy Score On Train Set: 0.41345
Accuracy Score On Test Set: 0.4046
Starting Epoch 3
--Epoch 3--
Mean Loss: 2.145845855474472
Accuracy Score On Train Set: 0.5007333333333334
Accuracy Score On Test Set: 0.4926
Starting Epoch 4
--Epoch 4--
Mean Loss: 1.9835724556446075
Accuracy Score On Train Set: 0.5891666666666666
Accuracy Score On Test Set: 0.5858
Starting Epoch 5
--Epoch 5--
Mean Loss: 1.7264388338724772
Accuracy Score On Train Set: 0.6707833333333333
Accuracy Score On Test Set: 0.6713
Starting Epoch 6
--Epoch 6--
Mean Loss: 1.3927107979853948
Accuracy Score On Train Set: 0.7254333333333334
Accuracy Score On Test Set: 0.7338
Configuration 11/11
{'lr': 0.002, 'batch_size': 100, 'hidden_layers': [200, 50],
'optimizer_type': 'SGD-Momentum'}
Starting Epoch 1
--Epoch 1--
Mean Loss: 1.5900424439211687
Accuracy Score On Train Set: 0.82725
Accuracy Score On Test Set: 0.8318
Starting Epoch 2
--Epoch 2--
Mean Loss: 0.4909058019518852
Accuracy Score On Train Set: 0.8859166666666667
Accuracy Score On Test Set: 0.8902
```

```
Starting Epoch 3
--Epoch 3--
Mean Loss: 0.37163644169767696
Accuracy Score On Train Set: 0.9024333333333333
Accuracy Score On Test Set: 0.9036
Starting Epoch 4
--Epoch 4--
Mean Loss: 0.3298041884849469
Accuracy Score On Train Set: 0.9101333333333333
Accuracy Score On Test Set: 0.9141
Starting Epoch 5
--Epoch 5--
Mean Loss: 0.30152752311279374
Accuracy Score On Train Set: 0.9156
Accuracy Score On Test Set: 0.918
Starting Epoch 6
--Epoch 6--
Mean Loss: 0.27810818088551365
Accuracy Score On Train Set: 0.9242833333333333
Accuracy Score On Test Set: 0.9276
```