Mini Project 1

Mehmet Ozan Baykan

EEE443, Fall 2023

1 Introduction

In this assignment, we will implement an FNN architecture for image classification on the MNIST set, without the use of any machine learning libraries. The main tasks included can be outlined as (1) data preprocessing, (2) model construction, (3) hyperparameter tuning for layer size, learning rate and nonlinear activation function, (4) hyperparameter tuning for mini-batch size, (5) hyperparameter tuning for L^2 regularization parameter λ.

2 Data Retrieval and Pre-Processing

Unfortunately, the website provided in the instructions was not publicly available as of now, hence we retrieved the dataset from Kaggle at www.kaggle.com/datasets/hojjatk/mnist-dataset/. The directory structure, the description, the number of samples in training and test sets, and the pixel resolution were identical, and we checked that the labels and the images were consistent. Next, as suggested in the instructions, we flattened the images with the method $flatten()$, normalized the pixel intensities by dividing each entry by the maximum amount of 255 using the method $normalized()$. Then, the labels were encoded into one-hot vectors of length 10 using the method $one\_hot()$. These will be used as the true labels in contrast to the output of the FNN.

3 Model Constrution

As in the instructions, we construct an FNN model with one hidden layer. To this end, the class $NN()$ is implemented. This will allow us to keep the parameters and metrics of the model in a neat way.

As the dimension of the flattened images is 784, the output should be compatible with the one-hot vectors, the model is initialized with input dimension = 784 and output dimension = 10.

The suggested sizes of 300, 500, and 1000 are initially used for hidden layer dimension, and likewise the learning rate is selected from 0.01, 0.05, and 0.09.

The weights are initialized with a modified version of the method suggested by Glorot and Bengio in their paper "Understanding the difficulty of training deep feedforward neural networks". The original distribution suggested in this paper is,

$$W^{(j)} \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}]$$

where $U[a, b]$ is the uniform distribution over $[a, b]$, and $W^{(j)}$ is the weight matrix for the transformation from the j^{th} layer to the $j+1^{th}$ layer, and n_j is the dimension of the j^{th} layer. In contrast, we utilize a narrower normal distribution with the same heuristic. Hence, our initialization distribution reads,

$$W^{(j)} \sim N(0, \frac{\sqrt{2}}{\sqrt{n_j + n_{j+1}}})$$

where N denotes the Gaussian distribution with mean 0 and variance $\frac{\sqrt{2}}{\sqrt{n_j+n_{j+1}}}$.

The models' activation functions are also defined as given by the instructions, to ReLU in the first layer and sigmoid in the second layer in the first case, and to tanh on both layer in the second case. The methods for these are defined as methods of the NN class, together with the corresponding methods for computing the gradients of the nonlinear activation functions during backpropagation.

All the abovementioned configurations of the model are handled by the $\_\_init\_\_()$ method of the NN class. On top, dictionaries for storing the model parameters and arrays for storing loss and accuracy metrics for the training and test sets are initialized.

The Mean Squared Error (MSE) function is used as the loss function:

$$MSE(Y, \hat{Y}) = \frac{}{1}_N \sum_{n=1}^{N} ||y_i - \hat{y}_i||_2^2$$

where Y is the matrix of true labels, $\hat{Y}$) is the matrix of predicted labels, y_i is the true label of the ith sample (corresponding to the ith column of Y), and $\hat{y}_i$ is the true label of the ith sample (corresponding to the ith column of $\hat{Y}$), and N is the number of samples in the training set. The MSE loss is implemented with the $cost()$ method in the NN class.

The input data is processed through the network using forward propagation, defined by the following equations in the first layer,

Tanh Activation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \tanh(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \tanh(Z^{[2]})$$

ReLU-Sigmoid Activation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = ReLU(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

For the backward propagation, where $\frac{\partial J}{\partial A^{[2]}}$ represents the derivative of the cost with respect to the output of the last layer, and similar notations for other derivatives:

Tanh Activation

$$\frac{\partial J}{\partial Z^{[2]}} = 2 \cdot (\hat{Y} - Y) \cdot (1 - tanh^2(Z^{[2]}))$$
$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{N} \cdot (\frac{\partial J}{\partial Z^{[2]}} A^{[1]T})$$
$$\frac{\partial J}{\partial b^{[2]}} = \frac{1}{N} \cdot \sum \frac{\partial J}{\partial Z^{[2]}}$$

$$\frac{\partial J}{\partial Z^{[1]}} = (W^{[2]T} \frac{\partial J}{\partial Z^{[2]}}) \cdot (1 - tanh^2(Z^{[1]}))$$
$$\frac{\partial J}{\partial W^{[1]}} = \frac{1}{N} \cdot (\frac{\partial J}{\partial Z^{[1]}} X^T)$$
$$\frac{\partial J}{\partial b^{[1]}} = \frac{1}{N} \cdot \sum \frac{\partial J}{\partial Z^{[1]}}$$

ReLU - Sigmoid Activation

$$\frac{\partial J}{\partial Z^{[2]}} = 2 \cdot (\hat{Y} - Y) \cdot \sigma(Z^{[2]})(1 - \sigma(Z^{[2]}))$$
$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{N} \cdot \frac{\partial J}{\partial Z^{[2]}} A^{[1]T}$$
$$\frac{\partial J}{\partial b^{[2]}} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{\partial J}{\partial Z^{[2]}}$$

$$\frac{\partial J}{\partial Z^{[1]}} = (W^{[2]T} \frac{\partial J}{\partial Z^{[2]}}) \cdot \mathbb{1}_{\{ReLU(Z^{[1]} > 0\}}$$
$$\frac{\partial J}{\partial W^{[1]}} = \frac{1}{N} \cdot \frac{\partial J}{\partial Z^{[1]}} X^T$$
$$\frac{\partial J}{\partial b^{[1]}} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{\partial J}{\partial Z^{[1]}}$$

4 Hyperparameter Tuning for Hidden Layer Size, Learning Rate, and Nonlinear Activation Function

For determining the best hidden layer size-learning rate-activation function combination among the given choices of hidden layer size$\in 300, 500, 1000$, learning rate$\in 0.01, 0.05, 0.09$ and activation function combinations of $Tanh - Tanh$ and $ReLU - Sigmoid$, we trained a model with each of the combinations possible with the given choices. The training was conducted for 100 epochs. The training/test loss and accuracy statistics are given in Figure 1 and Figure 2. As a result of these evaluations, models with parameter configurations (hidden dimension, learning rate, activation function) $= (1000, 0.09, 1), (500, 0.09, 1), (300, 0.09, 1)$, as can be seen in Table 4 in the Appendix.

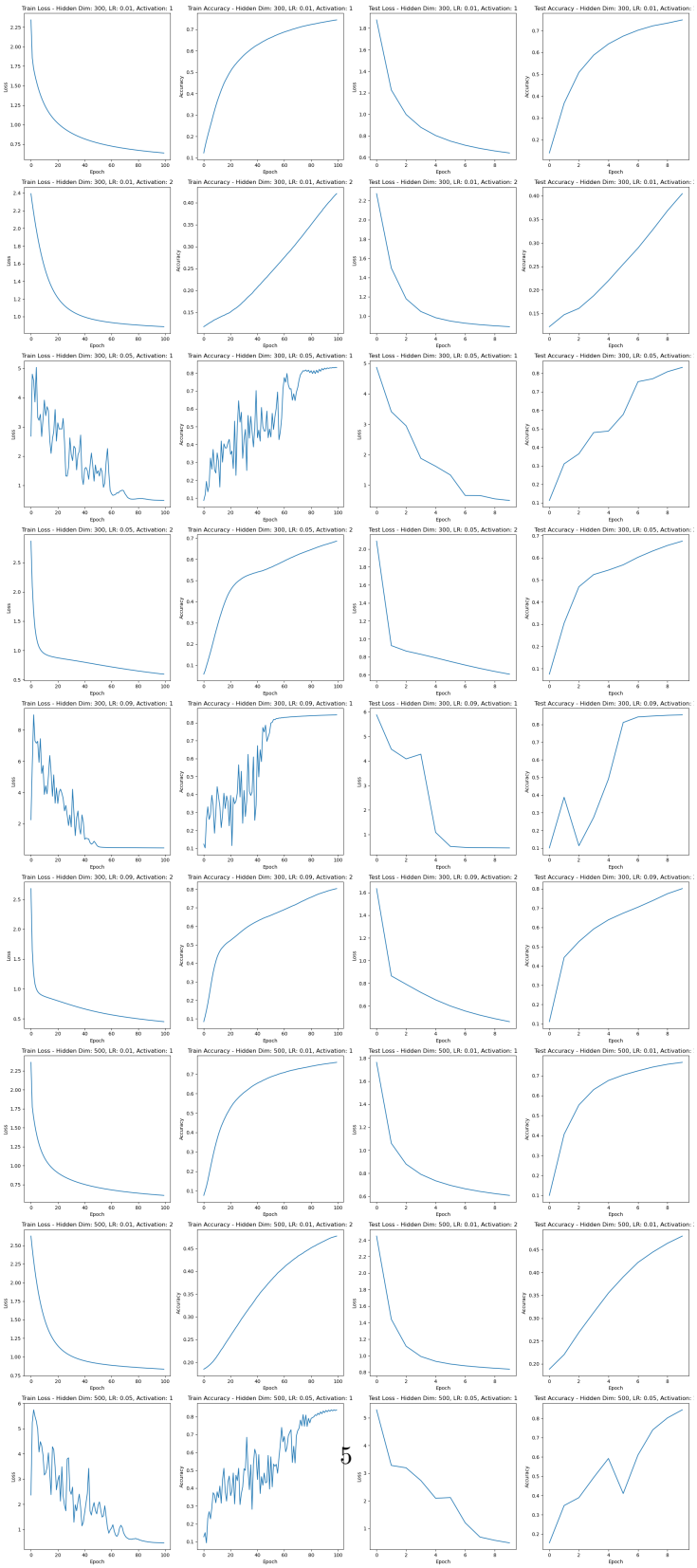

Figure 1: Figure 1: Train/Test loss and accuracy statistics of the first 9 configurations out of 18.

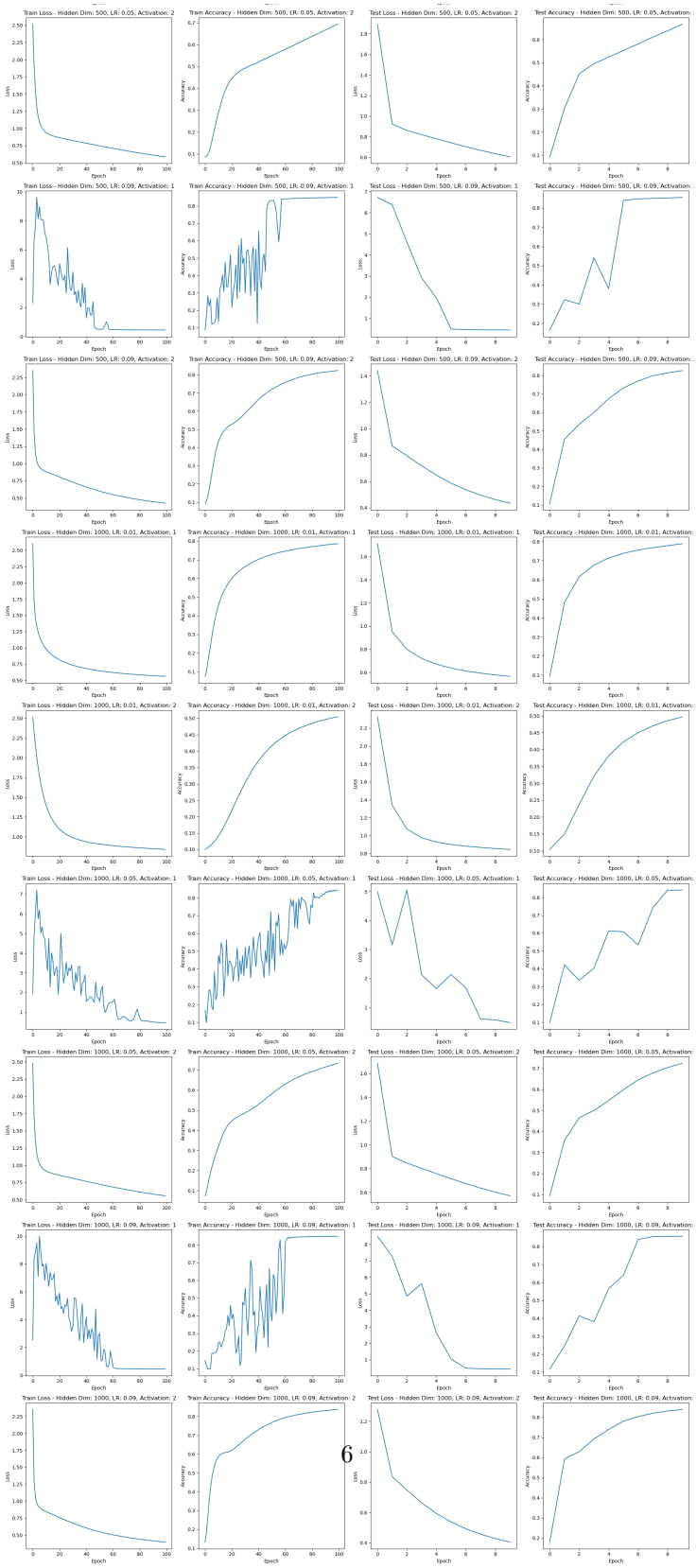

Figure 2: Figure 2: Train/Test loss and accuracy statistics of the last 9 configurations out of 18.

Since we were tuning the three parameters at a time, we also ran a 10-fold bootstrapping procedure to cross-validate the results. The results of the cross-validation are plotted in Figure 3. Also, the results can be seen in Table 5.

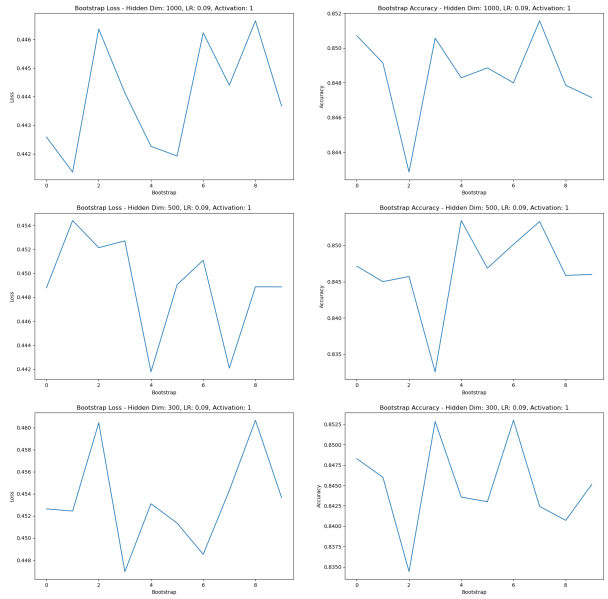

Figure 3: Figure 3: Bootstrap loss and accuracy statistics of the best performing 3 models among the 18 combinations for hidden dimension, learning rate, activation function.

In regards to the results of the cross-validation, we observe that the best combination is (hidden dimension, learning rate, activation function) $= (1000, 0.09, 1)$, so we continue the hyperparameter tuning procedure with this configuration.

Unfortunately, at this point, we had omitted recording the training time, hence, as a crude heuristic to compare these methods, inspired by the T_50 of a reaction in chemistry and biochemistry which gives a rough estimate of the equilibriating time of a complex system of molecules, we looked into the epoch at which the final training accuracy had reached half its value. The data on this is is Table 1.

Configuration	Train Half Epochs
(1000, 0.09, 1)	19
(500, 0.09, 1)	15
(300, 0.09, 1)	10

Table 1: Train and Test Half Epochs for Different Configurations

As a result, we see that as the hidden dimension increases, the optimization procedure also requires more epochs to equilibrate, yet, due to the success of the 1000 hidden dimension model in the cross-validation, we chose to continue with this model.

As a last comment, we should note that given the plots, the learning rate of 0.09 is most suitable, as the plots plateau near 100 epochs even with learning rate 0.09. Moreover, for the activation function, we would hypothesize that, the probabilistic interpretation that naturally follows from the output range of the tanh function and its larger spread over the output dimension may lend it superiority over the ReLU -sigmoid configuration. Moreover, as our input dimension is only smaller than 1000 among the possible hidden layer dimensions, it is understandable that increasing the dimension in the hidden layer allows the network to extract information during training and inference more efficiently.

5 Hyperparameter Tuning for Batch-Size

For this part, we trained three models as suggested in the exercise with (hidden dimension, learning rate, activation function, batch size) = (1000, 0.09, 1, 10), (1000, 0.09, 1, 50), (1000, 0.09, 1, 100) . The test loss and accuracy statistics are in Figure 4.

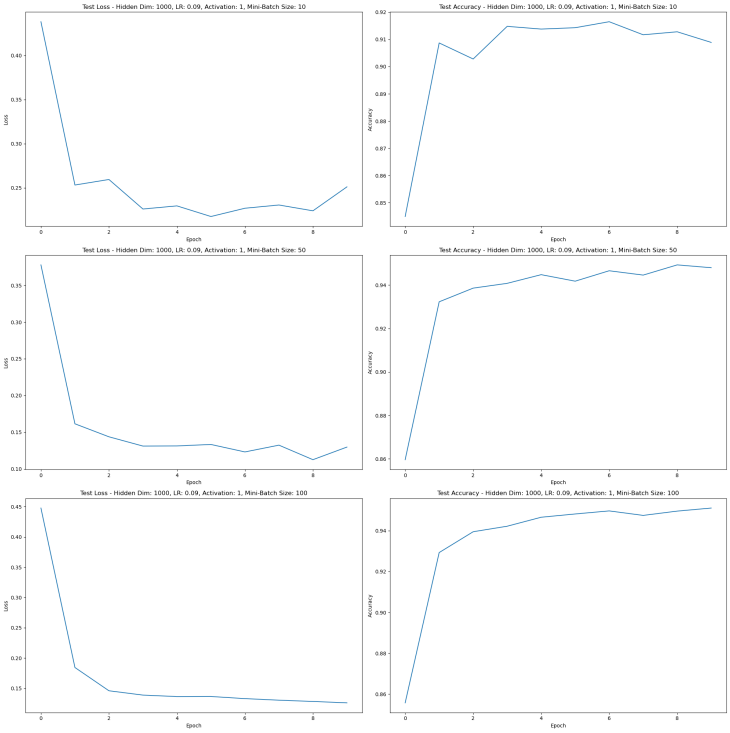

Figure 4: Figure 4: Test loss and accuracy plots for the three models with mini-batch = 10, 50, 100.

Table 2 below includes the final test loss and accuracy, and the total training time in seconds of these models.

Mini-Batch Size	Training Time (s)	Test Loss	Test Accuracy
10	8823.307301	0.251164	0.9089
50	2214.638463	0.129760	0.9480
100	991.429990	0.126046	0.9511

Table 2: Training and Test Metrics for Different Mini-Batch Sizes

As can be predicted, the lower the batch size, the longer the training time. In return, we would expect to get a more robust and powerful training result, yet, the model with mini-batch size 100 outperforms the other two in both these aspects. It's superiority in test accuracy may be attributed to the fact that the smaller the mini-batch size, the more chaotic the training is and the more individual samples introduce variance into the model fit. Hence, we conclude that a mini-batch size of 100 is the best-performing and most efficient among the three.

6 Hyperparameter Tuning for L2 Regularization Parameter λ

Here, we evaluate the parameter λ of the L2 regularization term in the loss function:

article amsmath

$$MSE_{\text{with L2}}(Y, \hat{Y}, W) = \frac{1}{N} \sum_{n=1}^{N} \|y_i - \hat{y}_i\|_2^2 + \frac{\lambda}{2} \sum_{allparameters} w_{ij}^2$$

The results are given in the plots in Figure 5, and the final values of the given metrics in Table 3.

λ	Training Time (s)	Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.010	1844.547873	0.116757	0.968283	0.128735	0.9588
0.001	1301.783485	0.109335	0.964067	0.120414	0.9555

Table 3: Results for different values of λ

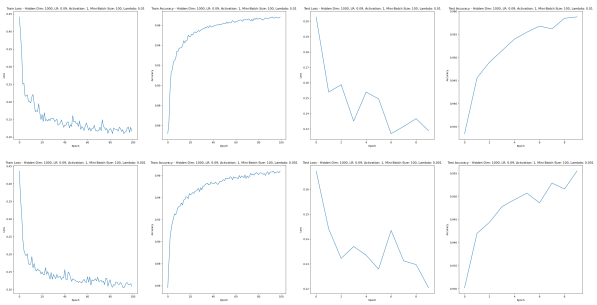

Figure 5: Figure 5: Test loss and accuracy plots for the two models with lambda = 1e-2, 1e-1.

Hence, we conclude that the model expectedly benefits from the L2 regularization, and the best model configuration is (hidden dimension, learning rate, activation function, batch size, lambda) = $(1000, 0.09, 1, 100, 0.01)$.

7 Appendix

Hidden Dim	Learning Rate	Activation	Train Loss	Train Accuracy	Test Loss	Test Accuracy
300	0.01	1	0.639287	0.744283	0.640905	0.7490
300	0.01	2	0.884837	0.420667	0.889066	0.4047
300	0.05	1	0.483492	0.831750	0.493135	0.8318
300	0.05	2	0.594612	0.685717	0.608415	0.6753
300	0.09	1	0.454044	0.843517	0.451684	0.8541
300	0.09	2	0.450331	0.803733	0.461213	0.8010
500	0.01	1	0.608763	0.760950	0.607574	0.7663
500	0.01	2	0.834720	0.477600	0.837162	0.4798
500	0.05	1	0.474221	0.838500	0.482831	0.8447
500	0.05	2	0.587269	0.693400	0.604987	0.6663
500	0.09	1	0.449328	0.848883	0.447063	0.8554
500	0.09	2	0.424218	0.823450	0.435059	0.8261
1000	0.01	1	0.564437	0.785667	0.564810	0.7888
1000	0.01	2	0.839317	0.504400	0.844896	0.4970
1000	0.05	1	0.461341	0.839117	0.475370	0.8415
1000	0.05	2	0.556830	0.733933	0.570016	0.7247
1000	0.09	1	0.444402	0.848750	0.441118	0.8573
1000	0.09	2	0.397038	0.837133	0.404752	0.8392

Table 4: Training statistics for the 18 configurations.

Hidden Dim	Learning Rate	Activation	Mean Bootstrap Loss	Bootstrap Loss SD	Mean Bootstrap Accuracy	Bootstrap Accuracy SD
1000	0.09	1	0.443958	0.001845	0.848500	0.002314
500	0.09	1	0.448976	0.003965	0.846600	0.005525
300	0.09	1	0.453408	0.004175	0.844943	0.005294

Table 5: Bootstrap statistics on the best performing three models in the first round of hyperparameter tuning.

8 Bibliography

Glorot, Xavier, and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR, 2010, pp. 249-256.