Part A

In Part A we had to study the influence of specific parameter changes. The effects of those changes are described in Part B of the report. However, we made two assumptions which influenced the result throughout the paper.

First, we decided to use a seed to make all results comparable over multiple runs. In Keras, setting a seed ensures reproducibility by initializing random number generators with a fixed seed value, providing consistent random initialization of model parameters and maintaining identical conditions across multiple runs. This doesn't mean that the same results can be achieved on different computers, however the results should remain consistent on the same PC.

Second, we defined the batch_size = 64 for shallow neural networks (NN) and 512 for deep NN, as it was also used in the provided code. The batch size is a hyperparameter for the gradient descent that controls the number of training samples to work through before the model's internal parameters are updated. Smaller batch size introduces more frequent updates to the model weights, which can lead to faster convergence, which is why it is suitable for shallow NN. We also tried using the same batch_size for both cases and noticed that the order of the best choice of other hyperparameters could change, however it did not influence the best choice.

Part B

Choosing the Number of Layers

In a NN model, there are usually three layers that process and analyze features of data. The first layer, the input layer, receives the input data and transports the features to the next layer. After the input layer are the hidden layers, which are responsible for learning and extracting features from the input data. Lastly, the output layer presents the output of the model after the hidden layers have processed different learning representations of the data along with any conclusions.

Typically in the creation of neural network models, the addition of more layers results in increased intricacy, versatility, and applicability. This is accomplished as a result of many factors like the ability to learn hierarchical representations of input data and the improved ability to utilize non-linear activation functions to model highly non-linear relationships found in data. Tradeoffs do exist though, with the main ones being differences in the risk of overfitting and varying training times. In our case, we decided to compare effectiveness based solely on accuracy since we noted minimal changes in runtime with each tested number of layers. When choosing the number of layers for our neural network, we decided to compare the effectiveness of a shallow network consisting of 1 layer, an intermediate network consisting of 2 layers, and a deep network with 3 layers. To measure this, we assessed the mean square errors (MSE) and mean absolute errors (MAE) under each number of layers, holding the activation function, number of neurons, optimizer, and number of epochs constant.

Our results, as shown in Table 1.1, conclude that the MSE and MAE are both minimized with 3 layers in the model. Using this knowledge, we proceeded to use a 3-layered NN for our optimal model.

Choosing the Activation Function

Once we determined that a three-layer deep neural network structure was ideal, we tested RelU, tanh, and sigmoid as different potential activation functions. An activation function introduces non-linearity to neural networks, enabling them to capture complex patterns and relationships in data by determining the output of a neuron based on its input. It also facilitates efficient gradient flow during training, helping the network learn and adapt to intricate features in the data. Out of the three activation functions, ReLu performed the best in terms of MSE and MAE by a significant margin. See Table 2.1.

Beyond the empirical evidence, using ReLu makes sense from an intuitive standpoint as well. ReLU has a simple mathematical form: f(x)=max(0, x). Option prices behave like the ReLu function, with a value of 0 when it is out the money, and infinite upside when it is in the money. On the other hand, tanh restricts outputs from -1 to 1 while sigmoid outputs between 0 and 1. The tanh function doesn't capture option pricing correctly, as an option cannot be worth less than 0, and can be worth more than 1. On the other hand, Sigmoid is mainly used to express probabilities so it is not ideal for predicting option prices. Hence, this step shows us the importance of choosing the correct activation function by accounting for the potential underlying relationship of the data set we train our model with. The right activation function can better help minimize the loss function through backpropagation, and introduce nonlinearity that approximates the true pattern of the data set.

Choosing the number of neurons

Once we had established that using a ReLu activation and a three-layer deep neural network was best, we went on to test 10, 50, and 100 neurons per layer. Increasing the number of neurons per layer of a deep neural network enhances its ability to learn complex patterns and nuances, providing greater analytical depth. However, this comes at the cost of increased computational complexity, longer training times, and a higher risk of overfitting, where the model may perform poorly on unseen data. Therefore, carefully balancing the number of neurons is essential for optimizing the network's performance, efficiency, and generalization capabilities. From the three options outlined, using 100 neurons per layer proved superior in terms of MSE, MAE as shown in Table 3.1.

Since these metrics were taken with respect to the validation set and not the training set, we do not need to consider the risk of overfitting the data with 100 neurons per layer, as it has already shown to be the most adaptable to the new data. We also do not need to consider the computational efficiency of higher numbers of neurons per layer, as we observed no meaningful increases in computation time for these relatively low values.

Choosing the optimizer

We opted for the Adam optimizer due to its outstanding performance while keeping all other hyperparameters constant, as evidenced by a notably lower Mean Squared Error (MSE) of 0.000023 compared to RMSProp and SGD momentum, which yielded MSE values of 0.000964 and 0.001685, respectively. It also demonstrated a better result on MAE as shown in Table 4.1. Regarding the convergence behaviour of the model, our observations from the DNN Training evolution plot across the three different optimizers revealed Adam's superiority. The values of the training loss function (logged MSE) exhibited a descending trend as the epochs increased, portraying a relatively smooth curve with less fluctuation (see Table 4.2).

We attribute Adam's superior performance to its combination of advantages from two other optimization algorithms—SGD and the RMSprop algorithm—coupled with its adaptability to the characteristic complexity of financial data; specific comparison can be found in Table 4.3. Stock and option prices are renowned for its intricate and dynamic patterns, leading to varying gradients during training. Adam's adaptive learning rate mechanism dynamically adjusts learning rates for each parameter, ensuring effective handling of these variations. The inclusion of momentum-like terms in Adam accelerates the optimization process by incorporating information from past gradients, mitigating potential oscillations in the parameter space. Furthermore, Adam's adaptive learning rates and momentum features equip the model with robustness against sudden changes in the data.

Choosing the epochs:

Epoch refers to one complete pass through the training dataset during the training phase, during which the neural network calculates errors and adjusts weights to optimize outcomes. If the number of epochs is too low, the neural network lacks sufficient practice, failing to capture accurate underlying patterns. Conversely, if the number of epochs is too high, the network may pick up noise and struggle to generalize to new data.

In our analysis, we considered three epoch values: 100, 500, and 1000. After reviewing the results, we opted for 100 epochs due to its balanced performance in terms of training MSE, test MSE, and convergence. Notably, with 100 epochs, the log MSE ranges between 10^-4 to 10^-5, slightly higher than the fluctuating log MSE around 10^-5 to 10^-6 observed with 500 epochs. Setting the epoch to 10000 resulted in increasing instability (see Table 5.1).

However, it's essential to note an overfitting problem when evaluating the testing output. We examine the MSE of the testing set across all three options. The MSE is the lowest when the number of epoch is 1000, at 1.2921. The MSE is 2.289 when epoch is set to 100 and 2.8034 when epoch is set to 500. As a result, we can conclude by balancing both the training MSE and testing MSE, the optimal epoch is 100.

Result

Our best model uses 3 hidden layers, RelU as activation function, 100 nodes for the hidden layers, adam as optimizer and epoch = 100.

Part C

In Part C we ran the model twice: once with the random split and the second time again with the best parameters discussed in Part B. However, we noticed the following issue, which we also discussed with the Professor:

The assignment asks to “randomly split the data into training set, validation set, and test set”. We did use a 80%-10%-10% split, as the original model also used 80% as training data. However, the original model as stated by the provided code had only two sets: training (80%) and testing set (20%), which was used as validation set. This means that our validation set for the random split has 50% less entries. Therefore, those models aren’t directly comparable.

Keeping this in mind, we did get a better result with our prior model, compared to the new model with random split (see Table C.2). We expected it, as the parameters were chosen in a way to optimize the model based on the non-random data set. Both models however have similar convergence properties (see Table C.1).

To compensate for this, we decided to additionally create a copy of our code, this time however with a new split (80-10-10) (this is the only thing we changed). We added the results for our “new” Part A in Table C.3. During the process, we saw that the selection of the hyperparameters didn’t change. From a MSE and MAE point of view, our model was also improved.

If we now compare our results, we still see that the original model with the new split has a better performance, supporting our argumentation (Table C.4)

Appendix

Table 1.1

NN Errors with Different Layers

Layers	MSE	MAE
1	0.000132	0.007840
2	0.000087	0.006193
3	0.000023	0.003419

Table 2.1

DNN Errors with Different Activators

Activation Function	MSE	MAE
RelU	0.000023	0.003419
tanh	0.001173	0.028538
sigmoid	0.038992	0.171201

Table 3.1

DNN Errors with Different Neurons Per Layer

Neurons (per layer)	MSE	MAE
100	0.000023	0.003419
50	0.000169	0.009834
10	0.033577	0.161895

Table 4.1

DNN Errors with Different Optimizers

Optimizer	MSE	MAE
Adam	0.000023	0.003419
RMSProp	0.000964	0.027500
Momentum	0.001685	0.031849

Table 4.2

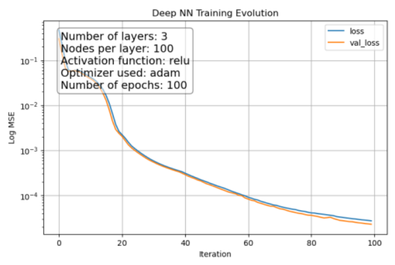
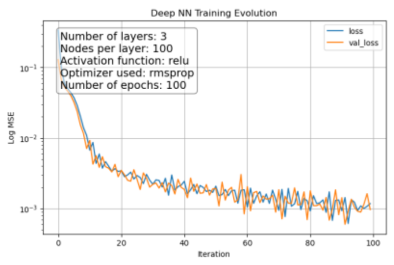
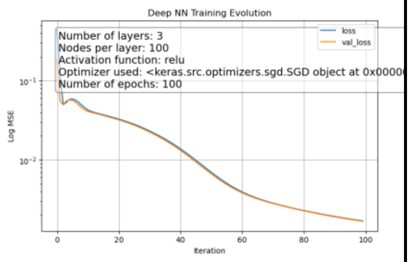
DNN Training evolution plot with Adam	DNN Training evolution plot with RMSProp	DNN Training evolution plot with SGD (momentum)
 Number of layers: 3 Nodes per layer: 100 Activation function: relu Optimizer used: adam Number of epochs: 100	 Number of layers: 3 Nodes per layer: 100 Activation function: relu Optimizer used: rmsprop Number of epochs: 100	 Number of layers: 3 Nodes per layer: 100 Activation function: relu Optimizer used: <keras.src.optimizers.sgd.SGD object at 0x0000 Number of epochs: 100

Table 4.3

	SGD	RMSprop	Adam
Learning Rates	Fixed learning rate	Adapts learning rates individually based on the running average of squared gradients	Combines adaptive learning rates for each parameter using gradient and squared gradient.
Momentum (Gradient Descent)	Utilises momentum to accelerate optimization.	Not included	Incorporates momentum-like terms for adaptive learning rates.
Bias Correction	No bias correction.	Often includes bias correction to mitigate early training issues.	Includes bias correction for first and second-order moment estimates.
Convergence	May be sensitive to the choice of learning rate.	Slow convergence	Tends to converge faster, often more robust due to adaptive rates and momentum.

Table 5.1

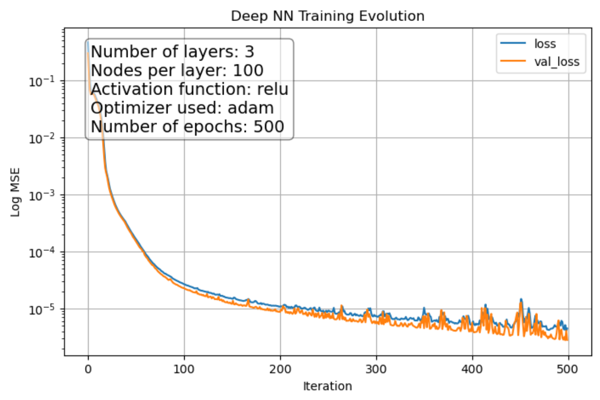
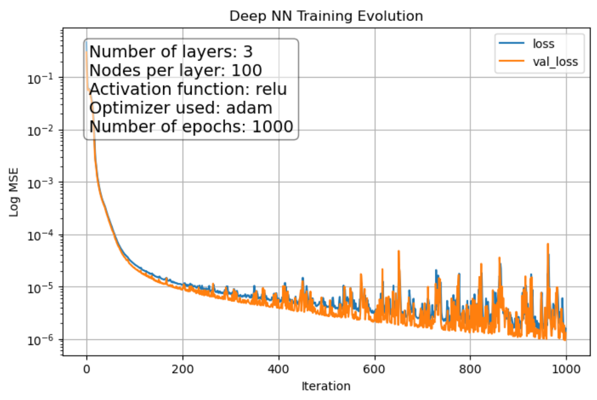
Using epoch = 500	Using epoch = 1000
 Number of layers: 3 Nodes per layer: 100 Activation function: relu Optimizer used: adam Number of epochs: 500	 Number of layers: 3 Nodes per layer: 100 Activation function: relu Optimizer used: adam Number of epochs: 1000

Table C.1: Comparison Random and previous split

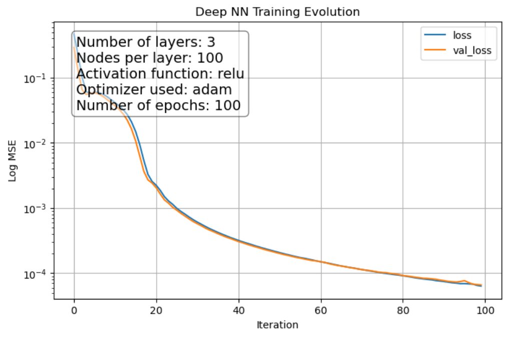
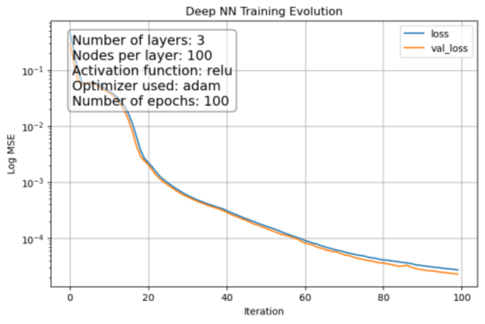
Split	**Random (80-10-10)**	**Normal (80-20) (Part A-B)**
Set size	Training size 2400 Validation size 300 Test size 300	Training size 2400 "Validation" size 600
Convergence		

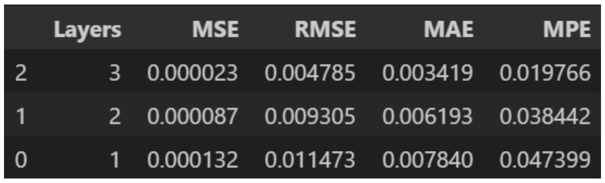
Table C.2: Comparison Random and previous split

Random	MSE	RMSE	MAE	MPE
no	0.000023	0.004785	0.003419	0.019766
yes	0.000066	0.008142	0.005320	0.032878

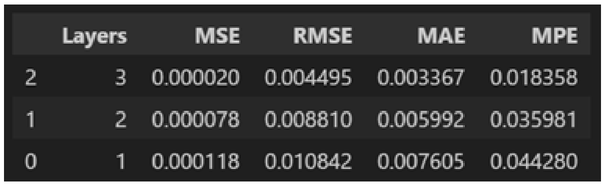
Table C.3: Comparison of previous model with old (80-20) and new split (80-10-10) for new Part A

Old split

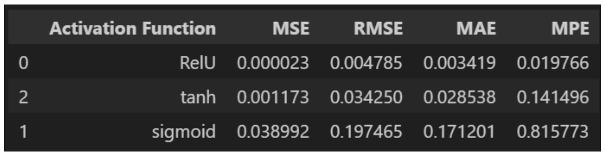
New split



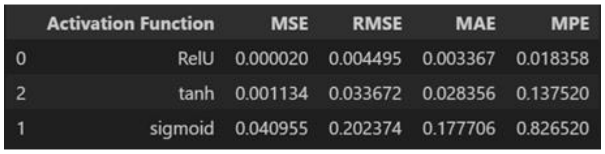
	Layers	MSE	RMSE	MAE	MPE
2	3	0.000023	0.004785	0.003419	0.019766
1	2	0.000087	0.009305	0.006193	0.038442
0	1	0.000132	0.011473	0.007840	0.047399



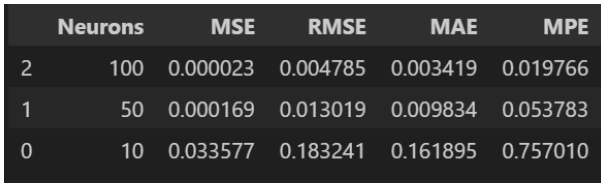
	Layers	MSE	RMSE	MAE	MPE
2	3	0.000020	0.004495	0.003367	0.018358
1	2	0.000078	0.008810	0.005992	0.035981
0	1	0.000118	0.010842	0.007605	0.044280



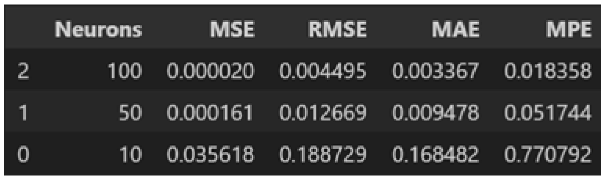
	Activation Function	MSE	RMSE	MAE	MPE
0	RelU	0.000023	0.004785	0.003419	0.019766
2	tanh	0.001173	0.034250	0.028538	0.141496
1	sigmoid	0.038992	0.197465	0.171201	0.815773



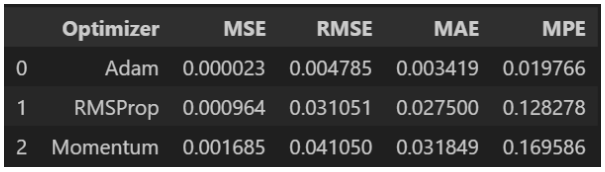
	Activation Function	MSE	RMSE	MAE	MPE
0	RelU	0.000020	0.004495	0.003367	0.018358
2	tanh	0.001134	0.033672	0.028356	0.137520
1	sigmoid	0.040955	0.202374	0.177706	0.826520



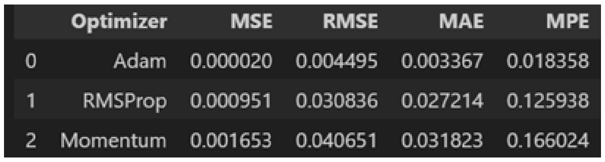
	Neurons	MSE	RMSE	MAE	MPE
2	100	0.000023	0.004785	0.003419	0.019766
1	50	0.000169	0.013019	0.009834	0.053783
0	10	0.033577	0.183241	0.161895	0.757010



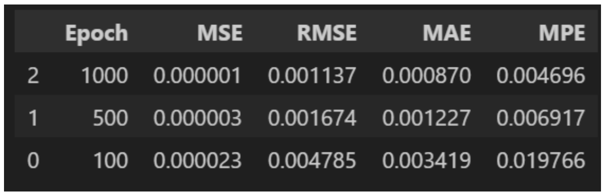
	Neurons	MSE	RMSE	MAE	MPE
2	100	0.000020	0.004495	0.003367	0.018358
1	50	0.000161	0.012669	0.009478	0.051744
0	10	0.035618	0.188729	0.168482	0.770792



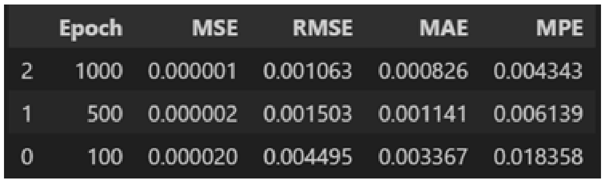
	Optimizer	MSE	RMSE	MAE	MPE
0	Adam	0.000023	0.004785	0.003419	0.019766
1	RMSProp	0.000964	0.031051	0.027500	0.128278
2	Momentum	0.001685	0.041050	0.031849	0.169586



	Optimizer	MSE	RMSE	MAE	MPE
0	Adam	0.000020	0.004495	0.003367	0.018358
1	RMSProp	0.000951	0.030836	0.027214	0.125938
2	Momentum	0.001653	0.040651	0.031823	0.166024



	Epoch	MSE	RMSE	MAE	MPE
2	1000	0.000001	0.001137	0.000870	0.004696
1	500	0.000003	0.001674	0.001227	0.006917
0	100	0.000023	0.004785	0.003419	0.019766

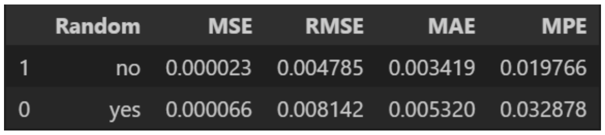


	Epoch	MSE	RMSE	MAE	MPE
2	1000	0.000001	0.001063	0.000826	0.004343
1	500	0.000002	0.001503	0.001141	0.006139
0	100	0.000020	0.004495	0.003367	0.018358

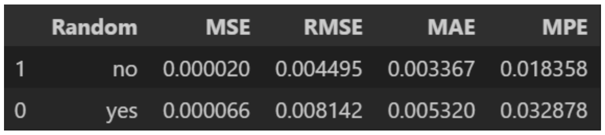
Table C.4: Final comparison of random vs. fixed split and old split (80-20) and new split (80-10-10)

Old split (80-20)

New split (80-10-10)



	Random	MSE	RMSE	MAE	MPE
1	no	0.000023	0.004785	0.003419	0.019766
0	yes	0.000066	0.008142	0.005320	0.032878



	Random	MSE	RMSE	MAE	MPE
1	no	0.000020	0.004495	0.003367	0.018358
0	yes	0.000066	0.008142	0.005320	0.032878