BLACK-SCHOLES CASE STUDY

PART A

The Black-Scholes model estimates the theoretical value of a derivative. It uses the price of the security in question, time and risk to evaluate its price. We aim to use artificial neural networks to approximate the nonlinear function of Black-Scholes to price financial derivatives. The computing power of the networks and its learning ability from test data will allow for accurate approximation. In order to get the best model, we will test different types of NN models.

We first test different models in terms of layers in their structures, with single, three-layer and a ten-layer model. As the number of layers increases, the expectation is that the model becomes more accurate in prediction, however, adds to its complexity. The results show that the model with 10 layers converge at the lowest MSE of 10^{-5} .

We now experiment with different activation functions. These functions define how the parameters will interact with each neuron, and which 'neurons' are to be activated. First we use Sigmoid, which is useful to predict probability as an output as it exists between 0 and 1. Moving on, we will use Tanh, as it allows us to input negative numbers; and finally with Relu, which is not restricted on the positive side, however, predicts 0 for any negative inputs. Relu shows the best results with an MSE of 10^{-5} compared to tanh that converges at 10^{-3} and Sigmoid at $4 * 10^{-2}$.

Next, we make a choice to choose the number of neurons in our network. As we increase the number of neurons, it increases the level of interaction of the input layer to the hidden layer and hence leads to more accurate results, as it improves the understanding of the data by the model. The number of neurons in each layer affects how many iterations are needed for the model to converge. For 100 neurons it converges after approximately 40 iterations, for 50 neurons it is around 60 and for 10 neurons it is closer to 80.

Now, we experiment with different optimizers. Optimizers represent different methods of reducing loss functions and improving learning rate. We apply 'ADAM' short for adaptive moment estimation, RMSprop that is a variation of gradient descent and SGD short for stochastic gradient descent. ADAM shows the best results, so that is applied onwards.

The last test seeks to investigate the impact of epochs. One epoch is the number of times the model scans the data. We expect that a larger number of epochs results in better precision in terms of MSE. We test with 100, 500, and 1000 epochs.100 epochs converges at an MSE of 10^{-5}, while the two other tests start to fluctuate around the same MSE. With these model parameters and this dataset 100 epochs is deemed to be the best model.

PART B

The best model we found had the following parameters: deep neural network with 10 layers, 50 nodes in each layer, utilizing optimizer adam, utilizing activator relu, and running with 100 epochs. To determine the best model, we experimented with each parameter, choosing the best option with the lowest MSE and using that model to experiment with the next parameter. We repeated this until we tested all 5 parameters, resulting in our best model which we stated above. The best model we generated resulted in a mean squared error (MSE) of 9.958287936359416e-06 (or 9.958e-6).

As the number of layers in the neural network structure increases, we observe in our models that the MSE decreases. Although this is true, we selected the model with 3 layers. This is because the MSE with 3 layers is significantly better than just 1 layer. While it's true that the model with 10 layers has an even lower MSE, the MSE is not notably different, but will increase the model complexity significantly. Regarding the activation functions, we found that using relu resulted in the lowest MSE compared to using sigmoid and tahn. Using the sigmoid activation function resulted in the highest MSE, this is likely because the sigmoid function is a logistic function used to normalize the output in the range of 0 to 1. While tahn is a hyperbolic tangent function giving values between -1 and 1. In comparison, relu’s only limit is that negative values are converted into 0, otherwise the output can be any positive value. Because of this, it can improve the prediction accuracy and reduce MSE.

For the number of neurons in hidden layers, we found that having 50 neurons resulted in the lowest MSE. As the number of neurons increases, it is expected that the MSE decreases, so using trial and error we can test different numbers of neurons until we reach the convergence point!

Layers	**10**
Activation	**Relu**
Neurons	**100**
Optimizer	**Adam**
Epochs	**100**

PART C

Now that the best model has been selected (refer to part B), we then randomly split data into a training set, validation set, and testing set. Recall that the best model chosen has 10 layers as the deep neural network structure, relu as the activation type, adam as optimizer, 100 as number of neurons in hidden layer, and lastly 100 as the multiple ephocs. Initially, there were two data sets: training and testing. In this section, the data was randomly split into 80% training set, 10% validation set, and 10% testing set. After running the code and using randomization in the model again, the results converged in a similar result as the one from part B). In which, the MSE is 1.743470356624771e-05 (or 1.743e-05). This is moderately higher than the one from part B) thus showing added variation with additional sets. Overall, the new randomly splitted data has provided reduction in accuracy though it also eliminates selection bias and minimize data corruption.