Project 2 - Trading costs

May 31, 2025

0.0.1 NOTE:

We ran the code on several computers and got varying results despite using the same package versions, etc. Interpretations below are given for results on a machine running Intel i5-1135G7 CPU with environment variable `TF_ENABLE_ONEDNN_OPTS=0`

1 Overview

In this project we were given the task of developing a neural network which trades as well as possible to hedge a European call option in the Black-Scholes model given different levels of transaction costs, (γ) and to evaluate the NN's performance at these different levels of transaction costs.

For this we designed 3 neural networks:

NN1 - Initial wealth network (pi) As in class, this NN aims to determine the optimal initial wealth

NN2 - The hedging network (hedge): The 'engine' of the wealth network, this NN aims to determine the optimal hedging position for each step. It takes in $(TTM, S_t, H_{t-1}, S_{t-1})$ to figure out H_t

NN3 - The wealth network (model_wealth): If NN2 is the engine, this is the entire car this NN takes in $[(TTM, S_t), (S_t, S_{t-1})]$, runs NN1, and NN2 to try and determine the optimal initial wealth, the optimal hedging positions, and outputs the terminal wealth it gets to using the determined strategy

For the parameters of the call option we chose: - $N = 20$ - $S_0 = 150$ - $K = 150$ - $T = 2$

And for the parameters of the Black-Scholes stock path generating model we chose - \$ = 0.02 \$ - $\sigma = 0.2$

The NN parameters were: - optimizer = "Adam" - epochs = 20 - batch_size = 128 - activator = "tanh" - $d = 2$ which is the number of hidden layers in hedging network - $n = 200$ which is the number of nodes in hedging network - $d_{V_0} = 0$ which is number of hidden layers in pi model

We trained the model on 40000 samples, and tested & evaluated on 10000 samples The models were trained on the same training and test sets

Further comments/explanations throughout the code

2 Setup

2.1 Libraries & Seed

[1]:
```
from keras.layers import Input, Dense, Add, Dot, Flatten, Lambda, Concatenate,␣
 ↪Reshape, Subtract, Multiply
from keras.models import Model
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np
import random
import scipy.stats as scipy
import tensorflow as tf

random.seed(5)
np.random.seed(5)
tf.random.set_seed(5)
```

```
2025-05-31 21:43:44.087169: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2025-05-31 21:43:44.117856: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2025-05-31 21:43:44.118404: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2025-05-31 21:43:44.782868: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

2.2 Constants

[2]:
```
#### MODEL CONSTANTS ####
N = 20 # time disrectization
S0 = 150 # initial value of the asset
strike = 150 # strike for the option
T = 2 # maturity
mu = 0.02  # excess return --- Q-dynamics: mu=0
sigma = 0.2 # volatility

#### NETWORK CONSTANTS ####
Ktrain = 40000 # size of training data
Ktest = 10000 # size of test data
epochs = 20
batch_size = 128
activator = "tanh" # activation function for all networks
# network structure
```

```
learnV0 = True # learn setup wealth. If set to False, then it uses the MC␣
↪estimate as initial wealth
d = 2 # number of hidden layers in hedging network
n = 200  # number of nodes in hedging network
prec = 2 # digits shown in print commands
TimePoints = np.linspace(0, T, N+1) # time points
m = 1 # dimension of price
d_V0 = 0 # number of hidden layers in pi model
```

2.3 Helper functions

[3]:
```
# returns time to maturity from t
def TimeConv(t):
     return T - t  # the NN expects a time to maturity as information



# BS model
def path(S0, mu, sigma, Timepoints, R):
    """
    Generates stock price path from Geometric Brownian motion. Returns an tuple␣
↪of shape (R, N + 1, 1)
    """
    N = len(Timepoints) - 1
    X = np.zeros((R, N + 1)) + np.log(S0) # the last 1 is for the dimension of␣
↪the model
    mu_log = mu - sigma ** 2 / 2
    for j in range(N):
        dt = Timepoints[j + 1] - Timepoints[j]
        dZ = np.random.normal(0, 1, R)
        increment = mu_log * dt + sigma * dZ * np.sqrt(dt)
        X[:, j+1] = X[:, j] + increment
    S = np.exp(X)
    return np.reshape(S,(R, N + 1, 1))



# payoff of European call
def f(S):
    return np.maximum(S[:, N, 0] - strike, 0)



# shapes paths into convenient format
def shape_inputs(paths):
    K, N, d = np.shape(paths)
    N = N - 1
    x = [np.zeros((K, N, 1 + d)), np.zeros((K, N, d))]
    for i in range(N):
        x[0][:, i, 0] = np.repeat(TimeConv(TimePoints[i]), K)
    x[0][:, :, 1] = paths[:, 0:N, 0]
    x[1][:, :, 0] = paths[:, 1:(N + 1), 0] - paths[:, 0:N, 0]
    return x
```

3 Neural networks

[4]:
```
#### NN1: Initial wealth ####
# Model name: pi
# architecture of the network --- expects (price) vector
# g(S0) = PriceOption
def build_pi_model():
    inp = Input(shape=(m,))
    V0 = inp

    for _ in range(d_V0):
        V0 = Dense(1, activation=activator)(V0)
    out = Dense(1, activation='linear', trainable=learnV0)(V0)

    return Model(inputs=inp, outputs=out)


#### NN2: Hedging network ####
# Model name: hedge
# architecture of the network --- expects (timeToMaturity, Price, Previous␣
 ↪Hedge position, Previous Stock price) vector
# g(timeToMaturity, Price, Previous Hedge position, Previous Stock price) =␣
 ↪PositionSize
def build_hedge_model():
    inp = Input(shape=(3 + m,))
    hedge_pos = inp

    for _ in range(d):
        hedge_pos = Dense(n, activation=activator)(hedge_pos)
    out = Dense(m, activation='linear')(hedge_pos)

    return Model(inputs=inp, outputs=out)


#### NN3: Wealth network ####
# Model name: model_wealth
# Architecture of the wealth network --- expects [(TimeToMaturity, Price), (S_t␣
 ↪- S_{t-1})] array
# g([(TimeToMaturity, Price), (S_t - S_{t-1})]) = TerminalWealth
def get_wealth_network(gamma, hedge, pi):
    """
    Creates a model wealth network for the specific trading cost multiplier␣
↪gamma given as an argument
    Returns the model wealth network
    """
    # (timeToMaturity, Price) vector
    Obs = Input(shape=(N,1+m))
    # (S_t - S_{t-1}) vector, i.e. price increments
    Incr = Input(shape=(N,m))
    inputs = [Obs, Incr]

    # estimating wealth with NN1
    first_price = Lambda(lambda x: x[:, 0, 1:(m + 1)])(Obs)
    V0 = pi(first_price)

    # initial previous hedge = zero
    prev_h = Lambda(lambda x: tf.zeros((tf.shape(x)[0], 1)))(Obs)
    # initial previous price = S_0
    prev_price = Lambda(lambda x: x[:, 0, 1:(1 + m)])(Obs)

    # store hedging positions
    h_slices = []

    # calculate hedging positions
    for t in range(N):
        # t-th (TTM, S_t) vector
        obsSlice = Lambda(lambda x, i=t: x[:, i, :])(Obs)
        # concatenate with previous hedge and previous price
        obs_with_prev = Concatenate(axis=1)([obsSlice, prev_h, prev_price])

        H = hedge(obs_with_prev)
        # store H (reshape for convenience)
        h_slices.append(Reshape((1, 1))(H))

        # update for next iteration
        prev_h = H
        prev_price = Lambda(lambda x, i=t: x[:, i, 1:(1 + m)])(Obs)

    # vector/layer of all positions H
    H = Concatenate(axis=1)(h_slices)

    # compute gains/losses from trading
    Incr_flat = Flatten()(Incr)
    H_flat = Flatten()(H)
    Gain = Dot(axes=1)([H_flat, Incr_flat])
    wealth_gain = Add()([V0, Gain]) # add gains/losses to V0

    # NOTE: This is where trading costs are calculated
    # compute variables for trading costs
    H_prev = Lambda(lambda x: x[:, :-1, :])(H)
    H_curr = Lambda(lambda x: x[:, 1: , :])(H)
    dH = Subtract()([H_curr, H_prev])
    S_prev = Lambda(lambda x: x[:, :-1, 1:(m + 1)])(Obs)

    # compute per-step trading cost
    cost_per_step = Multiply()([dH, S_prev])
    abs_cost_per_step = Lambda(lambda x: tf.abs(x))(cost_per_step)

    # compute total trading cost
    cost_sum = Lambda(lambda x: tf.reduce_sum(x, axis=1,␣
↪keepdims=True))(abs_cost_per_step)
    cost = Lambda(lambda x: gamma * x)(cost_sum)
    cost_flat = Flatten()(cost)

    # compute total wealth: V0 + Gain - Total Cost
    total_wealth = Subtract()([wealth_gain, cost_flat])

    # Define outputs
    out = [total_wealth, H]

    # compile model
    wealth = Model(inputs=inputs, outputs=out)
    wealth.compile(optimizer="adam", loss=["mse", "mse"],
                   loss_weights = [1.0, 0.0])

    return wealth
```

4 Fitting and evaluating

4.1 Datasets

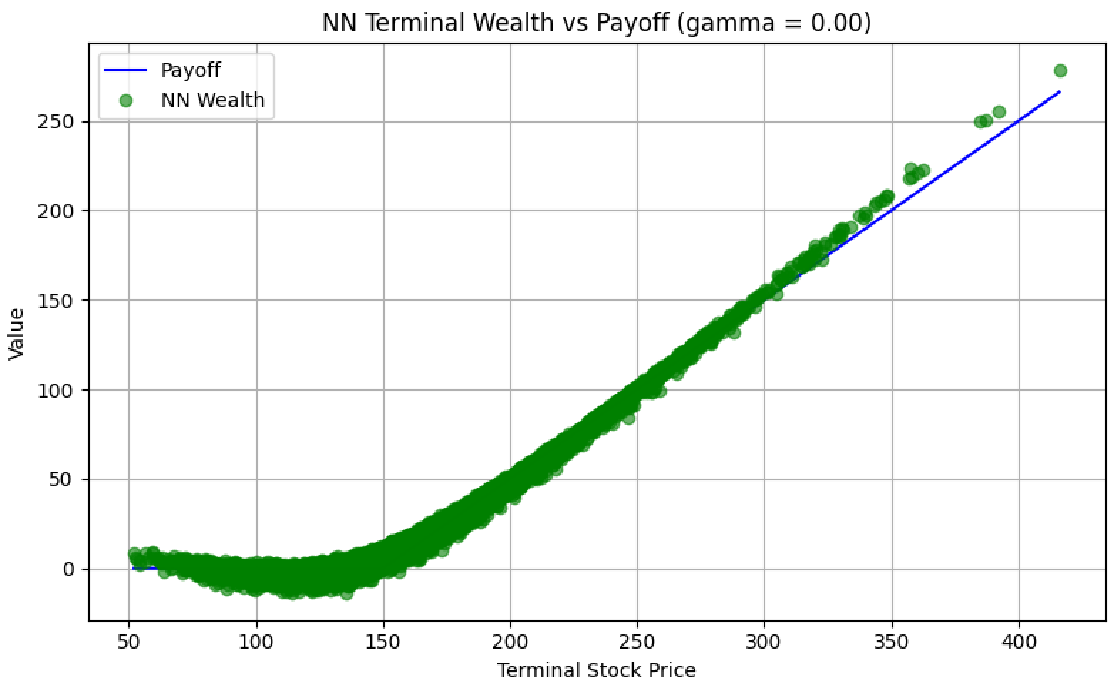
[5]:
```
#### Training set ####
trainpaths = path(S0, mu, sigma, TimePoints, Ktrain)
# convert paths to (TTM, Price) vectors
xtrain = shape_inputs(trainpaths)
# option payoffs
ytrainpayoffs = f(trainpaths).reshape(-1, 1)
# Vector does not affect model but needed for extracting H
ytraindummy = np.zeros((Ktrain, N, 1))
# Combining them makes ytrain
ytrain = [ytrainpayoffs, ytraindummy]

#### Test set ####
testpaths = path(S0, mu, sigma, TimePoints, Ktest)
```

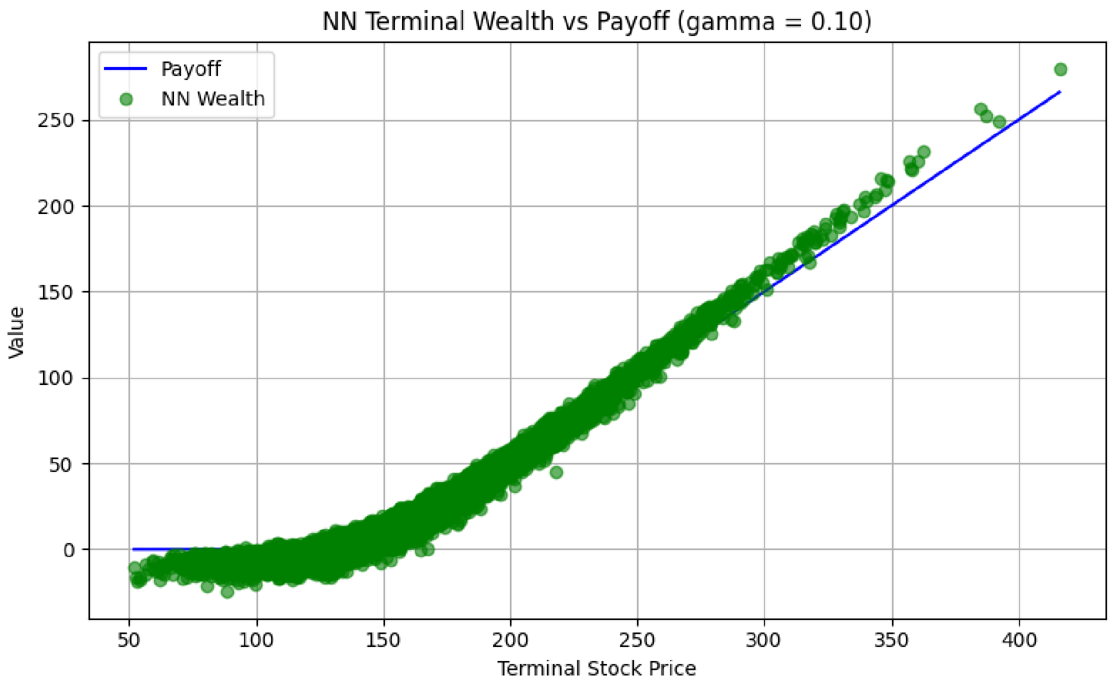
```
xtest = shape_inputs(testpaths)
ytest = f(testpaths).reshape(-1, 1) # only contains payoffs

#### Initialise gammas and lists containing data ####
gammas = np.linspace(0, 1, 11)
hedging_errors = []
hedging_volume = []
initial_wealths = []


#### Task 4
# select some sample path for the visualisation
path_index = 0
visual_path = testpaths[path_index:path_index + 1, :, :]

xtest_single = shape_inputs(visual_path)
hedge_positions = {}
```

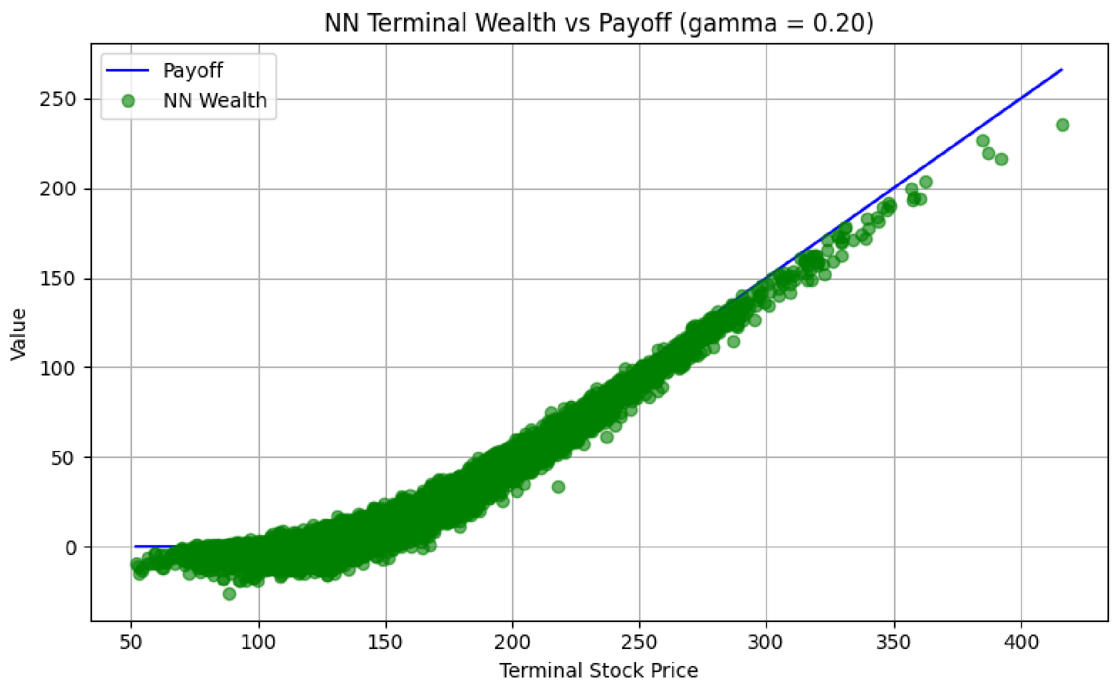
4.2 Calculating results

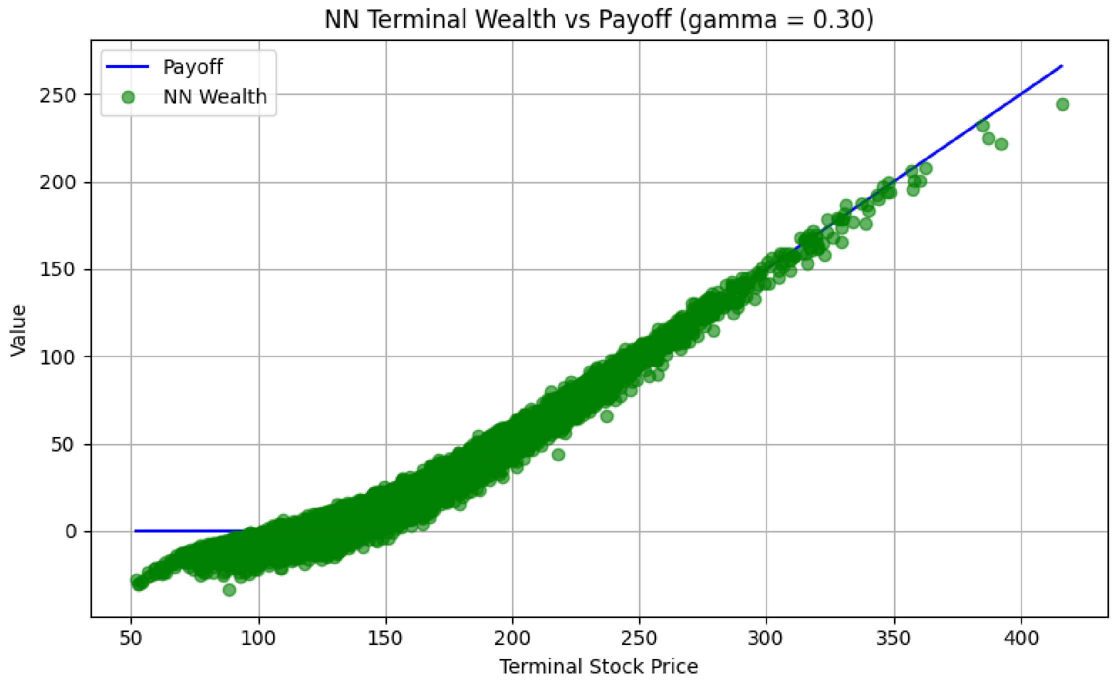
[6]:
```
for gamma in gammas:
    print(f"\nTraining & testing for   = {gamma:.2f}")

    # Build pi and hedge models
    hedge = build_hedge_model()
    pi = build_pi_model()
    V0_train = np.mean(ytrainpayoffs)
    pi.set_weights([np.array([[0]]), np.array([V0_train])])

    # Model training
    model_wealth = get_wealth_network(gamma, hedge, pi)
    model_wealth.fit(x=xtrain, y=ytrain, epochs=epochs, batch_size=batch_size,␣
 ↪verbose=0)

    # Get and append initial wealth from pi model
    V0_learned = pi.get_weights()[1][0] # The bias term is the learned initial␣
 ↪wealth
    initial_wealths.append(V0_learned)

    # Model testing & Getting hedging error
    # print(f"\nTesting")
    NNtest, H_vals = model_wealth.predict(xtest, verbose=0)
    error = np.mean((NNtest - ytest) ** 2)
    hedging_errors.append(error)

    # Calculating average hedging volume across all testset
    dH_vals = np.abs(np.diff(H_vals, axis=1))
    avg_volume = np.mean(np.sum(dH_vals, axis=1))
    hedging_volume.append(avg_volume)

    # Printing results
    print(f"Avg trading volume: {avg_volume}")
    print(f"Avg error test: {error}")
    print(f"Std Dev of NNtest - ytest: {np.std(NNtest - ytest):.2f}")

    # Individual NN terminal wealth vs option payoffs plot
    plt.figure(figsize=(8, 5))
    sort_idx = np.argsort(testpaths[:, N, 0])
    plt.plot(testpaths[sort_idx, N, 0], ytest[sort_idx], label='Payoff',␣
↪color='blue')
    plt.plot(testpaths[:, N, 0], NNtest, 'o', alpha=0.6, label='NN Wealth',␣
↪color='green')
    plt.title(f"NN Terminal Wealth vs Payoff (gamma = {gamma:.2f})")
    plt.xlabel("Terminal Stock Price")
    plt.ylabel("Value")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    # plt.show()
    plt.savefig(f"./plots/Gamma {gamma:.2f}.png")

    # Get single value for task 4
    NN_vals_single, H_vals_single = model_wealth.predict(xtest_single,␣
↪verbose=0)
    hedge_positions[gamma] = H_vals_single.reshape(-1)
```

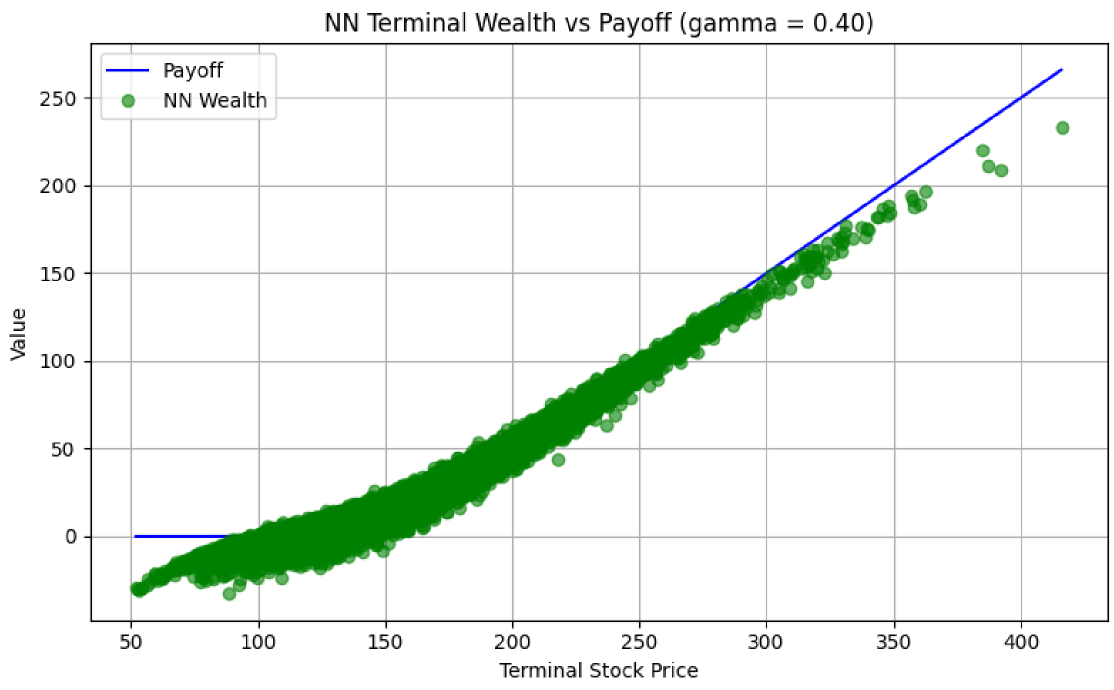
```
Training & testing for   = 0.00
Avg trading volume: 1.441604733467102
Avg error test: 16.33282092645169
Std Dev of NNtest - ytest: 4.04

Training & testing for   = 0.10
Avg trading volume: 1.0135620832443237
Avg error test: 40.95344687520104
Std Dev of NNtest - ytest: 6.29

Training & testing for   = 0.20
Avg trading volume: 0.79059237241745
Avg error test: 49.42346477666993
Std Dev of NNtest - ytest: 6.93

Training & testing for   = 0.30
Avg trading volume: 0.557379961013794
Avg error test: 68.78460210573849
Std Dev of NNtest - ytest: 7.89

Training & testing for   = 0.40
Avg trading volume: 0.4508795440196991
Avg error test: 70.76463103553944
Std Dev of NNtest - ytest: 8.07

Training & testing for   = 0.50
Avg trading volume: 0.3813835680484772
Avg error test: 83.37469811285612
Std Dev of NNtest - ytest: 9.10

Training & testing for   = 0.60
Avg trading volume: 0.3109961748123169
Avg error test: 89.314172045324
Std Dev of NNtest - ytest: 9.45

Training & testing for   = 0.70
Avg trading volume: 0.23973339796066284
Avg error test: 109.00394342321731
Std Dev of NNtest - ytest: 10.31

Training & testing for   = 0.80
Avg trading volume: 0.22116652131080627
Avg error test: 135.8267756856233
Std Dev of NNtest - ytest: 11.51

Training & testing for   = 0.90
Avg trading volume: 0.18248684704303741
Avg error test: 124.88027848862265
Std Dev of NNtest - ytest: 10.81

Training & testing for   = 1.00
Avg trading volume: 0.17781861126422882
Avg error test: 101.47985817585756
Std Dev of NNtest - ytest: 9.85
```

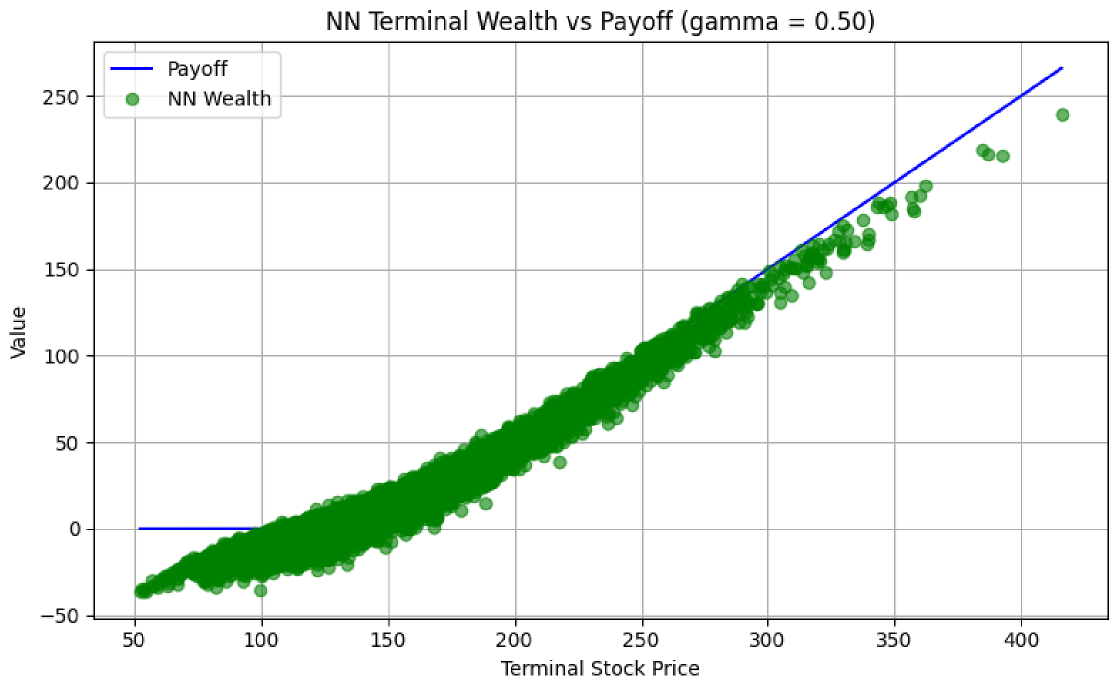

NN Terminal Wealth vs Payoff (gamma = 0.00)
Payoff
NN Wealth
Value
Terminal Stock Price
0
50
100
150
200
250
50
100
150
200
250
300
350
400

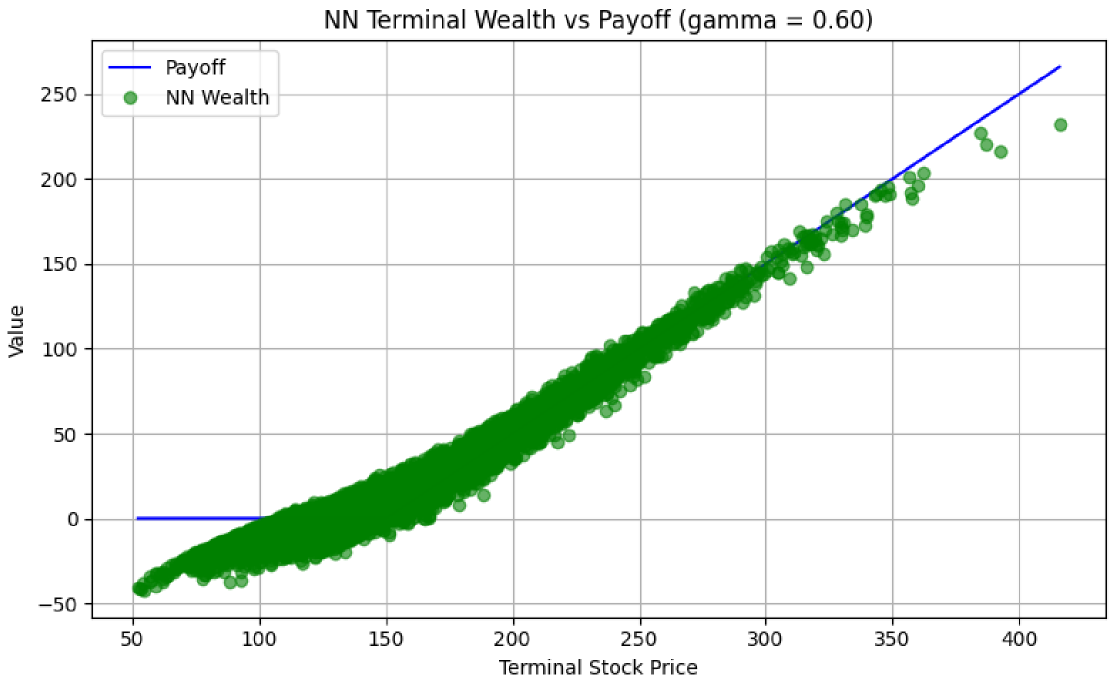

NN Terminal Wealth vs Payoff (gamma = 0.10)
Payoff
NN Wealth
Value
Terminal Stock Price
0
50
100
150
200
250
50
100
150
200
250
300
350
400

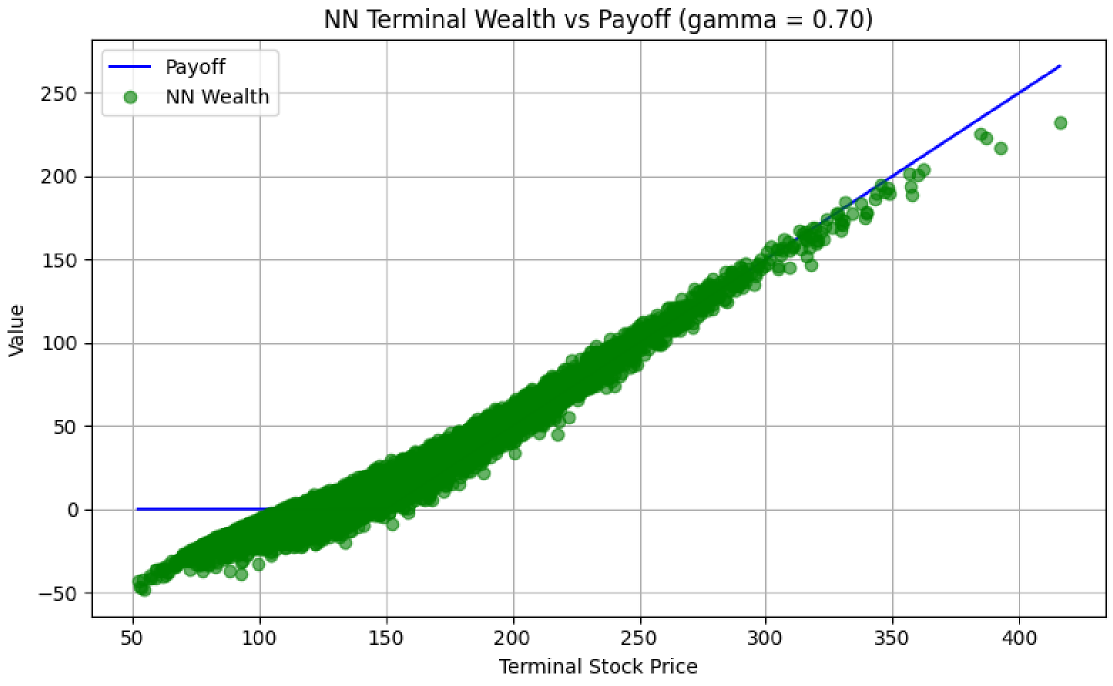

NN Terminal Wealth vs Payoff (gamma = 0.20)
Payoff
NN Wealth
Value
Terminal Stock Price
0
50
100
150
200
250
50
100
150
200
250
300
350
400

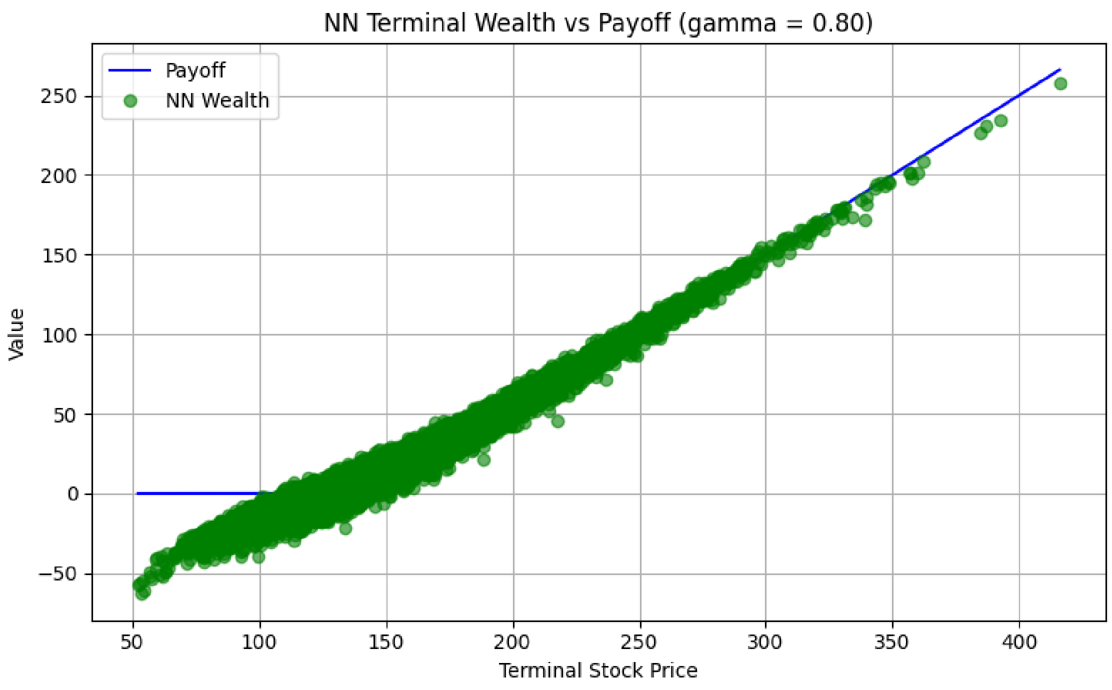

NN Terminal Wealth vs Payoff (gamma = 0.30)
Payoff
NN Wealth
Value
Terminal Stock Price
0
50
100
150
200
250
50
100
150
200
250
300
350
400

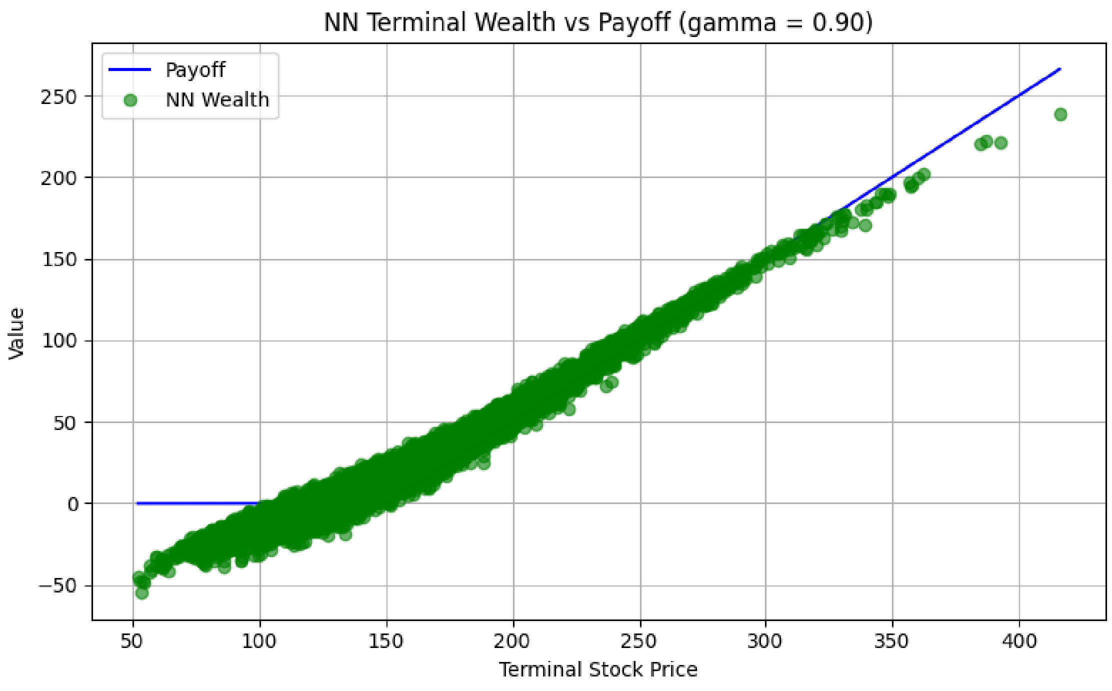

NN Terminal Wealth vs Payoff (gamma = 0.40)
Payoff
NN Wealth
Value
Terminal Stock Price

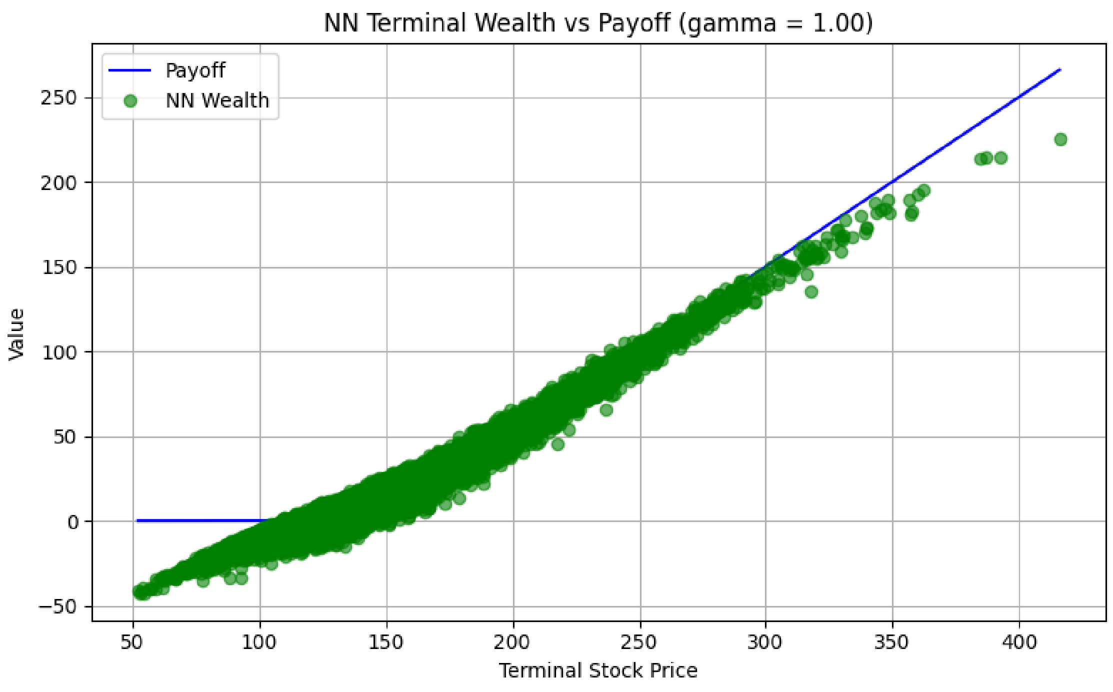

NN Terminal Wealth vs Payoff (gamma = 0.50)
Payoff
NN Wealth
Value
Terminal Stock Price


NN Terminal Wealth vs Payoff (gamma = 0.60)
Payoff
NN Wealth
Value
Terminal Stock Price
250
200
150
100
50
0
−50
50
100
150
200
250
300
350
400


NN Terminal Wealth vs Payoff (gamma = 0.70)
Payoff
NN Wealth
Value
Terminal Stock Price
250
200
150
100
50
0
−50
50
100
150
200
250
300
350
400


NN Terminal Wealth vs Payoff (gamma = 0.80)
Payoff
NN Wealth
Value
Terminal Stock Price
250
200
150
100
50
0
−50
50
100
150
200
250
300
350
400


NN Terminal Wealth vs Payoff (gamma = 0.90)
Payoff
NN Wealth
Value
Terminal Stock Price
250
200
150
100
50
0
−50
50
100
150
200
250
300
350
400



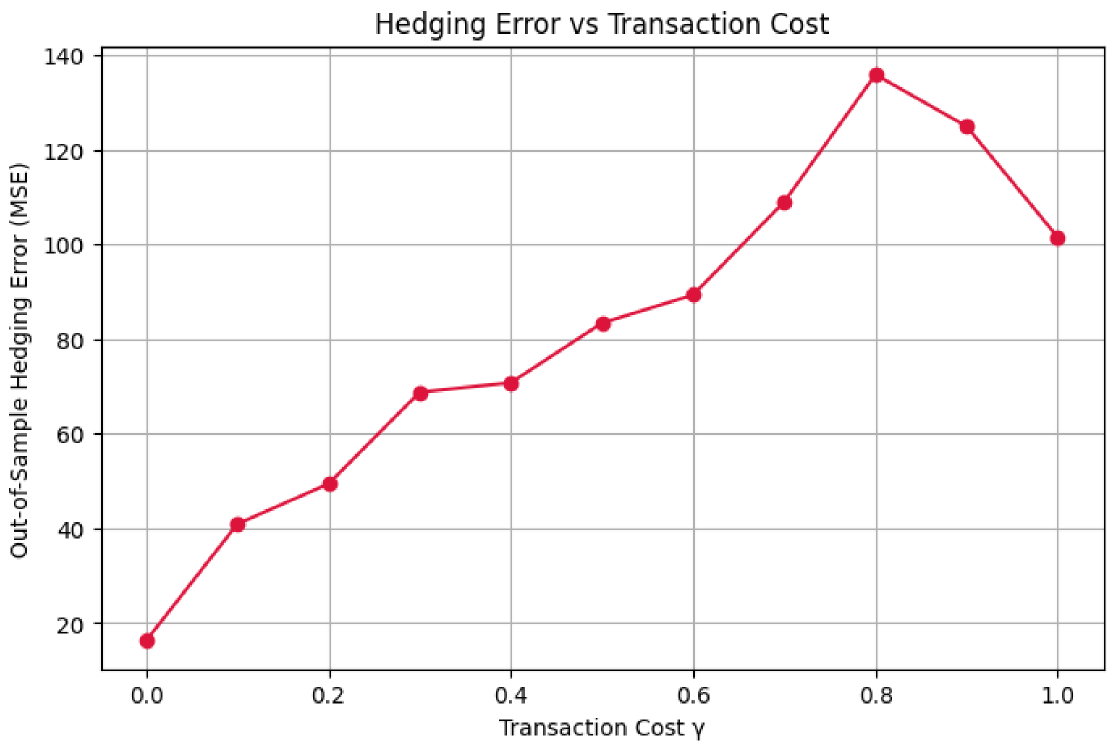
4.3 Remarks

For low γ we here see a great fit of the NN terminal wealth to the realised payoffs, with the NN even making more money than the call option at very low and very high terminal stock prices. As γ increases, as expected, this fit gets worse and worse, with the terminal wealth 'flattening out', indicating that the hedging strategy increasingly resembles a buy-and-hold strategy which loses the most money at the highest and lowest terminal stock prices.

5 Results

5.1 Hedging Error vs Transaction Cost

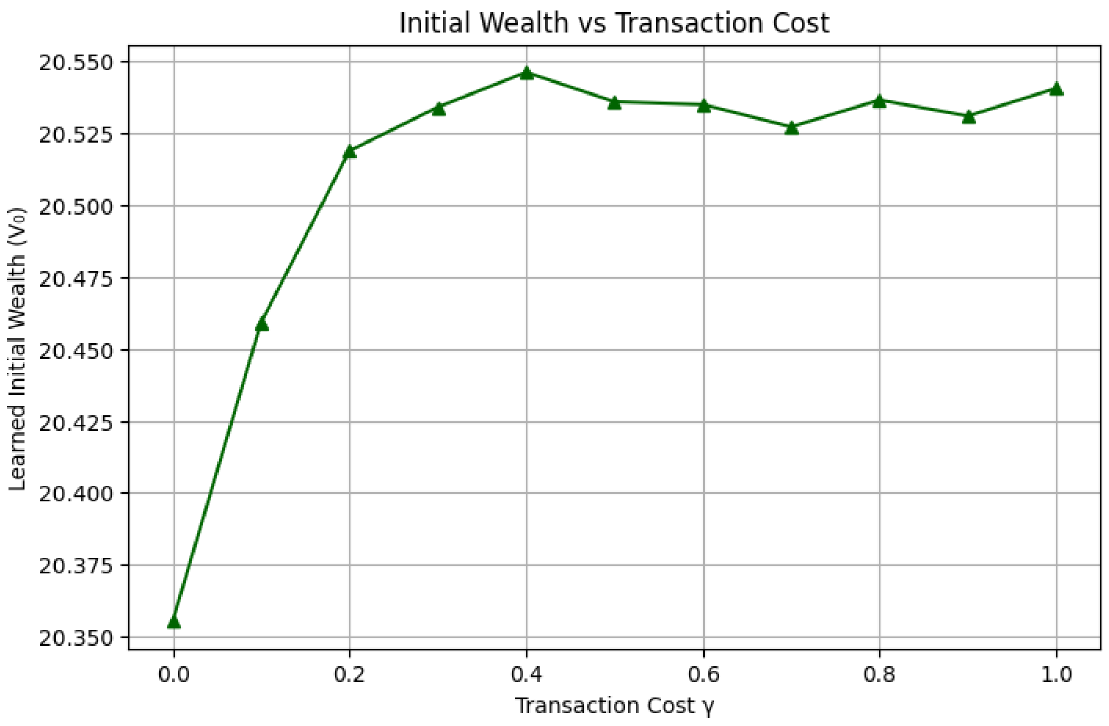
[7]:
```
plt.figure(figsize=(8, 5))
plt.plot(gammas, hedging_errors, marker='o', color='crimson')
plt.xlabel('Transaction Cost  ')
plt.ylabel('Out-of-Sample Hedging Error (MSE)')
plt.title('Hedging Error vs Transaction Cost')
plt.grid(True)
plt.show()
# plt.savefig("./plots/Hedging error.png")
```



As expected, hedging error increases with higher transaction costs almost linearly. What is interesting to note is the particularly bad fit of the model at $\gamma = 0.8$. Even more interesting is the seemingly low Out-of-Sample MSE at $\gamma = 1.0$ These point to differing hedging practices overall on this given training set for the highest γs, however the overall trend still holds and none of these MSEs exceed low or middle γ.

5.2 Initial Wealth vs Transaction Cost

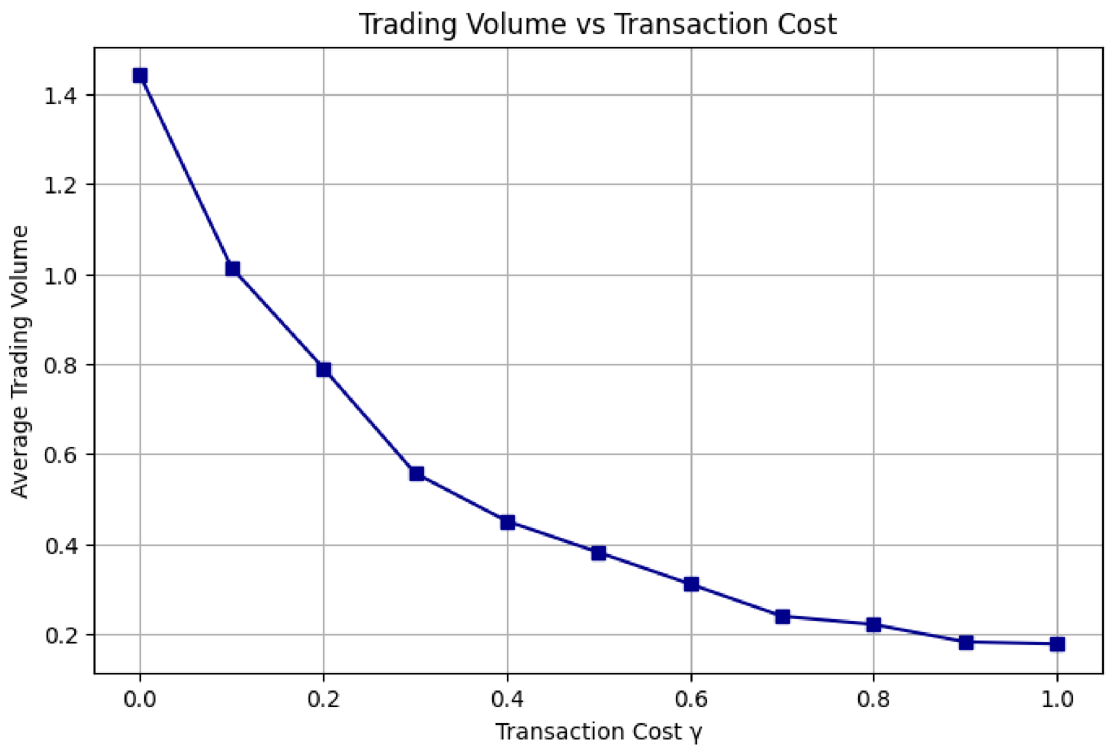
[8]:
```
plt.figure(figsize=(8, 5))
plt.plot(gammas, initial_wealths, marker='^', color='darkgreen')
plt.xlabel('Transaction Cost  ')
plt.ylabel('Learned Initial Wealth (V )')
plt.title('Initial Wealth vs Transaction Cost')
plt.grid(True)
plt.show()
# plt.savefig("./plots/Initial Wealth.png")
```



Here the initial wealth shoots up somewhat between transaction costs of 0% and 20%, where it remains relatively the same While this effect seems drastic on the plot, the y-axis shows that the difference is still relatively small

5.3 Trading Volume vs Transaction Cost

[9]:
```
plt.figure(figsize=(8, 5))
plt.plot(gammas, hedging_volume, marker='s', color='darkblue')
plt.xlabel('Transaction Cost  ')
plt.ylabel('Average Trading Volume')
plt.title('Trading Volume vs Transaction Cost')
plt.grid(True)
plt.show()
# plt.savefig("./plots/Trading Volume.png")
```

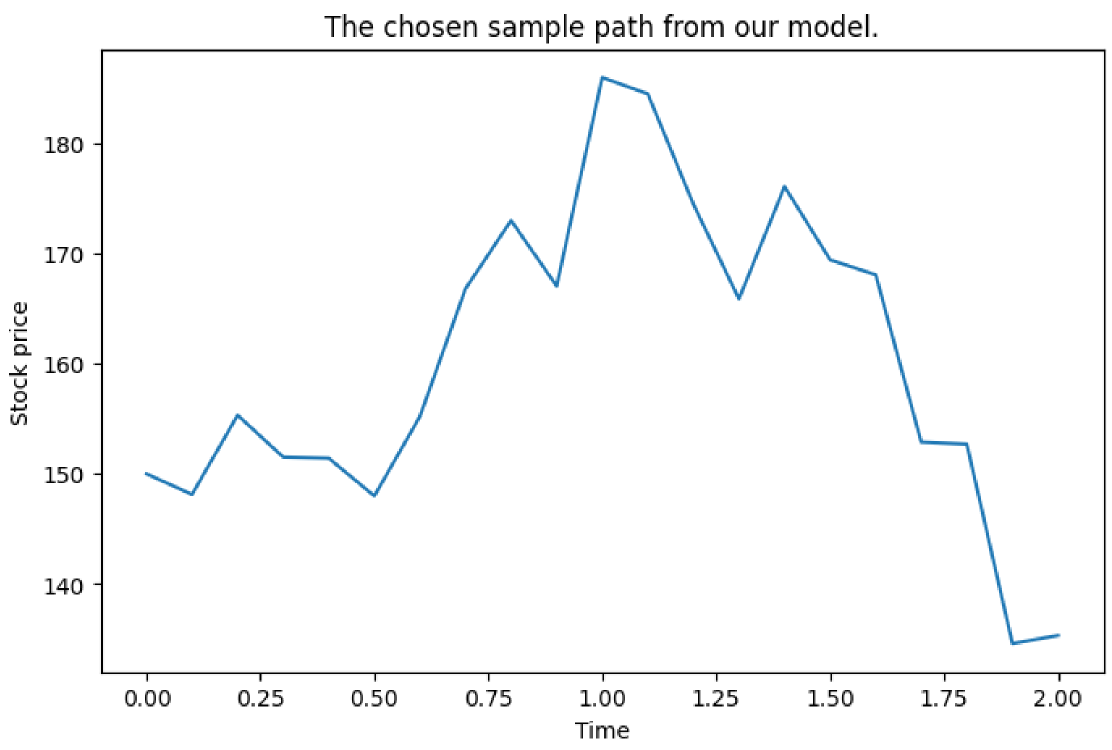


The average hedging volume steadily decreases as γ increases, as expected

5.4 Single path plots

5.4.1 Path chosen

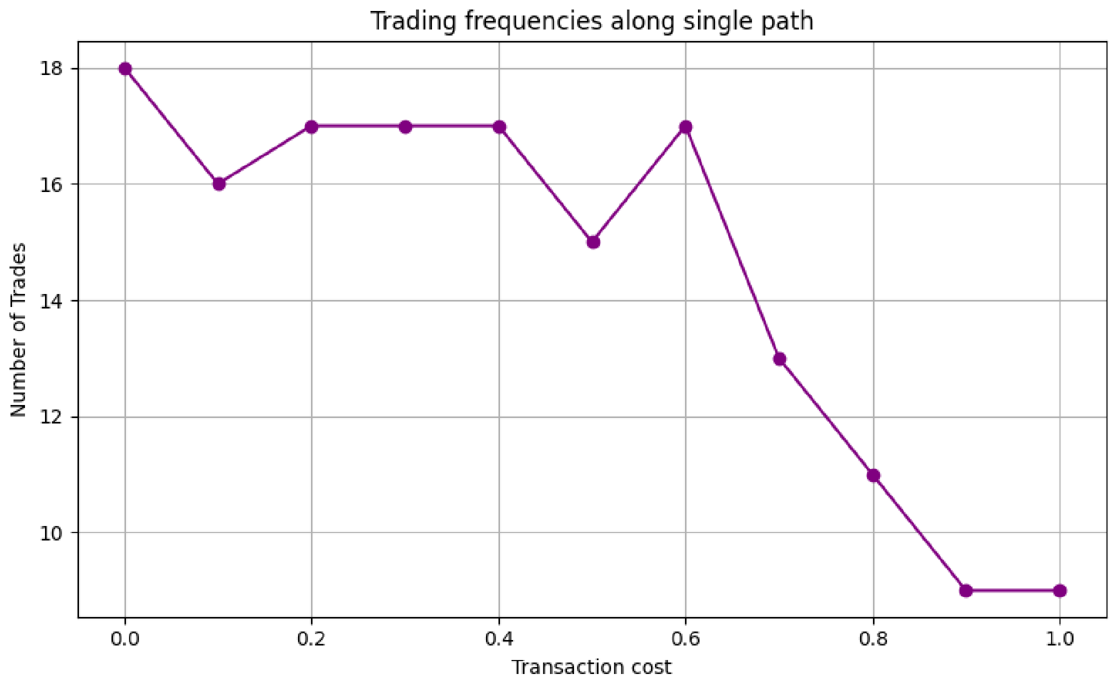
[10]:
```
plt.figure(figsize=(8, 5))
plt.plot(TimePoints, testpaths[path_index, :, 0])
plt.xlabel('Time')
plt.ylabel('Stock price')
plt.title(f"The chosen sample path from our model.")
plt.show()
# plt.savefig("./plots/Single path Chosen.png")
```



The sample path chosen exhibits relatively high volatility, making it an interesting choice to observe The model's behaviour on. Peaking after 1 year (in model time), the stock crashes, recovers for one tick, and crashes further. The call option hedged ends out of the money.

5.4.2 Number of trades

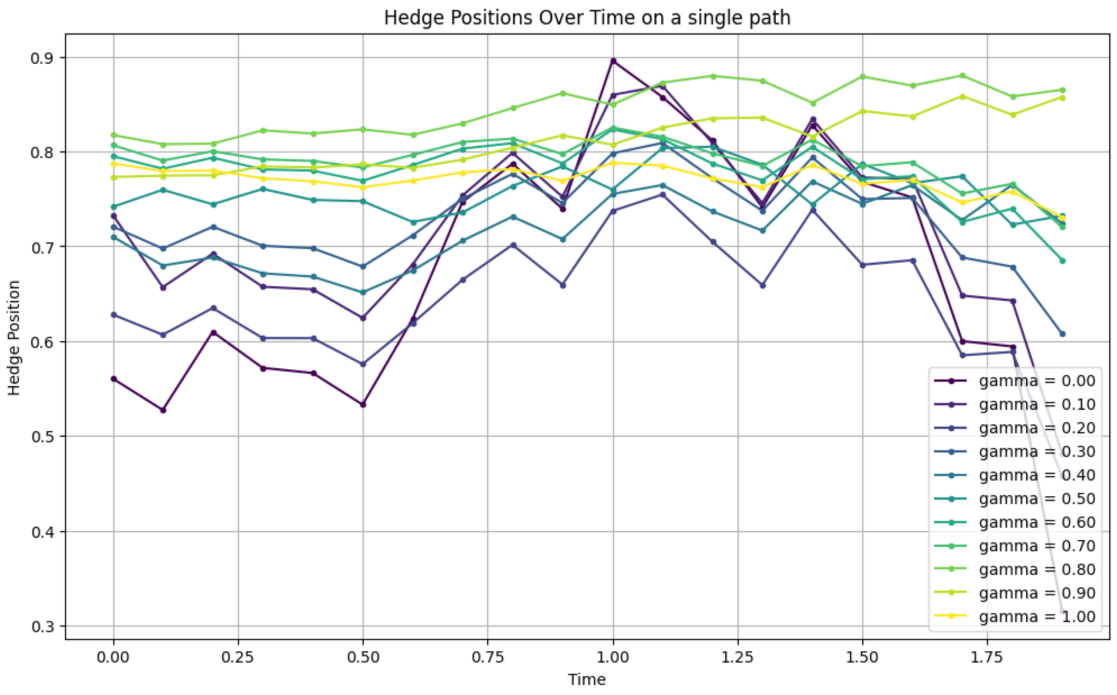
[11]:
```
trade_count = {}
trade_threshold = 1e-2 # only consider as trade if change is over a threshold

for gamma, H_positions in hedge_positions.items():
    # Calculate changes in hedge position
    H_all = np.insert(H_positions, 0, 0.0) # add zeroes as initial values
    dH = np.diff(H_all)
    trade_count[gamma] = np.sum(np.abs(dH) > trade_threshold)

plt.figure(figsize=(8, 5))
plt.plot(list(trade_count.keys()), list(trade_count.values()), marker='o',␣
 ↪linestyle='-', color='purple')
plt.xlabel('Transaction cost')
plt.ylabel('Number of Trades')
plt.title(f'Trading frequencies along single path')
plt.grid(True)
plt.tight_layout()
plt.show()
# plt.savefig("./plots/Single path Trading frequencies.png")
```



Here we counted each trade over 0.01 as a single trade As expected and much like in the aggregate case the trading frequency declines heavily, decreasing almost 100% in reaction to 100% trading costs.

5.4.3 Hedging positions over time

[12]:
```
colours = plt.cm.viridis(np.linspace(0, 1, len(gammas)))
plt.figure(figsize=(12, 7))
for i, gamma_plot, in enumerate(gammas):
    plt.plot(TimePoints[:-1], hedge_positions[gamma_plot], marker = '.',␣
 ↪linestyle = '-',
             label=f'gamma = {gamma_plot:.2f}',
             color = colours[i])

plt.xlabel('Time')
plt.ylabel('Hedge Position')
plt.title('Hedge Positions Over Time on a single path')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
# plt.savefig("./plots/Single path Hedge Positions.png")
```



Overall the most straightforward observation here is that for lower transaction costs, the algorithms followed the stock path movements more. As discussed previously, the difference in trading frequencies and volumes is quite evident on this chart, With hedging positions changing relatively little for high transaction costs and vice versa. Something interesting to notice is the high initial position in the high transaction cost neural networks. This aligns with the earlier observation of the almost buy-and-hold strategy for algorithms trained on high transaction costs. We also observe that the two worst models in the aggregate, the ones for 80% resp. 90% transaction costs hold the highest amounts in the underlying throughout the observation period, even when other algorithms decrease their positions