Deep Hedging

Alexis D. Plascencia

Physics Department and Center for Education and Research in Cosmology and Astrophysics (CERCA), Case Western Reserve University, Cleveland, OH 44106, USA

E-mail: aplascenciac@gmail.com

Abstract: In this report we overview the idea of Deep Hedging, which uses deep reinforcement learning to hedge a portfolio of derivatives. In contrast to the tradditional approach that uses the *greeks*, Deep Hedging works for general market conditions. We discuss the `Python` implementation and discuss some of the results.

Contents

1 Introduction

Deep Hedging was proposed in Ref. [1] by Hans Bühler, Lukas Gonon, Josef Teichmann and Ben Wood. In this approach, a neural network architecture learns how to hedge a portfolio of derivatives using historical (simulated) data without computing the *greeks*. Consequently, it can be applied for general market conditions. This report is based on lectures given by Josef Teichmann.

Suppose we have a portfolio of derivatives which represent our liabilities. In this work we consider a call option whose payoff depends on the underlying stock price so we have $f(S)$. The aim is to construct a portfolio in which this derivative is hedged.

Therefore, we want the value of the portfolio to vanish and the aim is to solve the following optimization problem

$$\min_{\delta} \mathbf{E}[(f(S_T) - p_0 - (\delta \cdot S)_T)^2] \tag{1.1}$$

where p_0 is the initial price of the derivative and

$$(\delta \cdot S)_T \equiv \sum_{k=0}^{n-1} \delta_k (S_{k+1} - S_k), \tag{1.2}$$

so δ_k corresponds to the number of shares held at time k. In other words, the agent is minimizing her loss at maturity. Transactions costs can be easily implemented by adding a new term to Eq. (1.1).

In this report we discuss a `Python` implementation of deep hedging for a call option. In Section 2 we go over the parameters of the model, in Section 3 we discuss the architecture of the Neural Network and the training. Finally, we present results and conclusions in Section 4.

2 Parameters

The stock prices are simulated using the Black-Scholes model with no drift. The parameters are set to:

```
N = 20 # Time steps
S0 = 1 # Initial stock price
strike = 1 # Strike for the call option
T = 1.0 # Maturity
sigma = 0.2 # Volatility
```

The initial value for the call option price is calculated using the Black-Scholes solution.

```
# Initial price for the call option:
priceBS = BS(S0,strike,T,sigma)
```

Taking zero dividend rate, the Black-Scholes solution is given by

$$C(S,t,T) = SN(d_1) - Ke^{-r(T-t)}N(d_2), \tag{2.1}$$

where

$$d_1 = \frac{\ln(S/K) + \sigma^2(T-t)/2}{\sigma\sqrt{(T-t)}}, \tag{2.2}$$

$$d_2 = d_1 - \sigma\sqrt{(T-t)}. \tag{2.3}$$

The variable `Ktrain` determines the number of generated paths on which the model will be trained.

```
# Number of trainable paths. This number can be 1e6
Ktrain = 10**5
```

3 Training

After we have generated all the `Ktrain` paths, we use them as input to train the model. The inputs for the training are:

1. The initial stock price S_0
2. The initial hedging being 0
3. The logarithmic increments of the stock price

We use the Adam optimizer, which is an adapted version of the stochastic gradient descent, to minimize the loss function

```
model_hedge.compile(optimizer='adam',loss='mean_squared_error')
```

where the loss function is given by

$$\mathcal{L} = [f(S_T) - p_0 - (\delta \cdot S)_T]^2 \,. \tag{3.1}$$

The number of hidden layers is $d \times N$. The strategy at the time step j will be calculated by the g_j Neural Network. The inputs are dynamical since at each time step j we use all the increments up to $j-1$. First, we initialize the Neural Network in an abstract way, and then we call the model with the actual data.

```
# Constructing the model and implementing the loss function
# Inputs contain: 1) The initial stock price S0
# 2) The initial hedging being 0
# 3) The increments of the log price

price = Input(shape=(m,))
hedge = Input(shape=(m,))

inputs = [price]+[hedge]

# Strategy at j is the hedging strategy at j, i.e. the Neural Network g_j
for j in range(N):
    strategy = price
    for k in range(d):
        strategy = layers[k+(j)*d](strategy)

    # Update the log price of the stock
    incr = Input(shape=(m,))
    logprice = Lambda(lambda x : K.log(x))(price)
    logprice = Add()([logprice, incr])

    # Update the price at time j+1
    pricenew = Lambda(lambda x : K.exp(x))(logprice)

    #Calculate the price increment
    priceincr = Subtract()([pricenew, price])

    # Update the value of the hedge
    hedgenew = Multiply()([strategy, priceincr])

    # This is only used for m > 1:
    #mult = Lambda(lambda x : K.sum(x,axis=1))(mult)

    # building up the discretized stochastic integral
    hedge = Add()([hedge, hedgenew])
    inputs = inputs + [incr]
    price = pricenew
```

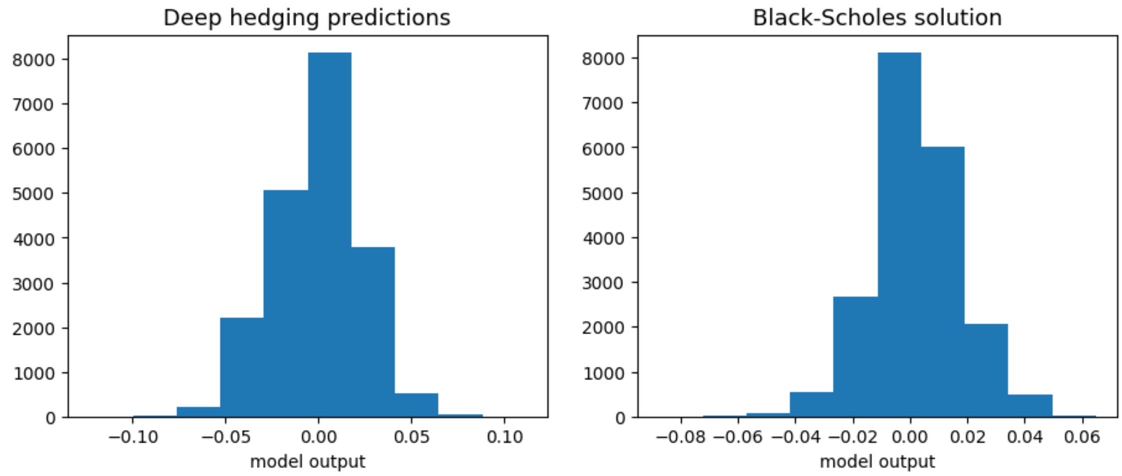

Figure 1: In the left (right) panel we show the predictions for the deep hedging model (Black-Scholes model). The results concentrate around zero as expected.

```
# Payoff of the call option
payoff = Lambda(lambda x : 0.5*(K.abs(x-strike) + (x-strike))
         - priceBS)(price)

# The output will be payoff-priceBS-hedge
# Loss function L = (payoff-priceBS-hedge)^2
outputs = Subtract()([payoff,hedge])

inputs = inputs
outputs = outputs

model_hedge = Model(inputs=inputs, outputs=outputs)
```

We simulate the data of the stock price paths that will be used for the training:

```
# Initial stock price
initialprice = S0

xtrain = ([initialprice*np.ones((Ktrain,m))] +
          [np.zeros((Ktrain,m))] +
          [np.random.normal(-(sigma)**2*T/(2*N),
          sigma*np.sqrt(T)/np.sqrt(N),(Ktrain,m)) for i in range(N)])

ytrain = np.zeros((Ktrain,1))
```

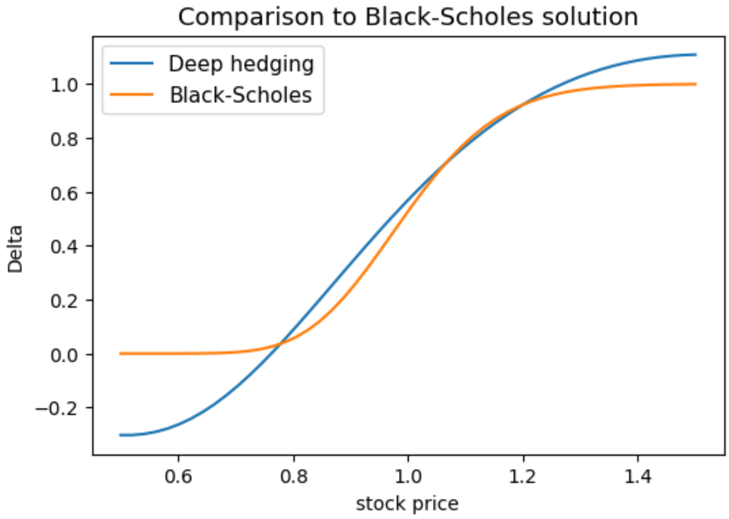

Figure 2: Comparison of the delta strategy as a function of the stock price. The blue (orange) line corresponds to the deep hedging (Black-Scholes model).

4 Results

On the left panel in Fig. 1 we show the predictions for the deep hedging model. The results are close to zero with an average value of 1.1×10^{-4} for the loss. We compare this with the analytic Black-Scholes solution, these results are shown on the right panel.

In Fig. 2 we show the Delta as a function of the stock price for a fixed time. The two lines show a very good overlap, except on the regions in the corners of the plot, since there was not enough data on these regions to generate good predictions.

References

[1] H. Buehler, L. Gonon, J. Teichmann and B. Wood, *Deep hedging*, *Quantitative Finance* **19** (2019) 1271–1291, [https://doi.org/10.1080/14697688.2019.1571683].