Neural Networks For Arbitrage Free Pricing

MATH 391 Summer Research Project

Shiyao Gu, Menglei Zhang, Qi Huang, Sizhe Jiang, Jierui Li, Langming Liang

Project Supervisor: Boutaib Youness, Samira Amiriyan
Module Supervisor: Corina Constantinescu

University of Liverpool

July, 2024

Outline

Background to The Study(MengLei Zhang)

What is option

The investor needs 5 tonnes of oil in October and expects the price to be higher than the current price in 3 months, so by purchasing the contract now at 100, the investor has the right to buy the oil at the current price after 3 months.

- How to price options?
- What is a good pricing?

Basic Definitions of Financial Markets(MengLei Zhang)

What is No-Arbitrage

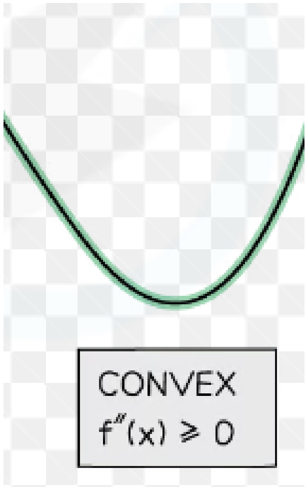
There is no way to make a profit in the market through risk-free

MARKET A
MARKET B
ARBITRAGE
BUY
SELL
INVESTOR

No-Arbitrage(MengLei Zhang)

Basic conditions that the no-arbitrage principle needs to fulfil for mathematical models

$$f \geq 0, \frac{\partial f}{\partial K} \leq 0, \frac{\partial f}{\partial T} \geq 0, \frac{\partial^2 f}{\partial K^2} \geq 0 \tag{1}$$

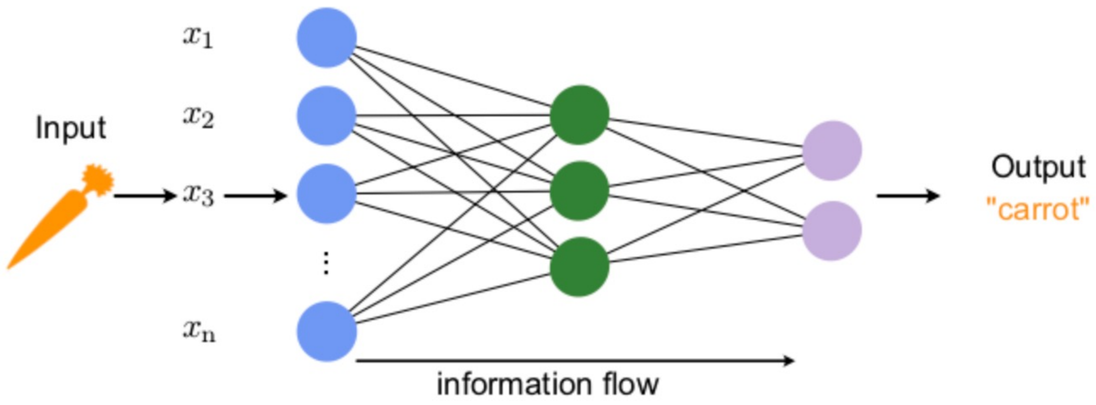


Note: In our subsequent model, K is taken to be $-K$, so both input parameters are increasing.

Basic Concepts(Qi Huang)

Definition

A neural network is composed of multiple units called neurons or nodes, which are typically distributed across different layers. These layers mainly include the input layer, hidden layers, and the output layer.



- The input data "carrot" is processed by the neural network, which is then able to recognize and output the label "carrot".

Training Loop(Qi Huang)

1. **Forward Propagation**: Input data → Predicted output
2. **Calculate Loss**: Predicted output → Error
3. **Back Propagation**: Error → Gradient calculation
4. **Update Weights**: Gradient → Adjust weights and biases

Forward Propagation

Output of layer = Activation Function(Weighted Sum of Inputs + Bias)

Back Propagation

Weight Update = Weight - $\eta \times \frac{\partial \text{Loss}}{\partial \text{Weight}}$

- **Forward Propagation**: Calculate and generate the output of the neural network.
- **Back Propagation**: Calculate the error and adjust the weights and biases of the network to reduce the error.

Universal Approximation Theorem(Qi Huang)

A class of functions $\hat{\mathcal{F}}$ from $\mathbb{R}^n$ to $\mathbb{R}$ is a **universal approximator** for a class of functions $\mathcal{F}$ from $\mathbb{R}^n$ to $\mathbb{R}$ if for any $f \in \mathcal{F}$, any compact domain $D \subset \mathbb{R}^n$, and any positive ϵ, one can find a $\hat{f} \in \hat{\mathcal{F}}$ with

$$\sup_{x \in D} |f(x) - \hat{f}(x)| \leq \epsilon.$$

It has already been shown that the class of artificial neural networks with one hidden layer:

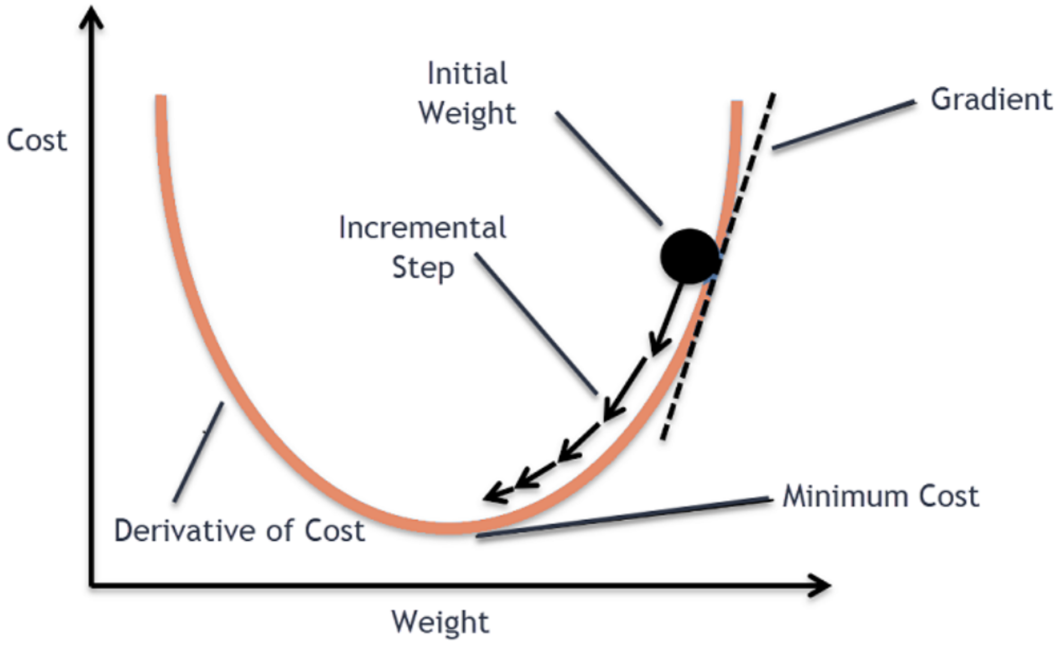
$$\hat{\mathcal{N}} = \left\{ f(x) = w_0 + \sum_{i=1}^{H} w_i \cdot h \left(b_i + \sum_{j} v_{ij} x_j \right) \right\},$$

- As long as there is a hidden layer with nonlinear activation functions, a feedforward neural network can approximate any continuous function from a finite-dimensional input space to real numbers.

Optimisation Strategy(LangMing Liang)

Definition

Gradient descent is an optimization algorithm which trains machine learning models by minimizing errors between predicted and actual results.



Optimisation Algorithms(LangMing Liang)

Main idea of gradient descent

Repeat $\omega := \omega - \alpha\frac{dJ(\omega,b)}{d\omega}$ and $b := b - \alpha\frac{dJ(\omega,b)}{db}$ then update ω and b.

- α: *learning rate which determines the size of the steps taken during each iteration of gradient descent.*
- $J(\omega, b) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$: *cost function (MSE)*

Other optimisation algorithms

- ***Stochastic Gradient Descent (SGD)***: SGD updates the parameters using one randomly chosen training example at a time.
- ***Adam(Adaptive Moment Estimation)*** : An extension to stochastic gradient descent, computing adaptive learning rates which is adjusted based on the estimates of the first and second moments of the gradients for each parameter. Adam also considers the exponentially weighted average of past gradients along with the current gradient which helps accelerate convergence.

Model Introduction(LangMing Liang)

Model Contribution

Our model dedicated to fit an actual funtion which can help us predict the option prices using the data from the real option market on the based of universe approximate theorem.

Constructing a basic frame of neural network

```
ANN.add(Dense(10,input_dim = 4, activation ='relu'))
ANN.add(Dense(10, activation = 'relu'))
ANN.add(Dense(10, activation = 'relu'))
ANN.add(Dense(10, activation = 'relu'))
ANN.add(Dense(1))
ANN.compile(loss = 'mean_squared_error', optimizer='adam')
ANN.fit(x_train, y_train)
y_pred = ANN.predict(x_test)
```

Forward and Backward Propagation(JieRui Li)

The role of forward and backward propagation in neural network training and how they are implemented in the Keras fit method.

- **How to call a method:** ANN.fit(X, Y, epochs, batch_size).
- **Epoch:** One complete traversal of the entire training dataset.
- **Batch:** Subset/Proportion/Part of the training dataset.

Forward Propagation

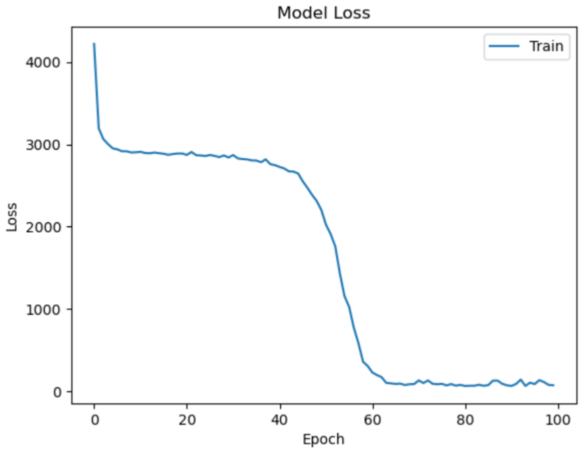
- At the beginning of each batch, the fit method inputs them into the first layer.
- The data passes through each layer until the output layer.
- Generate option predicted values.

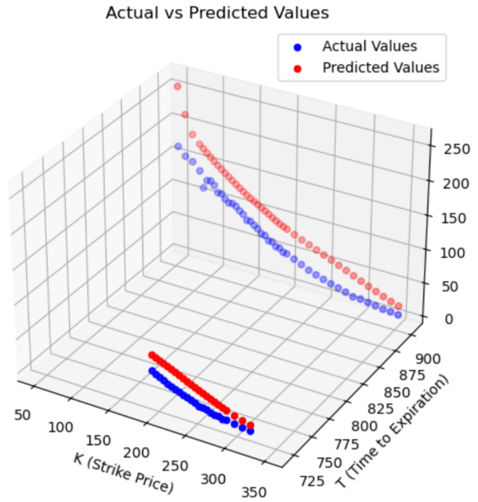
Backward Propagation

- Calculate the value of the loss function .
- The "fit" method is used to compute the gradients of the loss with respect to each parameter(chain rule).
- Update the weights by Adam optimizer.

Data Generation and Price Forecasting(JieRui Li)

- **Data Import:** Using pandas. Filtered the data for specific tickers(Tesla and Apple) and options(Put and call).
- **Data Splitting:** Spliting the data into training and testing sets.
- **Model Training:** Training data, observing changes, adjusting the parameters to make the model's loss function as small as possible.
- **Model Testing:** Testing data, calculating the size of MSE.
- **Visualisation:** A trend of the loss function and 3D Comparison for predicted and actual value.





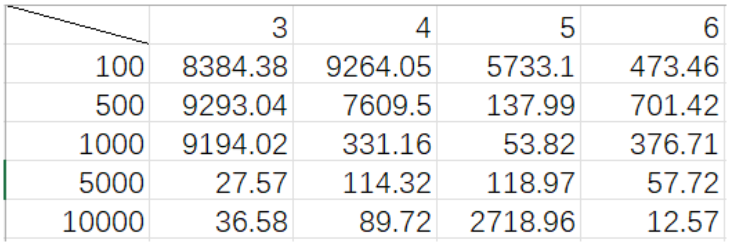
Neural Network Structure and Model Fitting(JieRui Li)

1.Two possible impact points:

- **Layers:** Increasing the number of hidden layers allows the model to capture more complex patterns.
- **Iteration:** More iterations allow the model more opportunities to learn from the data.Too many iterations may lead to overfitting.

2.Test Result:

- The data in this table is the size of the loss function on the test data after training with different parameters.



	3	4	5	6
100	8384.38	9264.05	5733.1	473.46
500	9293.04	7609.5	137.99	701.42
1000	9194.02	331.16	53.82	376.71
5000	27.57	114.32	118.97	57.72
10000	36.58	89.72	2718.96	12.57

3.Analysis:

- In general, the more layers and training times, the better the result of the test.
- Some problem maybe caused by overfitting.

The No-Arbitrage Pricing Model(ShiYao Gu)

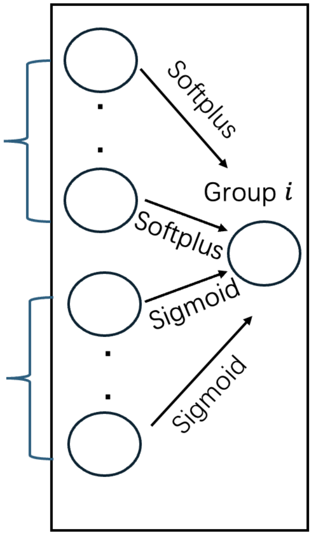
Models

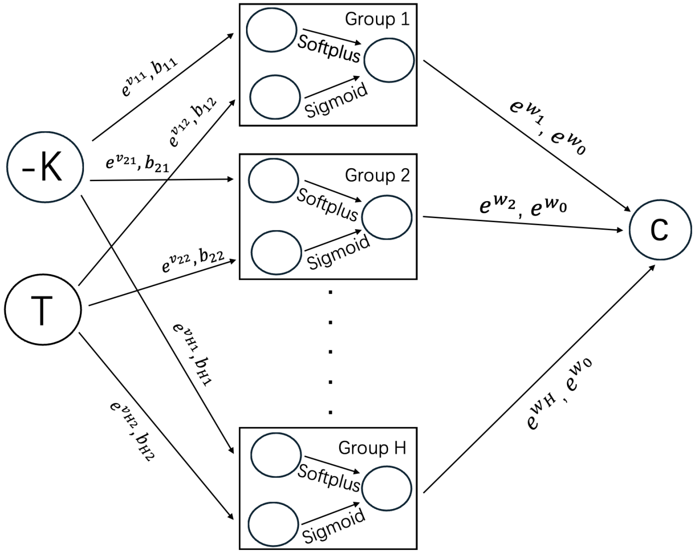
Primitive Model: $\hat{\mathcal{N}} = \left\{ f(x) = w_0 + \sum_{i=1}^{H} w_i \cdot h\left(b_i + \sum_j v_{ij} x_j\right) \right\}$

Modified Model: $c, n\hat{\mathcal{N}}_{++} = \{ f(x) = e^{w_0} + \sum_{i=1}^{H} e^{w_i} \left(\prod_{j=1}^{c} \zeta\left(b_{ij} + e^{v_{ij}} x_j\right)\right) \left(\prod_{j=c+1}^{n} h\left(b_{ij} + e^{v_{ij}} x_j\right)\right)$

For each parameter here, the model is both **monotonically increasing** and **convex** with respect to them.

For each parameter here, the model is **monotonically increasing** with respect to them.





Code Implementation(ShiYao Gu)

Example code:

```
# Step01: Define the forward propagation
def forward_propagation(V, b, w, w0, x):
  return result
# Step02: Import dataset
def read_option_data(file_path, ticker,
     current_date):
  return K_T_matrix.T, c_matrix.T
# Step03: Randomly splits training and
     testing datasets
def split_train_test(matrix1, matrix2, ratio
     ):
  return train_matrix1, test_matrix1,
       train_matrix2, test_matrix2
# Step04: Optimize the model using PyTorch
H = 10
optimizer = optim.Adam([V, b, w, w0], lr
     =0.01)
for epoch in range(num_epochs):
    optimizer.zero_grad()
    y_pred = forward_propagation(V, b, w, w0
         , train_x_apple)
    loss = mean_squared_error(y_pred,
         train_y_apple)
    loss.backward()
    optimizer.step()
# Step05: Make prediction model
def get_predict_function(V, b, w, w0):
  return predict
# Complete code can be found on GitHub
```

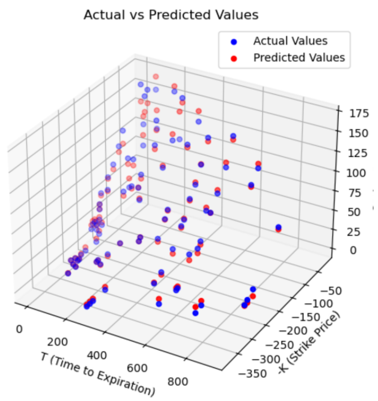
Questions:

1)Why is this an appropriate model?

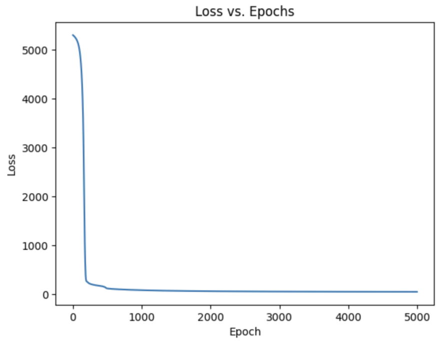
2)How to optimize the parameters?

Analysis of the Final Results of the Model(ShiYao Gu)

The performance of our model:



Convergence of model training:

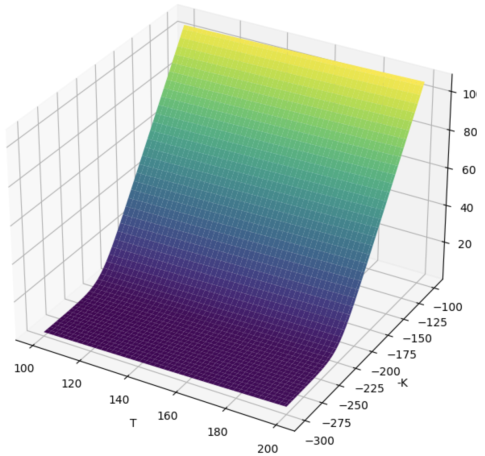


Relationship between loss function and number of neurons:

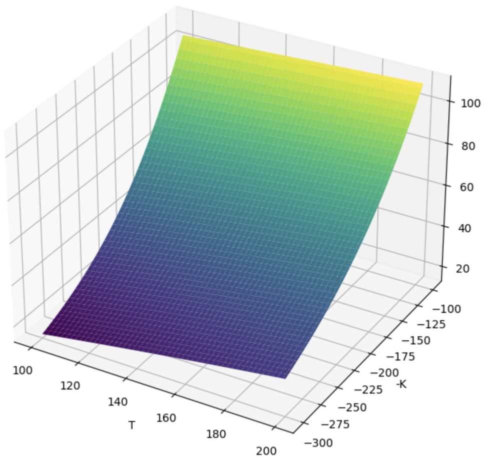
H	Loss
1	64.7333
2	62.379
3	45.1903
4	45.0423
5	44.711
10	44.7529
100	44.8589
1000	41.4791

Analysis of the Final Results of the Model(ShiYao Gu)

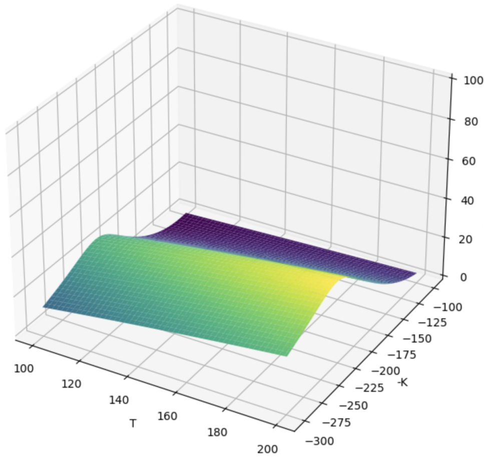
Our ultimate goal is to find a relationship between K, T, and c, that is $\hat{c} = f_{nn}(K, T)$. Then, we will compare it with the Black-Scholes Formula (setting all variables except K and T as constants). The following diagrams are: the relationship shown by our model, the relationship presented by the Black-Scholes Formula, and the difference between the two.



Our Model



Black-Sholes Formula



Black-Sholes Formula - Our Model

Conclusion(SiZhe Jiang)

The purpose of this project is to investigate the pricing strategy of European call options by using neural networks. We have demonstrated the non-parametric approach of pricing call options as opposed to traditional analytical approaches.

Model Implementation and Results

- The Training loop of forward and backward propagation.
- Optimisation Techniques. (Gradient decent, The Adam optimiser)
- The performance of the convergence during training.
- The performance of test data.

Conclusion(SiZhe Jiang)

However, the application of neural network in this project is of course facing its challenges, such as

The potential negative impact of performance caused by lack of data resources.

- Poor performance for small batch and epoch size.
- Bias and under-fitting.

The limitation of the neural network structure.

- Limited number of the hidden layers.
- Only two parameters are considered in our model.
- Computational intensity

Conclusion(SiZhe Jiang)

The biggest challenge

- NO MONEY!!

Conclusion(SiZhe Jiang)

Finally we are looking forward to see that different Neural network is also capable in option pricing and the implementation of other type of neural network.

Neural network in American call option pricing.

- Generative Adversarial Networks (GANs)
- Convolutional Neural network (CNNs)

Thank you very much

Any questions?