Evaluating Deep Learning Models On Monte Carlo-Simulated Basket Option Pricing Under Uniform Setting

STAT5293 Final Report

(Phyllis) Muqing Yang,
(Karry) Bingjiang Xia,
(Colin) Jianfeng Chen

Background

In today's financial world, making smart decisions about investments is crucial, especially when it comes to complex areas like option pricing and futures trading. We need tools that can accurately predict market movements, and do it quickly and efficiently, especially Deep learing derivatives.

- **Motivation**

To see how well deep learning can forecast derivatives prices.

- **Model**

CNNs, LSTMs, DNNs and Transformers for comparing their accuracy, acc-speed trade-off and acc-cost trade-off.

- **Objective**

To seek for the optimal model for making predictions in the real-world trading

Introduction

- Inspired by '**Deeply Learing Derivatives**'
published by Ryan Ferguson & Andrew Green (2018)


arXiv > q-fin > arXiv:1809.02233

Quantitative Finance > Computational Finance

[Submitted on 6 Sep 2018 (v1), last revised 17 Oct 2018 (this version, v4)]

Deeply Learning Derivatives

Ryan Ferguson, Andrew Green

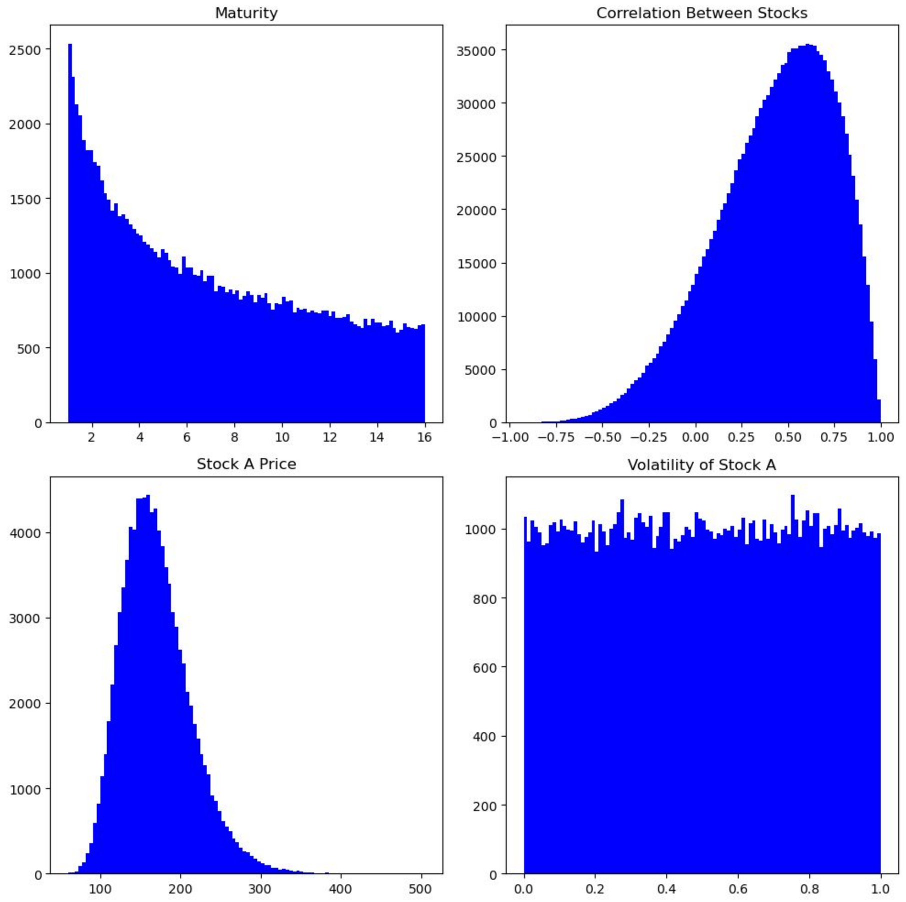
- Basket option defintion
  - A **European call option**, meaning it can only be exercised at expiration.
  - depends on the **difference** between the strike price and the price of the worst-performing stock in the basket.
  - If all stocks are above the strike price at expiration, the option expires worthless.
  - If at least one stock is below the strike price at expiration, the option's value is the difference between the strike price and the price of the worst-performing stock.

Data Generation

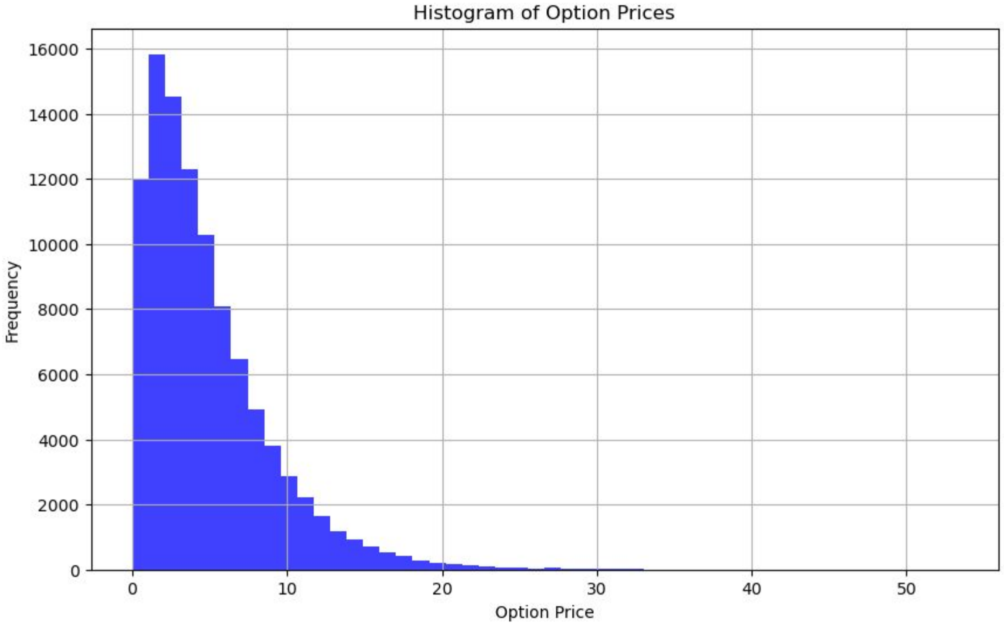
- Each basket of option: containing **6** stocks
- **Monte Carlo simulation:** V = max(0, min(Stock1, Stock2, Stock3, Stock4, Stock5, Stock6) – K)
  - Forward prices
  - Volatilities
  - Correlation matrices
  - Option maturity time
  - Strike price
- Captures real-world characteristics and complexities of financial markets.
- Sample size: **100,000** baskets option

Data Generation

Inputs	Distribution	Paramater	Values
Forward Price	Normal	Mean	0.5
		Standard Deviation	0.25
Volatility	Uniform	Range	[0,1]
Maturity Time	Uniform (squared)	Range	[1,4]
Correlation	Beta	Alpha	5
		Beta	2
Strike Price	Constant		100
Risk-free Rate	Constant		0.05



Data Generation



- European call option on a worst-of basket with six underlying stocks.
- 100,000 Monte Carlo simulation steps for precise option price generation.
- High-quality training and testing data or deep learning models.

- MinMaxScaler for data standardization
- 70% training data, 30% testing data.

Models Introduction

Model	Description
CNN	CNNs represent a specialized kind of ANN known for their local connectivity and shared weights architecture, frequently employed in the field of computer science for their robust capacity to distill information.
LSTM	LSTM networks, a specialized form of ANNs, are predominantly applied to time-series data for regression and classification tasks.
CNN/ LSTM	CNN-LSTM is a type of deep neural network that combines convolutional neural networks and long short-term memory networks
DNN	A DNN is a sophisticated iteration of an ANN, characterized by its deeper structure of hidden layers. This complexity enables a DNN to excel at tasks like regression and classification by capturing a more thorough and efficient representation of the input data.
Transformer	The Transformer, ibecome a mainstay in Computer Vision (CV) and Natural Language Processing (NLP), the Self-Attention mechanism of the Transformer is also well-suited for time series prediction tasks.

Implementation Details

- In order to keep uniform setting, we chose the following hyperparamters for each model
  - 1/3/5 layers/blocks
  - 128 neurons/filters per layer/block
  - Optimizer: Adam with learning rate 0.001
  - Epochs: 50
  - Batch size: 256
  - Activation function: relu for CNN and DNN, tanh for LSTM
  - Tensorflow, Keras
- Also we ensured that all models are trained and tested on same set training set and testing set.
- More details on implementation of CNN-LSTM and Transformer are shown in the report.

Difficulties Encountered in Hardware/Library

```
Epoch 40/50
274/274 [==============================] - 1s 4ms/step - loss: 2.2826e-04 - val_loss: 0.1901
Epoch 41/50
274/274 [==============================] - 1s 4ms/step - loss: 2.2669e-04 - val_loss: 0.1871
Epoch 42/50
274/274 [==============================] - 1s 4ms/step - loss: 2.2300e-04 - val_loss: 0.1884
Epoch 43/50
274/274 [==============================] - 1s 4ms/step - loss: 2.2034e-04 - val_loss: 0.1926
Epoch 44/50
274/274 [==============================] - 1s 4ms/step - loss: 2.1666e-04 - val_loss: 0.1898
Epoch 45/50
274/274 [==============================] - 1s 3ms/step - loss: 2.1306e-04 - val_loss: 0.1846
Epoch 46/50
274/274 [==============================] - 1s 3ms/step - loss: 2.1007e-04 - val_loss: 0.1909
Epoch 47/50
274/274 [==============================] - 1s 3ms/step - loss: 2.0184e-04 - val_loss: 0.1909
Epoch 48/50
274/274 [==============================] - 1s 4ms/step - loss: 2.0905e-04 - val_loss: 0.1895
Epoch 49/50
274/274 [==============================] - 1s 4ms/step - loss: 1.9805e-04 - val_loss: 0.1942
Epoch 50/50
274/274 [==============================] - 1s 4ms/step - loss: 1.9757e-04 - val_loss: 0.1863
```

```
Epoch 40/50
274/274 [==============================] - 1s 3ms/step - loss: 1.9628e-04 - val_loss: 3.1510e-04
Epoch 41/50
274/274 [==============================] - 1s 3ms/step - loss: 1.8687e-04 - val_loss: 3.7087e-04
Epoch 42/50
274/274 [==============================] - 1s 3ms/step - loss: 1.8675e-04 - val_loss: 3.0381e-04
Epoch 43/50
274/274 [==============================] - 1s 3ms/step - loss: 1.7744e-04 - val_loss: 3.0493e-04
Epoch 44/50
274/274 [==============================] - 1s 3ms/step - loss: 1.7821e-04 - val_loss: 2.9927e-04
Epoch 45/50
274/274 [==============================] - 1s 3ms/step - loss: 1.7385e-04 - val_loss: 2.9518e-04
Epoch 46/50
274/274 [==============================] - 1s 3ms/step - loss: 1.7370e-04 - val_loss: 3.0816e-04
Epoch 47/50
274/274 [==============================] - 1s 3ms/step - loss: 1.6669e-04 - val_loss: 2.8956e-04
Epoch 48/50
274/274 [==============================] - 1s 3ms/step - loss: 1.6942e-04 - val_loss: 3.7112e-04
Epoch 49/50
274/274 [==============================] - 1s 3ms/step - loss: 1.6422e-04 - val_loss: 2.9092e-04
Epoch 50/50
274/274 [==============================] - 1s 3ms/step - loss: 1.6881e-04 - val_loss: 2.8392e-04
```

- The test loss doesn’t decrease at all on my computer. Hardware? Different version of library?
- To fix this problem and ensure all calculations are done in uniform setting, the final results are produced on the same device.

Difficulties Encountered in Transformer

```
# train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=512,validation_data=(X_test, y_test))
```

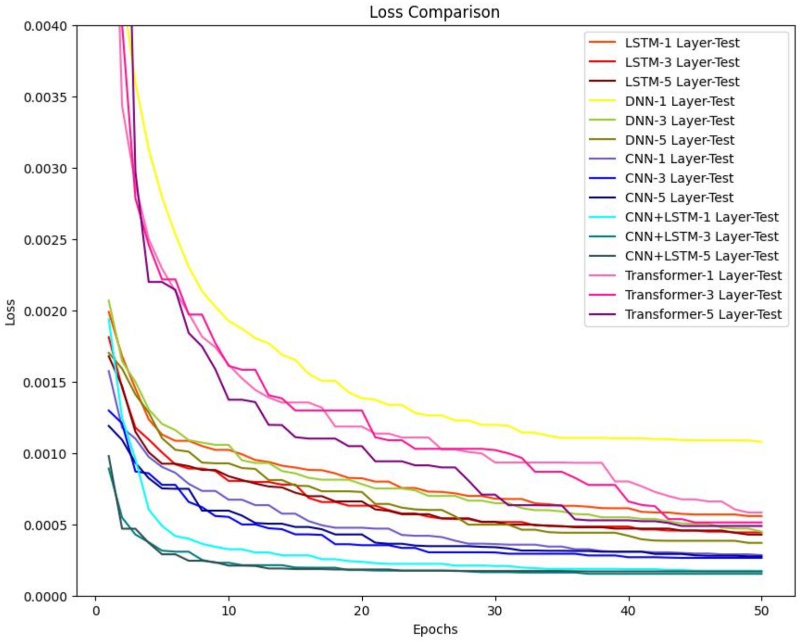
```
Epoch 1/50
---------------------------------------------------------------------------
ResourceExhaustedError                    Traceback (most recent call last)
<ipython-input-37-09c90f734c88> in <cell line: 6>()
      4
      5 # train the model
----> 6 history = model.fit(X_train, y_train, epochs=50, batch_size=512,validation_data=(X_test, y_test))

                                 1 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

ResourceExhaustedError: Graph execution error:
```

GPU RAM
14.8 / 16.0 GB

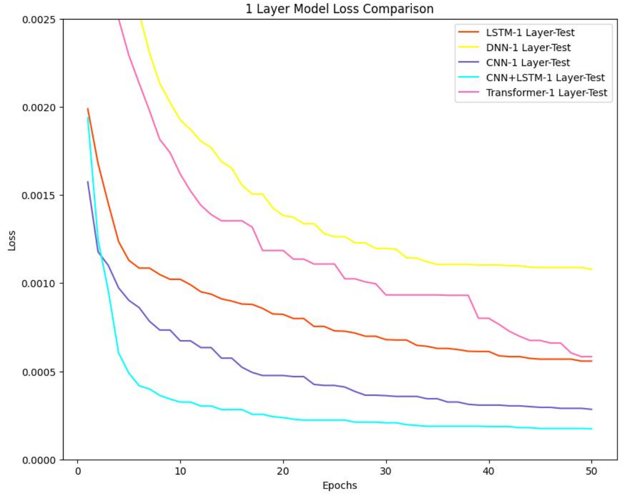
- Encountered GPU memory limitations on the available V100 GPUs with 16GB memory each when choosing 512 or 1024 as the batch size
- Run the model training on larger GPU
- Finally choose 256 as our batch size

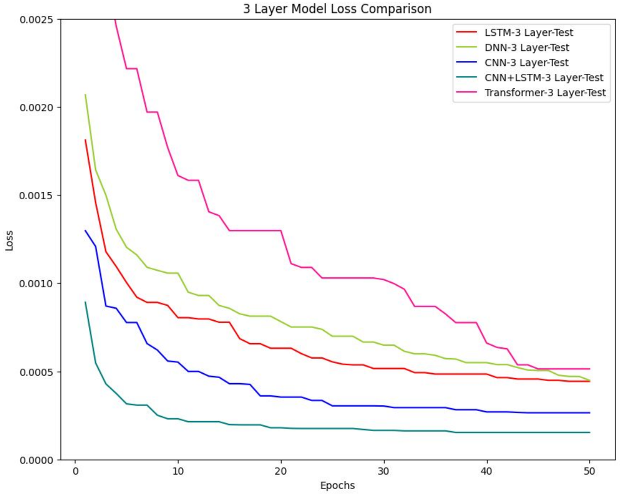
Results - Accuracy

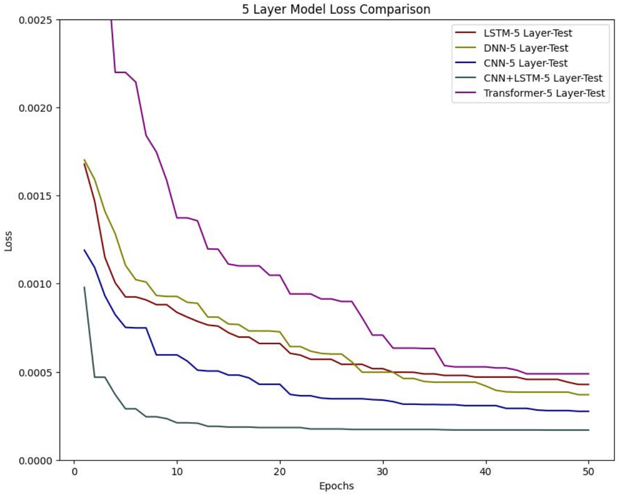
- Most models seem to converge towards a certain loss values (0.0010-), although the rate and smoothness of convergence vary.
- Generally, models with less layers tend to start with a higher initial loss (Transformer except).
- Generally, models with more layers tend to have a smaller minimum loss.



Results - Accuracy







1 Layer:
CNN-LSTM>CNN>LSTM>T
rans-former>DNN

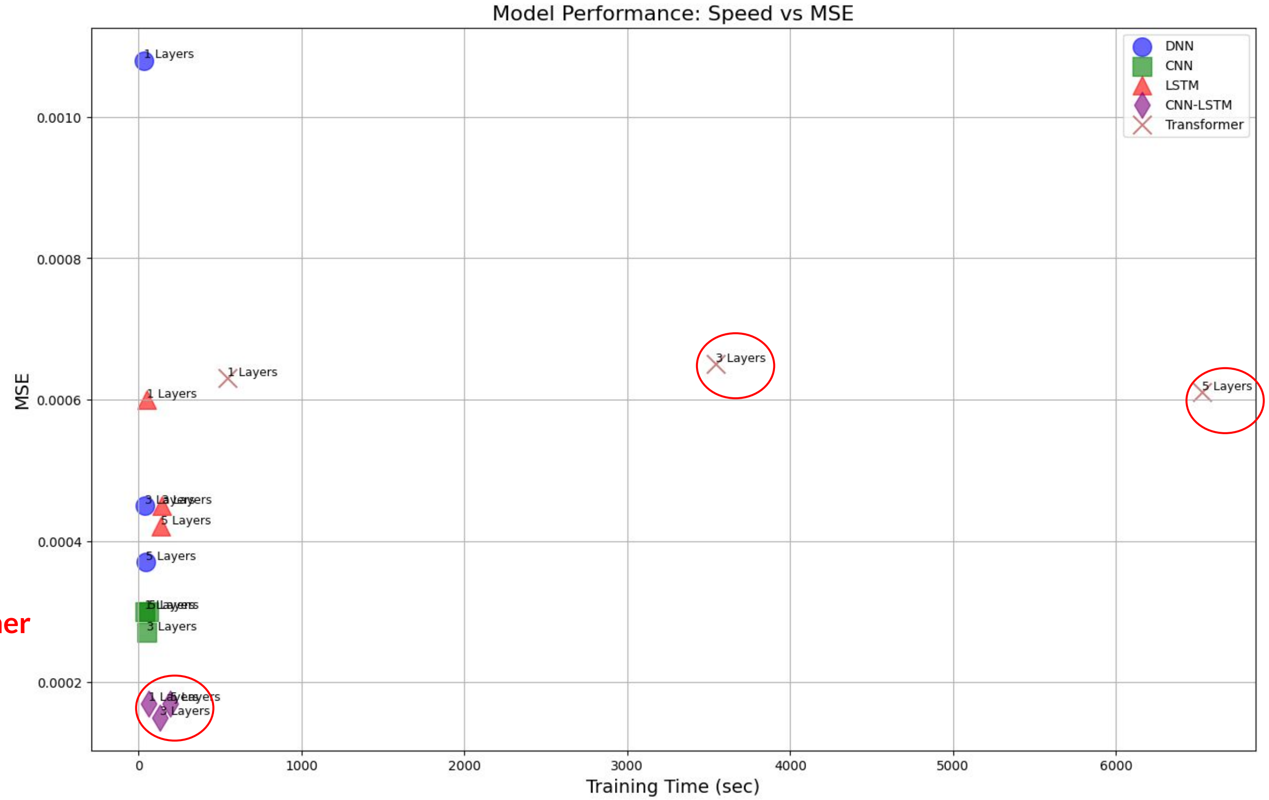
3 Layers:
CNN-LSTM>CNN>LSTM>
DNN>Transformer

5 Layers:
CNN-LSTM>CNN>LSTM>T
rans-former>DNN

Results - Speed VS MSE

- 1 layer:
  - All models perform fair
- 3 layers:
  - Transformer worst
  - CNN-LSTM best
- 5 layers:
  - Transformer worst
  - CNN-LSTM best

CNN-LSTM > CNN > LSTM = DNN > Transformer



Results - Cost VS MSE

- 1 layer:
  - DNN worst
- 3 layers:
  - CNN best in trade-off
- 5 layers:
  - CNN best in trade-off

CNN > LSTM > CNN-LSTM = DNN > Transformer

Results - Summary

- **Accuracy**: The CNN-LSTM model consistently achieves the lowest MSE, indicating superior performance. CNN also performed well, highlighting the importance of spatial feature extraction.

- **Speed**: Accuracy will improve as training time increases. Among them, CNN-LSTM performs best in terms of speed-acc trade-off.

- **Cost**: The relationship between mse and the total number of parameters is complex. CNN-LSTM has the largest number of parameters and the highest accuracy. CNN performs best in terms of cost-acc trade-off.

Findings

- **Overall,**
    - DNN, LSTM, CNN, and CNN-LSTM are efficient. Considering both accuracy and speed, CNN-LSTM is our best model.
    - Transformer is super computationally expensive. As it also demonstrates the worst prediction performance, it is not necessary to implement Transformer on this task.
- **In terms of model complexity,**
    - CNN and CNN-LSTM archives the best performance when 3 layers are implemented. This might suggest overfitting issue when more layers are implemented and CNN and CNN-LSTM doesn't require the same level of complexity to achieve better performance compared to other models.
    - The performance DNN, LSTM, and Transformer always gets better when increasing number of layers, so more hyperparameter tuning is suggested.

Future Research Suggestion

- **Test on real market data:**
  - Simulated data has provided a good baseline, but real-world conditions introduce complexities such as market volatility and external economic factors
  - to better understand models' robustness in dynamic environments
- **Try other "fancier" models:**
  - For example, Exponential Lévy Neural Network, which blends ANN with the exponential Lévy process
  - Empirically test to evaluate their efficiency and accuracy in handling issues like overfitting in deep learning and parameter estimation in traditional models.

Thank you for your time and attention 🙂