blsc_architecture_put

March 20, 2025

[255]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU
from keras.optimizers import Adam
from scipy.stats import norm
```

[256]:

```
def blsc(S, K, T, r, sigma, call=True):
    """
    Black-Scholes option pricing model.
    S: Current stock price
    K: Option strike price
    T: Time to expiration (in years)
    r: Risk-free interest rate
    sigma: Volatility of the underlying asset
    call: True for call option, False for put option
    """
    d1 = (np.log(S / K) + ((r + (0.5 * (sigma ** 2))) * T)) / (sigma * np.
 ↪sqrt(T))
    d2 = d1 - (sigma * np.sqrt(T))

    if call:
        return ((S * norm.cdf(d1)) - (K * np.exp(-r * T) * norm.cdf(d2)))
    else:
        return ((K * np.exp(-r * T) * norm.cdf(-d2)) - (S * norm.cdf(-d1)))
```

[257]:

```
def error(x,y):
        error = np.sum(np.abs(np.array(x) - np.array(y)))*100/np.sum(np.
 ↪array(y))
        return error
```

[258]:

```
df = pd.read_csv('nvda_options_data.csv')
```

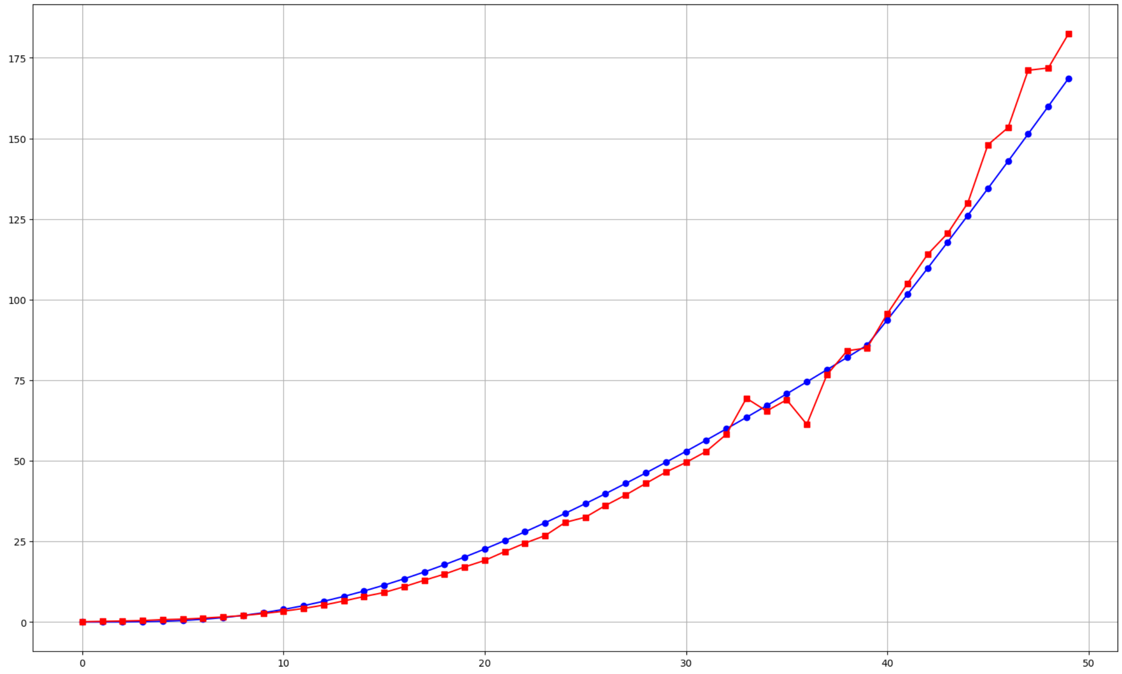
[259]:

```
def algo():
    y_blsc = blsc(df['Stock_Price'], df['strike'], df['Time_to_Expire'],␣
 ↪df['Risk_Free_Rate'], df['IV'], call=False)
    loss = error(y_blsc,df['PutPrice'])
    print('Loss:',loss)
    plt.figure(figsize=(20, 12))
    plt.plot(y_blsc, label='Predicted', marker='o', linestyle='-', color='blue')
    plt.plot(df['PutPrice'], label='Actual', marker='s', linestyle='-',␣
 ↪color='red')
    plt.grid()
    plt.show()
    return y_blsc
```

[260]:

```
blsc_price = algo()
```

```
Loss: 7.159392691777871
```



[261]:

```
df['blsc_price'] = blsc_price
```

[262]:

```
#Put
X = df[['strike', 'IV','Stock_Price',␣
 ↪'Time_to_Expire','Risk_Free_Rate','blsc_price']]
y = df[['PutPrice']]

X.head()
```

[262]:

```
   strike        IV  Stock_Price  Time_to_Expire  Risk_Free_Rate  blsc_price
0     5.0  0.566315   117.889999        1.820671          0.0422    0.000041
1    10.0  0.566315   117.889999        1.820671          0.0422    0.002695
2    15.0  0.566315   117.889999        1.820671          0.0422    0.022218
3    20.0  0.566315   117.889999        1.820671          0.0422    0.085540
4    25.0  0.566315   117.889999        1.820671          0.0422    0.224070
```

[263]:

```python
print(f"X shape: {X.shape}, Y shape: {y.shape}")
```

```
X shape: (50, 6), Y shape: (50, 1)
```

[264]:

```python
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=500)
```

[265]:

```python
X_train, y_train = tf.convert_to_tensor(X_train, dtype=tf.float32), tf.
 ↪convert_to_tensor(y_train, dtype=tf.float32)
X_test, y_test = tf.convert_to_tensor(X_test, dtype=tf.float32), tf.
 ↪convert_to_tensor(y_test, dtype=tf.float32)
```

[266]:

```python
# Hyperparams
n_units = X_train.shape[1]
n1_units = 400
layers = 4
```

[267]:

```python
model = Sequential()
model.add(Dense(n_units, input_dim=X_train.shape[1]))
model.add(LeakyReLU())
for _ in range(layers - 1):
    model.add(Dense(n1_units))
    model.add(LeakyReLU())

model.add(Dense(1, activation='relu'))
```

```
/Users/aadityatrivedee/tf_lib/env/lib/python3.10/site-
packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[268]:

```python
model.summary()
```

```
Model: "sequential_11"


 Layer (type)                     Output Shape                  Param #

 dense_55 (Dense)                 (None, 6)                          42
 leaky_re_lu_44 (LeakyReLU)        (None, 6)                          0

 dense_56 (Dense)                  (None, 400)                    2,800

 leaky_re_lu_45 (LeakyReLU)        (None, 400)                        0

 dense_57 (Dense)                  (None, 400)                  160,400

 leaky_re_lu_46 (LeakyReLU)        (None, 400)                        0

 dense_58 (Dense)                  (None, 400)                  160,400

 leaky_re_lu_47 (LeakyReLU)        (None, 400)                        0

 dense_59 (Dense)                  (None, 1)                        401
```

```
 Total params: 324,043 (1.24 MB)

 Trainable params: 324,043 (1.24 MB)

 Non-trainable params: 0 (0.00 B)
```

[269]:

```
model.compile(loss='mae', optimizer=Adam(learning_rate=0.001))
```

[270]:

```
batch_size = 32
losses = model.fit(X_train, y_train, validation_data=(X_test,␣
 ↪y_test),batch_size=batch_size, epochs=30, verbose=1)
```

```
Epoch 1/30
2/2              1s 138ms/step - loss:
46.4444 - val_loss: 61.2511
Epoch 2/30
2/2              0s 34ms/step - loss:
44.6196 - val_loss: 60.8937
Epoch 3/30
2/2              0s 33ms/step - loss:
47.7373 - val_loss: 60.2524
Epoch 4/30
2/2              0s 32ms/step - loss:
46.5957 - val_loss: 59.2201
Epoch 5/30
2/2              0s 32ms/step - loss:
44.6221 - val_loss: 57.6975
Epoch 6/30
```

```
2/2             0s 33ms/step - loss:
42.9354 - val_loss: 55.4279
Epoch 7/30
2/2             0s 33ms/step - loss:
39.8995 - val_loss: 52.2467
Epoch 8/30
2/2             0s 33ms/step - loss:
40.1202 - val_loss: 47.9989
Epoch 9/30
2/2             0s 33ms/step - loss:
36.8835 - val_loss: 42.2895
Epoch 10/30
2/2             0s 32ms/step - loss:
34.6895 - val_loss: 35.0036
Epoch 11/30
2/2             0s 33ms/step - loss:
28.6890 - val_loss: 26.0684
Epoch 12/30
2/2             0s 32ms/step - loss:
20.7766 - val_loss: 14.3313
Epoch 13/30
2/2             0s 32ms/step - loss:
12.4458 - val_loss: 4.7504
Epoch 14/30
2/2             0s 32ms/step - loss:
4.3004 - val_loss: 17.4798
Epoch 15/30
2/2             0s 32ms/step - loss:
9.8014 - val_loss: 22.2880
Epoch 16/30
2/2             0s 32ms/step - loss:
12.6885 - val_loss: 18.6077
Epoch 17/30
2/2             0s 32ms/step - loss:
9.7197 - val_loss: 10.6945
Epoch 18/30
2/2             0s 33ms/step - loss:
5.9978 - val_loss: 3.5243
Epoch 19/30
2/2             0s 33ms/step - loss:
2.7925 - val_loss: 5.4614
Epoch 20/30
2/2             0s 33ms/step - loss:
5.3286 - val_loss: 5.0740
Epoch 21/30
2/2             0s 33ms/step - loss:
4.4222 - val_loss: 2.9755
Epoch 22/30
```

```
2/2              0s 33ms/step - loss:
1.7864 - val_loss: 6.9663
Epoch 23/30
2/2              0s 32ms/step - loss:
3.5932 - val_loss: 8.4820
Epoch 24/30
2/2              0s 32ms/step - loss:
4.6892 - val_loss: 5.2472
Epoch 25/30
2/2              0s 32ms/step - loss:
2.8974 - val_loss: 2.7220
Epoch 26/30
2/2              0s 32ms/step - loss:
1.8764 - val_loss: 3.1831
Epoch 27/30
2/2              0s 33ms/step - loss:
2.5887 - val_loss: 2.6065
Epoch 28/30
2/2              0s 33ms/step - loss:
1.7775 - val_loss: 3.8694
Epoch 29/30
2/2              0s 33ms/step - loss:
1.9904 - val_loss: 3.4445
Epoch 30/30
2/2              0s 32ms/step - loss:
1.6257 - val_loss: 2.9650
```

[271]:

```
model.evaluate(X_test[:3], y_test[:3],batch_size=batch_size)
```

```
1/1              0s 33ms/step - loss:
5.9434
```

[271]:

```
5.94343900680542
```

[272]:

```
model.predict(pd.DataFrame(X_test).iloc[0:10])
```

```
1/1              0s 41ms/step
```

[272]:

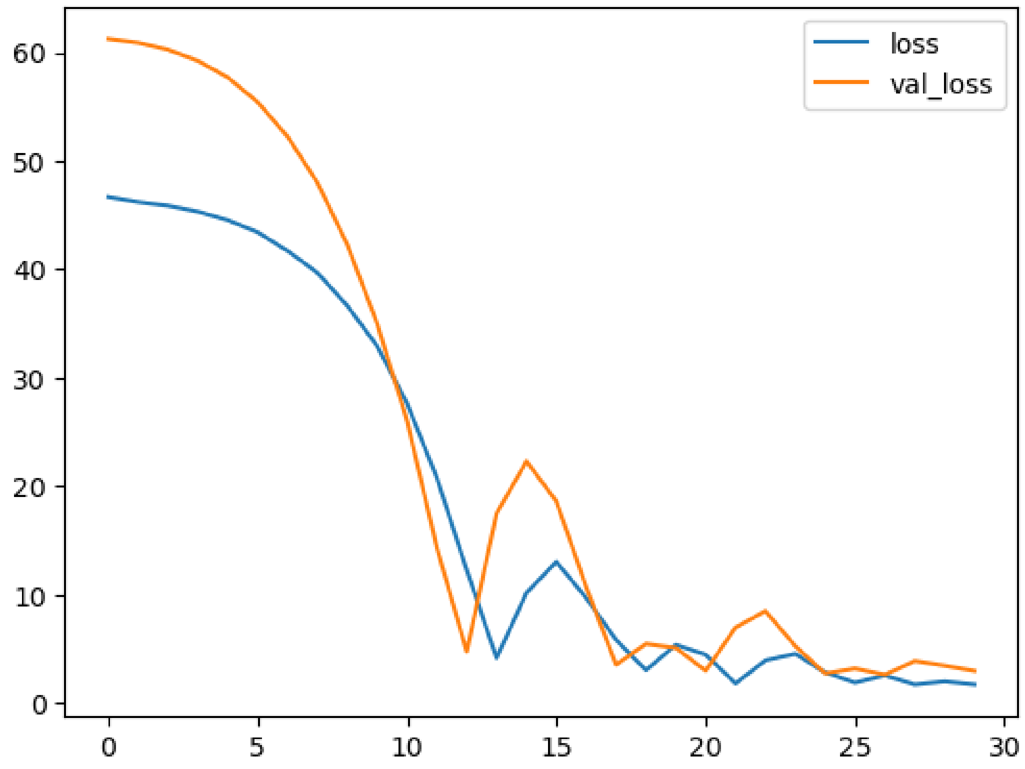
```
array([[ 74.973755],
       [ 29.853188],
       [123.67975 ],
       [ 24.647491],
       [141.83694 ],
       [132.72156 ],
       [ 96.90907 ],
       [  4.651142],
       [  0.      ],
       [  0.      ]], dtype=float32)
```

```
[273]: pd.DataFrame(y_test).iloc[0:10]
```

```
[273]:             0
        0   61.310001
        1   30.900000
        2  120.559998
        3   24.469999
        4  148.050003
        5  130.000000
        6   95.620003
        7    5.250000
        8    0.730000
        9    0.090000
```

```
[274]: loss_df = pd.DataFrame(losses.history)
       loss_df.loc[:,['loss','val_loss']].plot()
```

```
[274]: <Axes: >
```



```
[275]: def model_error(x,y):
           error = np.sum(np.abs(model.predict(pd.DataFrame(x)) - pd.
        ↪DataFrame(y)))*100/(np.sum(np.array(y)))
```

```
    return error
print('Mean Square Percentage Error in train:', model_error(X_train, y_train))
print('Mean Square Percentage Error in test:', model_error(X_test, y_test))
```

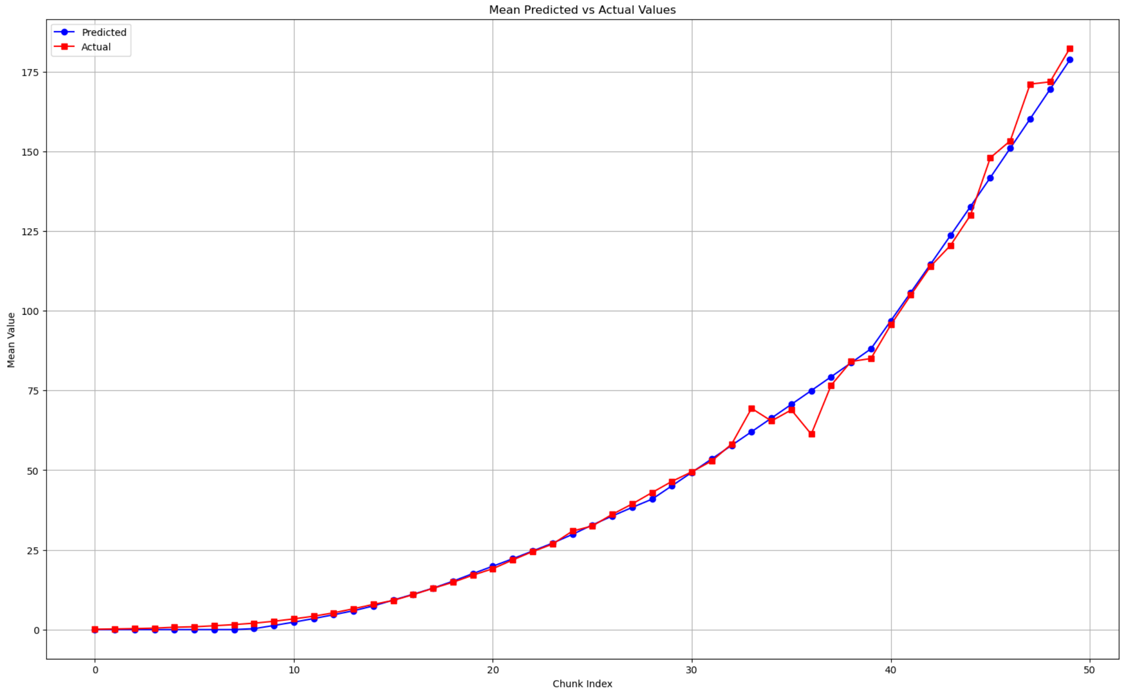
```
2/2               0s 28ms/step
Mean Square Percentage Error in train: 0    3.020914
dtype: float32
1/1               0s 15ms/step
Mean Square Percentage Error in test: 0    4.805724
dtype: float32
```

```
/Users/aadityatrivedee/tf_lib/env/lib/python3.10/site-
packages/numpy/core/fromnumeric.py:86: FutureWarning: The behavior of
DataFrame.sum with axis=None is deprecated, in a future version this will reduce
over both axes and return a scalar. To retain the old behavior, pass axis=0 (or
do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
/Users/aadityatrivedee/tf_lib/env/lib/python3.10/site-
packages/numpy/core/fromnumeric.py:86: FutureWarning: The behavior of
DataFrame.sum with axis=None is deprecated, in a future version this will reduce
over both axes and return a scalar. To retain the old behavior, pass axis=0 (or
do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

[276]:

```
plt.figure(figsize=(20, 12))
plt.plot(model.predict(pd.DataFrame(X)), label='Predicted', marker='o',␣
 ↪linestyle='-', color='blue')
plt.plot(pd.DataFrame(y), label='Actual', marker='s', linestyle='-',␣
 ↪color='red')
plt.title('Mean Predicted vs Actual Values')
plt.xlabel('Chunk Index')
plt.ylabel('Mean Value')
plt.grid()
plt.legend()
plt.show()
```

```
2/2               0s 20ms/step
```



[277]:

```python
model.save('hybrid_put.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```