general_architecture_call

March 20, 2025

[1]:
```
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
```

[2]:
```
df = pd.read_csv('nvda_options_data.csv')
df = df.dropna()
```

[3]:
```
X = df[['strike', 'IV','Stock_Price', 'Time_to_Expire','Risk_Free_Rate']]
y = df[['CallPrice']]
X.head()
```

[3]:

	strike	IV	Stock_Price	Time_to_Expire	Risk_Free_Rate
0	5.0	0.566315	117.889999	1.820671	0.0422
1	10.0	0.566315	117.889999	1.820671	0.0422
2	15.0	0.566315	117.889999	1.820671	0.0422
3	20.0	0.566315	117.889999	1.820671	0.0422
4	25.0	0.566315	117.889999	1.820671	0.0422

[4]:
```
X.head()
```

[4]:

	strike	IV	Stock_Price	Time_to_Expire	Risk_Free_Rate
0	5.0	0.566315	117.889999	1.820671	0.0422
1	10.0	0.566315	117.889999	1.820671	0.0422
2	15.0	0.566315	117.889999	1.820671	0.0422
3	20.0	0.566315	117.889999	1.820671	0.0422
4	25.0	0.566315	117.889999	1.820671	0.0422

[5]:
```
print(f"X shape: {X.shape}, Y shape: {y.shape}")
```

```
X shape: (50, 5), Y shape: (50, 1)
```

[6]:
```
X = StandardScaler().fit_transform(X)
```

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=500)
```

[8]:
```
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)
```

```
2025-03-20 13:48:59.903130: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M3
2025-03-20 13:48:59.903162: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2025-03-20 13:48:59.903167: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
2025-03-20 13:48:59.903197: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2025-03-20 13:48:59.903208: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

[9]:
```
# Hyperparams
n_units = X_train.shape[1]
n1_units = 400
layers = 4
```

[10]:
```
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)
```

[11]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 5)	30
leaky_re_lu (LeakyReLU)	(None, 5)	0
dense_1 (Dense)	(None, 400)	2,400
leaky_re_lu_1 (LeakyReLU)	(None, 400)	0
dense_2 (Dense)	(None, 400)	160,400
leaky_re_lu_2 (LeakyReLU)	(None, 400)	0
dense_3 (Dense)	(None, 400)	160,400
leaky_re_lu_3 (LeakyReLU)	(None, 400)	0
dense_4 (Dense)	(None, 1)	401

Total params: 323,631 (1.23 MB)

Trainable params: 323,631 (1.23 MB)

Non-trainable params: 0 (0.00 B)

[12]:
```
model.compile(loss='mae', optimizer=Adam(learning_rate=0.001))
```

[13]:
```
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
2025-03-20 13:49:00.549122: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
2/2             1s 261ms/step - loss:
42.8725 - val_loss: 39.8130
Epoch 2/30
2/2             0s 52ms/step - loss:
42.9740 - val_loss: 39.3245
Epoch 3/30
2/2             0s 47ms/step - loss:
```

```
43.5751 - val_loss: 38.6020
Epoch 4/30
2/2             0s 50ms/step - loss:
41.5413 - val_loss: 37.5169
Epoch 5/30
2/2             0s 47ms/step - loss:
40.5886 - val_loss: 35.8920
Epoch 6/30
2/2             0s 50ms/step - loss:
38.4194 - val_loss: 33.5429
Epoch 7/30
2/2             0s 76ms/step - loss:
34.9284 - val_loss: 30.3032
Epoch 8/30
2/2             0s 50ms/step - loss:
33.2522 - val_loss: 25.9341
Epoch 9/30
2/2             0s 50ms/step - loss:
28.3201 - val_loss: 20.3325
Epoch 10/30
2/2             0s 47ms/step - loss:
23.7598 - val_loss: 14.0995
Epoch 11/30
2/2             0s 46ms/step - loss:
15.0656 - val_loss: 6.2820
Epoch 12/30
2/2             0s 92ms/step - loss:
6.3635 - val_loss: 7.2300
Epoch 13/30
2/2             0s 45ms/step - loss:
8.3107 - val_loss: 10.9863
Epoch 14/30
2/2             0s 45ms/step - loss:
11.9252 - val_loss: 9.5735
Epoch 15/30
2/2             0s 45ms/step - loss:
10.7955 - val_loss: 5.9096
Epoch 16/30
2/2             0s 46ms/step - loss:
7.2465 - val_loss: 2.3114
Epoch 17/30
2/2             0s 46ms/step - loss:
3.7318 - val_loss: 4.7192
Epoch 18/30
2/2             0s 53ms/step - loss:
5.3522 - val_loss: 5.6188
Epoch 19/30
2/2             0s 45ms/step - loss:
```

```
5.9139 - val_loss: 3.9370
Epoch 20/30
2/2             0s 73ms/step - loss:
3.9261 - val_loss: 1.7932
Epoch 21/30
2/2             0s 47ms/step - loss:
3.0508 - val_loss: 3.0642
Epoch 22/30
2/2             0s 45ms/step - loss:
4.4266 - val_loss: 3.1360
Epoch 23/30
2/2             0s 96ms/step - loss:
3.9024 - val_loss: 2.1387
Epoch 24/30
2/2             0s 46ms/step - loss:
2.2400 - val_loss: 3.2035
Epoch 25/30
2/2             0s 46ms/step - loss:
2.7482 - val_loss: 4.1954
Epoch 26/30
2/2             0s 46ms/step - loss:
3.5647 - val_loss: 3.0262
Epoch 27/30
2/2             0s 46ms/step - loss:
2.6371 - val_loss: 2.3288
Epoch 28/30
2/2             0s 45ms/step - loss:
1.9169 - val_loss: 2.0676
Epoch 29/30
2/2             0s 95ms/step - loss:
2.5128 - val_loss: 1.9864
Epoch 30/30
2/2             0s 45ms/step - loss:
2.2168 - val_loss: 2.8343
```

[14]:
```
model.evaluate(X_test[:3], y_test[:3],batch_size=batch_size)
```

```
1/1             0s 55ms/step - loss:
1.1585
```

[14]:
```
1.1585124731063843
```

[15]:
```
model.predict(pd.DataFrame(X_test).iloc[0:3])
```

```
1/1             0s 53ms/step
```

[15]:
```
array([[16.361498],
       [31.606623],
       [ 8.356339]], dtype=float32)
```

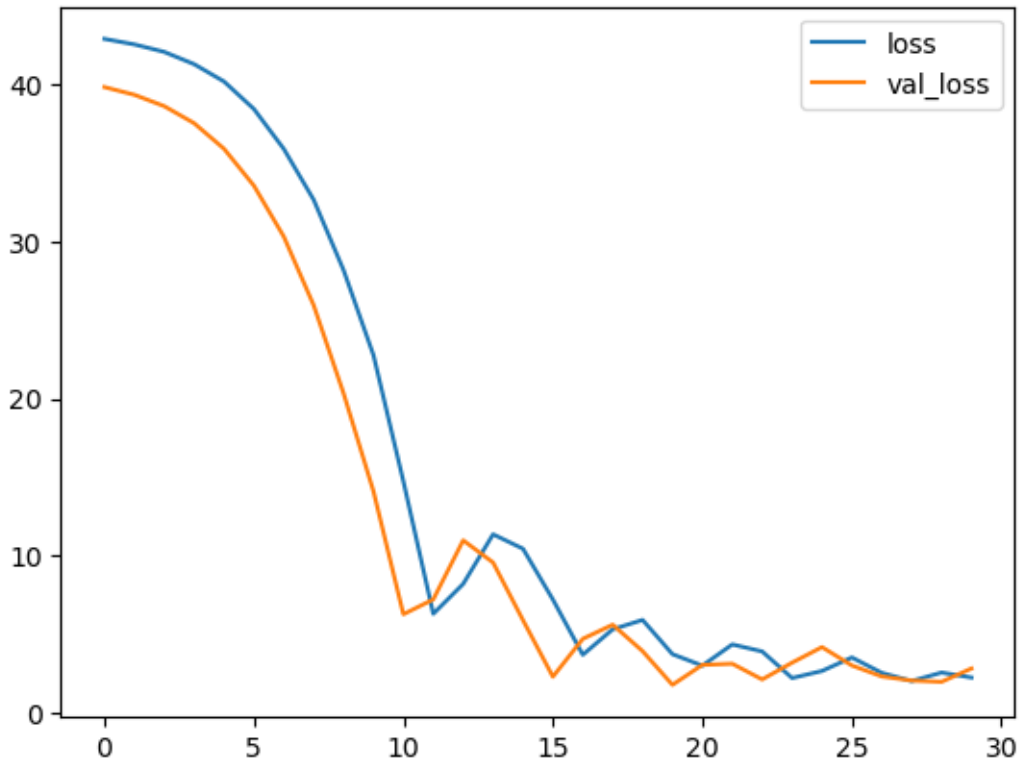
[16]:
```
pd.DataFrame(y_test).iloc[0:3]
```

[16]:

	0
0	17.299999
1	32.599998
2	9.900000

[17]:
```
loss_df = pd.DataFrame(losses.history)
loss_df.loc[:,['loss','val_loss']].plot()
```

[17]: <Axes: >



[18]:
```
def error(x,y):
    error = np.sum(np.abs(model.predict(pd.DataFrame(x)) - pd.
 ↪DataFrame(y)))*100/(np.sum(np.array(y)))
    return error
```

[19]:
```
print('Mean Square Percentage Error in train:', error(X_train, y_train))
print('Mean Square Percentage Error in test:', error(X_test, y_test))
```

```
2/2             0s 21ms/step
Mean Square Percentage Error in train: 0    5.603084
dtype: float32
```

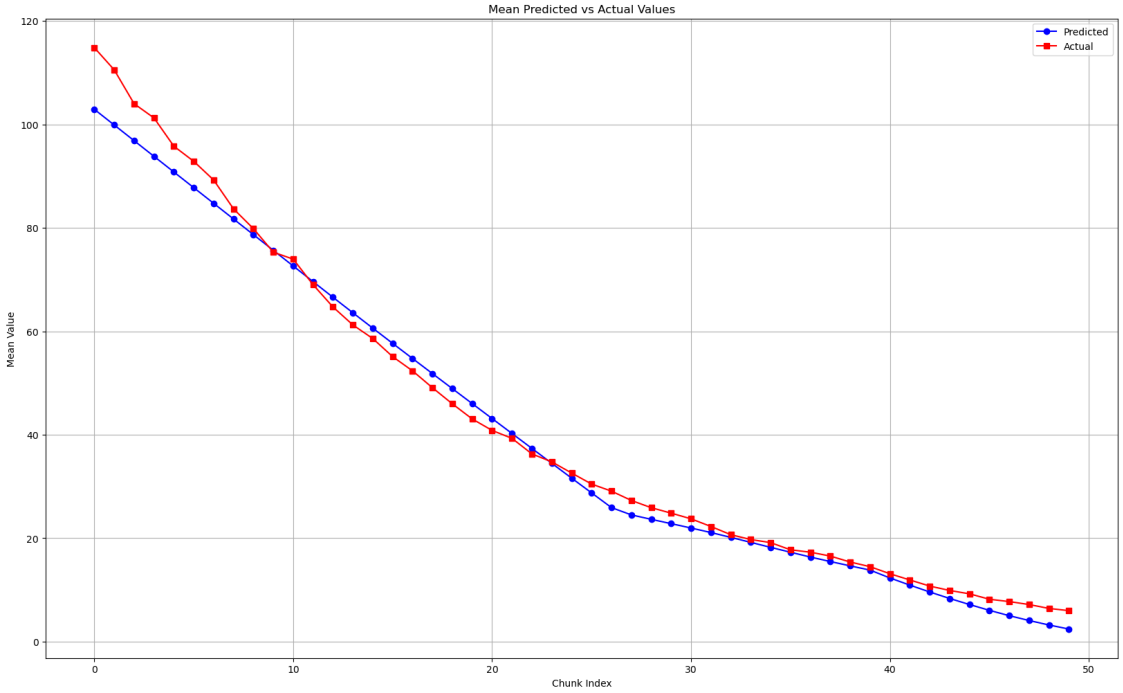
```
1/1             0s 77ms/step
Mean Square Percentage Error in test: 0    7.050473
dtype: float32
```

```
/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)
```

[20]:
```
def aggregate(data, size=1):
        return np.array([np.mean(data[i:i + size]) for i in range(0, len(data),␣
 ↪size)])
```

[24]:
```
plt.figure(figsize=(20, 12))
X_plt=model.predict(pd.DataFrame(X))
plt.plot(aggregate(X_plt), label='Predicted', marker='o', linestyle='-',␣
 ↪color='blue')
plt.plot(aggregate(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 10ms/step
```



[23]:
```
model.save('generalarch_call.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')`.
```