Lab Logbook Requirement:

- *Plot 4 graphs:*
  1. *Precision during training graph*
  2. *More detailed Precision graph*
  3. *Training accuracy graph*
  4. *More detailed Accuracy graph*

¶

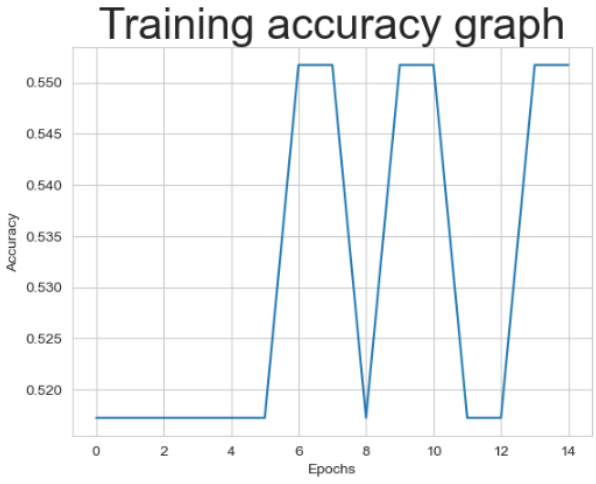
NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

Precision during training

[280]:
```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
import matplotlib.pyplot as plt


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)
```

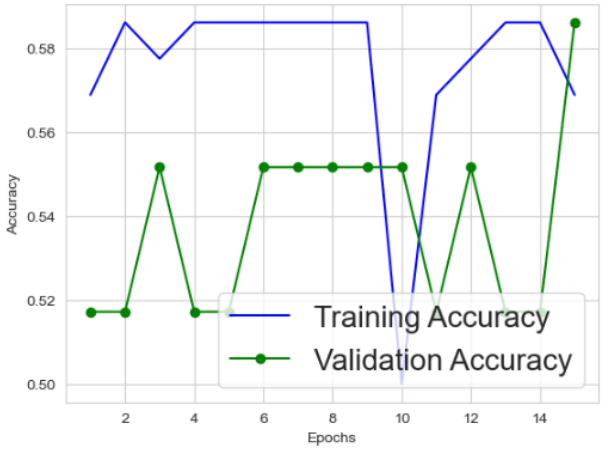
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))


model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


model.fit(X_train, y_train, epochs=20, batch_size=8, verbose=1, validation_split=0.2)


y_pred = (model.predict(X_test) > 0.5).astype(int)
precision_values = [precision_score(y_test[:i+1], y_pred[:i+1]) for i in range(len(y_test))]


plt.plot(precision_values)
plt.xlabel('Samples')
plt.ylabel('Precision')
plt.title('Precision during training')
plt.show()
```

```
Epoch 1/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 8s 86ms/step - accuracy: 0.6213 - loss: 0.6871 - val_accuracy: 0.5172 - val_loss: 0.6905
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5919 - loss: 0.6855 - val_accuracy: 0.5172 - val_loss: 0.6887
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5701 - loss: 0.6864 - val_accuracy: 0.5172 - val_loss: 0.6883
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6151 - loss: 0.6628 - val_accuracy: 0.5172 - val_loss: 0.6899
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5518 - loss: 0.6906 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5695 - loss: 0.6979 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5799 - loss: 0.6797 - val_accuracy: 0.5517 - val_loss: 0.6918
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5804 - loss: 0.7042 - val_accuracy: 0.5517 - val_loss: 0.6901
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.5763 - loss: 0.6813 - val_accuracy: 0.5172 - val_loss: 0.6892
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5349 - loss: 0.7019 - val_accuracy: 0.5517 - val_loss: 0.6892
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5925 - loss: 0.6821 - val_accuracy: 0.5517 - val_loss: 0.6887
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5529 - loss: 0.6778 - val_accuracy: 0.5172 - val_loss: 0.6890
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5066 - loss: 0.6645 - val_accuracy: 0.5172 - val_loss: 0.6893
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6401 - loss: 0.6639 - val_accuracy: 0.5517 - val_loss: 0.6882
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5553 - loss: 0.6868 - val_accuracy: 0.5517 - val_loss: 0.6869
```



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import Callback


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))


class AccuracyCallback(Callback):
    def __init__(self):
        self.train_accuracy = []
        self.val_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        self.train_accuracy.append(logs['accuracy'])
        self.val_accuracy.append(logs['val_accuracy'])


model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


accuracy_callback = AccuracyCallback()
model.fit(X_train, y_train, epochs=15, batch_size=8, verbose=1, validation_data=(X_val, y_val), callbacks=[accuracy_callback])


epochs = range(1, len(accuracy_callback.train_accuracy) + 1)
plt.plot(epochs, accuracy_callback.train_accuracy, label='Training Accuracy', color='blue')
plt.plot(epochs, accuracy_callback.val_accuracy, label='Validation Accuracy', color='green', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 99ms/step - accuracy: 0.6340 - loss: 0.6865 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5411 - loss: 0.6914 - val_accuracy: 0.5172 - val_loss: 0.6886
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5361 - loss: 0.6769 - val_accuracy: 0.5517 - val_loss: 0.6877
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5294 - loss: 0.6881 - val_accuracy: 0.5172 - val_loss: 0.6884
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5850 - loss: 0.6820 - val_accuracy: 0.5172 - val_loss: 0.6890
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6284 - loss: 0.6660 - val_accuracy: 0.5517 - val_loss: 0.6928
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6018 - loss: 0.6597 - val_accuracy: 0.5517 - val_loss: 0.6906
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6017 - loss: 0.6935 - val_accuracy: 0.5517 - val_loss: 0.6886
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6411 - loss: 0.6592 - val_accuracy: 0.5517 - val_loss: 0.6882
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5075 - loss: 0.6570 - val_accuracy: 0.5517 - val_loss: 0.6870
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5645 - loss: 0.6803 - val_accuracy: 0.5172 - val_loss: 0.6855
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5788 - loss: 0.6932 - val_accuracy: 0.5517 - val_loss: 0.6872
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5461 - loss: 0.6774 - val_accuracy: 0.5172 - val_loss: 0.6867
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5766 - loss: 0.6606 - val_accuracy: 0.5172 - val_loss: 0.6861
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5700 - loss: 0.6740 - val_accuracy: 0.5862 - val_loss: 0.6840
```




```python
# Calculate the prediction vector

# Verify and reshape X_test
print("Original X_test shape:", X_test.shape)
if len(X_test.shape) == 2:
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
print("Reshaped X_test shape:", X_test.shape)

# Check model input shape
print("Expected input shape:", model.input_shape)

# Make predictions
pred = model.predict(X_test)
print("Predictions:", pred)
```

```
Original X_test shape: (29, 10, 1)
Reshaped X_test shape: (29, 10, 1)
Expected input shape: (None, 10, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
Predictions: [[0.5556013 ]
 [0.4996255 ]
 [0.6276852 ]
 [0.63105965]
 [0.6306144 ]
 [0.49330345]
 [0.5002903 ]
 [0.4950856 ]
 [0.49369597]
 [0.63282484]
 [0.63177073]
 [0.5741288 ]
 [0.5018753 ]
 [0.50046986]
 [0.5219916 ]
 [0.6392977 ]
 [0.49119493]
 [0.50059724]
 [0.50257075]
 [0.50231224]
 [0.500361  ]
 [0.50128865]
 [0.63047624]
 [0.5686562 ]
 [0.63633466]
 [0.6273813 ]
 [0.5925585 ]
 [0.5055052 ]
 [0.49101394]]
```

```
print(pred)
```

```
[[0.5556013 ]
 [0.4996255 ]
 [0.6276852 ]
 [0.63105965]
 [0.6306144 ]
 [0.49330345]
 [0.5002903 ]
 [0.4950856 ]
 [0.49369597]
 [0.63282484]
 [0.63177073]
 [0.5741288 ]
 [0.5018753 ]
 [0.50046986]
 [0.5219916 ]
 [0.6392977 ]
 [0.49119493]
 [0.50059724]
 [0.50257075]
 [0.50231224]
 [0.500361  ]
 [0.50128865]
 [0.63047624]
 [0.5686562 ]
 [0.63633466]
 [0.6273813 ]
 [0.5925585 ]
 [0.5055052 ]
 [0.49101394]]
```

[298]:
```
len(pred)
```

[298]:
```
29
```

[ ]:

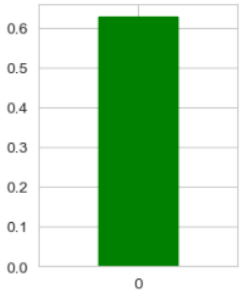
[308]:
```
import random

pred = model.predict(X_test)

# Check: we take a random element random.randint() and look: what is the difference between test and predict

n_rec = random.randint(0, X_test.shape[0])
print(n_rec)


print("Predicted probability:", pred[n_rec], ", right answer:", y_test[n_rec])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
5
Predicted probability: [0.49330345] , right answer: 1
```

```
classes=['0 is Flat', '1 is Trend']

index = random.randint(0, y_test.shape[0])
print('Right answer: ', y_test[index])

x = X_test[index]
x = np.expand_dims(x, axis=0)

prediction = model.predict(x)
sample = x

ans = round(float(prediction))

fig = plt.figure(figsize=(5,3))

ax = fig.add_subplot(1, 2, 2)
bar_list = ax.bar(np.arange(1), prediction[0], align='center')
if ans == y_test[index]:
    bar_list[0].set_color('g')
else:
    bar_list[0].set_color('r')

ax.set_xticks(np.arange(1))
ax.set_xlim([-1, 1])
ax.grid('on')

plt.show()
print('Predicted answer: {}'.format(classes[ans]), "\n ")
print('Right answer: {}'.format(classes[y_test[index].astype(int)]) )
print(classes)
```

```
Right answer:  1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
```

```
C:\Users\ccs\AppData\Local\Temp\ipykernel_7292\2073967640.py:12: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before per
forming this operation. (Deprecated NumPy 1.25.)
```




```
Predicted answer: 1 is Trend

Right answer: 1 is Trend
['0 is Flat', '1 is Trend']
```