main

May 3, 2024

1 Subject: Machine Learning, 1st March 2024 to 3rd May 2024.

1.1 Topic: Group Project (Task B)

1.2 Learning Outcomes:

- MO2: Select and apply machine learning algorithms to formulate solutions to different types of machine learning problems, taking into account criteria such data availability and characteristics, and problem-specific requirements for balancing speed, accuracy, and explainability.
- MO3: Implement and evaluate contemporary machine learning solutions to application problems using a range of contemporary frameworks.
- MO4: Demonstrate an awareness of the ethical and societal implications of machine learning solutions.

NOTE: To pylint your code type "pip install nbqa pylint // nbqa pylint my_notebook.ipynb".

1.3 Dependencies

[79]:
```
import subprocess
import os


current_directory = os.path.dirname(os.path.abspath("main.ipynb"))
current_directory += "/"

def install(requirements):
    """
    Install all the relevent project dependencies.
    """

    try:
        if os.path.isdir('.venv'):
            activate_script = os.path.join('.venv', 'bin', 'activate')
            subprocess.check_call(['source', activate_script], shell=True)

        with open(requirements, 'r') as f:
            requirements = f.read().splitlines()
            subprocess.check_call(['pip', 'install'] + requirements)
```

```
        print("Installed dependencies.")

    except FileNotFoundError:
        print(f"File '{requirements}' not found.")
    except subprocess.CalledProcessError:
        pass

if __name__ == "__main__":
    install(current_directory + "project/requirements.txt")
```

```
Installed dependencies.
```

1.4 Model Variables

Below are the variables we can use to fine-tune our models.

[80]:
```
N_TIMESTEP = 10
TEST_SIZE = 0.15
UNITS = 128
ACTIVATION_FUNCTION = "relu" # Non-linear
RNN_ACTIVATION_FUNCTION = "sigmoid" # Linear
RNN_OPTIMISER = "adam"
LOSS_ERROR = "mse"
EPOCH = 32
PATIENCE = 5
BATCH_SIZE = 32
```

2 Preprocessing our data

[81]:
```
import pandas as pd
import datetime as dt
import warnings
from pandas.errors import SettingWithCopyWarning

warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)

raw_data = pd.read_csv(current_directory + "project/raw_data/BTC-USD.csv")
df = raw_data[["Date", "Close"]]

def to_datetime(s: str):
    year, month, day = s.split("-")
    return dt.datetime(
        int(year),
        int(month),
        int(day)
    )
```

```
close = df["Close"]
dates = df["Date"].apply(to_datetime)
print(df)
```

```
            Date         Close
0     2020-03-08   8108.116211
1     2020-03-09   7923.644531
2     2020-03-10   7909.729492
3     2020-03-11   7911.430176
4     2020-03-12   4970.788086
…            …             …
1458  2024-03-05  63801.199219
1459  2024-03-06  66106.804688
1460  2024-03-07  66925.484375
1461  2024-03-08  68300.093750
1462  2024-03-09  68498.882812

[1463 rows x 2 columns]
```

2.0.1 Partitioning and Normalisation

Get the training set, validation set and test set for the future model to make predictions.

[82]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Prepare X and y
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_close_price = 'ScaledClosePrice'
df[scaled_close_price] = scaler.fit_transform(df[["Close"]])


# N_TIMESTEP = 10 #
n_samples = len(dates)
print(n_samples)

X = []
y = []
for i in range(len(df) - N_TIMESTEP):  # Adjusted loop iteration range
    # Historical close prices
    X.append(df[scaled_close_price].values[i:i+N_TIMESTEP])

    # Future close price from dataset
    y.append(df[scaled_close_price].values[i+N_TIMESTEP])

X = np.array(X)
y = np.array(y)
```

```
print(X, "length:", len(X))
print(y)
```

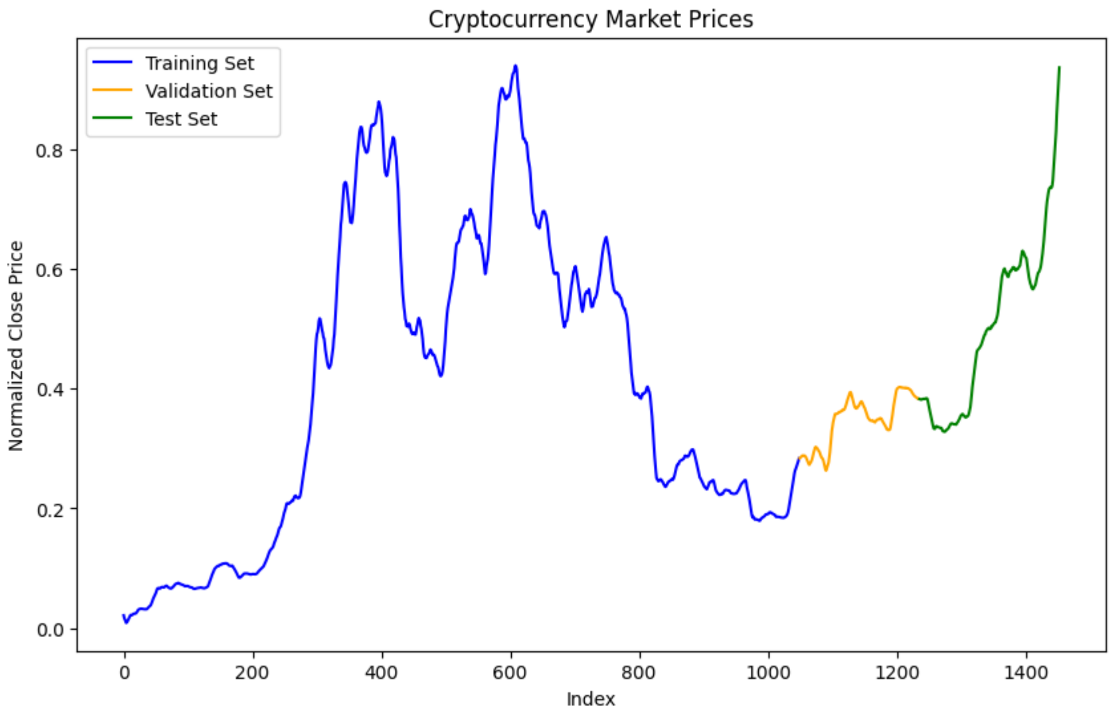
```
1463
[[4.93848924e-02 4.64811114e-02 4.62620738e-02 … 6.63528250e-03
  6.87757042e-04 4.01147413e-03]
 [4.64811114e-02 4.62620738e-02 4.62888444e-02 … 6.87757042e-04
  4.01147413e-03 4.21310275e-03]
 [4.62620738e-02 4.62888444e-02 0.00000000e+00 … 4.01147413e-03
  4.21310275e-03 1.92104736e-02]
 …
 [7.79995284e-01 8.20339147e-01 9.05646568e-01 … 9.97348122e-01
  9.26053447e-01 9.62346138e-01]
 [8.20339147e-01 9.05646568e-01 8.85082340e-01 … 9.26053447e-01
  9.62346138e-01 9.75233030e-01]
 [9.05646568e-01 8.85082340e-01 9.04636680e-01 … 9.62346138e-01
  9.75233030e-01 9.96870848e-01]] length: 1453
[0.0042131  0.01921047 0.01932988 … 0.97523303 0.99687085 1.        ]
```

[83]:
```
# Split the data into Samples, steps, then features
X = np.reshape(X, (X.shape[0], N_TIMESTEP, 1))
print(len(X))

# Split the data into train, test and validation
# First, split into train and test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, shuffle=False
)

# Calc validation remainder: (0.15 * (1 - 0.15)) / (1 - 0.15)
validation_size = (TEST_SIZE * (1.0 - TEST_SIZE)) / (1.0 - TEST_SIZE)

# Split the remaining data into validation and new train sets
X_train, X_validation, y_train, y_validation = train_test_split(
    X_train, y_train, test_size=validation_size, shuffle=False
)

print("Training set:", len(X_train)/len(close), "%")
print("Validation set:", len(X_validation)/len(close), "%")
print("Test set:", len(X_test)/len(close), "%")
```

```
1453
Training set: 0.7170198222829802 %
Validation set: 0.127136021872864 %
Test set: 0.14900888585099112 %
```

[84]:

```
import matplotlib.pyplot as plt

# Calculate the indices for train, validation, and test sets
train_start, train_end = 0, len(X_train) - 1
validation_start, validation_end = len(X_train), len(X_train) +␣
↪len(X_validation) - 1
test_start, test_end = len(X_train) + len(X_validation), len(X_train) +␣
↪len(X_validation) + len(X_test) - 1

plt.figure(figsize=(10, 6))

# Plot train set
plt.plot(range(train_start, train_end + 1), np.mean(X_train[:, :, 0], axis=1),␣
↪color='blue', label="Training Set")

# Plot validation set
plt.plot(range(validation_start, validation_end + 1), np.mean(X_validation[:, :
↪, 0], axis=1), color='orange', label="Validation Set")

# Plot test set
plt.plot(range(test_start, test_end + 1), np.mean(X_test[:, :, 0], axis=1),␣
↪color='green', label="Test Set")

plt.xlabel('Index')
plt.ylabel('Normalized Close Price')
plt.title("Cryptocurrency Market Prices")
plt.legend()
plt.show()
```



2.1 Creating the models

[85]:
```
import tensorflow as tf

try:
    from keras.api.models import Sequential
    from keras.api.layers import LSTM, Dense
except ImportError:
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import LSTM

def create_model(
    units: int,
    activation: str = "relu",
    recurrent_activation: str = "sigmoid",
    optimizer: str = "adam"
) -> Sequential:

    model = Sequential()
    model.add(LSTM(
        units,
        activation=activation,
        recurrent_activation=recurrent_activation,
        return_sequences=False, # Display hidden layer
```

```
        input_shape=(N_TIMESTEP, 1) # Timestep, features.
    ))

    # Output Layer. Updates dimensionality.
    model.add(Dense(
        1
    ))

    model.compile(optimizer=optimizer, loss=LOSS_ERROR)
    return model

lstm_model = create_model(
    units=UNITS,
    activation=ACTIVATION_FUNCTION,
    recurrent_activation=RNN_ACTIVATION_FUNCTION,
    optimizer=RNN_OPTIMISER
)

print(
    X.shape,
    y.shape,
    X_train.shape,
    y_train.shape,
    X_test.shape,
    y_test.shape,
    X_validation.shape,
    y_validation.shape
)
```

```
(1453, 10, 1) (1453,) (1049, 10, 1) (1049,) (218, 10, 1) (218,) (186, 10, 1)
(186,)
```

```
c:\Program Development\UWE Bristol\Year 2\Machine Learning\github\v1\uni-
machine-learning\.venv\Lib\site-packages\keras\src\layers\rnn\rnn.py:204:
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__(**kwargs)
```

[86]:
```
# # Visualise the data:
target_variable = 'Next Close'
X_reshaped = np.reshape(X, (X.shape[0], -1))
X_df = pd.DataFrame(X_reshaped, columns=[f"Close_{i}" for i in range(X.
 ↪shape[1])])
y_df = pd.DataFrame(y, columns=[target_variable])
data_df = pd.concat([X_df, y_df], axis=1)

# Display the DataFrame
```

```
print(data_df)
print(len(data_df[target_variable]))
```

```
       Close_0   Close_1   Close_2   Close_3   Close_4   Close_5   Close_6  \
0     0.049385  0.046481  0.046262  0.046289  0.000000  0.009333  0.003614
1     0.046481  0.046262  0.046289  0.000000  0.009333  0.003614  0.006635
2     0.046262  0.046289  0.000000  0.009333  0.003614  0.006635  0.000688
3     0.046289  0.000000  0.009333  0.003614  0.006635  0.000688  0.004011
4     0.000000  0.009333  0.003614  0.006635  0.000688  0.004011  0.004213
…          …         …         …         …         …         …         …
1448  0.733539  0.736091  0.779995  0.820339  0.905647  0.885082  0.904637
1449  0.736091  0.779995  0.820339  0.905647  0.885082  0.904637  0.898170
1450  0.779995  0.820339  0.905647  0.885082  0.904637  0.898170  0.916076
1451  0.820339  0.905647  0.885082  0.904637  0.898170  0.916076  0.997348
1452  0.905647  0.885082  0.904637  0.898170  0.916076  0.997348  0.926053

       Close_7   Close_8   Close_9  Next Close
0     0.006635  0.000688  0.004011    0.004213
1     0.000688  0.004011  0.004213    0.019210
2     0.004011  0.004213  0.019210    0.019330
3     0.004213  0.019210  0.019330    0.019114
4     0.019210  0.019330  0.019114    0.013529
…          …         …         …         …
1448  0.898170  0.916076  0.997348    0.926053
1449  0.916076  0.997348  0.926053    0.962346
1450  0.997348  0.926053  0.962346    0.975233
1451  0.926053  0.962346  0.975233    0.996871
1452  0.962346  0.975233  0.996871    1.000000

[1453 rows x 11 columns]
1453
```

2.2 Creating the Models

[87]:
```
from tensorflow.keras.callbacks import EarlyStopping

# We use this object to monitor the loss of the model.
# After 'patient' consecutive attempts if no improvement we stop running the
 ↪model.
# We use val_loss since we are fine-tuning with validation data.
loss_callback = EarlyStopping(
    monitor='val_loss',
    patience=PATIENCE,
    restore_best_weights=True
)

# Fit the model.
```

```
# Since we dont have explicit labels for validation monitor performance
# during training instead.
model_data = lstm_model.fit(
    X_train, y_train,
    epochs=EPOCH,
    batch_size=BATCH_SIZE,
    validation_data=(X_validation, y_validation),
    callbacks=[loss_callback],
    verbose=False
)

# Pass in true values to evaluate loss
model_loss = lstm_model.evaluate(X_test, y_test)
predictions = lstm_model.predict(X_test)
```

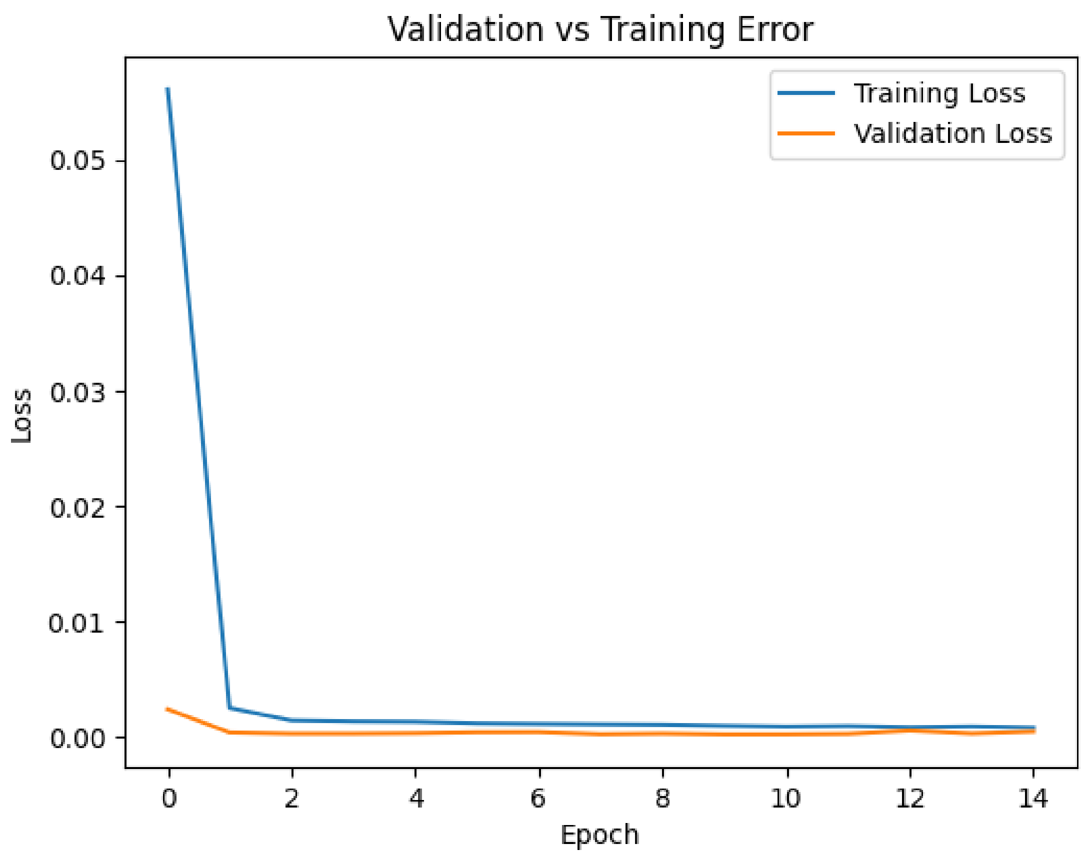
```
7/7               0s 2ms/step - loss:
4.0049e-04
7/7               0s 18ms/step
```

2.2.1 Plotting single instance LSTM

[88]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# mse_score = mean_squared_error(y_test, predictions)
# print(mse_score)

# Plot training and validation loss curves
training_loss = model_data.history['loss']
validation_loss = model_data.history['val_loss']

print(min(training_loss), min(validation_loss))
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title("Validation vs Training Error")
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
0.0008328879484906793 0.0002507037133909762
```

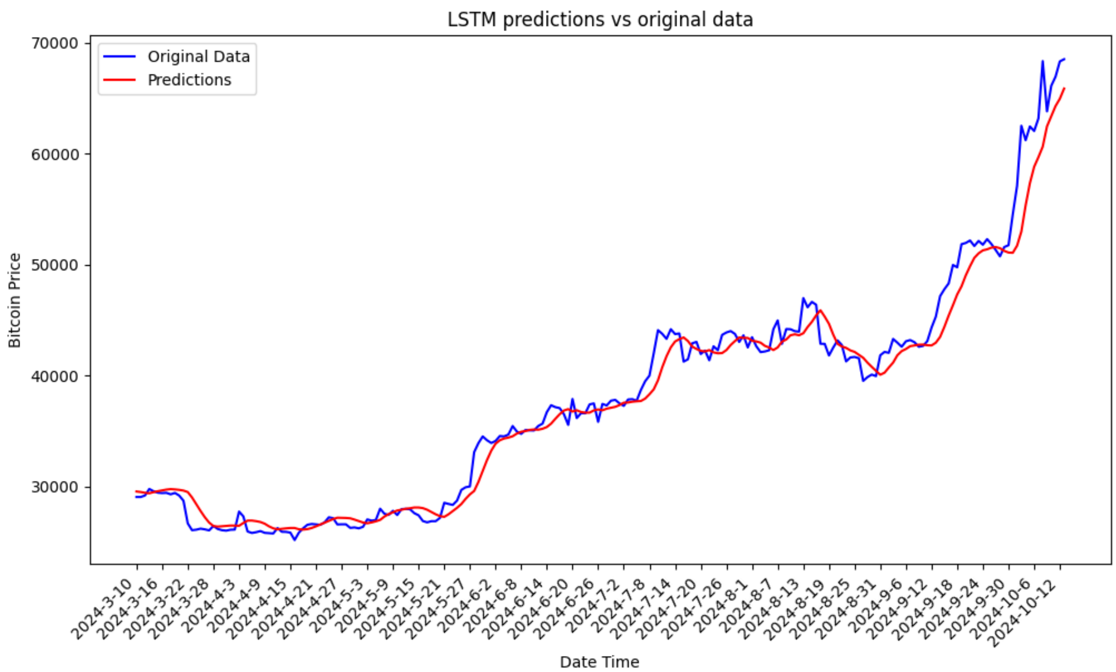
[88]:
```
<matplotlib.legend.Legend at 0x22a51e11a50>
```



[89]:
```python
max_dates = 6

# Reverse scaling on predictions
predictions_original = scaler.inverse_transform(predictions)
y_test_original = scaler.inverse_transform(y_test.reshape(-1, 1))

last_date_str = df['Date'].iloc[-1]
last_date = to_datetime(last_date_str)
new_dates = []
for i in range(len(predictions)):
    new_date = last_date + dt.timedelta(days=i+1)
    new_dates.append("%s-%s-%s" % (new_date.year, new_date.month, new_date.day))

plt.figure(figsize=(10, 6))
plt.plot(new_dates, y_test_original, label='Original Data', color='blue')
plt.plot(new_dates, predictions_original, label='Predictions', color='red')
plt.xlabel('Date Time')
plt.xticks(new_dates[::max_dates], rotation=45, ha='right')
plt.tight_layout()  # Adjust layout to prevent clipping of labels
```

```
plt.ylabel('Bitcoin Price')
plt.title('LSTM predictions vs original data')
plt.legend()
```

[89]: <matplotlib.legend.Legend at 0x22a52f6df50>



2.3 Grid Search the Models

[ ]:
```
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import (
    mean_squared_error
)

search_params = {
    "units": [32, 64, 128],  # Number of LSTM units
    "activation": ["relu", "tanh"],  # Activation function for LSTM units
    "recurrent_activation": ["sigmoid", "tanh"],  # Recurrent activation␣
 ↪function for LSTM units
    "optimizer": ["adam", "rmsprop"]  # Optimizer for training the model
}

class Node:
    def __init__(self, score, model, params, timestep):
        self.score = score
        self.model = model
```

```
        self.params = params
        self.timestep = timestep


best_model = None
best_score = float('inf')
search_array = []  # Store all grid searched nodes to get data
counter = 0
for params in ParameterGrid(search_params):
    search_model = create_model(
        **params
    )

    search_model.fit(
        X_train,
        y_train,
        epochs=EPOCH,
        batch_size=BATCH_SIZE,
        callbacks=[loss_callback],
        verbose=0
    )

    y_pred = search_model.predict(X_validation)
    mse_score = mean_squared_error(y_validation, y_pred)

    if mse_score < best_score:
        best_score = mse_score
        best_model = search_model
        best_params = params
        search_array.append(Node(best_score, best_model, best_params, counter))

    counter += 1
```

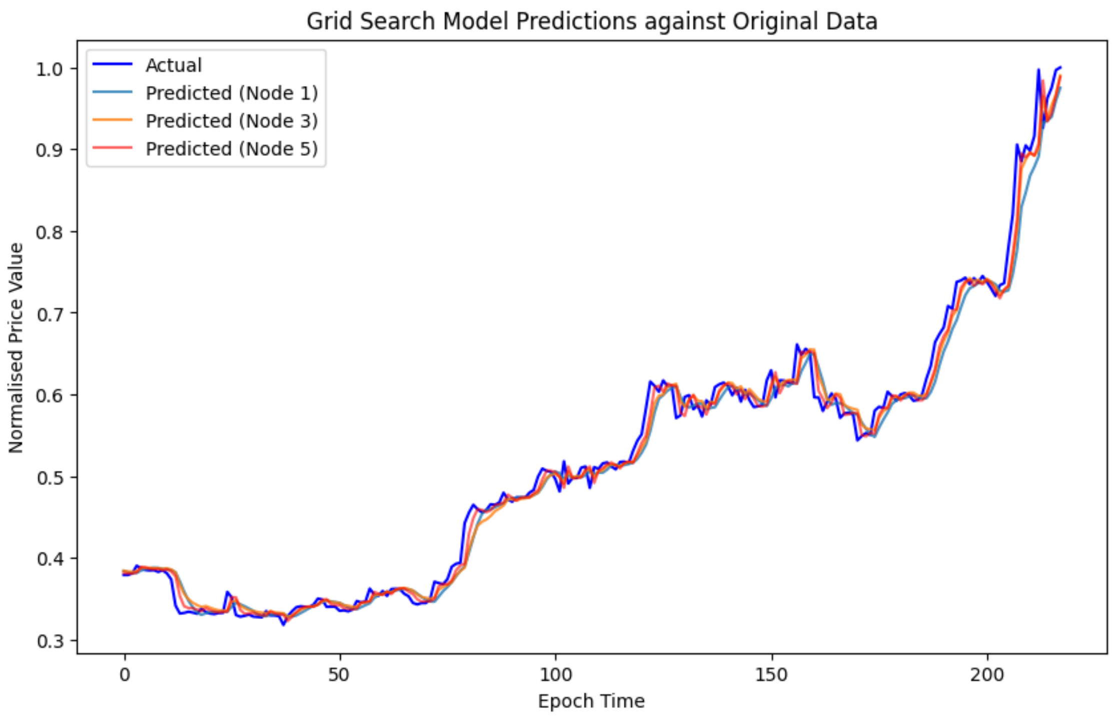
2.3.1 Plotting Predictions

[91]:
```
# Display all grid search Nodes
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual', color="blue")
for i, node in enumerate(search_array):
    node.predictions = node.model.predict(X_test)

    # Reduce coupling of data
    if i % 2 == 0:
        if i==4:
            plt.plot(node.predictions, label=f'Predicted (Node {i+1})',␣
 ↪color="red", alpha=0.6)
            continue
```

```
        plt.plot(node.predictions, label=f'Predicted (Node {i+1})', alpha=0.8)

plt.xlabel('Epoch Time')
plt.ylabel('Normalised Price Value')
plt.title('Grid Search Model Predictions against Original Data')
plt.legend()
```

```
7/7              0s 997us/step
7/7              0s 1ms/step
7/7              0s 997us/step
7/7              0s 1ms/step
7/7              0s 1ms/step
```

[91]: <matplotlib.legend.Legend at 0x22a68e96d50>



2.4 Checking error scores

[92]:
```
# Dataframe all the prediction performance
for i, node in enumerate(search_array):
    print(f"Predicted Node {i+1} MSE:", node.score, node.timestep, node.params)
```

```
Predicted Node 1 MSE: 0.00019165788285618928 0
Predicted Node 2 MSE: 0.0001861332278829973 2
Predicted Node 3 MSE: 0.00016952102826509865 3
```

```
Predicted Node 4 MSE: 0.0001272269517841474 4
Predicted Node 5 MSE: 0.00010715549457547691 17
```