Time and Series Forecasting with LSTM- Recurrent Neural Network

We will be implementing a time series forecasting model using the LSTM algorithm in Keras. We will train the model on the monthly airline passenger dataset and used it to make predictions for the next 12 months.

`Explanation of the code:`

In the context of the time series forecasting algorithm used in this article, instead of manually calculating the slope and intercept of the line, the algorithm uses a neural network with LSTM layers to learn the underlying patterns and relationships in the time series data. The neural network is trained on a portion of the data and then used to make predictions for the remaining portion. In this algorithm, the prediction for the next time step is based on the previous n_inputs time steps, which is similar to the concept of using y(t) to predict y(T+1) in the linear regression example. However, instead of using a simple linear equation, the prediction in this algorithm is generated using the activation function of the LSTM layer. The activation function allows the model to capture non-linear relationships in the data, making it more effective in capturing complex patterns in time series data.

The activation function used in the LSTM model is the rectified linear unit (ReLU) activation function. This activation function is commonly used in deep learning models because of its simplicity and effectiveness in dealing with the vanishing gradient problem. In the LSTM model, the ReLU activation function is applied to the output of each LSTM unit to introduce non-linearity in the model and allow it to learn complex patterns in the data. The ReLU function has a simple thresholding behavior where any negative input is mapped to zero and any positive input is passed through unchanged, making it computationally efficient.

`(Link to dataset:`
`https://github.com/jbrownlee/Datasets/blob/master/airline-passengers.csv)`

The code imports three important libraries: numpy, pandas, and matplotlib. The pandas library is used to read in the ‘airline-passengers.csv’ file and set the ‘Month’ column as the index, which allows the data to be analyzed over time. The code then uses the matplotlib library to create a line plot showing the number of airline passengers over time. Finally, the plot is displayed using the ‘plt.show’ function. This code is useful for anyone interested in analyzing time series data, and it demonstrates how to use pandas and matplotlib to visualize trends in data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('airline-passengers.csv', index_col='Month',
parse_dates=True)
df.index.freq = 'MS'
df.shape
df.columns
plt.figure(figsize=(20, 4))
plt.plot(df.Passengers, linewidth=2)
plt.show()
```

This code creates two new data frames ‘df_train’ and ‘df_test’ by splitting an existing time series data frame ‘df’ into training and testing sets. The ‘nobs’ variable is set to 12, which means that the last 12 observations of ‘df’ will be used for testing, while the rest of the data will be used for training. The training set is stored in ‘df_train’ and consists of all rows in ‘df’ except for the last 12 rows, while the testing set is stored in ‘df_test’ and consists of only the last 12 rows of ‘df’. The ‘shape’ attribute is then used to print the number of rows and columns in each data frame, which confirms that the splitting was done correctly. This code is useful for preparing time series data for modeling and testing purposes by splitting it into two sets.

```
nobs = 12
df_train = df.iloc[:-nobs]
df_test = df.iloc[-nobs:]
df_train.shape
df_test.shape
```

This code snippet demonstrates how to use the ‘TimeseriesGenerator’ class from Keras and the ‘MinMaxScaler’ class from scikit-learn to generate input and output arrays for a time series forecasting model. The code first creates an instance of the ‘MinMaxScaler’ class and fits it to the training data set (‘df_train’) in order to scale the data. The scaled data is then stored in ‘scaled_train’ and ‘scaled_test’ data frames. The number of time steps (‘n_inputs’) is set to 12, and the number of features (‘n_features’) is set to 1. A ‘TimeseriesGenerator’ object is created with the ‘scaled_train’ data and a window length of ‘n_inputs’ and a batch size of 1. Finally, a loop is used to iterate over the ‘generator’ object and print out the input and output arrays for each time step. The ‘X’ and ‘y’ variables represent the input and output arrays for each time step, respectively. The ‘flatten()’ method is used to convert the input array into a 1D array for easier printing. Overall, this code is useful for preparing time series data for forecasting models using a sliding window approach.

```
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_train)
scaled_train = scaler.transform(df_train)
scaled_test = scaler.transform(df_test)
n_inputs = 12
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length =
n_inputs, batch_size =1)

for i in range(len(generator)):
    X, y = generator[i]
    print(f' \n {X.flatten()} and {y}')
```

This code returns the shape of an array or matrix 'X'. The 'shape' attribute is a property of NumPy arrays and returns a tuple representing the dimensions of the array. The code does not provide any additional context, so it is unclear what the shape of 'X' is. The output will be in the format (rows, columns).

```
X.shape
```

This code demonstrates how to create an LSTM neural network model for time series forecasting using Keras. Firstly, the necessary Keras classes are imported, including 'Sequential', 'Dense', and 'LSTM'. The model is created as a 'Sequential' object and an LSTM layer is added with 200 neurons, the 'relu' activation function, and an input shape defined by 'n_inputs' and 'n_features'. The LSTM layer output is then passed to a 'Dense' layer with a single output neuron. The model is compiled with the 'adam' optimizer and the mean squared error ('mse') loss function. The 'summary()' method is used to display a summary of the architecture, including the number of parameters and the shapes of the input and output tensors for each layer. This code can be useful for creating an LSTM model for time series forecasting, as it provides an easy-to-follow example that can be adapted to different data sets and forecasting problems.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

model = Sequential()
model.add(LSTM(200, activation='relu', input_shape = (n_inputs,
n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.summary()
```

This code trains the LSTM neural network model using the ‘fit()’ method in Keras for 50 epochs. The ‘TimeseriesGenerator’ object generates batches of input/output pairs for the model to learn from. The ‘fit()’ method updates the model parameters using backpropagation based on the loss function and optimizer defined during model compilation. By training the model, it learns to make predictions on new, unseen data based on patterns learned in the training data.

```
model.fit(generator, epochs = 50)

plt.plot(model.history.history['loss'])

last_train_batch = scaled_train[-12:]

last_train_batch = last_train_batch.reshape(1, 12, 1)

last_train_batch

model.predict(last_train_batch)
```

This code uses the trained LSTM neural network model to make predictions on a new data point. The last 12 data points from the training data are selected, scaled, and reshaped into the appropriate format for the model. The ‘predict()’ method is called on the model with the reshaped data as input, and the output is the predicted value for the next time step in the time series. This is an essential step in using the LSTM model for time series forecasting.

```
scaled_test[0]
```

This code prints the first element of the scaled test data array. The ‘scaled_test’ variable is a NumPy array of the test data that has been transformed using the ‘MinMaxScaler’ object. Printing the first element of this array shows the scaled value for the first time step in the test data.

```
#Forecasting


y_pred = []

first_batch = scaled_train[-n_inputs:]
current_batch = first_batch.reshape(1, n_inputs, n_features)

for i in range(len(scaled_test)):
    batch = current_batch
    pred = model.predict(batch)[0]
    y_pred.append(pred)
    current_batch = np.append(current_batch[:,1:, :], [[pred]], axis = 1)


y_pred


scaled_test
```

This code generates predictions for the test data using the trained LSTM model. It uses a for loop to loop over each element in the scaled test data. In each iteration, the current batch is used to make a prediction using the 'predict()' method of the model. The predicted value is then added to the 'y_pred' list and the current batch is updated. Finally, the 'y_pred' list is printed along with the 'scaled_test' data to compare the predicted values with the actual values. This step is crucial in evaluating the performance of the LSTM model on the test data.

```
df_test

y_pred_transformed = scaler.inverse_transform(y_pred)

y_pred_transformed = np.round(y_pred_transformed,0)
```

```
y_pred_final = y_pred_transformed.astype(int)

y_pred_final
```

This code transforms the predicted values generated in the previous step back to the original scale using the ‘inverse_transform()’ method of the scaler object. The transformed values are rounded to the nearest integer using the ‘round()’ function and converted to integers using the ‘astype()’ method. The resulting array of predicted values, ‘y_pred_final’, is printed to show the final predicted values for the test data. This step is important for evaluating the accuracy of the LSTM model’s predictions on the original scale of the data.

```
df_test.values, y_pred_final

df_test['Predictions'] = y_pred_final

df_test
```

The code above shows the predicted values generated by the LSTM model being added to the original test dataset. First, the ‘values’ attribute is used to extract the values of the ‘df_test’ dataframe, which are then paired with the predicted values ‘y_pred_final’. Then, a new column called ‘Predictions’ is added to the ‘df_test’ dataframe to store the predicted values. Finally, the ‘df_test’ dataframe is printed with the newly added ‘Predictions’ column. This step is important to visually compare the actual values of the test dataset with the predicted values and evaluate the accuracy of the model.

```
plt.figure(figsize=(15, 6))
plt.plot(df_train.index, df_train.Passengers, linewidth=2, color='black',
label='Train Values')
plt.plot(df_test.index, df_test.Passengers, linewidth=2, color='green',
label='True Values')
plt.plot(df_test.index, df_test.Predictions, linewidth=2, color='red',
label='Predicted Values')
plt.legend()
plt.show()
```

This code block is generating a plot using the matplotlib library. It first sets the figure size, and then plots the training data as a black line, the true test values as a green line, and the predicted test values as a red line. It also adds a legend to the plot and displays it using the show() method.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

sqrt(mean_squared_error(df_test.Passengers, df_test.Predictions))
```

This code calculates the root mean squared error (RMSE) between the actual passenger values in the test set (df_test.Passengers) and the predicted passenger values (df_test.Predictions). RMSE is a commonly used metric to evaluate the performance of regression models. It measures the average distance between the predicted values and the actual values, taking into account the square of the differences between them. RMSE is a useful metric because it penalizes large errors more heavily than small errors, making it a good indicator of the overall accuracy of a model's predictions.

The model performed well with a root mean squared error of 30.5. The visualization of the true, predicted, and training values showed that the model was able to capture the overall trend and seasonality in the data. This demonstrates the power of LSTM in capturing complex temporal relationships in time series data and its potential for making accurate predictions.