Long Short-Term Memory (LSTM) Networks

PHASE 3 – DEVELOPMENT PART 1

LSTM is a type of recurrent neural network (RNN) architecture that has proven to be highly effective in handling sequences of data, making it particularly useful for time series forecasting, including electricity price prediction.

The working of LSTM is mentioned as the following,

1. **MEMORY CELLS** : Unlike traditional neural networks, LSTM networks have a more complex structure that includes memory cells. These cells can store information over long periods of time, allowing them to capture long-term dependencies in data.

2. **GATES** : LSTMs have three types of gates (input gate, forget gate, and output gate) that control the flow of information through the cell. These gates use sigmoid and tanh activation functions to regulate which information is passed through and which is discarded.

- **INPUT GATE** : Determines how much of the new information is stored in the cell.

- **FORGET GATE**: Controls which information should be discarded from the cell's memory.

- **OUTPUT GATE**: Determines what part of the cell's memory should be outputted.

3. **SEQUENTIAL PROCESSING** : LSTMs process data sequentially, which means they consider each element in the sequence one at a time. This allows them to capture complex temporal patterns in the data.

4. **BACK PROPAGATION THROUGH TIME (BPTT)** : LSTMs are trained using the backpropagation algorithm, which adjusts the weights of the network to minimize the prediction error. In the case of LSTMs, this process is extended over time to handle sequences.

**EFFECTS OF LSTM ON ELECTRICITY PRICES PREDICTION :**

- **CAPTURING LONG TERM DEPENDENCIES** : LSTMs are designed to capture dependencies over long sequences, which is crucial for modeling electricity price patterns influenced by various factors.

- **HANDLING SEASONING AND TRENDS :** LSTMs can automatically learn and adapt to seasonal variations, trends, and other complex patterns present in electricity price data.

- **DYNAMIC PERFORMANCE** : LSTMs are capable of adapting their internal state based on new information in the sequence, making them suitable for dynamic environments where patterns may change over time.

- **ROBUSTNESS TO IRREGULARITIES** : LSTMs can handle missing data or irregularly sampled time series, which is common in real-world datasets.

It's worth noting that while LSTMs are powerful, their effectiveness can be influenced by factors like data quality, feature engineering, and hyperparameter tuning. Therefore, careful preprocessing and experimentation are often crucial for achieving optimal results.

**CODING :**

The coding for electricity prices prediction using the LSTM (Long Short-Term Memory) algorithm. Here's a basic outline in Python using the TensorFlow and Keras libraries:

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

# Load your dataset

# Assuming you have a CSV file with a column 'price' representing the electricity prices

# Replace 'your\_dataset.csv' with the actual file path

data = pd.read\_csv('your\_dataset.csv')

prices = data['price'].values.reshape(-1, 1)

# Normalize the data

scaler = MinMaxScaler()

prices = scaler.fit\_transform(prices)

# Create sequences for training

sequence\_length = 10 # Adjust this based on your dataset and requirements

x, y = [], []

for i in range(len(prices) - sequence\_length):

X.append(prices[i:i+sequence\_length])

y.append(prices[i+sequence\_length])

X = np.array(X)

y = np.array(y)

# Split data into training and testing sets

train\_size = int(0.8 \* len(X))

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Build the LSTM model

model = Sequential([

LSTM(units=50, activation='relu', input\_shape=(sequence\_length, 1)),

Dense(units=1)

])

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

# Evaluate the model

loss = model.evaluate(X\_test, y\_test)

# Predict future prices

predicted\_prices = model.predict(X\_test)

# Inverse transform the predictions to get actual prices

predicted\_prices = scaler.inverse\_transform(predicted\_prices)