## **LSTM**

Reference: https://medium.com/@techwithjulles/recurrent-neural-networks-rnns-and-long-short-term-memory-lstm-creating-an-lstm-model-in-13c88b7736e2

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.preprocessing import MinMaxScaler
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import LSTM, Dense
In [ ]: # Load dataset
        df = pd.read_csv("coin_Bitcoin.csv") # Replace with your actual file name
        # Convert Date column to datetime and sort by date
        df['Date'] = pd.to_datetime(df['Date'])
        df.sort_values('Date', inplace=True)
        # Filter data for Bitcoin (BTC)
        df = df[df['Symbol'] == 'BTC']
        # Select 'Close' price for prediction
        data = df[['Close']].dropna()
        # Scale data using MinMaxScaler (0 to 1)
        scaler = MinMaxScaler()
        data_scaled = scaler.fit_transform(data)
        # Function to create sequences of past prices for LSTM input
        def create_sequences(data, seq_length):
            X, y = [], []
            for i in range(len(data) - seq_length):
                X.append(data[i:i + seq_length])
                y.append(data[i + seq_length])
            return np.array(X), np.array(y)
        # Define sequence Length (past 30 days of data)
        seq_length = 30
        X, y = create_sequences(data_scaled, seq_length)
        # Split into training and test sets (80-20 split)
        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:]
        # Reshape input to LSTM format (samples, timesteps, features)
        X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
        X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
        # Build LSTM model
        model = Sequential([
            LSTM(50, activation='relu', input_shape=(seq_length, 1)),
            Dense(1)
        ])
        # Compile model
        model.compile(optimizer='adam', loss='mse')
        # Train model
        history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.
        # Evaluate model
        loss = model.evaluate(X_test, y_test)
        print(f"Test Loss: {loss}")
        # Predict on test data
```

y\_pred = model.predict(X\_test)

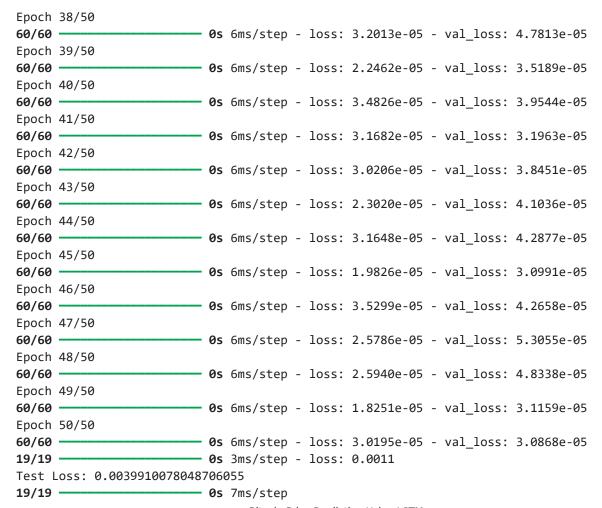
```
# Inverse transform predictions to original scale
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)

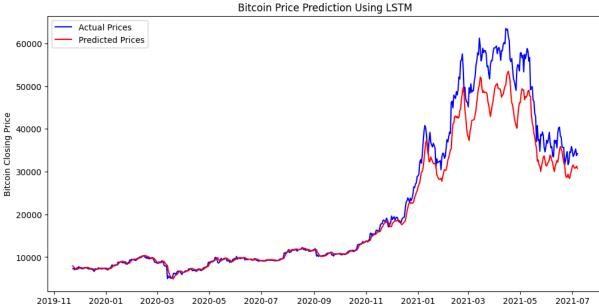
# Plot actual vs. predicted values
plt.figure(figsize=(12, 6))
plt.plot(df['Date'].iloc[-len(y_test):], y_test_inv, label="Actual Prices", color='plt.plot(df['Date'].iloc[-len(y_test):], y_pred_inv, label="Predicted Prices", coloplt.xlabel("Date")
plt.ylabel("Bitcoin Closing Price")
plt.title("Bitcoin Price Prediction Using LSTM")
plt.legend()
plt.show()
```

## Epoch 1/50

d:\SEM\_2\_SETU\ml\ML\_Algorithms\_shon\.venv\Lib\site-packages\keras\src\layers\rnn\rn
n.py:200: 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 lay
er in the model instead.
 super().\_\_init\_\_(\*\*kwargs)

<b>60/60</b> Epoch	2/50	2s	9ms/step	-	loss:	0.0016 - val_	_loss: 1.25	518e-04
60/60		0s	7ms/step	-	loss:	1.0166e-04 -	val_loss:	9.6208e-05
	3/50	4.	0		1	F 0000- 0F		0 2707 - 05
<b>60/60</b> Epoch		15	9ms/step	-	1055:	5.9898e-05 -	val_loss:	8.3/9/e-05
60/60		0s	7ms/step	-	loss:	8.6365e-05 -	<pre>val_loss:</pre>	9.5659e-05
Epoch <b>60/60</b>	5/50	0s	6ms/step	_	loss:	7.9265e-05 -	val loss:	7.5030e-05
Epoch	6/50		·				_	
<b>60/60</b> Epoch	7/50	0s	6ms/step	-	loss:	9.0848e-05 -	val_loss:	7.6648e-05
60/60		0s	6ms/step	-	loss:	6.0278e-05 -	val_loss:	7.7550e-05
Epoch <b>60/60</b>		0s	6ms/step	_	loss:	5.2126e-05 -	val loss:	6.6621e-05
Epoch	9/50							
	10/50	0s	6ms/step	-	loss:	5.8266e-05 -	val_loss:	6.2593e-05
60/60		0s	6ms/step	-	loss:	5.1379e-05 -	<pre>val_loss:</pre>	6.5979e-05
•	11/50	0s	6ms/step	_	loss:	5.5722e-05 -	val_loss:	9.0798e-05
•	12/50	0-			1	4 7471 - 05		7 7545 - 05
	13/50	05	oms/step	-	1055:	4.7471e-05 -	va1_10SS:	7.7545e-05
	14/50	0s	6ms/step	-	loss:	4.6370e-05 -	val_loss:	5.7879e-05
		0s	6ms/step	-	loss:	4.3920e-05 -	val_loss:	4.9284e-05
Epoch <b>60/60</b>	15/50	95	6ms/sten	_	loss:	4.9304e-05 -	val loss:	5.3675e-05
Epoch	16/50							
	 17/50	0s	6ms/step	-	loss:	3.1572e-05 -	val_loss:	5.1680e-05
60/60		0s	6ms/step	-	loss:	3.4439e-05 -	<pre>val_loss:</pre>	7.1594e-05
•	18/50 ——————	0s	6ms/step	_	loss:	4.8687e-05 -	val_loss:	5.3579e-05
	19/50	00	Ems/ston		10001	2 26200 05	val loss:	4 26620 BE
Epoch	20/50							
	21/50	0s	6ms/step	-	loss:	3.0847e-05 -	val_loss:	4.2904e-05
60/60		0s	6ms/step	-	loss:	2.8515e-05 -	val_loss:	4.8903e-05
	22/50	0s	6ms/step	_	loss:	4.1565e-05 -	val_loss:	4.3432e-05
•	23/50	00	Ems/ston		10001	2.9455e-05 -	val loss:	4 2261a AF
Epoch	24/50	03	ollis/step	-	1055.	2.94336-03 -	va1_1055.	4.33016-03
	25/50	0s	6ms/step	-	loss:	3.5904e-05 -	val_loss:	4.0590e-05
60/60		0s	6ms/step	-	loss:	2.7253e-05 -	val_loss:	3.9369e-05
	26/50 	0s	6ms/step	_	loss:	2.7592e-05 -	val loss:	4.0000e-05
Epoch	27/50							
	28/50					2.5348e-05 -		
		0s	6ms/step	-	loss:	3.6410e-05 -	val_loss:	5.7197e-05
•	29/50	0s	6ms/step	-	loss:	3.2339e-05 -	val_loss:	3.6141e-05
	30/50	۵s	6ms/sten	_	lossi	3 27040-05 -	val loss:	3 59290-05
Epoch	31/50							
	32/50	0s	6ms/step	-	loss:	2.4707e-05 -	val_loss:	4.1802e-05
60/60		0s	6ms/step	-	loss:	2.3553e-05 -	val_loss:	4.6998e-05
•	33/50	0s	6ms/step	-	loss:	2.2645e-05 -	val_loss:	6.4497e-05
	34/50		·				_	
Epoch	35/50							
	36/50	0s	6ms/step	-	loss:	2.6562e-05 -	val_loss:	3.4977e-05
60/60		0s	6ms/step	-	loss:	2.6623e-05 -	val_loss:	3.4274e-05
Epoch <b>60/60</b>		0s	7ms/sten	_	loss:	2.2995e-05 -	val loss:	4.4249e-05
,			-, P					





Date