VIT BANGALORE

Name : **DHARSHANI ANANDKUMAR**

Reg. No : **23MSP3068**

Project Title : **WEATHER PREDICTION USING RNN**

Faculty Name : **Prof.Ramya M**

Academic year: **2023-2024**

Index

Problem Statement:

This project aims to develop a deep learning model using Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network, to predict daily maximum temperatures in Seattle accurately. The need for precise weather forecasting is critical in sectors such as agriculture and urban planning, where accurate weather data can significantly influence decision-making processes.

The project addresses several challenges including data integrity, overfitting, and generalization to unseen data. Advanced techniques are employed to enhance the model's performance and robustness. Dropout regularization is used to prevent overfitting by randomly omitting subsets of features during training. Early stopping is implemented to halt training when validation performance no longer improves, ensuring the model does not over-train. Additionally, model checkpoints are utilized to save the state of the model periodically, allowing the recovery of the most effective model configuration observed during training.

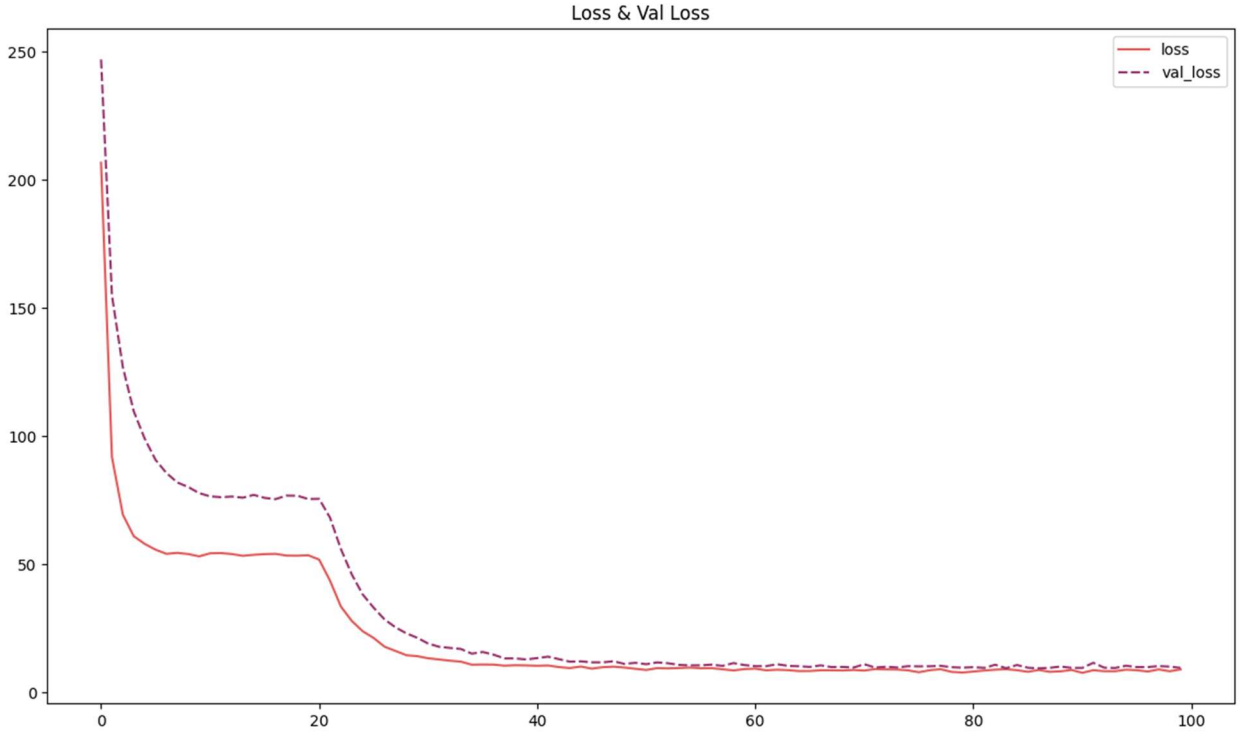
By incorporating these techniques, the project not only aims to achieve high accuracy in temperature predictions but also serves as a platform to explore the effectiveness of LSTM networks in weather prediction. This approach helps improve the reliability and utility of predictive weather models, ultimately supporting better-informed strategic planning in weather-sensitive sectors.

Code :

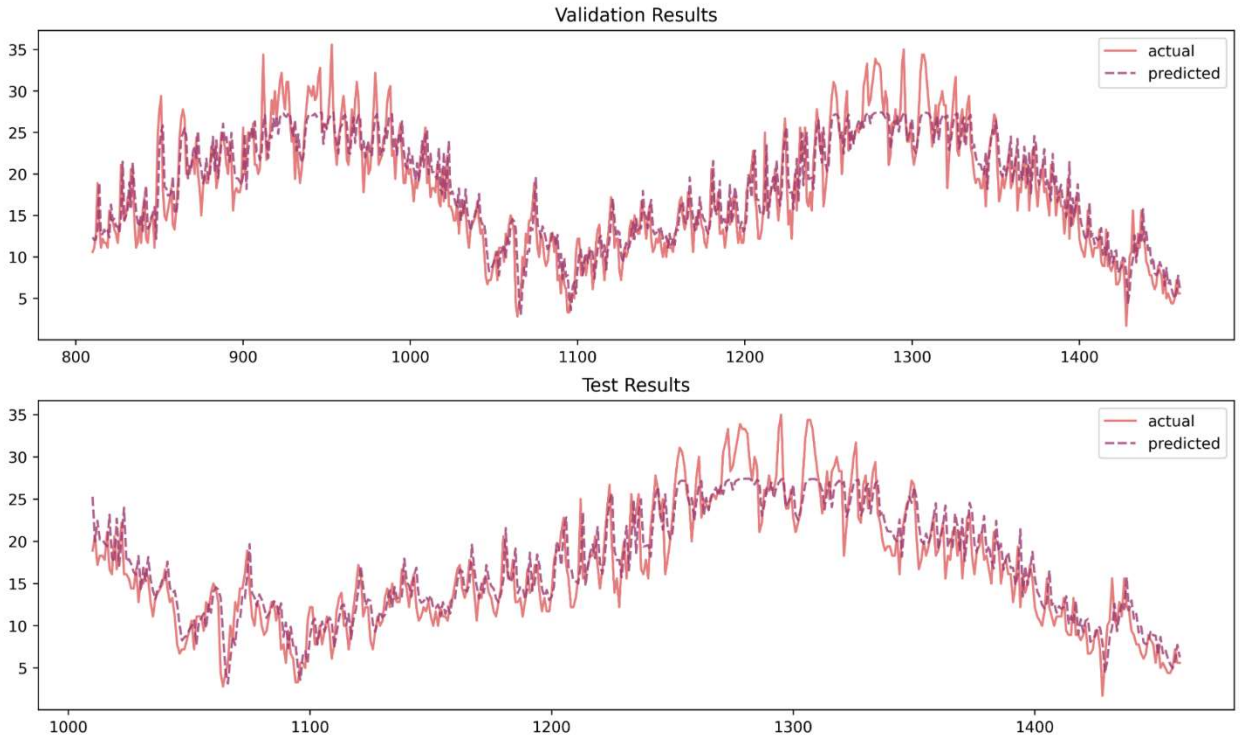
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('/content/seattle-weather.csv')

df

df.isnull().sum()

df.duplicated().sum()

#column temp_max converted into numpy array
training_set = df.iloc[:,2:3].values
training_set

len(training_set)

def df_to_XY(df,window_size=10):
 X_train=[]
 y_train=[]

 for i in range(10,len(training_set)):
    X_train.append(training_set[i-10:i,0])
    y_train.append(training_set[i,0])

 X_train, y_train = np.array(X_train), np.array(y_train)
 return X_train, y_train

WINDOW = 10
X,y = df_to_XY(df,WINDOW)
print(len(X),len(y))
X_train = X[:800]
y_train = y[:800]
X_val = X[800:1000]
y_val = y[800:1000]
X_test = X[1000:]
x_test = y[1000:]

#Reshaping(To add new dimensions)
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
X_val = np.reshape(X_val,(X_val.shape[0],X_val.shape[1],1))
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))


#Building the RNN
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

regressor = Sequential()

#Addinf the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units=50, return_sequences = True, input_shape=(X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return_sequences = True))
regressor.add(Dropout(0.2))
```

```
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))

#Output layer
regressor.add(Dense(units=1))

#Compiling
regressor.compile(optimizer='adam',loss='mean_squared_error')

from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam


#fitting the rnn to the training set
history=regressor.fit(X_train,y_train,validation_data=(X_val,y_val),epochs=100, batch_size=32)

his = pd.DataFrame(history.history)

his

import seaborn as sns
his.columns
history_loss = his[['loss', 'val_loss']]

plt.figure(figsize=(14, 8))
plt.title("Loss & Val Loss")
sns.lineplot(data=history_loss, palette="flare")

plt.show()

train_pred = regressor.predict(X_train).flatten()
val_pred = regressor.predict(X_val).flatten()
test_pred = regressor.predict(X_test).flatten()

pred = np.concatenate([train_pred,val_pred,test_pred])
df_pred = pd.DataFrame(df["temp_max"].copy())
df_pred.columns=["actual"]
df_pred = df_pred[WINDOW:]
df_pred["predicted"] = pred

fig,axes = plt.subplots(2,1,figsize=(14,8),dpi=400)

plt.subplot(2,1,1)
plt.title("Validation Results")
sns.lineplot(df_pred[800:],alpha=0.8,palette="flare",linestyle=None)

plt.subplot(2,1,2)
plt.title("Test Results")
sns.lineplot(df_pred[1000:],alpha=0.8,palette="flare",linestyle=None)
```

Results :



The graph displays the training loss (solid line) and validation loss (dashed line) over 100 epochs. Initially, both losses decrease sharply, indicating rapid learning. After the initial drop, both lines show smaller fluctuations and stabilize, with the validation loss closely tracking the training loss, suggesting the model generalizes well without overfitting.



Correlation Between Actual and Predicted Values:
Both plots show a good correlation between actual and predicted values, which suggests that the model used for these predictions has a decent performance. The predicted values follow the trends and shifts in the actual values quite closely.
Performance on Validation vs. Test Set:
Validation Results: The match between the actual and predicted data in the validation set appears very tight, which indicates effective learning from the training data without significant overfitting at this stage.
Test Results: The predictions in the test set also closely follow the actual data but show a slightly greater deviation than in the validation results. This is common as the test data may include patterns or anomalies not fully captured during training.

Downward Trend: Both sets of results show a notable downward trend towards the end of the range, and the model predicts this trend reasonably well. This might indicate a seasonal change or a time-related trend in the weather parameter being predicted.

Conclusion:

The deep learning project successfully implemented an LSTM-based Recurrent Neural Network to forecast Seattle's daily maximum temperatures using historical weather data. Through rigorous preprocessing, including handling missing values and duplicates, the model's data integrity was meticulously maintained. The architecture featured several LSTM layers incorporated with dropout techniques to enforce regularization, significantly reducing the risk of overfitting, as reflected in the declining trends of both training and validation loss across 100 epochs.

The model demonstrated its effectiveness in capturing temporal patterns and yielded promising predictive accuracy on both training and validation sets. However, discrepancies observed in the test set predictions suggest a potential for further refinement to enhance model performance. Adjustments such as hyperparameter tuning, experimenting with additional layers, or exploring alternative activation functions could potentially improve the model's accuracy and generalization capability.

This project not only illustrates the capabilities of LSTM networks in accurately modeling time-series data but also underscores their potential applicability in real-world scenarios. Enhanced predictive accuracy in weather forecasting could significantly benefit sectors such as agriculture and urban planning, where precise weather information is crucial for operational and strategic decision-making. Moving forward, integrating additional environmental variables and extending the model to multi-variate forecasts could offer deeper insights and improved accuracy, further pushing the boundaries of what modern RNNs can achieve in complex forecasting tasks. This project sets a foundational framework for future explorations and advancements in the field of neural network-based weather prediction.