RNN/LSTM

Sequence prediction – feed output back into itself as input.

Vanishing Gradients impact back calc. of weights & biases. ReLU activation function, batch normalization, choose different weight initialization help alleviate these issues.

Exploding Gradients – gradient clipping. These all slow down training time.

LSTM & GRU Units – Fix these issues.

**RNN:**

1. Test Train Split (take percentage of data from end of data set)
   1. test\_percent = ?
   2. test\_point = np.round(len(df) \* test\_percent)
   3. test\_index = int((len(df) – test\_point)
   4. train = df.iloc[:test\_ind]
   5. test = df.iloc[test\_ind:]
2. Scale
   1. from sklearn.preprocessing import minmaxscaler
   2. scaler = minmaxscaler()
   3. scaler.fit(train)
   4. scaled\_train = scaler.transform(train)
   5. scaled\_test = scaler.transform(test)
3. Create Batches
   1. from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
   2. length = 2 (long enough to determine a cycle/seanality, (eg.,year)
   3. batch\_size = 1
   4. generator = timeseriesgenerator(scaled\_train, scaled\_train, length=length, batch\_size=batch\_size)
   5. print(len(scaled\_train)), print(len(generator)), x,y = generator[0], print(x), print(y)
4. Create Model
   1. from tensorflow.keras.models import sequential
   2. from tensorflow.keras.layers import dense, simplernn, lstm
   3. n\_features = ?
   4. model = sequential()
   5. number\_of\_neurons = ? (related to batch size)
   6. model.add(simplernn(number\_of\_neurons, input\_shape=(length, n\_featrues)))
   7. model.add(dense(1))
   8. model.compile(optimizer=’adam’, loss=’mse’)
   9. print(model.summary())
   10. model.fit\_generator(generator, epochs=5)
5. Evaluate Losses
   1. losses = pd.dataframe(model.history.history)
   2. losses.plot()
6. Prediction
   1. print(model.predict(first\_eval\_batch))
   2. print(scaled\_test)
   3. test\_predictions = []
   4. first\_eval\_batch = scaled\_train[-length:]
   5. current\_batch = first\_eval\_batch.reshape((1, length, n\_features))
   6. for i in range(len(test)):
      1. current\_pred = model.predict(current\_batch)[0]
      2. test\_predictions.append(current\_pred)
      3. current\_batch = np.append(current\_batch[:, 1:, :], [[current\_pred]], axis=1)
   7. print(test\_predictions)
   8. print(scaled\_test)
   9. true\_predictions = scaler.inverse\_transform(test\_predictions)
   10. print(true\_predictions)
   11. test[‘predictions’] = true\_predictions
   12. test.plot(figsize=(12, 8))
7. Add EarlyStopping
   1. from tensorflow.keras.callbacks import earlystopping
   2. early\_stop = earlystopping(monitor=’val\_loss’, patience=2)
   3. validation\_generator = timeseriesgenerator
   4. length = 49
   5. generator = timeseriesgenerator(scaled\_train,scaled\_train, length=length,batch\_size=1)
   6. validation\_generator = timeseriesgenerator(scaled\_test,scaled\_test, length=length,batch\_size=1)

**LSTM:**

1. Create model
   1. model = Sequential()
   2. model.add(LSTM(50,input\_shape=(length, n\_features)))
   3. model.add(Dense(1))
   4. model.compile(optimizer='adam', loss='mse')
   5. model.fit\_generator(generator,epochs=20, validation\_data=validation\_generator,callbacks=[early\_stop])
2. Test
   1. test\_predictions = []
   2. first\_eval\_batch = scaled\_train[-length:]
   3. current\_batch = first\_eval\_batch.reshape((1, length, n\_features))
   4. for i in range(len(test)):
   5. current\_pred = model.predict(current\_batch)[0]
   6. test\_predictions.append(current\_pred)
   7. current\_batch = np.append(current\_batch[:,1:,:],[[current\_pred]],axis=1)
   8. true\_predictions = scaler.inverse\_transform(test\_predictions)
   9. test['LSTM Predictions'] = true\_predictions
   10. test.plot(figsize=(12,8))
3. Forecast (use entire dataset)
   1. full\_scaler = MinMaxScaler()
   2. scaled\_full\_data = full\_scaler.fit\_transform(df)
   3. generator = TimeseriesGenerator(scaled\_full\_data, scaled\_full\_data, length=length, batch\_size=1)
   4. model = Sequential()
   5. model.add(LSTM(50, input\_shape=(length, n\_features)))
   6. model.add(Dense(1))
   7. model.compile(optimizer='adam', loss='mse')
   8. model.fit\_generator(generator,epochs=6)
   9. forecast = []
   10. first\_eval\_batch = scaled\_full\_data[-length:]
   11. current\_batch = first\_eval\_batch.reshape((1, length, n\_features))
   12. for i in range(len(?)):
   13. current\_pred = model.predict(current\_batch)[0]
   14. forecast.append(current\_pred)
   15. current\_batch = np.append(current\_batch[:,1:,:],[[current\_pred]],axis=1)
   16. forecast = scaler.inverse\_transform(forecast)
   17. print(df), print(len(forecast))
   18. forecast\_index = np.arange(50.1,55.1,step=0.1)
   19. plt.plot(df.index,df['Sine'])
   20. plt.plot(forecast\_index,forecast)

**NOTES:**

* After each prediction cycle (e,g,, 1 week)/7 days) include missing actuals in training data and retrain.