CIS600 EVOLUTIONARY MACHINE LEARMNING

Homework2

Abstract

This report includes summarization of homework2 implementation.

Anagha Dinesh Fatale

SUID438039600

- ## Assignment:

To apply Evolution Strategies (or Particle Swarm Optimization) to a neural network model for a prediction task using LSTM

- ## Data Set:

I have used International airlines passenger's data.
I have created 40% of data as test data and 60% as training data using train_test_split() method. This database consists of year and number of passengers.

- ## Recurrent Neural Network:

Long ShortTerm Memory neural network is implemented using Keras.

```
model_s = build_model(LOSS)

#build_model function is called#
#Sequential model for neural network#
model = Sequential()

#Layers are added to RNN model with 1 input neurons, LSTM layer with 4 neurons and 1 neuron as output layer #
model.add(LSTM(4, input_shape = (1, window_size)))
model.add(Dense(1))
model.compile(loss = 'mse', optimizer = "sgd" )

#Evaluation of LSTM Model#
model_s.fit(x_train, y_train,epochs = 10, batch_size = 8,verbose = 2)
scores = model_s.evaluate(x_train, y_train, batch_size=BATCH_SIZE, verbose=0)

#Prediction is performed on x_train data#
pred = scaler.inverse_transform(model_s.predict(x_train))
orig_data = scaler.inverse_transform([y_train])

#Root Mean Squared Error metric is calculated for loss#
train_score = math.sqrt(mean_squared_error(orig_data[0], pred[:, 0]))
```

RMSE Score without pyswarm Optimization is: 56.98

- ## Pyswarm Optimization Steps:

The fit function for PSO is called as below:

```
for i, p in enumerate(self.particles):
         local_score = p.get_score(x, y)
            if local_score < self.global_best_score:
                self.global_best_score = local_score
                self.global_best_weights = p.get_best_weights()
         print("PSO -- Initial best score :")
        print(self.global_best_score)
        bar = ProgressBar(steps, updates=20)

        for i in range(steps):
            for j in range(num_batches):
```

```
          x_ = x[j*batch_size:(j+1)*batch_size,:]
          y_ = y[j*batch_size:(j+1)*batch_size]

          for p in self.particles:
              local_score = p.step(x_, y_, self.global_best_weights)
              #print("....",local_score)
              if local_score < self.global_best_score:
                  self.global_best_score = local_score
                  self.global_best_weights = p.get_best_weights()
```

1. For Pyswarm Optimization, the initially the local score that is the best score is obtained for given LSTM, that is 94.64.
2. For given every step, for every particle the best score is calculated as above.
3. Then p.get_score() function is called which gets initial weights of the model and find the best -weights for the model and returns best local score model.

```
def get_score(self, x, y, update=True):
        print(self.model.metrics_names)
        #5 local_score = self.model.evaluate(x, y, verbose=0)
        pred = scaler.inverse_transform(self.model.predict(x))
        # print(pred[:0])
        # Prepare Y data to also be on the original scale for interpretability.
        orig_data = scaler.inverse_transform([y])
        # print(orig_data[0])
        # Calculate RMSE.
        local_score = math.sqrt(mean_squared_error(orig_data[0], pred[:, 0]))
        print("Local RMSE score:",local_score)
        if local_score < self.best_score and update:
            self.best_score = local_score
            self.best_weights = self.model.get_weights()

        return local_score
```

4. Then new model is build using this score then it is compiled, and values are predicted.
5. The new RMSE is calculated for current model (that is the score for this model) and then it is compared with best_score.
6. This value is then returned to optimizer.

- **Output:**

```
RMSE Score without pyswarm Optimization is: 56.980781835071234

PSO -- Initial best score:
94.64

RMSE Score with pyswarm Optimization is: 22.17
```

• Comparison of results:

	Keras (Backpropagation)	**Genetic Algorithm**
RMSE	56.98	22.17
Efforts Made	1. To apply different activation and optimization functions. 2. LSTM with different 3. Optimizer='standard gradient deviation' found to produce best results. 4. Implemented different number of epochs and batch sizes to get results.	1. To reduce RSME, different particles and number of steps were implemented. 2. Total particles = 30 and total steps = 10 found best to get results. 3. I have used batch_size = 32. 4. The acceleration=0.1, local_rate=1.0, global_rate=1.0 are paramters used for Pyswarm optimization

• Summary:

Hence, Recurrent Neural Network when it is optimized by PySwarm provided with less loss (root mean squared error) compared to Recurrent Neural Network model without PySwarm optimization.

• References:

1. https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks
2. https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
3. https://github.com/mike-holcomb/PSOkeras/blob/master/example.py
4. https://github.com/mike-holcomb/PSOkeras/tree/master/psokeras
5. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
6. http://aqibsaeed.github.io/2017-08-11-genetic-algorithm-for-optimizing-rnn/