Deep Learning 2020 **(Due: 2 Dec 2020)**

Submission Assignment #4

Instructor: Jakub Tomczak *Name:* Asif Anwar, *Netid:* aar216

1 Introduction

*For this paper **Task A: Sentiment classification** from assignment 4 has been completed. Considering the page limitation, all the tasks are implemented in a single hypothesis. In the Methodology section each parts of **Task A** have been explained in details. Three Jupiter notebooks are shared as the code along with gist link for easier access which can be found in appendix section.*

2 Problem statement

Hypothesis: *The **mean accuracy** on validation dataset for **LSTM**, is the heights, followed by the **Elman** network, followed by the **MLP**, given the best hyperparameter for each network.*

H0: $\mu_{LSTM} = \mu_{Elman} = \mu_{MLP}$, **HA:** $\mu_{LSTM} > \mu_{Elman} > \mu_{MLP}$

3 Methodology

Overview: *The methodology of given four parts of the assignments are describe below.*

Loading the data: *The IMDB dataset has been imported. It was found that the minimum length of data is 10 and maximum length is 2514 where the mean length is 240. Considering there are lot of outliers in the dataset a **Variable length Dataloader** has been introduced. The data loader is designed to supply each batch with different length and size of mini batch to reduce extra padding.*

Non-recurrent model: *The non recurrent network consists of one embedding layer and two linear layers and Relu activation function is used.*

Elman RNN: *A new class Elman was create to implement this recurrent network. An instance of the class was introduced as an layer of the MLP network. An object 'hidden' was created to reserve short time weights of the layer which used for training the elman network.*

Torch RNN: *The Network was on top of the MLP network, where between linear layers pytorch recurrent layers were introduced.*

4 Experiments

Non-reccurent model: *For training this network the custom Dataloader has been used with variable batch size and length, the starting batch size is 64. The model was trained with adam optimizer with learning rate of 0.001 for 3 epochs. Cross entropy loss function is used for the training. And every 20 batches training and validation loss and accuracy was observed.*

Elman RNN: *For training this network the custom Dataloader was modified little to give same batch size while changing the length of each batch as needed. The fixed batch sized was needed to keep the 'hidden context layer' same. The batch size was 100. The model was trained with adam optimizer with learning rate of 0.001. Due to slow training process of the network, the training kept to only 1 epoch.*

Torch RNN: *This network is also training with custom dataloader with variable batch lenght. Even This network can process larger batch size, to keep the reading similar the batch size and step size was kept same as Elman Network. The adam optimizer with 0.001 learning rate was used for this network. And total 3 epochs were used to train the network.*

5 Results and discussion

Non-reccurent model: *The validation accuracy, $\mu_{MLP} = 63.44\%$ for this network. In the figure 1, we can notice that the both training and validation reduces during the training. however the reduction rate is very slow and there are lots of fluctuation. The training accuracy also has fluctuation but some times it reaches to 100%. But in the validation accuracy has less fluctuation and gradually reaches over 60% mark*

Elman RNN *The Elman network performs significantly better than the MLP network. After adding the Elman layer, the validation accuracy increased from 63.44% to 81.16%. In figure 3. we can observed that the training loss gradually reduces from 0.6 to 0.4 and training accuracy fluctuate and get stable at 80%. Note: both training loss and accuracy begin with zero in the graph, which is due to in accurate reading in step 1.*

Torch RNN: *The LSTM network using pytorch default function performs the best over validation dataset. The validation accuracy is 84.1% which is almost 3% higher than Elman network and 20% higher than MLP network. In figure 3, it can be noticed the both training and validating accuracy has more smoother upward trend than any other network. And the training time is also less for this network than other two networks.*

Test Results: *Based on the experiment its been evident that LSTM perform best on the validation dataset. Hence the test dataset was loaded and train over LSTM dataset for 3 epochs with same hyperparameters. Before the training the model provides 50.01% base accuracy and after the training we see that the model able to predict 84.55%.*

6 Conclusion

From the above experiments we observed that the validation accuracy of the MLP, Elman and LSTM are 63.44%, 81.16% and 84.1%. Hence we can rn reject the null hypothesis basis on this data and confirm that HA: $\mu_{LSTM} > \mu_{Elman} > \mu_{MLP}$. That means the mean accuracy on validation dataset for LSTMis the heights followed by Elman, followed by MLP.

We can observed that the LSTM performs quite good over the test data provided the tuned hyperparameters of the network. The test mean accuracy of the network raised to 84.1% from the baseline accuracy of 50.01%

Note: *The significant level can be tested to prove the hypothesis based on p_value testing for each mean in future analysis.*

Appendix

A code snippet

6.1 Variable Dataloader

```
def varLenLoader(X, y, batch_size):
  X = np.array(X)
  total_len = len(X)
  bin = total_len*0.1
  decay = int(total_len/bin)
  for i in np.arange(0, X.shape[0], batch_size):
    x_temp = X[i:i + batch_size]
    y_mini = y[i:i + batch_size]

    ln = np.array([len(x) for x in x_temp])
    batch_length = ln.max()
    x_min = []
    for x in x_temp:
```

```
      leng = len(x)
      if leng < batch_length:
        x_min.append(x+[w2i['.pad'] for x in range(batch_length - leng)])
      else: x_min.append(x)
    x_min = torch.tensor(x_min, dtype=torch.long)
    y_mini = torch.tensor(y_mini)
    yield (x_min, y_mini)

    if i % decay == 0:
      if batch_size > 4:
        batch_size -= 1
      else:
        batch_size == 4
```

6.2 Non-recurrent model

```
class NonRNN(nn.Module):
  def __init__(self, num_embeddings, embedding_dim=300):
    super(NonRNN, self).__init__()
    self.embedding = nn.Embedding(num_embeddings, embedding_dim)
    self.fc1 = nn.Linear(300, 300)
    self.fc2 = nn.Linear(300, 2)

  def forward(self, sentence):
    x = self.embedding(sentence)
    x = F.relu(self.fc1(x))
    x = torch.max(x, 1).values
    x = self.fc2(x)
    return x
```

6.3 Elman Network

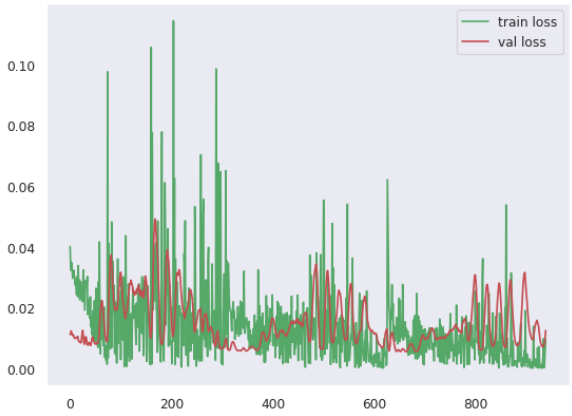
```
class Elman(nn.Module):
  def __init__(self, i_size, hsize, o_size):
    super(Elman, self).__init__()
    self.fc1 = nn.Linear(i_size+hsize, hsize)
    self.fc2 = nn.Linear(hsize, o_size)
  def forward(self, x, hidden=None):
    b,t,e = x.size()
    if hidden is None:
      hidden = torch.autograd.Variable(torch.zeros((b, e)).type(torch.FloatTensor),
    requires_grad=True)
    outs = []
    for i in range(t):
      inp = torch.cat([x[:,i,:], hidden], dim=1)
      hidden = torch.tanh(self.fc1(inp))
      out = self.fc2(hidden)
      outs.append(out[:, None, :])
    return torch.cat(outs, dim=1), hidden
```

B Codes

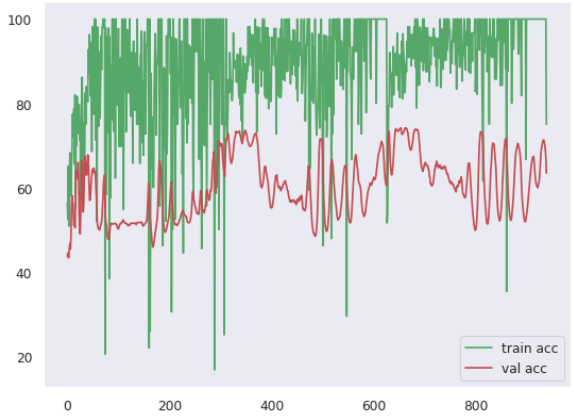
The Jupiter Notebook of the experiment is published in the below GIST for easier viewing.

- *https://colab.research.google.com/gist/tamal2000/4a2d293be9984dea49228258d6f9b20f/part_1_assignment_4.ipynb*
- https://colab.research.google.com/gist/tamal2000/357e691fb7ad75327b5763796b4b70e7/part-2-elman.ipynb
- https://colab.research.google.com/gist/tamal2000/09f0d9049bb401c57c95e5d26e6250f1/part3-lstm.ipynb
- https://colab.research.google.com/gist/tamal2000/42709436538ff4ca760d61859ba1e1e5/part4-lstm-test.ipynb

C Charts and graphs



(a) Training and validation loss



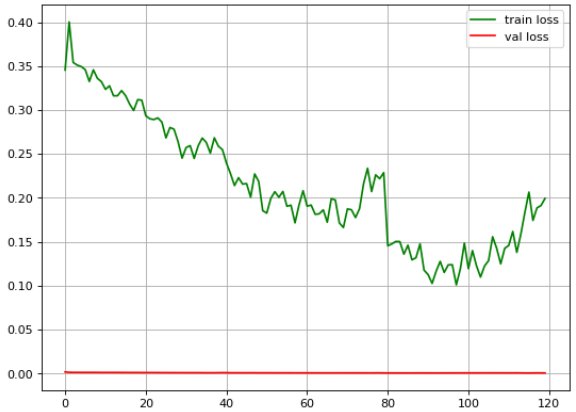
(b) Training and validation accuracy

Figure 1: Non recurrent MLP

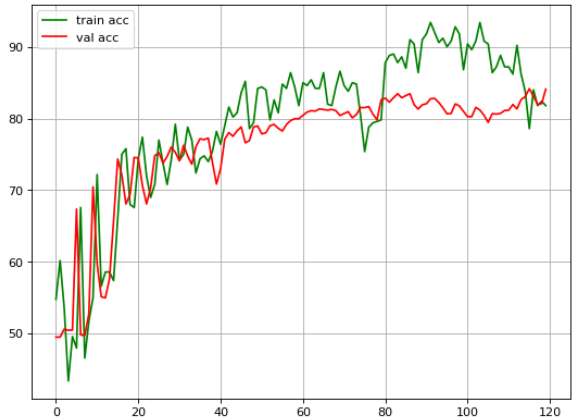
(a) Training Loss

(b) Training Acc

Figure 2: Elman recurrent network



(a) Loss per 20 batches



(b) Accuracy per 20 batches

Figure 3: LSTM recurrent network