

## IST 597 Assignment – 5

Vaibhav Singh

PSU ID – 997500644

Access ID – vxs5308

### **RNN and LSTM**

We know that in general LSTM takes more execution time and provide more accurate results. Keeping this in mind, I have tried to verify this behavior using the given models.

Initially, I have split the training data into train set and validation set in an 80/20 split I have used `dataset.take()` and `dataset.skip()` for this purpose.

My models were overfitting a lot as could be seen from very high train set accuracies(99%) and low validation and test set accuracies(85%).

To add an element of regularization, I have added the functionality to add a dropout layer during training (with `keep_prop = 0.8`) and created model instances accordingly.

Following are some results that I obtained –

I had to settle down to a batch size of 1024 due to the high execution times for smaller batches. Hence, I couldn't document performance for smaller batch sizes.

Following are the models that I have experimented with –

Optimizer	RNN or LSTM?	Hyperparameters
Adam	LSTM	<code>lr = 1e - 3</code>
Adam	RNN	<code>lr = 1e - 3</code>
RMSProp	LSTM	<code>lr = 0.01</code>
Adam	LSTM	<code>Keep_prob = 0.8</code>
Adam	RNN	<code>Keep_prob = 0.8</code>

## Model performance - Speed

Following are the execution times seen for various models –

```
[115] time_taken_lstm_adam
      319.1764781475067

[116] time_taken_lstm_dropout_adam
      358.74462509155273

[117] time_taken_lstm_rmsprop
      399.9376394748688

▶ time_taken_lstm_dropout_adam
      358.74462509155273

▶ time_taken_ugrnn_adam
  □→ 211.75835967063904

[120] time_taken_ugrnn_dropout_adam
      190.24637842178345
```

As expected, LSTM takes much more time to execute when compared to RNN models and LSTM with RMSprop tends to perform the slowest even with a learning rate of 0.01 when compared with Adam with learning rate =  $1e-3$ . Adding dropout tends to slow down LSTM even further.

## Model performance - Test set accuracies

In terms of overall test set accuracies, following are the results obtained for these models -

```
✓ [126] keys = all_models.keys()
0s      for key in keys:
          print (f"{key}: {list(all_models[key].history['test_acc'])[0]}")

      lstm_model_adam: 0.8485599756240845
      ugrnn_model_adam: 0.8241599798202515
      rmsprop_lstm_model: 0.7240399718284607
      ugrnn_dropout_adam_model: 0.8343200087547302
      lstm_dropout_adam_model: 0.8428400158882141
```

As expected, LSTM delivers better accuracy than RNN and doesn't seem to work well with RMSProp. With better hyperparameter tuning, we would be able to get even better test set accuracies.

## Model stability – Training accuracies

```
▶ keys = all_models.keys()
  for key in keys:
    print ([f"{key}: {list(all_models[key].history['train_acc'])}"])

lstm_model_adam: [0.837, 0.9202, 0.96795, 0.98885, 0.99365, 0.9967, 0.999, 0.99955]
ugrnn_model_adam: [0.70445, 0.89745, 0.9148, 0.9595, 0.98175, 0.99255, 0.99665, 0.99755, 0.9987, 0.99915]
rmsprop_lstm_model: [0.65705, 0.8507, 0.82865, 0.9082, 0.95375, 0.9807, 0.98685, 0.99435, 0.9962, 0.998]
ugrnn_dropout_adam_model: [0.7711, 0.88455, 0.94675, 0.97605, 0.98785, 0.99305, 0.99555, 0.979, 0.98995]
lstm_dropout_adam_model: [0.63625, 0.88955, 0.94175, 0.97785, 0.99005, 0.9949, 0.9962, 0.99875, 0.99875]
```

LSTM with dropout tends to achieve high train set accuracies in fewer epochs and similar results are seen for validation set as well. With RMSProp, the validation set accuracies are inconsistent and low.

Dropout regularization provides slight improvement in terms of overall accuracy of models.

### Interpretability

In terms of accuracy of sentiment classification, LSTM with Adam tends to provide more accurate results for different reviews. Similarly, as expected RMSProp tends to perform worse sentiment classification.

For example –

Apart from the reviews that are already there in the notebook, I have added following two reviews for analysis –

- "Hailed as a classic, but oddly unengaging and Nicholson doesn't help"
- "The movie was meh"

```
new_reviews_1 = [review_score_11,review_score_21]
scores_1 = [1,2]
with tf.device(device):
    for original_review, score in zip(new_reviews_1, scores_1):
        print (f"Testing review : {original_review}")
        for key in keys:
            indexed_review, seq_length = process_new_review(original_review)
            indexed_review = tf.reshape(tf.constant(indexed_review), (1,-1))
            seq_length = tf.reshape(tf.constant(seq_length), (1,))
            logits = all_models[key].predict(indexed_review, seq_length, False)
            pred = tf.argmax(logits, axis=1).numpy()[0]
            print('Model : %s - sentiment : %s'
                  %(key, sent_dict[pred]))
        print()
```

```
Testing review : Hailed as a classic, but oddly unengaging and Nicholson doesn't help
Model : lstm_model_adam - sentiment : negative
Model : ugrnn_model_adam - sentiment : positive
Model : rmsprop_lstm_model - sentiment : positive
Model : ugrnn_dropout_adam_model - sentiment : positive
Model : lstm_dropout_adam_model - sentiment : positive

Testing review : The movie was meh.
Model : lstm_model_adam - sentiment : negative
Model : ugrnn_model_adam - sentiment : negative
Model : rmsprop_lstm_model - sentiment : positive
Model : ugrnn_dropout_adam_model - sentiment : negative
Model : lstm_dropout_adam_model - sentiment : negative
```

For review - "Hailed as a classic, but oddly unengaging and Nicholson doesn't help"

- Ideally this should be classified as a negative review.

Similarly, for review - "The movie was meh"

- This should be classified as negative sentiment

So, in general default LSTM model gives better results. But, the consistency is still low.

Overall, with access to better compute strength and accuracy being a priority, we should choose LSTM models with Adam.

### Problems faced

One major problem was that we needed to upload all files every-time colab disconnected. Apart from that, it was very difficult to try smaller batch sizes due to the impact on execution times.

## Transformers

I am using the following suggestion as reference for my approach –

*“The suggestion is to start with a Small BERT (with fewer parameters) since they are faster to fine-tune. If you like a small model but with higher accuracy, ALBERT might be your next option. If you want even better accuracy, choose one of the classic BERT sizes or their recent refinements like Electra, Talking Heads, or a BERT Expert.”*

Apart from that, I am using AdamW and trying to learn more about various pre-trained BERT models as AdamW is the recommended optimizer for transformers and some other optimizers won't work well as per the transformer architecture.

### Small bert

- Batch size = 256, the time taken is high and the accuracy takes a hit.
  - Should be because there are smaller number of transformer blocks here to take up the processing load.
    - Time :  $224 * 5 = 1100$  secs
- With batch size of 32
  - takes  $110 * 5 = \text{Approx } 550\text{secs}$ 
    - Overall and gives a better accuracy than with bigger batch size
- So, with small BERT, larger batch sizes are not ideal
- With batch size = 32 and  $\text{lr} = 3e - 4$ ,
  - the accuracies still saturate at similar max numbers.

### Albert

- Batch size = 32
  - Gives much better accuracy of 87% even in just 1 epoch but takes a lot of time - 441 secs just for one epoch.
  - Validation accuracy doesn't seem to improve with epochs though.
- Batch size – 128
  - Execution takes similar time and validation accuracy saturates at a similar peak of 88%

### Electra Base

- As mentioned in the suggestions, Electra provides better accuracies

- An epoch takes 400secs with batch size 32 which is better than the above models.
  - But validation accuracy doesn't improve with epochs
- One interesting observation apart from this is for review - "The movie was meh.". Ideally, this is a negative review, but Small Bert gives a very high score to this review which doesn't seem to be correct. On the other hand, more accurate models like Electra gives it a low score of 0.018 which was the initial expectation.
- I also tried some other sentences like the ones mentioned here and got results as per the expectations
  - 'The movie was boringly awesome but awesomely boring and interestingly dull',
  - 'The movie was dull and drowsy',
  - 'It was a WOW movie'

### **Talking Heads Base**

- I wanted to try this model but couldn't proceed due to resource limitations. Increasing batch size tends to flag the following error in this case - OOM when allocating tensor

### **Electra Small**

- Trains very fast (140 secs per epoch) and delivers good accuracy of 87% during evaluation and sentiment classification.

Overall, this should be a recommended model if time is of the essence. This also generates correct prediction scores for tried reviews as per initial expectations.

Apart from that, based on my tests, given substantial compute strength and a large dataset to process, we should try to use bigger models like **Electra Base**

Transformers are usually touted to perform faster due to parallelization capabilities, but I couldn't confirm this in numbers while comparing with the tried LSTM models. I am guessing that with better compute powers and with some hyperparameter tuning, I should have been able to see that. It was implicit that transformers performed faster since I couldn't run LSTM with smaller batch sizes. The execution time was substantially higher. Also, different pretrained models can take different amounts of time since a particular model might favor a particular set of hyperparameters as shown above.

Overall, the biggest issue I faced was the time of execution.