NLP Assignment 2: Sentence Representations
CSE 538 Fall 2019

Anmol Shukla, SBU ID - 112551470

26^{th} October 2019

2 Model Implementation

- DAN
  - In the **init** method of **DanSequenceToVector**, I created all the Dense layers of size "input_dim" with *tanh* activation function.
  - For **call** method - we have been given the word embeddings for a batch of sentences that are padded with tokens. To obtain the correct embeddings for a sentence, we need to use the sequence_mask and multiply with vector_sequence to put a 0 across the the embeddings for all the padding tokens. But before this, I have expanded the dimension of sequence_mask so that it is of shape (batch_size, max_tokens_num, 1) and could be multiplied using broadcasting with vector_sequence.
  - The next step is **dropout**. For dropout, I followed the DAN paper approach. I generated a tensor of *sequence_mask.shape* having elements drawn from uniform distribution. Any element having a value less than "dropout" would have a 0 and all other elements will have 1. This is in line with DAN paper approach as they use Bernoulli trials and assume that dropout probability of every word is independent of each other (the Bernoulli independent trial assumption). Therefore, we need not worry about paddings as they are already 0 and their dropout does not affect other words. I then multiplied this mask (after expanding its dimensions) with the vector_sequence.
  - For **calculating the average vector**, I did not use "tf.reduce_mean(vector_sequence, axis = 1)" because the denominator cannot simply be the total number of words/tokens. We have to exclude the dropped out words and the padding words (as per the equation 8 of DAN paper). Therefore, after masking padded tokens and dropout, I counted the number of non-zeros in the mask tensor and used that to divide from avg_vector_sequence using tf.math.divide_no_nan to avoid NaN when computing loss in case of bigrams i.e. when both the tokens would be dropped, we would be dividing by 0. Alternatively, we could also ignore word dropout in such cases.
  - The next step is to **call the Dense layers** one by one. The input for the first layer will be our average_vector. The input for all the successive layers will be the output of previous layer. "combined_vector" will simply be the output of last layer and "layer_representations" will simply be output of all the layers stacked together.
- GRU
  - In the **init method** of GRU, we will declare all our GRU layers just as DAN. Each GRU layer will have *tanh* activation and we will set return_sequences = True, return_state = True so that we get input for the next layer i.e. output of the current layer and also the last state of the current layer.
  - In the **call method**, we will call every dense layer one by one such that the sequence output of one layer is the input for next layer. At every iteration, we will also get the last state of the current layer which we will save in a list.
  - The last state output will be the combined_vector and the output of every state will form layer_representations.

- Probing Classifier
  - In the **init method** method of ProbingClassifier we will define a single classification layer that will be Dense layer having 2 units.
  - In the **call method**, we will call the classifier with the given inputs which will return the last layer output and layer representation of all the layers.
  - To get the representation for a particular layer, I used unstack method to get a list of all the layers. Then, I extracted the representation for a particular layer by indexing into the list and passed that to the classification layer that was declared in init method. This would give us the logits which is the output of our probing model.

3 Analysis

3.1 Learning Curves

- **Increasing the training data**
  From Figure 1 and Figure 2, we can see that as the training data increases, the performance of DAN and GRU improves. However, we can see that for a smaller dataset, DAN performs better than GRU. In other words, to reach the same accuracy of 0.90, DAN requires a smaller set of training data than GRU as evident in the graph.

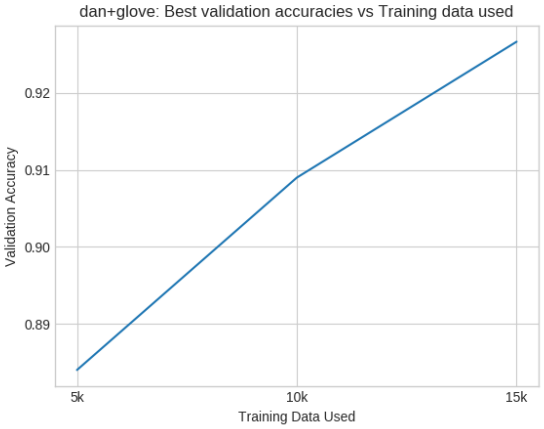

Figure 1: DAN accuracy vs training data size

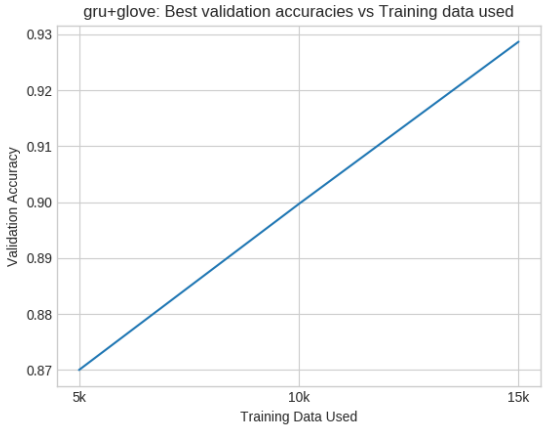

Figure 2: GRU accuracy vs training data size

- **Increase training time**
  From Figure 3 we can see that as we increase the number of epochs, the training loss decreases for DAN and then it flattens out after the 20-25th epoch. This indicates that the loss is converging very slowly and we could perhaps use early stopping as the validation loss, evident by Figure 4, keeps increasing after the 10th epoch. This is because our model starts overfitting the training data leading to high variance.

  From Figure 5, we can see that training accuracy improves with number of epochs and then it flattens out at around 0.98-0.99 which means that DAN has fit the training data (in fact, we could say that it has overfit the data). Figure 6 shows that validation accuracy increases but once the model overfits the training data, validation accuracy goes down a bit and shows high variance.

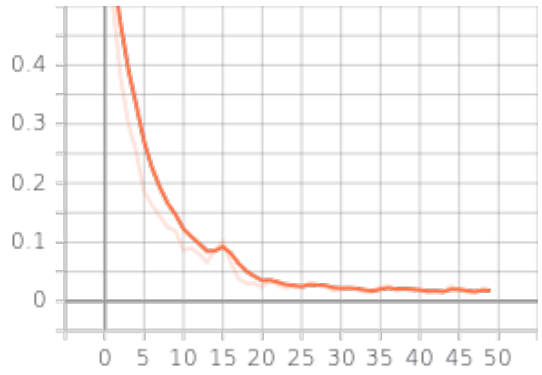

Figure 3: Training loss vs epochs

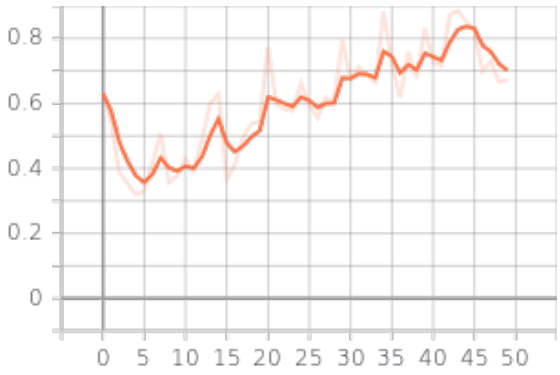

Figure 4: Validation loss vs epochs

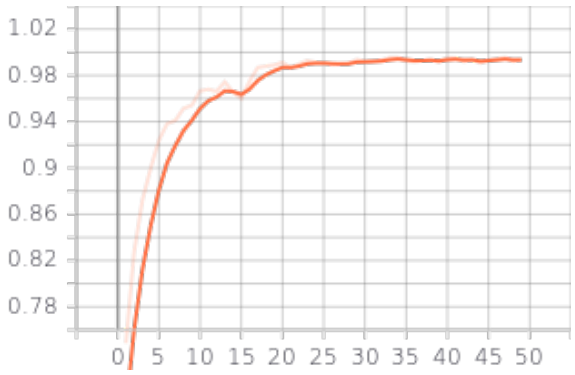

Figure 5: Training accuracy vs epochs

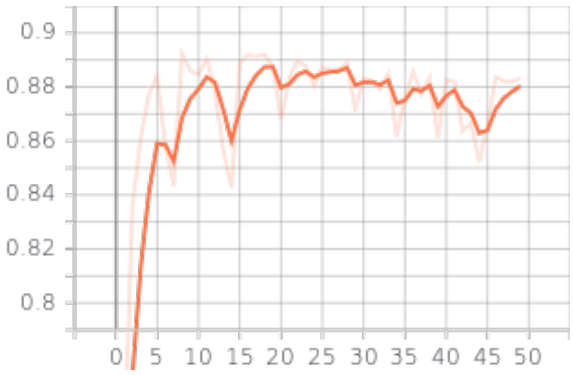

Figure 6: Validation accuracy vs epochs

3.2 Error Analysis

- Advantage of DAN - DAN requires very less training time to achieve a similar performance to GRU. It took me 10 seconds to train a DAN model while it took me 20 mins to train a GRU model. Also, DAN's linear transformations produced a vector which better differentiated between sentences that had very subtle differences such as "this is cool" and "this is okay".
- Advantage of GRU - GRU respects and considers the word order unlike the BOW model of DAN. Hence, it was able to do well on tasks such as bigram task where the order of the words mattered.

DAN fails in the cases where there is a double negative such as "I do not hate this movie" while GRU is able to predict the correct sentiment. I verified this by putting the examples in jsonl and running the predict.py. I also tried all the sentences given in the DAN research paper (Table 3), and DAN gave me similar results as indicated. Here is a short table (I have not put all the sentences so as to keep this table short):

Sentence	DAN	GRU	Ground Truth
I do not hate this movie	0	1	1
I do not like this movie	0	0	0
this movie was not good	0	0	0
this movie was good	1	1	1

However, GRU does better on bigram tasks because it considers the order of the word with the help of memory i.e. statefulness. For example, GRU predicted that "movie this" was not a valid bigram while DAN predicted that it was a valid bigram. DAN did not consider the order of the two words and since they had co-occurred in the training data, DAN predicted that they were bigrams. I used the models generated in probing task for this prediction.

4 Probing Tasks

4.1 Probing Sentence Representation for Sentiment Task

In the figure 7 and 8 we can see that as number of layers increase, the accuracy for both DAN and GRU increase. However, DAN achieves a better accuracy with 3 layers than GRU with 3 layers. This proves the DAN paper's claim that the simple feedforward layers are able to learn as good as GRU. However, there is no improvement on adding the 4th layer for DAN while GRU certainly improves after adding the 4th layer.

Layer 1 - DAN has an accuracy of 0.857 and GRU has an accuracy of 0.82. Layer 2 - DAN has same accuracy of 0.857 and GRU improves to 0.845. Layer 3 - DAN improves to 0.864 and GRU improves to 0.850. Layer 4 - DAN flattens out and does not improve while GRU improves to 0.855.

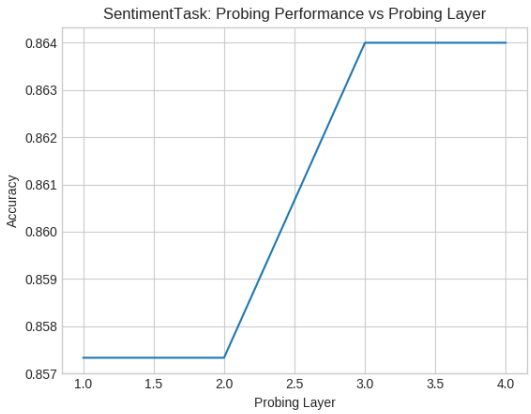

Figure 7: DAN accuracy vs num_layers

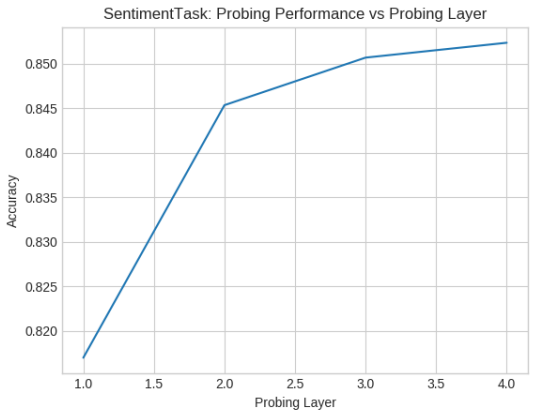

Figure 8: GRU accuracy vs num_layers

4.2 Probing Sentence Representations for Bigram Order Task

The accuracy for GRU on this task was around 0.63 while for DAN it was around 0.50 as shown in the figure 9. This was expected as DAN is a BOW model that does not consider word order and as such, it cannot differentiate between "good movie" and "movie good". Therefore, I noticed that it got many bigram predictions wrong and basically behaved as a random classifier with accuracy of 0.50. Although GRU did a bit better than DAN, the accuracy was still quite low, indicating that GRU is not as good on this task as it is on sentiment analysis. **(graph on last page)**.

4.3 Analysing Perturbation Response of Representations

Figure 10 shows the perturbation response. It basically shows how well DAN is able to differentiate between slightly different sentences where there is only a difference of single word and how the difference becomes pronounced over the layers. Therefore, as the network gets deeper, negative sentences are increasingly different from the original positive sentences. However, Figure 11 shows a different story. In case of GRU, the difference between slightly differenct sentences remains the same with the addition of GRU layers which shows that GRU cannot differentiate between sentences like "this movie was ok" and "this movie was cool" **(graph on last page)**.

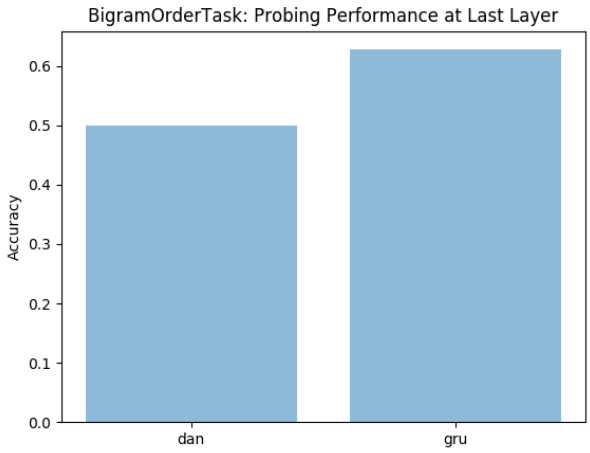

Figure 9: Accuracy of DAN & GRU on reversed bi-gram task

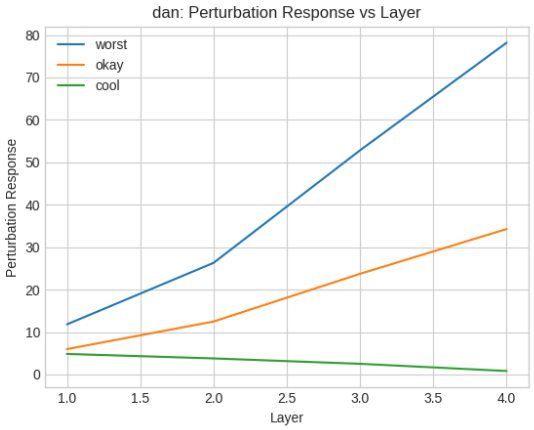

Figure 10: Training accuracy vs epochs

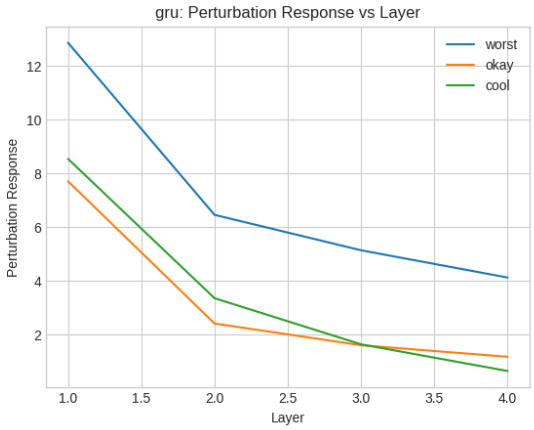

Figure 11: Validation accuracy vs epochs