Date: 10/25/2019 Name: Kaustubh Sable; SBU ID: 112608972

1. Model Implementation:

DAN:
DAN refers to the Deep Averaging Networks. In this model, we have to average the input word embeddings and then apply a series of non-linear layers.

For this, we first declare the non-linear layers in the **__init__** function of class DanSequenceToVector. Then, in the call function, we first implement masking. Masking is implemented in order to inform the model about the padding which is done to make all the samples of uniform length. For masking, I have converted the input sequence_mask to a 3-D tensor and then multiplied it with the input vector_sequence.

Once, masking is done, we implement dropout. Dropout refers to ignoring some of the neurons/word embeddings during the training phase. It is a regularization technique used to control/prevent the overfitting of the model. We have implemented dropout using tf.random.uniform which generates a mask of random numbers. Using self.dropout, we adjust the mask to fit our requirement and multiply this mask to our vector_sequence. Note that dropout is implemented when “training”=True.

After implementing dropout, we pass the vector_sequence through an array of non-linear layers. Also, we capture the representations from each layers and return it along with the output of final layer.

GRU:
GRU refers to Gated Recurrent Unit. In case of GRUs, we declare the GRU layers in the **__init__** function. We set return_sequences and return_state to True as this allows us to stack multiple GRUs on top of each other.

Inside the call function, we first pass the inputs through one GRU layer along with the mask specified. Then its output is passed to next GRU layers. Then in a similar fashion as in DAN, we return the representations from each layers and the output of final layer.

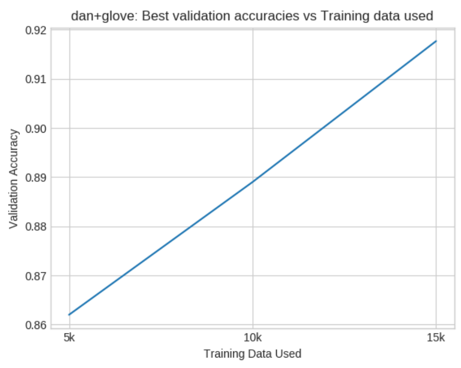
Probing Model:
In case of probing model, we extract the representation from single required layer from our pretrained model and attach a linear classifier after that (I have used Softmax for this). In the call function, we first pass our inputs through the pretrained model layer. This gives as output a dictionary from which we extract the layer representations. Then, we extract the representation for the specific layer and pass it through the activation function (linear classifier) which gives us the logits required.

2. Analysis:

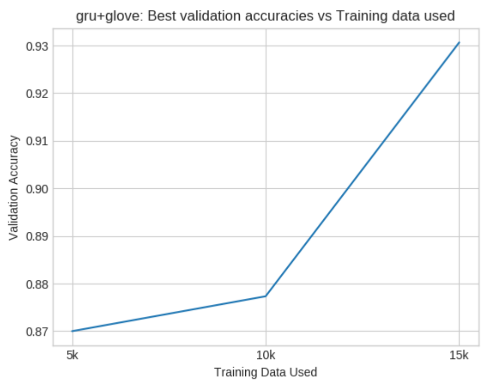
Learning Curves:

• Increasing the training data:

Performance Graph for DAN

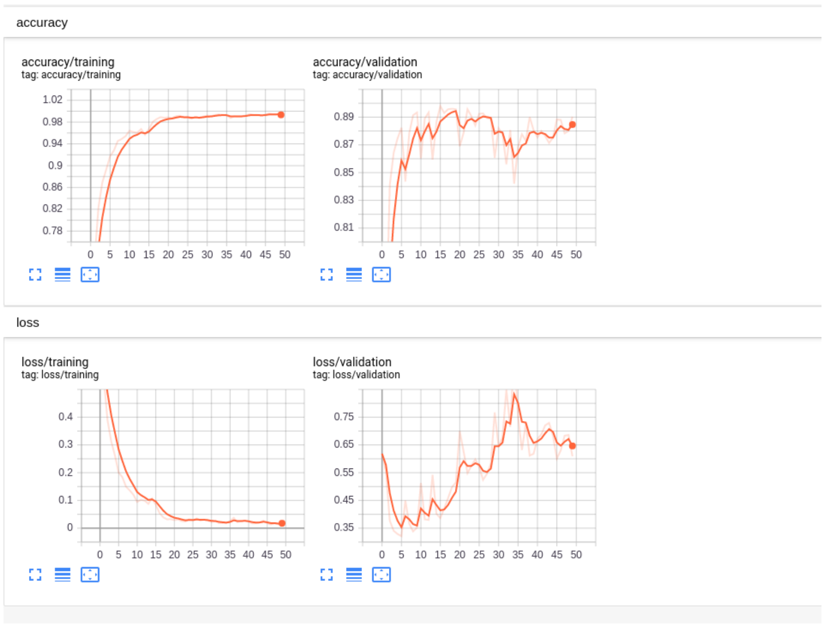


Performance Graph for GRU



From the graphs and from the accuracy of the executed models, we can see that for both the models, accuracy increases as we increase the training data. However, it is seen that the accuracy increases linearly in case of DAN while in case of GRU, it increases drastically after a point. This might be due to the fact that DAN is a complex model and needs more data to better train the model.

• Increase training time (number of epochs):



When we increase the number of epochs we can see that the training loss decreases constantly. However, the validation loss spikes around epoch #32. So, what’s happening around this, is that training loss is decreasing and validation loss is increasing. This is a typical feature of overfitting. So it is possible that after 30-35 epochs, the DAN model overfits and hence the loss increases.

Error Analysis:

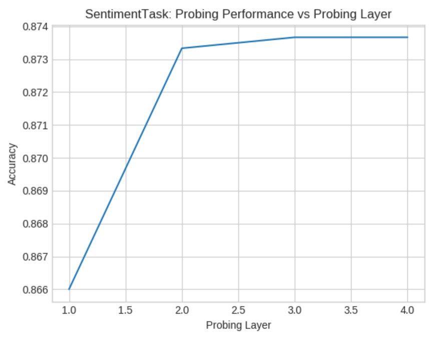
• **Explain one advantage of DAN over GRU and one advantage of GRU over DAN**:
Advantage of DAN over GRU:
One of the major advantages of DAN over GRU is that the DAN model learns faster than the GRU model. While training the models, I have observed that DAN model takes around 10 seconds to train for the imdb_sentiment_5k dataset over 5 epochs. However, the GRU model takes around 20 minutes to train for the same configuration. Training time
One of the major advantage of GRU over DAN is that GRU is able to capture long-range dependencies better than GRU. This is due to the reason that it uses the concept of gates which pass the information to the next layers. In case of our model, we can observe this during perturbation analysis that GRU learns the difference early on as compared to DAN. Also, GRU learns contextual information better than DAN. This is evident from accuracy of the bigram order task.

• **Use the above to show and explain failure cases of GRU that DAN could get right and vice-versa:**
Failure case of GRU that DAN could get right:
For this, we can refer the words “okay” and “cool” from the perturbation analysis. In case of GRU, after couple of layers, it tries to treat both the words as synonyms as they might have been used in similar contexts in some cases. However, in case of DAN model, it learns to differentiate the words more accurately as the number of layers increase.
Failure case of DAN that GRU could get right:
In case of DAN, as it does averaging of all the words, it loses the important differences between them. However, GRU learns the contextual features from the corpus. This is evident from the accuracy of both of models on the bigram order task(where order of the words matter).

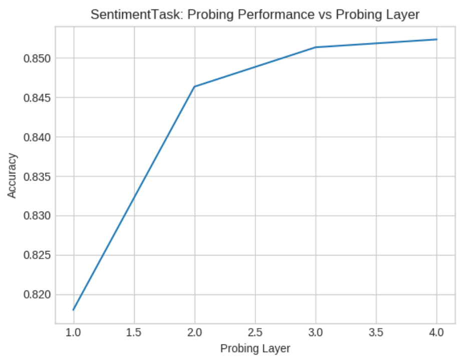
3. Probing Tasks:

Probing Sentence Representation for Sentiment Task:

Performance for DAN

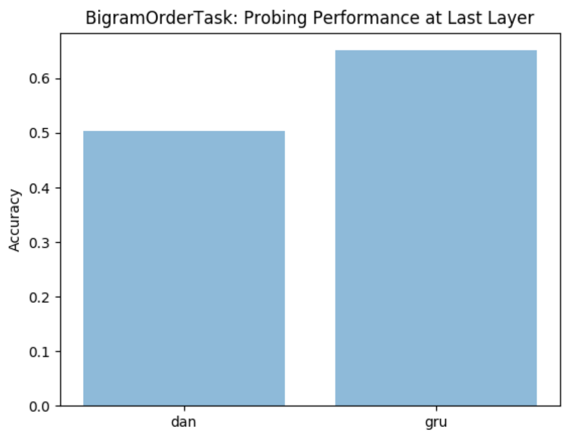


Performance for GRU



From the graph we can see that the accuracy of the both the models increase as the model becomes deeper. However, the DAN model learns the embeddings faster and the accuracy flattens out after couple of layers. However, in case of GRU, it takes more time and layers to learn the features and hence the accuracy gradually increases as the model gets deeper. This is also the reason why the final accuracy of the GRU is still less than the accuracy of DAN for 4 layers.

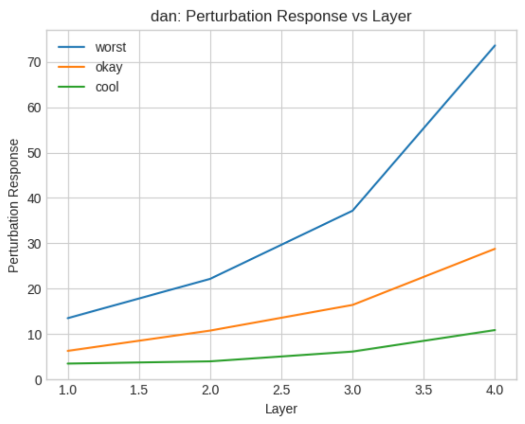
Probing Sentence Representations for Bigram Order Task



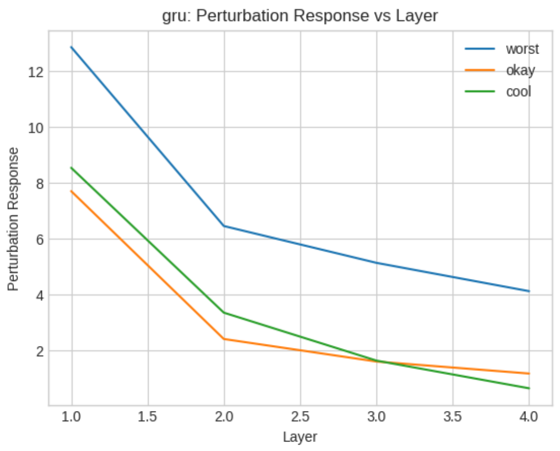
From the above graph of sentence representation for Bigram Order Task, we can see that the GRU model performs better on the new dataset as compared to the DAN model. This is due to the reason that DAN cannot learn contextual features and nor can it capture long range dependencies. This is the reason that the GRU model, which captures the contextual dependencies efficiently, performs well on the bigram order task.

Analyzing Perturbation Response of Representations

Perturbation Response of DAN



Perturbation Response of GRU



From the above graph, we can see that the Perturbation Response of DAN shows a good response when the words are changed. The word “worst” which has a negative connotation is well separated from the word “cool” (having positive connotation). Also, the representations get better as the model gets better. This is due to the fact that though the averaging minimizes the important differences between the words, but as we increase the layers it is able to recover these differences.

However, in case of GRU, the perturbation response shows that the GRU model has learnt the differences well and as the layers increase the difference almost remains constant. Towards the last layers, however we see that the model learns that there is less differences between the word “okay” and “cool”. This might be because of the long range dependencies it has learnt and figured out that these words are sometimes used interchangeably in some sentences.