# Assignment – 2

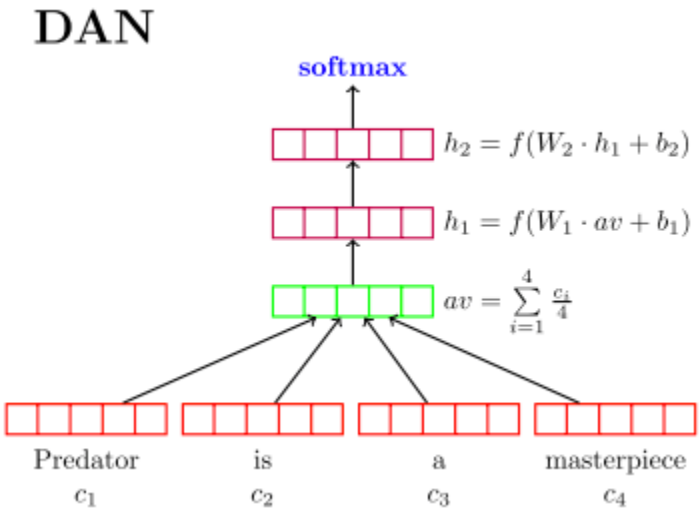
# Sentence Representation


## Anurag Dutt

## SBUID: 113019209


## Model Implementation

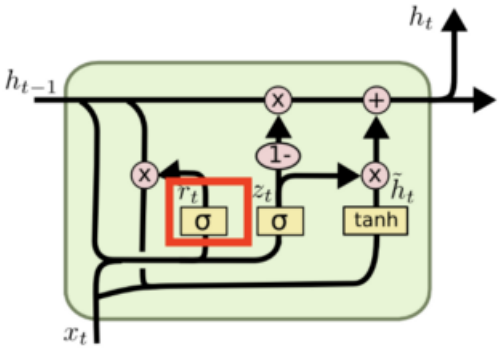
- ***Deep Averaging Networks:***




As mentioned in the representation given in the paper, (image taken from the Iyyer et al. Paper) Deep averaging networks or DANs for short average out the representation of each word embedding in a sentence, to get a single representation for the entire sentence. The output is then passed through multiple non-linear hidden layers, with the final output received from the output layer is passed to a softmax function, which does the sentiment classification. For the purposes of our implementation, we needed to implement the DAN classifier barring the softmax function. The vector sequence input provided was a 3-dimensional tensor representing [Batch_Size = 64, Max_tokens_in_sentence = 209, Word_embeddings_in_sentence = 50]. We are also given a sequence mask which we are given to understand points out the padded tokens in the sentence. We apply a Boolean Mask (sequence_mask is a boolean matrix) of sequence mask over the vector sequence of word embeddings and then apply a dropout (generated from a bernoulli distribution of the dropout probability) for each word in the sentence, thus getting a boolean for whether each word in a sentence should be included in the analysis or dropped out. The we calculate dropout is by generating a numpy.random.rand(n_words) > dropout probability. This generates a boolean True-False list of n_words length, where probability of each

element being true is 1-dropout_probability and each element being false is dropout_probability. We discard the words which are indexed False in the list, thus ensuring dropout probability for each word is consistent. We collect all the remaining word embeddings for the remaining words in a sentence, average out to get the sentence representation. In the init call, we define the input, output and the hidden layers based on the num_layer input. We pass a 2-dimensional tensor [Batch_Size x sentence representation (size = word embedding) which is the average sentence representation of all words]. The output of each layer is stored in the layer representation tensor, and the combined vector stores the final output which is to be forwarded to the softmax. The function returns combined vector and layer representation.

- ***Gated Recurrent Units:-***




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

(Image taken from https://towardsdatascience.com/what-is-a-recurrent-nns-and-gated-recurrent-unit-grus-ea71d2a05a69)

Gated Recurrent Units are an implementation of Recurrent Neural Networks where it uses two gates namely – update gate and reset gate, to carry information over multiple future timesteps thus excerabating the issue of vanishing gradients to a large extent. The update gate decides on how much information to keep and reset gate decides how much information to forget. For implementing GRU we start by initiating hidden layers in the GruSequenceToVectorClass., based on the num_layer input. We used tf.keras.layers.GRU. We used a tanh activation and set return_sequences and return_state as True. Return state gives us the layer output of each layer and Return Sequence returns the entire time step sequence of the word vocabulary input. We use vector_sequence as input which is a tensorized input of size [Batch_Size = 64, Max_tokens_in_sentence = 209, Word_embeddings_in_sentence = 50]. Keras.GRU accepts a 3D tensor as input and returns both the state and sequence. We pass on the sequence as the input to our next layer and store the state as our layer representation for the current

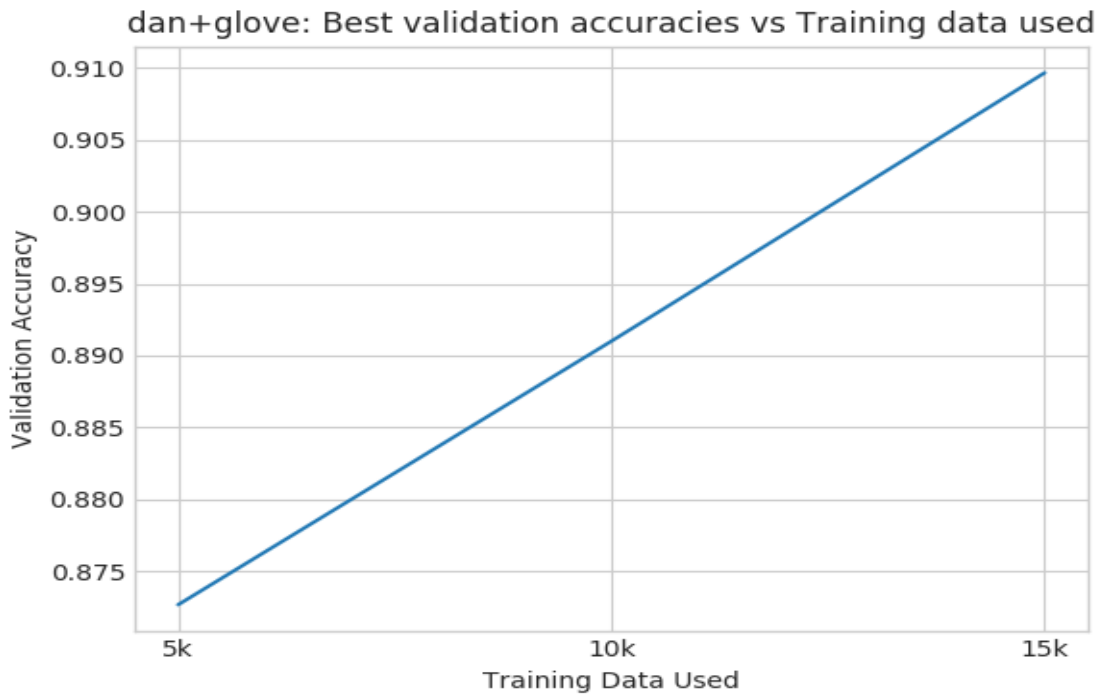
layer. The state of last layer is returned as the combined_vector output and passed to softmax(not in the function scope) and we return the layer representation as well.

- ***Probing Model:-***

For implementing the probing model, we initiate classes_num, which is the the output size of the linear classifier. We also initiate a linear classifier (to do this we implement a Dense layer with classes_num as output size and a linear activation function, since the layer is supposedly a linear classifier. We then pass the input tensor to the loaded pre-trained model and get the dictionary for the logits and layer representations. We pass the layer representation of the nth layer (where n is provided as input) through the linear classifier and get our logits, which the function returns. We do this whole process to determine how each layer learns, and how much contribution does each layer make to the learning process.

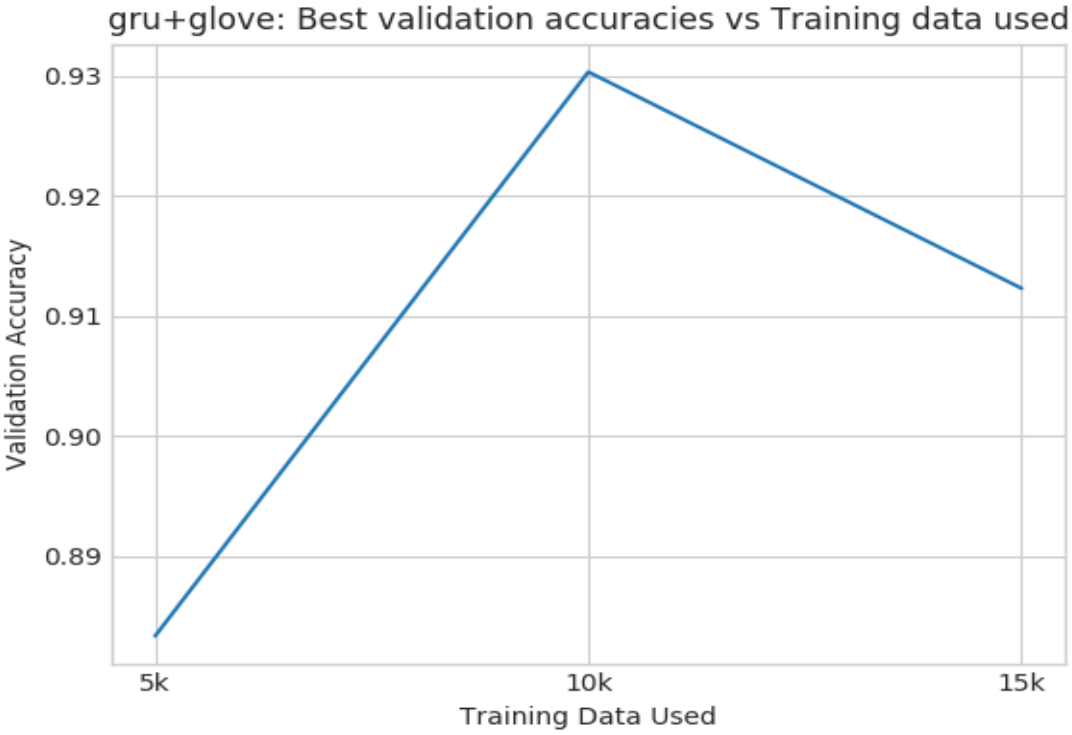
## Error Analysis and Learning Curves

- ***Effect of increasing training data:-***
  - **DAN:**



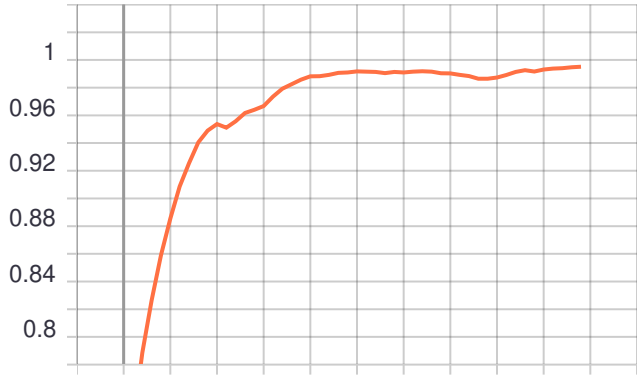
As we can clearly see that for DANs there is a direct correlation between increasing training data and DANs. Exposing the model to more batches and instances, leads to better prediction accuracy. If we have implement dropout correctly, our model will continue to learn as the training size increase and correct implementation of dropout ensures our model is not overfitted.

- **GRU:**



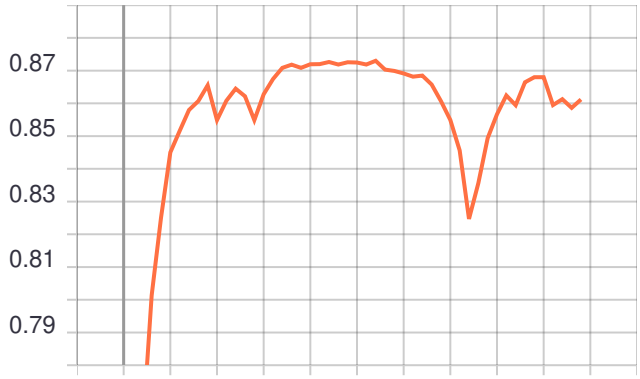

We can clearly see that increasing training data increases accuracy on the validation set. However, increasing the activation, by a much higher amount decreases the validation accuracy. This observation might be because, increasing training data by a large amount might lead to overfitting. Since, we are not implementing dropout (Note that dropout here refers to dropping nodes rather than dropping words), there is a possibility that GRU overfits on the training set, which explains a small drop in validation accuracy (Drop of close to 1.5%).

- ***Effect of training time:***
  - **Training Accuracy vs Epochs:**



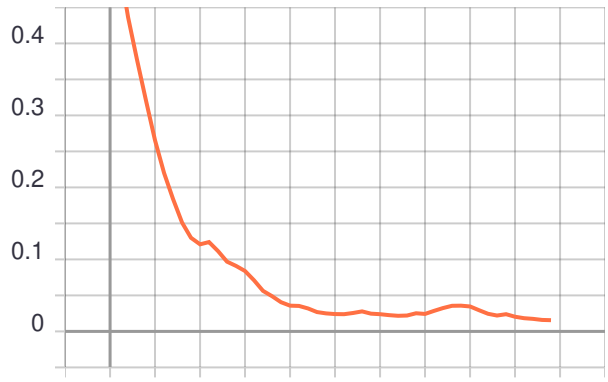

The epochs are on the x-axis and training accuracy is on the y-axis. As expected the training accuracy increases and almost goes to 1 as training time increases.

  - **Validation Accuracy vs Epochs:**



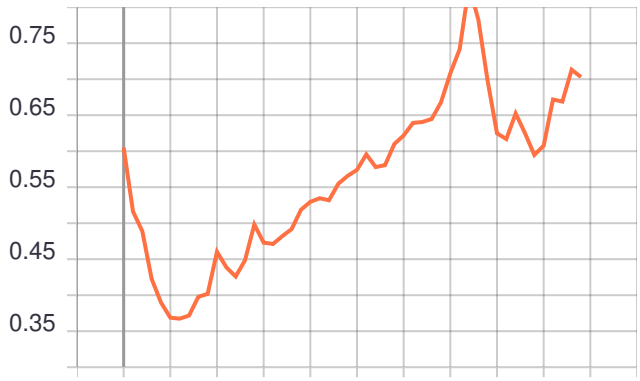

The epochs are on the x-axis and the validation accuracy is on the y-axis. Clearly, as expected the validation accuracy of the model increases in the beginning but after sufficient epochs it stops learning after a while and the validation accuracy even decreases as a result the model overfitting on the training set.

  - **Training Loss vs Epochs**



We can see that training loss consistently decreases and almost goes to zero, as we increase the number of epochs. The model keeps fitting itself to the training set better and better.

- **Validation Loss vs Epochs**




The validation loss initially decreases, then increases and then finally decreases, after sufficient number of epochs.

- ***Error Analysis:-***

### *3.1 Why GRU outperforms DAN*

Theoretically we know that, GRU should perform better than DANs on prediction tasks because of the sequential nature of the model. However, for classification task, GRU performs only slightly and marginally better than DANs. GRU also tends to learning to learn faster than DAN, the sense that, it takes lesser number of training iterations (epochs) to train GRU, as compared to DAN. One interesting observation to note is that GRU is essentially a sequential model and as such will perform better than DAN in tasks such as Bigram ordering and prediction. RNN's also tend to preserve the individual word representations and as such will learn the association between certain positive and negative words and the corresponding sentiments slightly better than DAN's, although this necesarrily might not always be the case, as DANs hold the compositional semantic of the sentence well. DAN is also susceptible to syntactic word components and cases such as double negatives where sequence might matter.

### *3.2 Why we observe the corresponding arguments for GRU*

We do observe, that GRU outperforms DAN by 10% in the Bigram Ordering task. Thus our analysis proves that for any task requiring sequential input of the word vector, GRU will outperform compositional models such as DAN. All in all, it was observed that GRU's have higher validation

accuracy as compared to DAN's. True to the theory, in our case, GRU slightly outperforms DAN in the sentiment classification task as well.

***3.1 Why DAN outperforms GRU?***

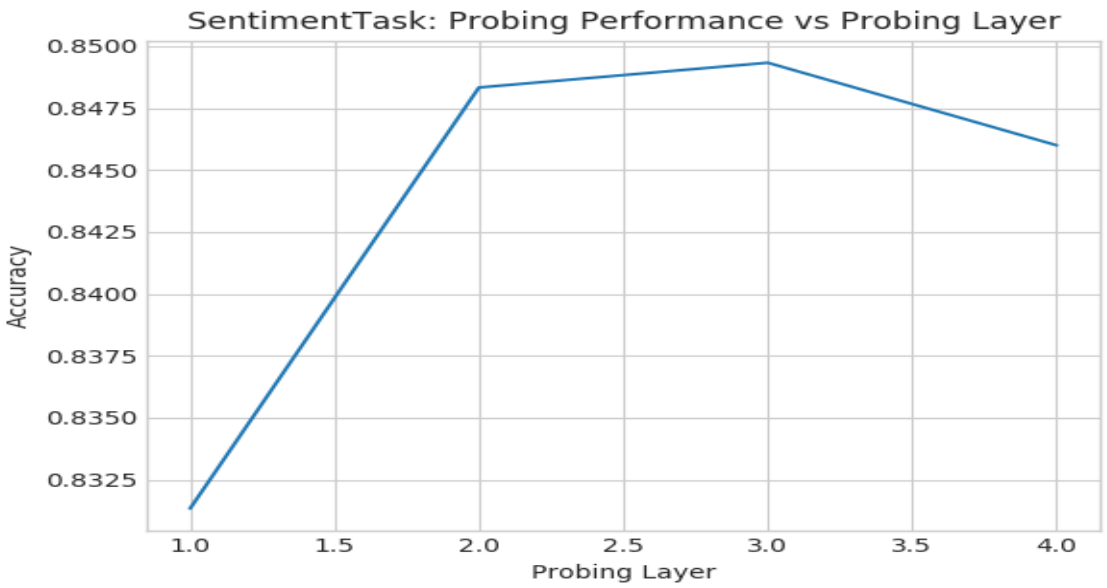
GRU takes exponential time, because it is a network type algorithm. In our modelling scenario, GRU take almost 4 times the amount DAN takes, for each epoch. As such, the computational cost associated with GRU is much higher than DAN. Surprisingly despite not being a sequential model DAN performs very close to GRU in the sentiment classification task. DANs also tend to perform well with noisy data, even improving on such experiements. DANs in general perform better than sequential models on aggregation tasks as well, and due to proper implementation of Dropout are less susceptible to overfitting as well.

**3.2 Our model supporting DANs case**

On an average each epoch for DAN takes about 54 seconds to execute, where as each epoch for DAN takes about 198 seconds to execute. Hence the computational cost and memory requirement associated with DAN is much less than GRU. DANs are much faster to train, and in conjunction with Dropout, are excellent at avoiding overfitting. If we look at the graph, for effect of training data, we can see that GRU tends to overfit while DANs are much less susceptible to overfitting as compared to GRU because despite the large training size, the model still continues learning and improving on the accuracy of the validation set.

## Probing Tasks

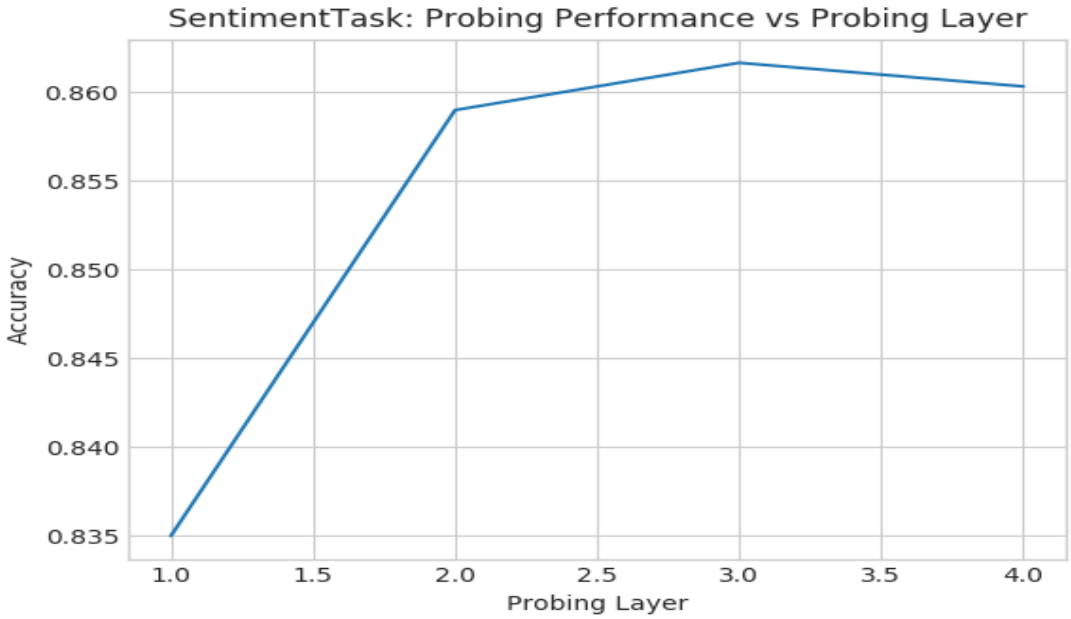
- ***Probing sentence representation for Sentiment task:-***
  - **DANs**




From the plot, we can infer that the first and second layers aid the most in learning as compared to the first and last layer. Clearly we can see that the third layer had maximum accuracy in the sentiment prediction task, followed by the second layer and then the third layer as compared to the first layer.

| | Probing Layer | Accuracy (Approximate inferred values) |
|---|---|---|
| 1. | Layer 1 | 0.8310 |
| 2. | Layer 2 | 0.8485 |
| 3. | Layer 3 | 0.8495 |
| 4. | Layer 4 | 0.8460 |

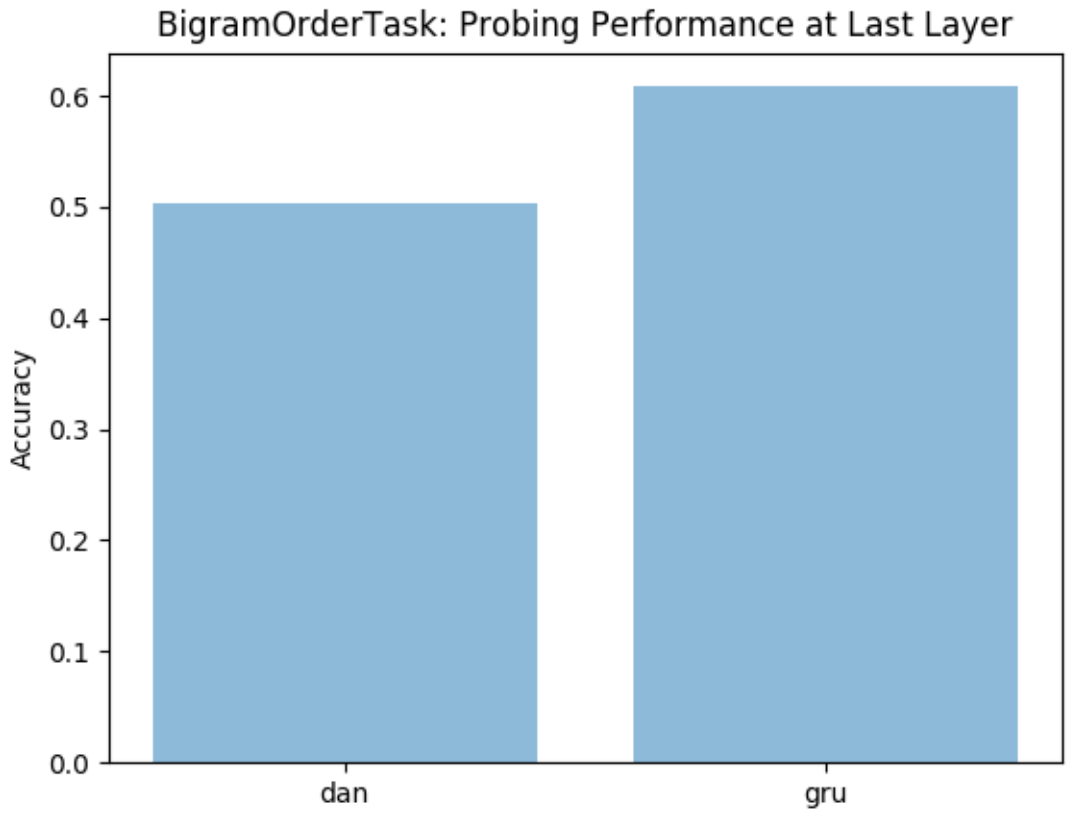
- **GRU**



GRU has a very similar plot shape for probing performance as compared to DAN's, although GRU layers have slightly better accuracy as compared to DANs. Again, the third layer showed the maximum prediction accuracy, followed by the fourth layer, followed by second layer and, lastly the first layer.

| | Probing Layer | Accuracy(Approximate inferred values) |
|---|---|---|
| 1. | Layer 1 | 0.8310 |
| 2. | Layer 2 | 0.8485 |
| 3. | Layer 3 | 0.8495 |
| 4. | Layer 4 | 0.8460 |

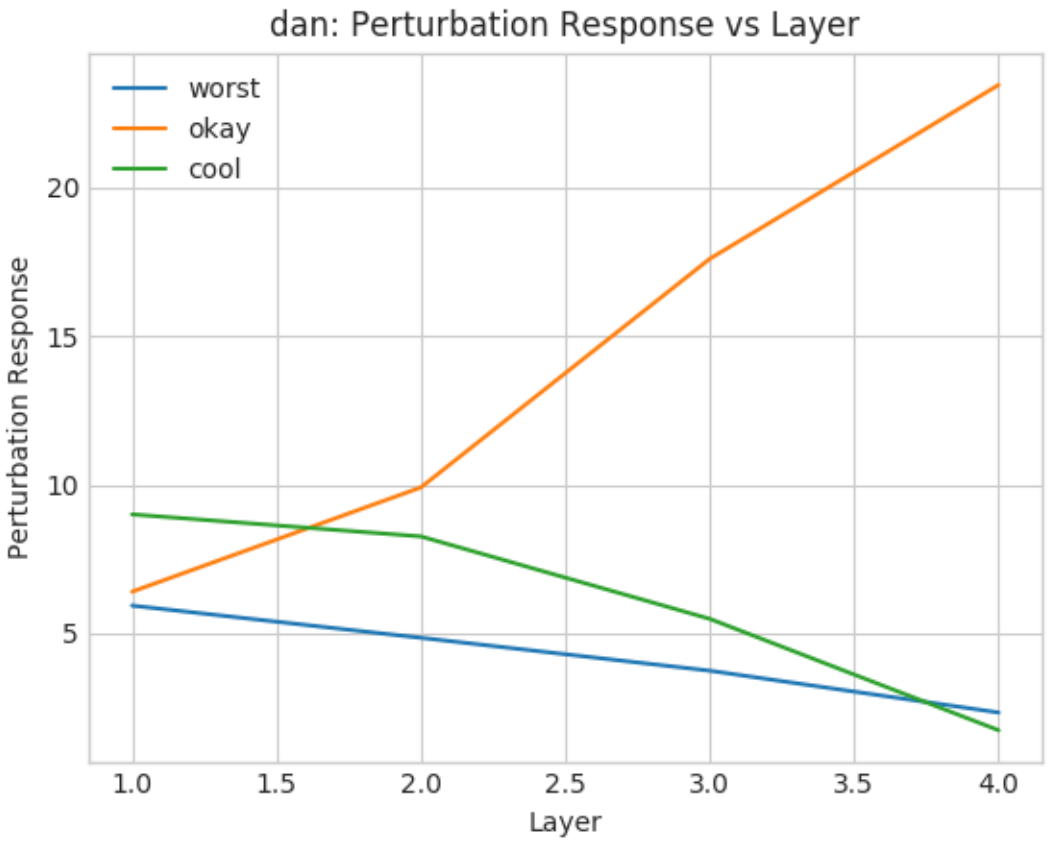
- ***Probing sentence representation for Bigram task:-***




For the bigram order, DAN performs no better than 50%. This is expected because during the learning process of DAN, sequence of the word embeddings is completely lost. So predicting the Bigram, is essentially a random choice with 50-50 probability in case of DAN. This is corroborated by our observation where accuracy of DAN for

predicted BiGram order is 0.5. GRU is a Recurrent Neural Network based model and thus takes care of the sequential ordering. Hence GRU gives a much better accuracy as compared to DANs.

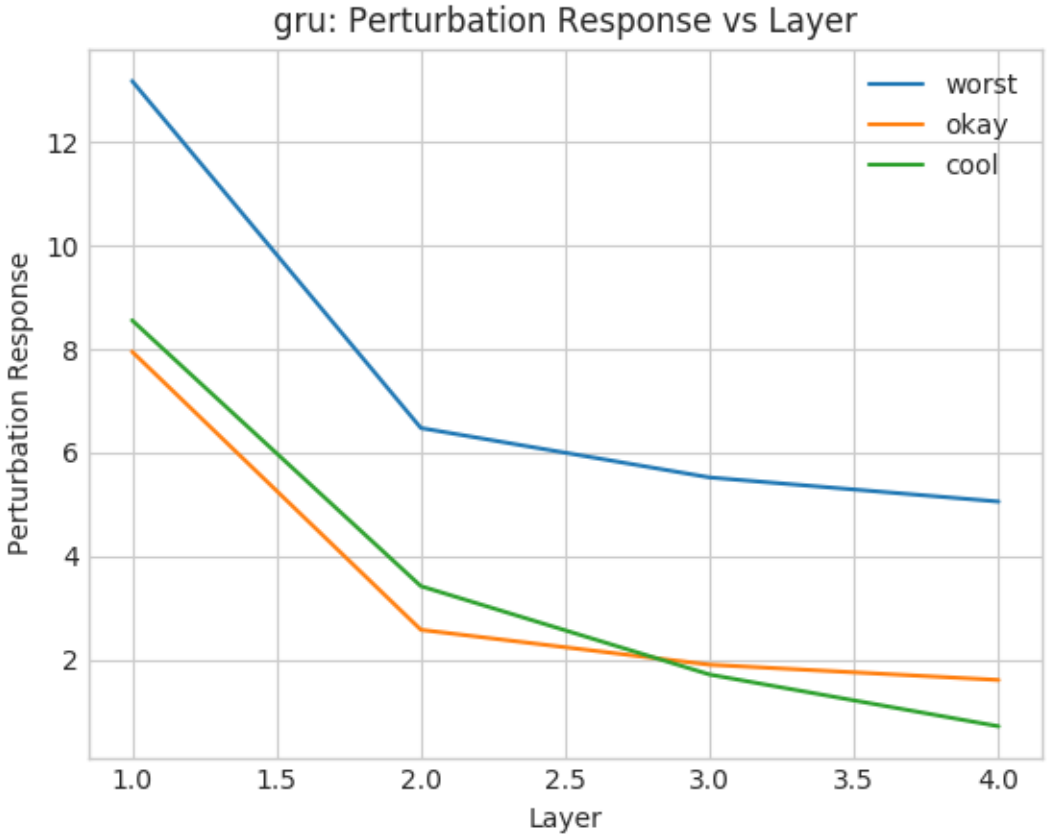
- **Analyzing Perturbation response of Representations:-**
  - ***DANs***




Clearly we can see that the perturbation response of the tag “okay” increases. The words “worst” and “okay” show a much lower perturbation response as compared to “okay”. This might be because the model tends to learn more for neutral words as compared to extreme positive or negative sentiment words. However, the learning might not necesarrily be correct as the word representation are lost while averaging. (Please check

my sanity test casae, where I recreate this graph with different hyperparameters and get a much more intuitive result)

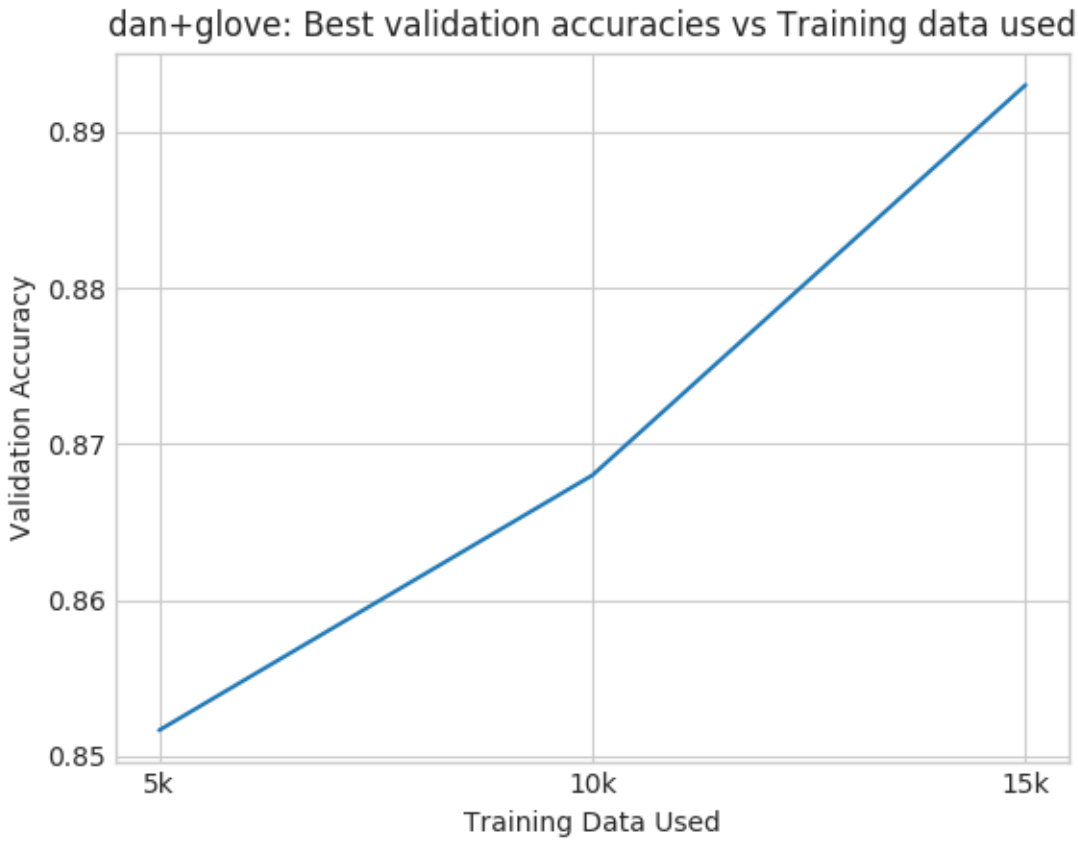
- ***GRU***



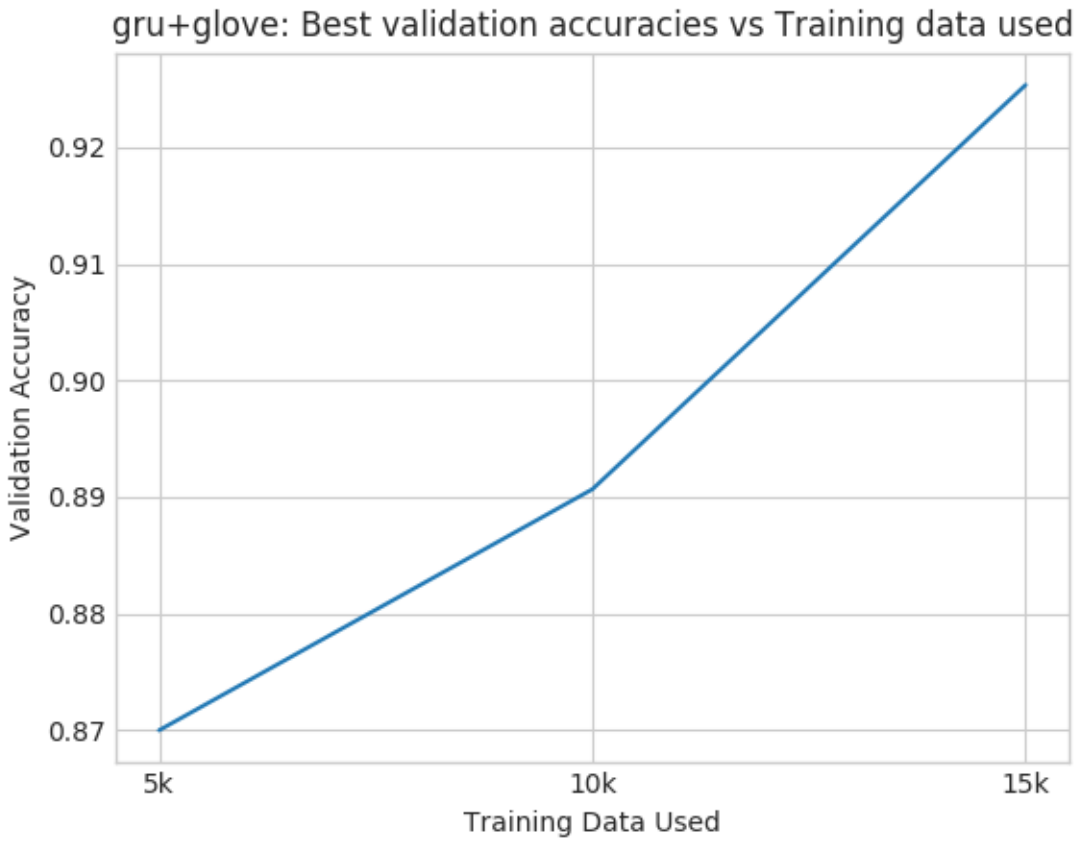

The Perturbation response for worst is although is higher for the word worst as compared to the words okay and cool. This learning looks objective as we can classify the word worst as a much stronger emotion as compared to the words “cool” and “okay”, and hence might be related to the sentiment classification much strongly than the words okay and cool.

## ***Sanity Check***

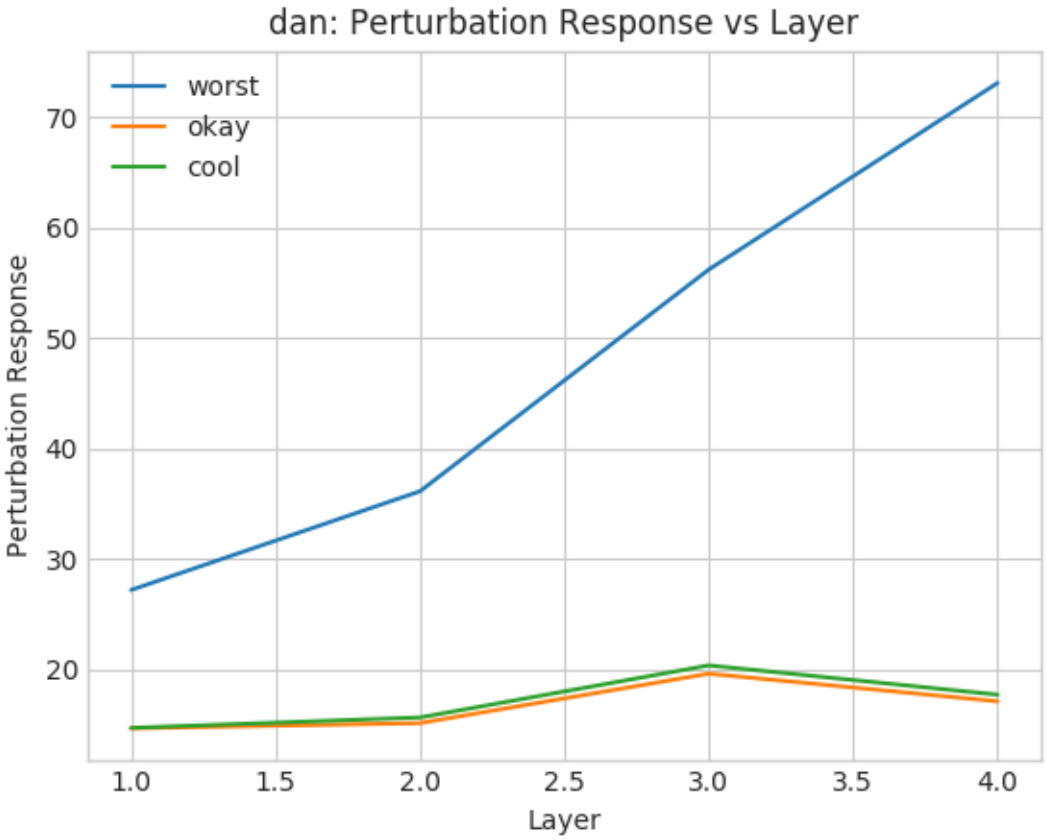
Since, we were initially getting a large value for training accuracy for both DAN and GRU (upwards of 0.92), I postulated that both my models were susceptible to overfitting. As per the instructions of the TA, I performed a sanity check by reducing the training time and as such setting the training epochs to 4, retraining the model, executing the probing part again and recreating the plots again. We considerably reduced our training loss to approximately 0.88, which also brings us in the range of expected outputs. I am attaching the plots of the sanity checked models trained with 4 epochs for your consideration.



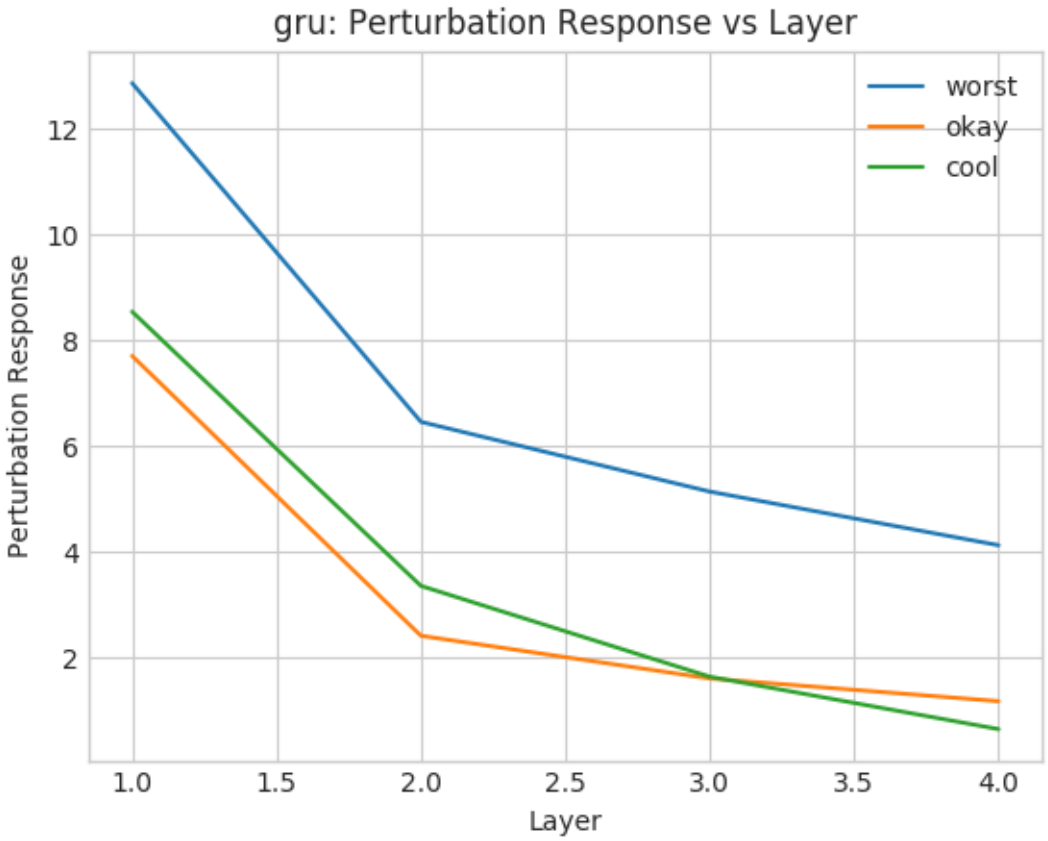

The training data for DAN doesn't change much, and hence we were correct in our assumption that DAN was not susceptible to overfitting.



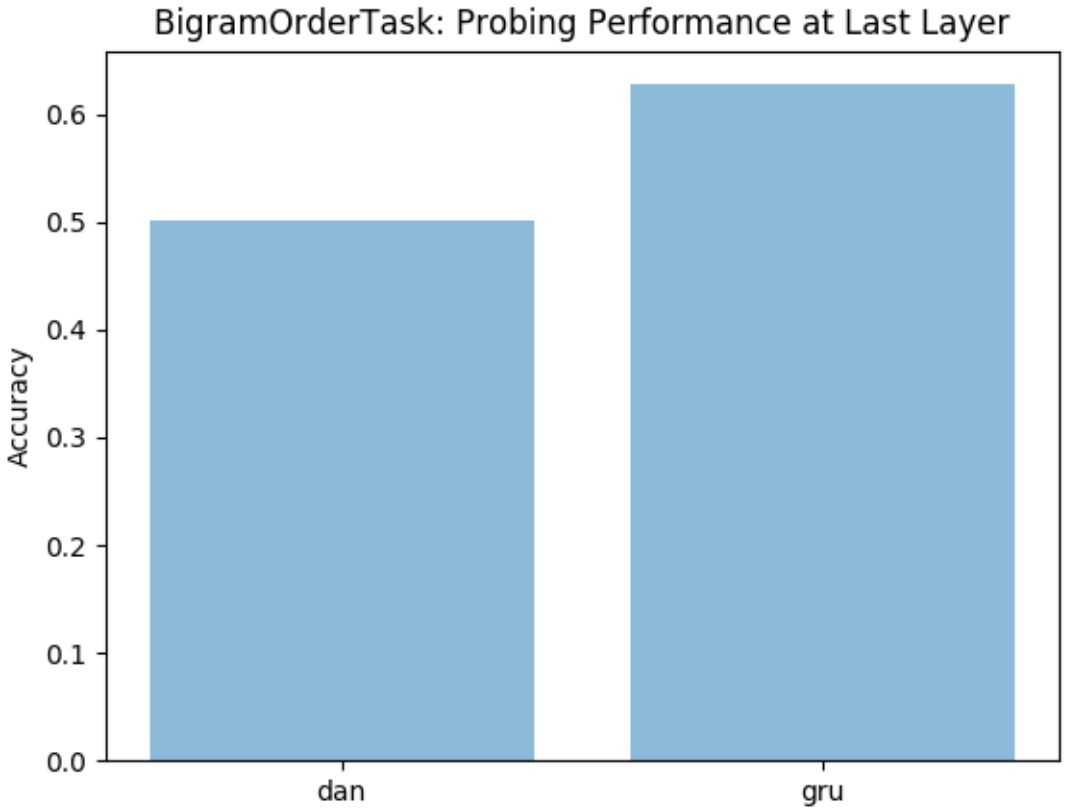

As we can see the validation accuracy of GRU improves with training data, in contrast to our previous case where it was degrading. Hence, we might assume that GRU was susceptible to overfitting for a higher training time.



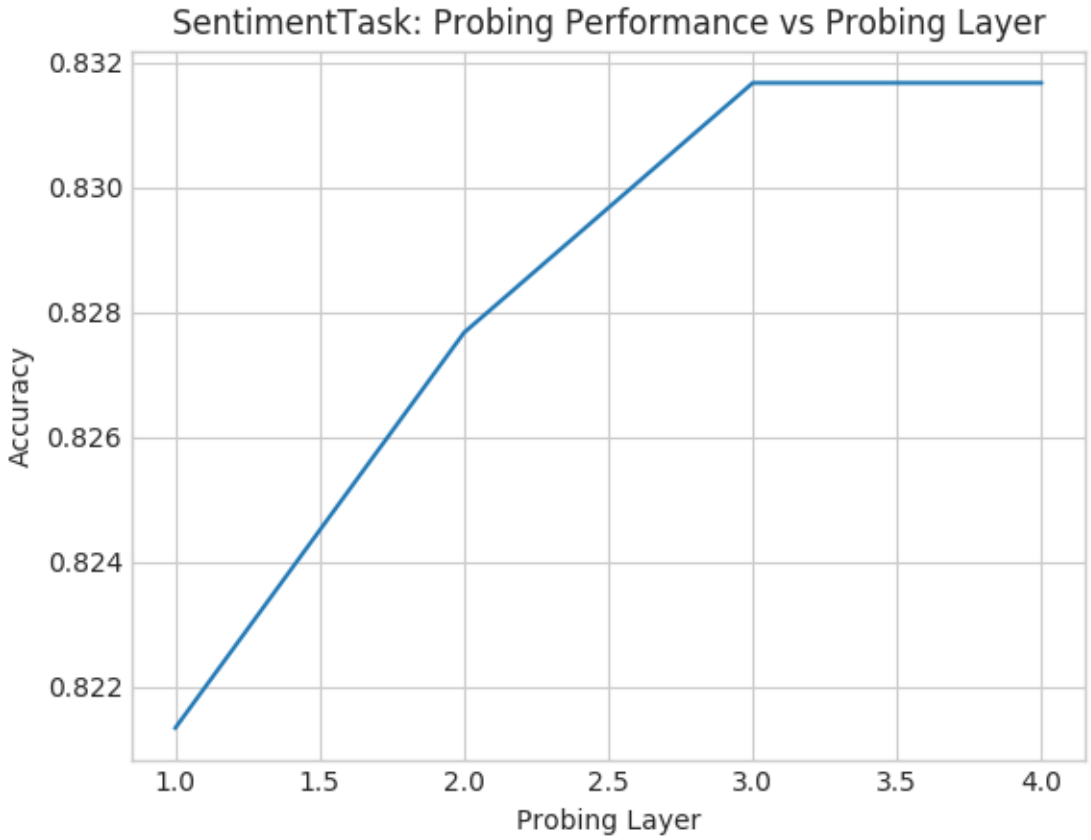

DAN gives a much more intuitive Pertubation response, after reducing our epochs and performing sanity check. The word “worst” as professed earlier is much more drastic and associated with a negative sentiment than the words cool and okay, which are more neutral. As such it is imperative that our model will learn more from the “worst” as compared to “okay” or “cool”.



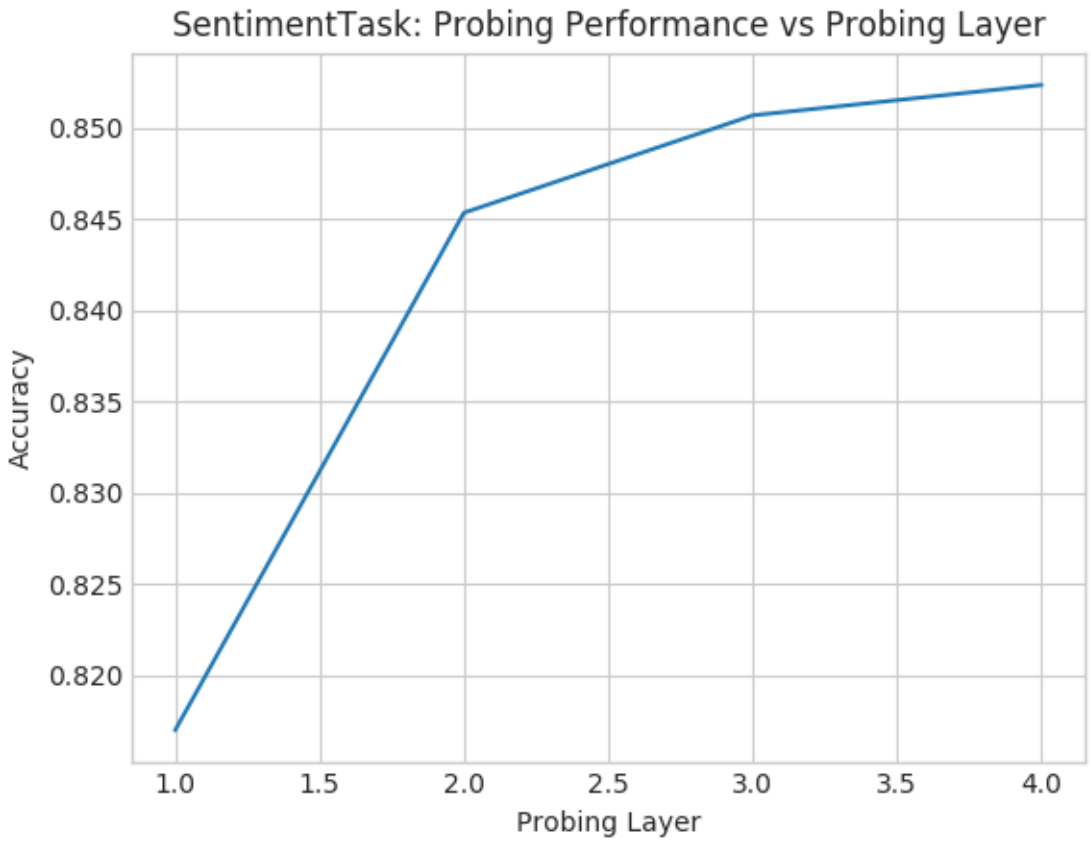

The perturbation response for GRU doesn't necesarrily change for the GRU. Thus we can interpret that GRU was performing the sentiment classification correctly in the previous iteration as well. However, our learning is reinforced that the model learns more from the stronger word "worst" as compared to more neutral words "okay" and "cool".




The bigram ordering task has the exactly same result as before, and doesn't change at all.




The probing task is also essentially same albeit slight differences. We observe that for DANs learning increases for every subsequent layer till third layer and then remains the same. Hence we can infer that instead of going down as in the previous case, it stays the same(although as you might observe the validation accuracy is slightly lower). Both these observations might be attributed to the fact that we are training our model for a lesser training time.




Same observation for GRU. GRU continues to learn in all 4 layers, and each layer has a higher validation accuracy than the previous. Again this might be attributed to the fact that we have countered overfitting by training for smaller times.

**Wget link:-**

https://docs.google.com/uc?export=download&id=1dRqMUMIHw3E3M_GKyB-SlxCiJ9w5rr3D