1. Model Explanations:

a. DAN: Deep Averaging Network
   i. Averaging can minimize important differences. By adding more transformations we can recover these differences in later layers.
   ii. Initially to get the fixed length vector containing the averages of the word embedding vectors I used tensorflow's tf.reduce_mean.
   iii. Once I had my fixed length vector I needed my non linear transformations to finally get the output.
   iv. For which I used a feedforward network with the non-linearity being a tanh function.
   v. These feed forward networks can be thought of as a stack of logistic regression models which is helping us giving the desired output.
   vi. To build the feedforward I used keras' inbuilt Dense method with activation set as tanh.
   vii. For dropout, I utilised the sequence_mask parameter and updated it using the given dropout value to get the dropout to be handled during averaging.
   viii. The number of times I ran the loop was equal to the num_layers parameter provided
   ix. Once we have the final vector output, get the last state in the last layer as the combined vector.
   x. To get layer representations I saved every layer information during looping to create stacks and then converted that list of layer information to a tensor and stored it as layer_representations.

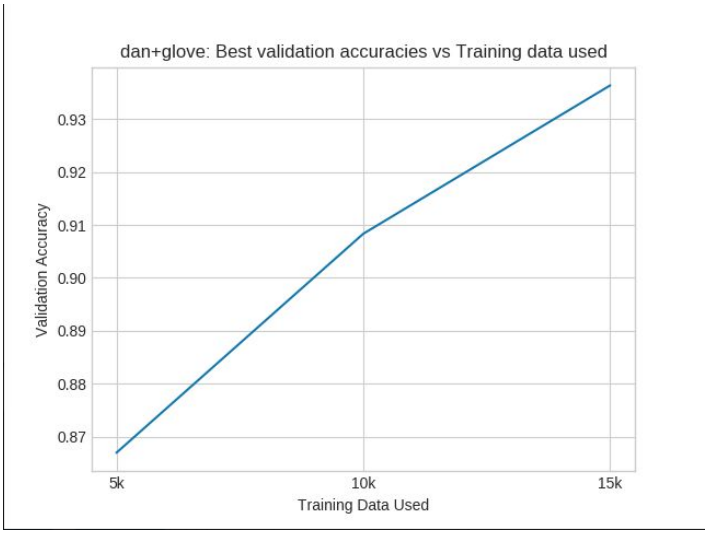
b. GRU: Gated Recurrent Unit
   i. Gated recurrent unit does away with the forget gate and input gate of the LSTM model and in fact merges them to one update gate. It can also merge the hidden and cell states.
   ii. It does multiple transformation over many layers. Usually 2-3 layers are good but more layers bring diminished layers. Usually training is slow.
   iii. The information in a GRU has to pass through hidden state.
   iv. For implementing GRU I used keras' inbuilt methodology of GRU and set the parameters return_sequences as True and used activation function as tanh
   v. The number of stacks was equal to the num_layers parameter provided.
   vi. Similar to DAN I used convert to tensor function to get the layer representations

2. Probing Method:

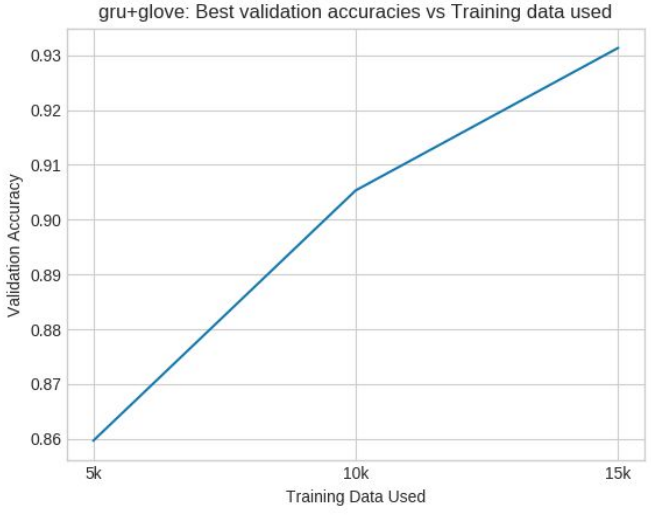
a. For probing I used the pretrained models for both DAN and gru and used the inputs parameter to retrieve the corresponding combined vector and layer representations.
b. For my linear classifier I used keras' inbuilt Dense methodology.
```
tf.keras.layers.Dense(self.class_num, activation='softmax',
use_bias=True, kernel_initializer='glorot_uniform',
bias_initializer='zeros',)
```
c. Here class_num tells me the number of labels, the activation function used was softmax and I also used a bias parameter for my linear classifier.

3. Learning Curves:

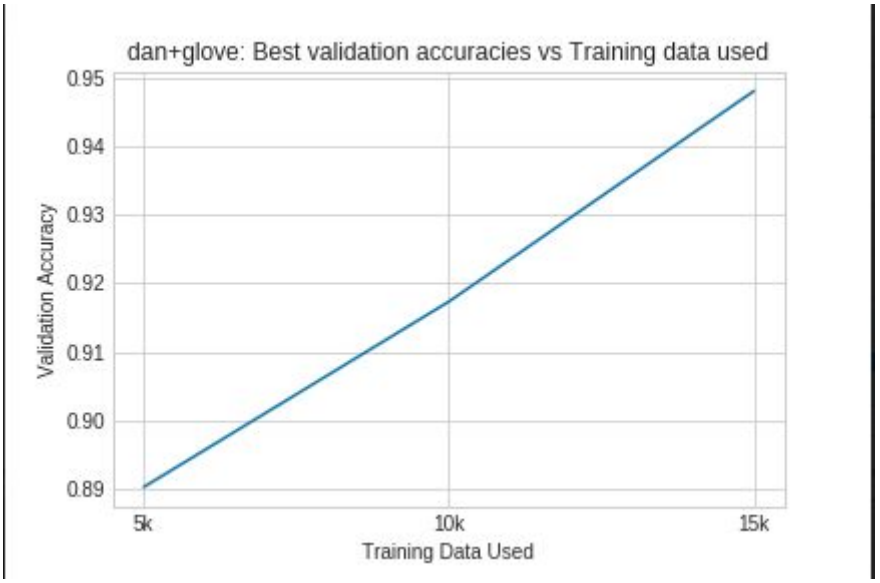
a. Performance_against_data_size_dan_with_glove, performance increases as the training data increases



b. Performance_against_data_size_gru_with_glove, performance increases as the training data increases



c. Performance_against_data_size_dan_with_glove with 50 epoch: performance increases as the training data increases as well as the number of epochs as it should happen.

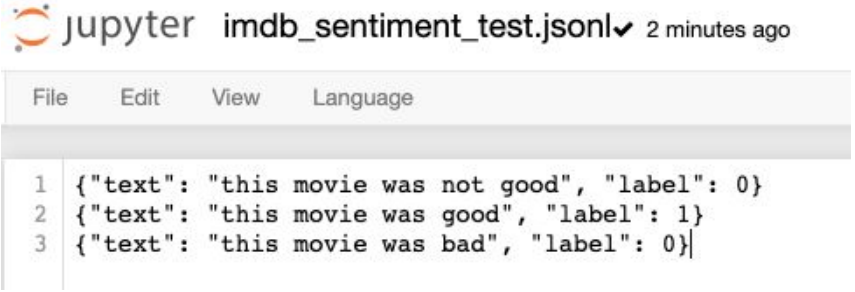


4. Error Analysis

a. **Case 1: DAN advantage over GRU: GRU is not good for learning simple languages as it cannot perform unbound counting. Whereas DAN performs well for simple language learning**

b. Below are the screenshots for analysis

c. Dataset tested on:



DAN Performance guessed all correct (accuracy: 1.0)

```
In [53]: !python3 predict.py serialization_dirs/main_dan_5k_with_emb \
                  data/imdb_sentiment_test.jsonl \
                  --predictions-file my_predictions.txt

100%|██████████| 3/3 [00:00<00:00, 121.21it/s]
2019-10-26 22:53:53.122021: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7f9fe5c8bc10 executing co
mputations on platform Host. Devices:
2019-10-26 22:53:53.122064: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host, Defa
ult Version
100%|██████████| 1/1 [00:00<00:00, 40.00it/s]
Making predictions
100%|██████████| 1/1 [00:00<00:00, 10.11it/s]
Saving predictions to filepath: my_predictions.txt

In [54]: !python3 evaluate.py data/imdb_sentiment_test.jsonl my_predictions.txt

Accuracy: 1.0
```

GRU Performance: could not guess all correct (0.67 accuracy)

```
In [55]: !python3 predict.py serialization_dirs/main_gru_5k_with_emb \
                  data/imdb_sentiment_test.jsonl \
                  --predictions-file my_predictions.txt

100%|██████████| 3/3 [00:00<00:00, 1980.31it/s]
2019-10-26 22:55:50.012602: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7fc91552beb0 executing co
mputations on platform Host. Devices:
2019-10-26 22:55:50.012657: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host, Defa
ult Version
100%|██████████| 1/1 [00:00<00:00, 3761.71it/s]
Making predictions
100%|██████████| 1/1 [00:00<00:00,  2.16it/s]
Saving predictions to filepath: my_predictions.txt

In [56]: !python3 evaluate.py data/imdb_sentiment_test.jsonl my_predictions.txt

Accuracy: 0.67
```

d. Case 2: GRU is good for testing double negations, however DAN cannot test double negations

In the sentence we have a double negation: “This movie was **not bad**”

Dataset tested on:

DAN Performance: could not guess it (accuracy: 0.0)

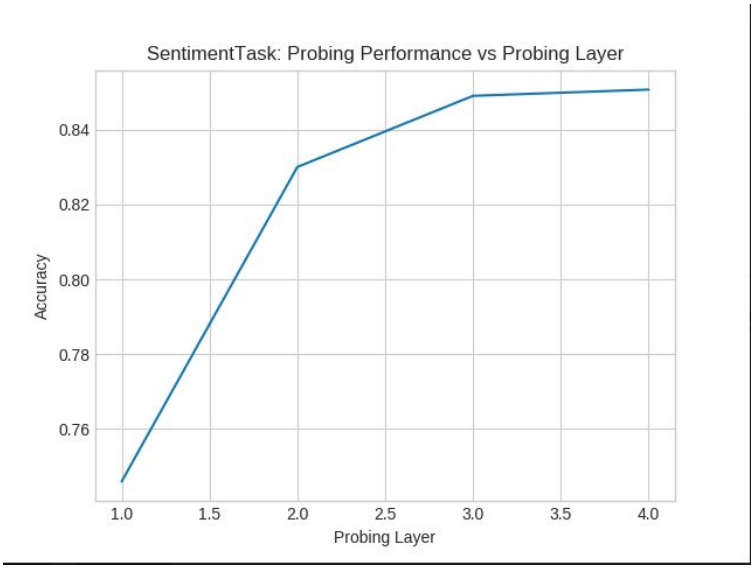
```
In [59]: !python3 predict.py serialization_dirs/main_dan_5k_with_emb \
                    data/imdb_sentiment_test.jsonl \
                    --predictions-file my_predictions.txt

100%|██████████| 1/1 [00:00<00:00,  4.24it/s]
2019-10-26 23:01:53.433357: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7f8201093450 executing co
mputations on platform Host. Devices:
2019-10-26 23:01:53.433397: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host, Defa
ult Version
100%|██████████| 1/1 [00:00<00:00, 4029.11it/s]
Making predictions
100%|██████████| 1/1 [00:00<00:00,  6.86it/s]
Saving predictions to filepath: my_predictions.txt

In [60]: !python3 evaluate.py data/imdb_sentiment_test.jsonl my_predictions.txt

Accuracy: 0.0
```

GRU Performance: guessed it correct (accuracy 1.0)

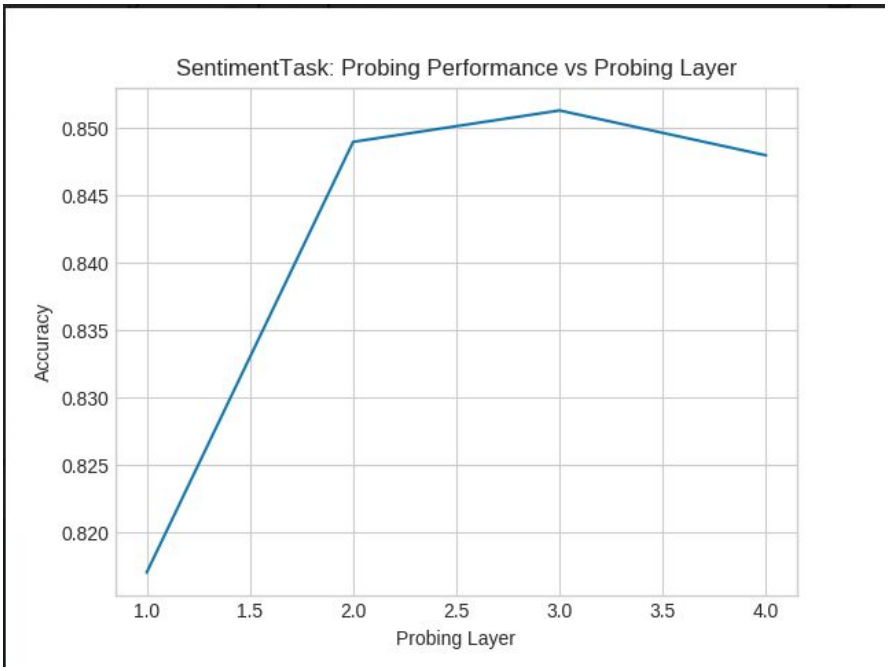
```
In [57]: !python3 predict.py serialization_dirs/main_gru_5k_with_emb \
                    data/imdb_sentiment_test.jsonl \
                    --predictions-file my_predictions.txt

100%|██████████| 1/1 [00:00<00:00, 1069.70it/s]
2019-10-26 22:59:47.067576: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7fe6dc5dac50 executing co
mputations on platform Host. Devices:
2019-10-26 22:59:47.067611: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host, Defa
ult Version
100%|██████████| 1/1 [00:00<00:00, 3030.57it/s]
Making predictions
100%|██████████| 1/1 [00:00<00:00, 10.46it/s]
Saving predictions to filepath: my_predictions.txt

In [58]: !python3 evaluate.py data/imdb_sentiment_test.jsonl my_predictions.txt

Accuracy: 1.0
```

5. Probing Tasks: plots and observations.

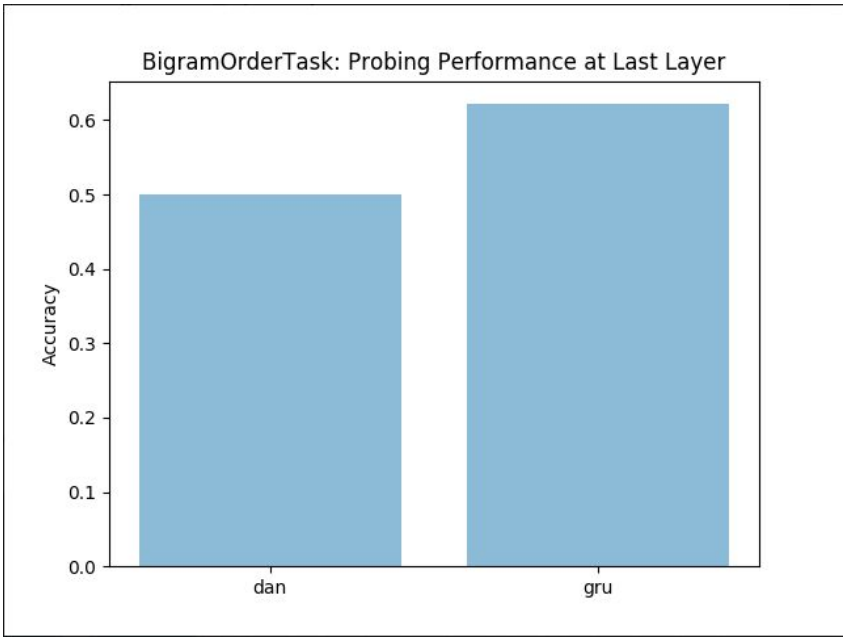
a. **For sentiment task DAN performance**. It gave an accuracy of around 84.5%. Accuracy increased with the number of layers in DAN as it should because more the number of feedforward networks, the more logistic regression like action we do and better the performance.



b. **For sentiment task GRU performance**. It gave an accuracy of around 85.5% Accuracy increased with the number of layers till 3 but then it decreased as it should have. This is because more layers diminishes GRU performance.

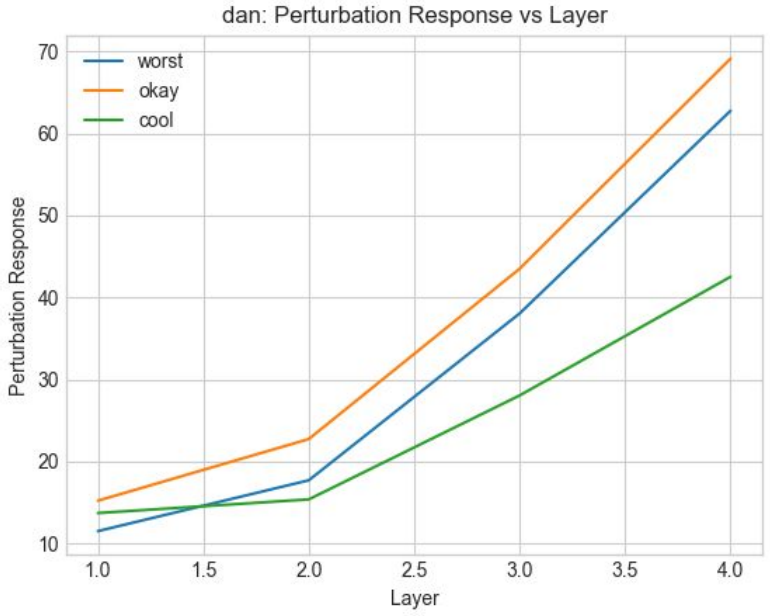


c. **Bigram Plots**: gru performs better as it remembers the ordering of sequences and uses layered stacks.

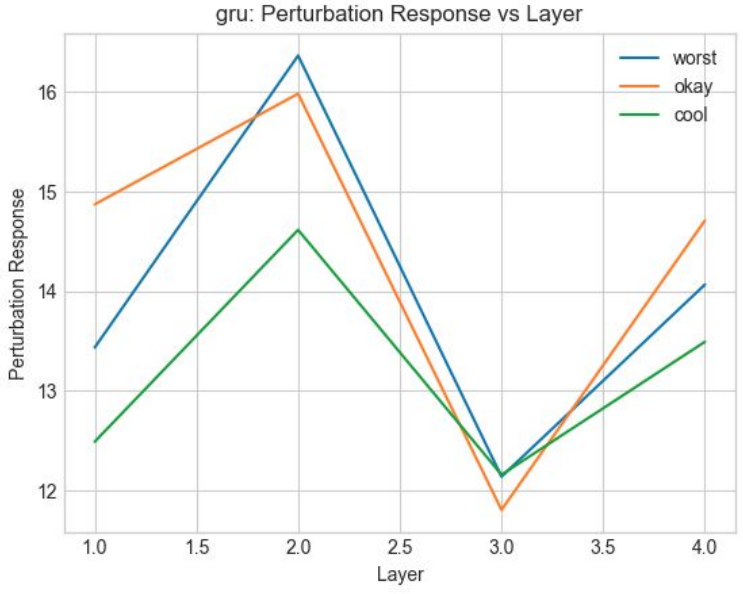


6. Perturbation Analysis

a. **<u>For DAN</u>**: as layers increase the difference between the observations increase for DAN as said in the reference paper. The first layer cannot pick up the differences properly. But as the layers increase the differences are visible properly

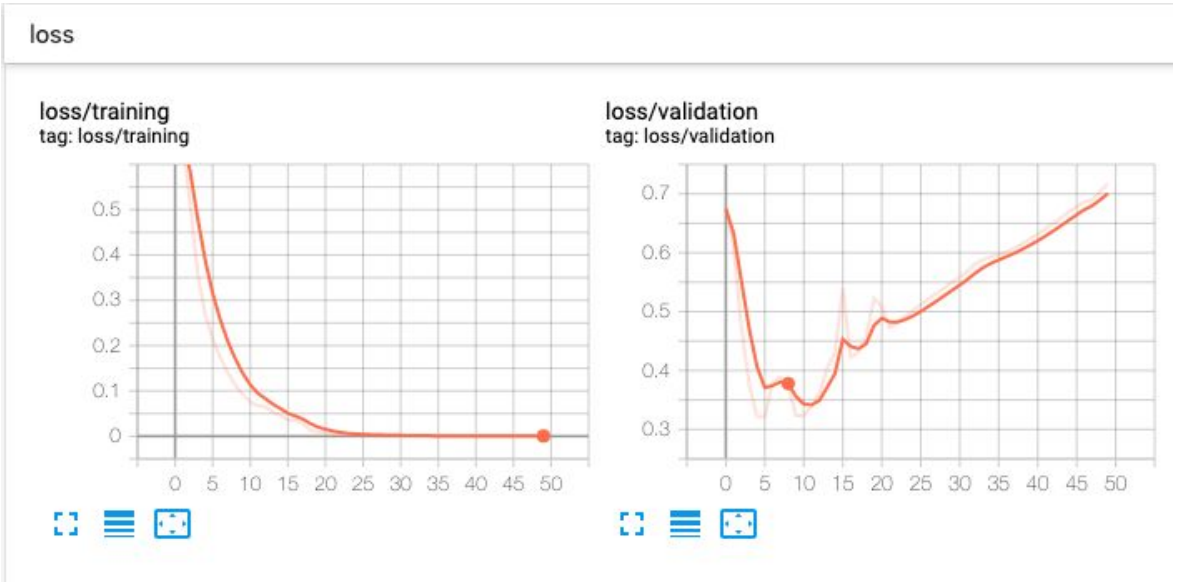


b. **Perturbation Analysis for GRU:** performance increases till 2 layers and then diminishes as is the case with GRU. Also, the difference in gap between these subtle differences is visible from the first layer itself unlike DAN. Thereby proving the effectiveness of GRU as compared to DAN.



c. Train/validation accuracy and losses on the tensorboard

Loss: with number of epochs the training loss decreased, but validation loss saw an increase after 10-15 epochs



Accuracy: with number of epochs the training accuracy increased and then became stable after 15 epochs, but validation accuracy saw a dip after 10 epochs and again an increase after 15 epochs then later it also became almost stable