Stony Brook University
Department Of Computer Science

CSE 538 Natural Language Processing - Fall 2019
Instructor: Niranjan Balasubramanian

Assignment 2

Madhusmita Dash (SBU ID: **112715430**)

1. Model implementation:

DAN: For implementing DAN, the steps were as follows:
a. In _init_(), define the keras.layers.Dense() with the output dimension as the embedding size and activation as Relu.

Rest of the steps were completed in _call_():
b. Apply the sequence mask to the vector input sequence to consider only the actual words and ignore the padding.
c. If the model is called in training mode, apply dropout to the input words in a sentence by sampling from a random uniform distribution and retaining those word indexes which have a probability more than the dropout probability a.k.a Bernoulli distribution.
d. For each feed-forward layer, apply the Dense layer to get the layer representation of the corresponding layer. The last layer output is the final sentence representation i.e. combined representation.

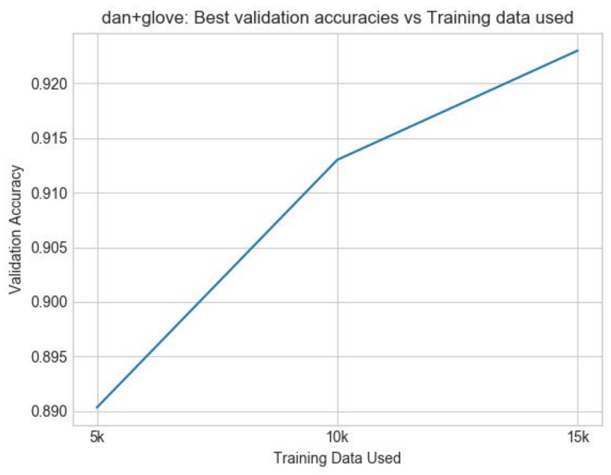
GRU: For implementing GRU, the steps were as follows:
a. In _init_(), define the keras.layers.GRU() with the output dimension as the embedding size. Set return_sequence = True to return the generated representation at each step to use it as input to the next layer with dimension as [batch_size,max_word_count,embedding_size]. Set return_state = True to return the final representation at the end of the sentence to use as our layer representation of the corresponding layer.
b. In _call_(), apply the GRU layer for the given num_layers to the input and also pass the sequence_mask to apply to the input.
c. Save the layer representation of each layer and store the last GRU layer output as the combined output after applying all the layers.

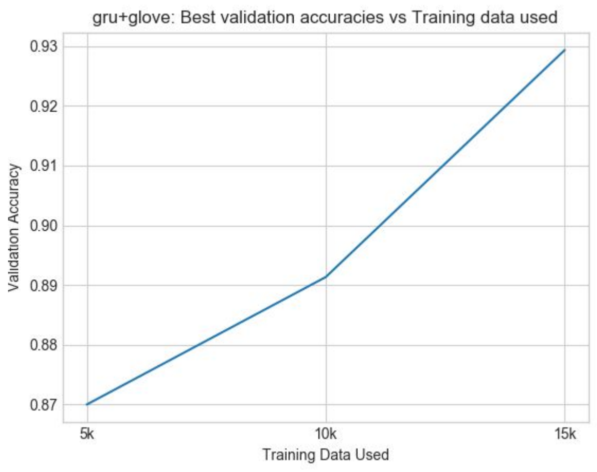
Probing model:
a. We extract the pre-trained model and pass our inputs to get the layer representation of each layer.
b. To apply the feed-forward layer, we take the layer representation of the given layer number (passed as a parameter) and get the logits vector of shape [batch_size, num_classes] on which softmax will be applied later to get the classified label.

2. Analysis:

Learning curves:

a. Increasing the training data

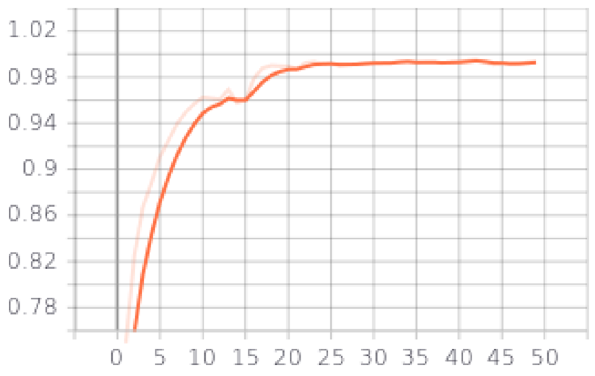


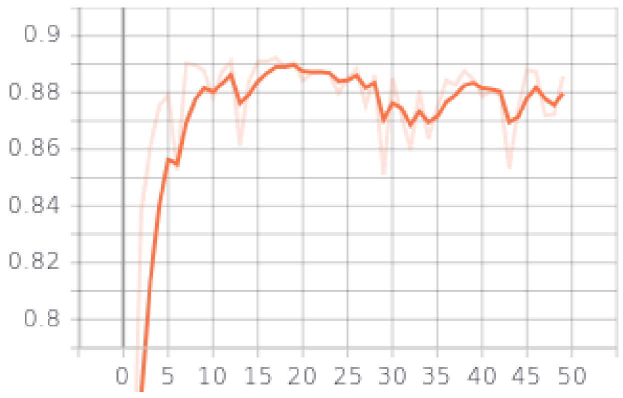


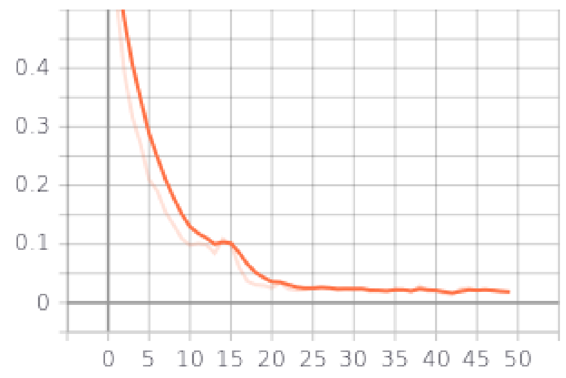
The accuracy increases more for GRU on increasing the training data than it does for DAN. The increase is steeper for GRU as more training data is provided.

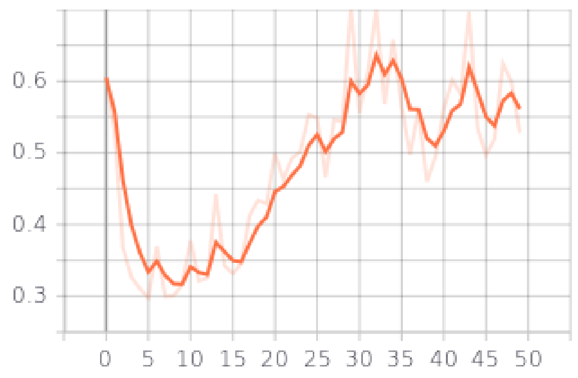
b. Increase training time (number of epochs)

The plots are as follows:
Top left: training accuracy, Top right: validation accuracy,
Bottom left: training loss, Bottom right: validation loss









The training accuracy increases and becomes stable after a few epochs (10-15) but the validation accuracy decreases after 10-15 epochs which means it is overfitting. This can be verified by checking the validation loss which increases after 10-15 epochs. The training error always reduces as we fit the data more and more.

2. Error analysis:

a. **Explain one advantage of DAN over GRU and one advantage of GRU over DAN.**

DAN is useful when we are learning only sentence-level labels (e.g. coarse-grained tasks like document classification). It uses Relu instead of sigmoid and tanh and does not use any complex function which is often the cause of vanishing gradients in GRU. This may be useful when the tasks where it is infeasible to consider fine-grained sentences. It magnifies the differences among meaningful words as the nonlinearity increases.

GRU is better than DAN in cases when the order of the words matter i.e context of the sentence. Though DAN amplifies the differences in words on increasing non-linearity, it does not consider order.

b. **Use the above to show and explain failure cases of GRU that DAN could get right and vice-versa. Use one of your trained models to do this analysis.**

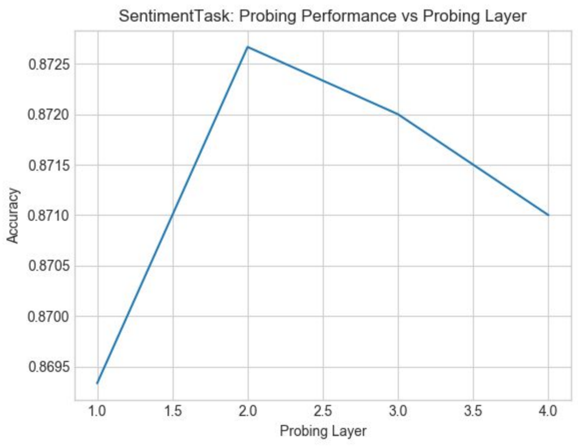
DAN performs well on perturbation tasks to amplify the differences between meaningful polarized words ‘ok’, ‘cool’, ‘worst’.

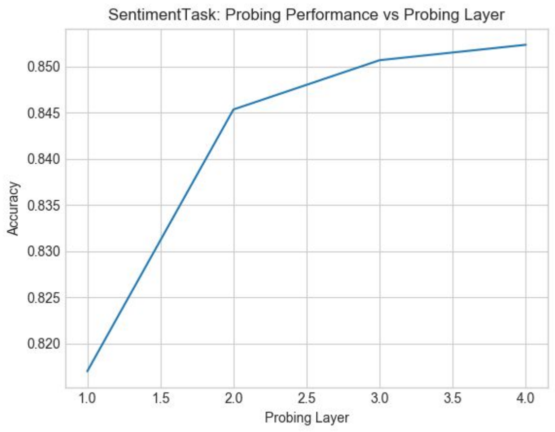
GRU performs better on bigram tasks to distinguish ‘New York’ from ‘York New’ whereas DAN considers both as same.

3. Probing Tasks:

a. Probing Sentence Representation for Sentiment Task

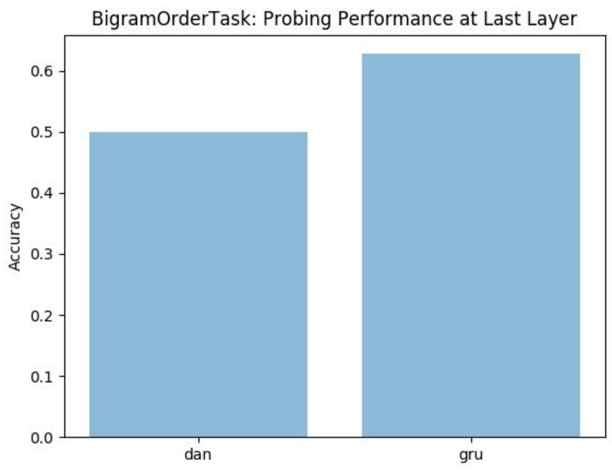
Left: DAN, Right: GRU





On the sentiment analysis task, DAN performs well for 2-3 non-linear layers but then the accuracy drops on adding more non-linearity as it amplifies the differences more. For GRU, the accuracy increases overall as we take the representation from later layers as it takes more words from the sentence to get the sentiment.

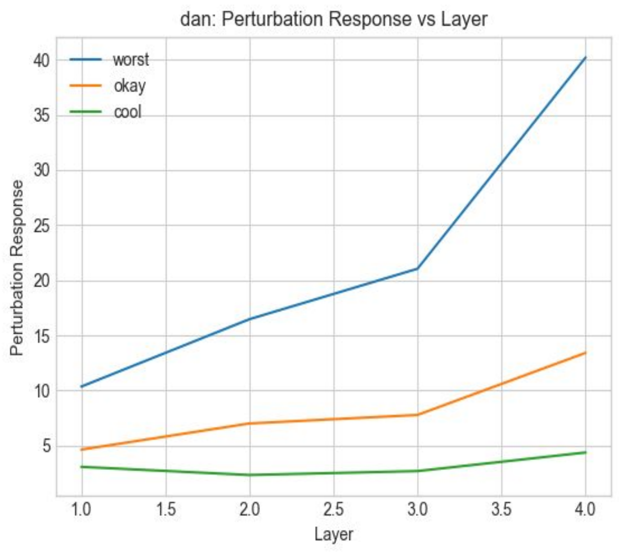
b. Probing Sentence Representations for Bigram Order

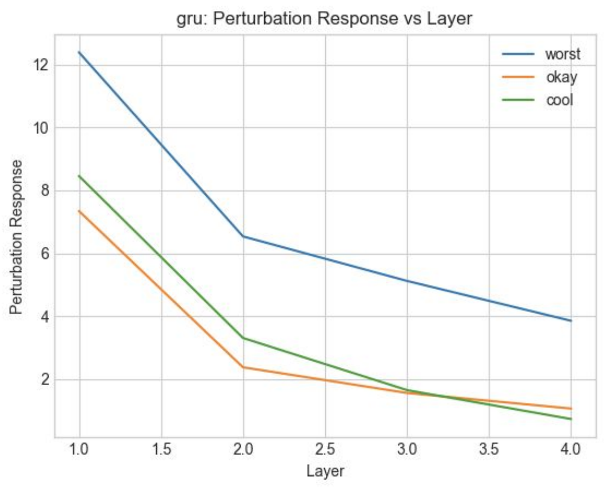


For the bigram task, we can see that DAN has 50% accuracy as it considers 'New York' and 'York New' as same. This is because it is a bag of words model and does not consider the order of words.

GRU performs well on the bigram task as it is context-dependent.

c. Analyzing Perturbation Response of Representations





In the perturbation test, for a sample sentence "the film performances were awesome", if we change the word awesome to ‘worst’, ‘okay’ and ‘cool’, we notice that DAN amplifies the differences among these words as the nonlinearity increases. Thus as increasingly negative words are perturbed, the DAN distinguishes more and more between these words.

However, in GRU we see that the differences among the negatively and positively polarized are meaningful. But as the number of layers increases, the differences start to reduce a little bit.

References:

https://arxiv.org/abs/1412.3555
https://people.cs.umass.edu/ miyyer/pubs/2015acldan.pdf