CSE 538 NLP Assignment - 2 Report
114707888, Shiva Sankeerth Reddy Yarradla

Section 1: Model Implementation

- Deep Averaging Network

As mentioned in the provided paper, **Deep Unordered Composition Rivals Syntactic Methods for Text Classification**, Deep Averaging Network works in the three steps:

1. It takes the vector average of the embeddings associated with the input sequence of word tokens.
2. It passes the said average through one or more feed-forward layers.
3. Perform a linear classification in the final layer.

My Implementation:

1. In the __init__ method, I first initialized my list of layers using torch.nn.ModuleList() and appended my Linear (torch.nn.Linear) and ReLU (torch.nn.ReLU) layers 'num_layers-1' times. I append another Linear layer to the end.
2. In the forward method, I used the Bernoulli Distribution, as mentioned in the given paper, to create a random tensor of shape same as vector_sequence. Then I chose instances where probability is greater than the dropout value i.e. 0.25, this becomes my word_drop_mask. I then applied the dropout mask on the vector_sequence.
3. According to the paper, the first layer of the network expects the average of the 'vector_sequence' as input. I calculated the sum of the vector_sequence from above as the numerator and then calculated the number of 1's in the 'sequence_mask' to get the number of words as the denominator. Then I calculated the average and passed it to the first layer.
4. Then I looped over each layer and passed the output of the first layer to the second layer and so on, while appending the output to a layer representation list. Then I returned the last layer's output as 'combined_vector' and the entire stack of the final output as the 'layer_representations'.

- Gated Recurrent Unit (GRU):

We use GRUs as an improvement over LSTM since LSTM involves computations for a large number of gates and GRU reduces the number of gates to just 2. The gates control the amount of information going to the output and since there are only two vectors that need to be computed it is an improvement over LSTM.

My Implementation:

1. In the __init__ method, I first initialized the GRU model using torch.nn.GRU with the number of recurrent layers as 'num_layers'. This model expects features in the dimension of input_dim.
2. In the forward method, as padding is required to accommodate variable length texts in a minibatch and since we do not want the GRU to process the trailing padding token, I used torch.nn.utils.rnn.pack_padded_sequence to pack a tensor containing padded sequences. I passed these padded sequences into the model.

3. We get the ‘output’ tensor containing the output features from the last layer of the GRU and a ‘final_hidden_states’ tensor containing the final hidden state for each element in the batch.
4. The layer_representation is the transpose of the above hidden state tensor and the combined_vector is the last element of the hidden state tensor.

- Probing Model:

A probing model helps us to analyze the performance of a given model layer by layer.

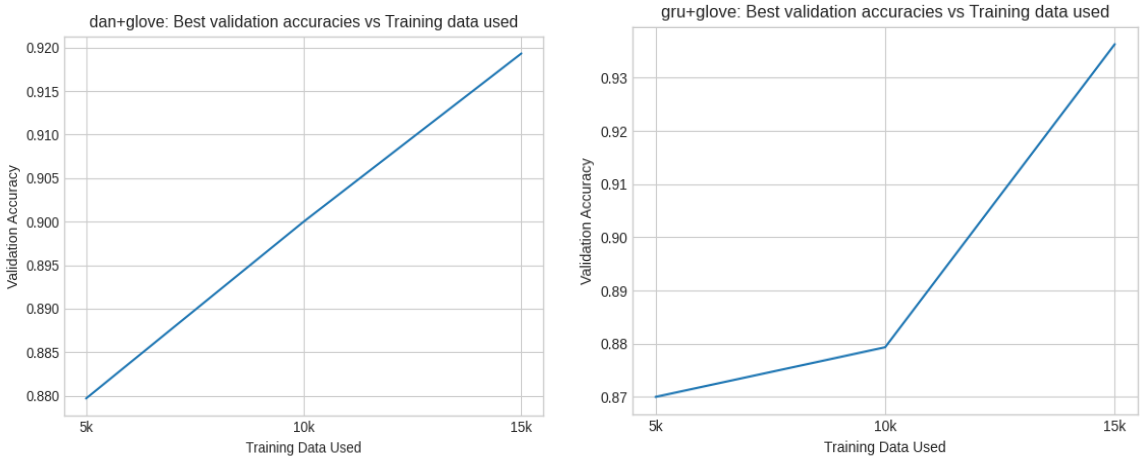
My Implementation:

1. In the __init__ method, I initialized my probing layer as a Liner layer using nn.Linear. The input dimension is input_dim and the output dimension is the class_num.
2. In the forward method, I applied the pre-trained model on the given input and derived the outputs.
3. I extracted the layer representations from the output and got the (n-1)th layer representation from it.
4. I then applied my linear probing model on the above (n-1)th layer representation to get logits.

Section 2: Analysis and Probing Tasks

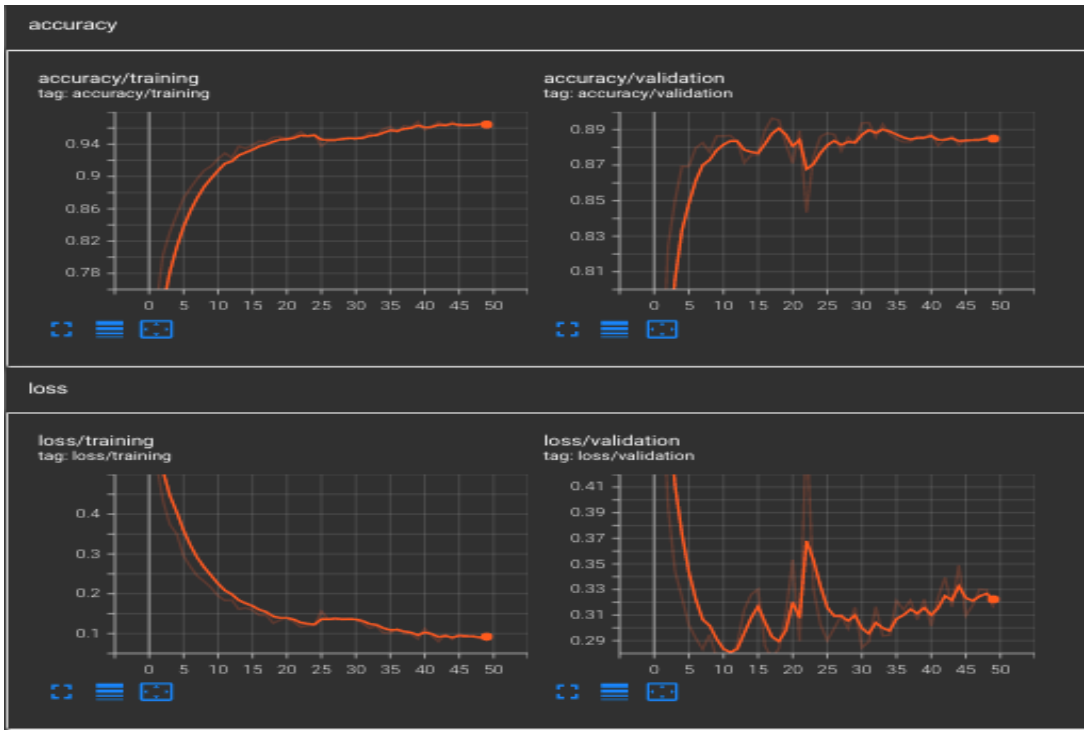
Learning Curves:

- Performance with respect to training dataset size



As can be seen from the plots, DAN's performance increases steadily as we increase the data size whereas GRU starts good, compared to DAN, but then its performance dips a little with the 10k IMDb file and then increases suddenly.

- Increasing the Training Time:



Increasing the training time has the following effects:

1. Accuracy: For training, it increases till some epoch 25, and then there is a slight decrease. But, from then on it becomes moves at constant. For validation, it starts with an increase till epoch 10, but there is a lot of variation in the region where validation loss was having fluctuations.
2. Loss: The training loss looks like a mirror image to the training accuracy and keeps on decreasing till epoch 25. Whereas, the validation loss fluctuates a lot after decreasing till epoch 10.

Error Analysis:

- Advantage of Deep Averaging Network (DAN) over GRU:
    - During the implementation, I observed that DAN takes less time to train in comparison with GRU. Moreover, from the above plot, I can infer that DAN is better at handling large training data than GRU. It also has better performance in terms of accuracy on the sentiment analysis task.

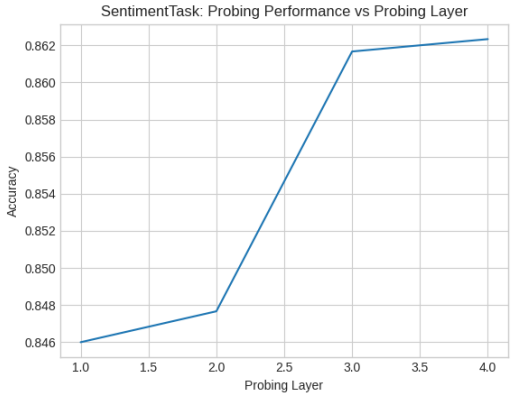
- Advantage of GRU over DAN:
    - GRU considers sequential information of the input which is ignored in DAN since DAN computes average, this results in GRU preserving important information. GRU also doesn't suffer from the vanishing/exploding gradient problem. It also performs better in the bigram ordering task than DAN.

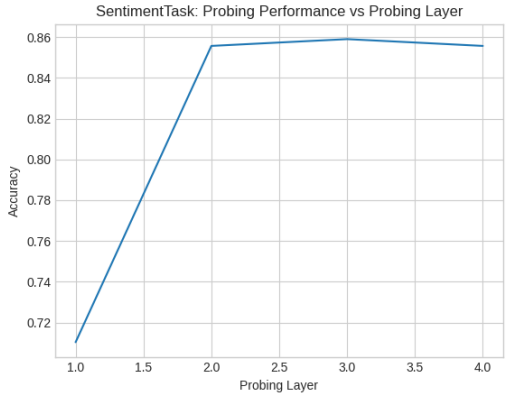
- Cases where GRU fails:
    - GRU tend to make mistake when the sequences get longer.
    - For example, 'The movie that Shireen told me to watch the other day at her place when we were planning to go camping turned out to be awesome'. Here, this sentence is very long and the subject is present in the beginning and the sentiment is present at the end.
    - By computing the average of the sequence, DAN only keeps the most important parts of the sentence representations. This way it gets rid of the problem coming from long sequences as above.

- Cases where DAN Fails:
    - Any task that will be affected by the order in which words appear will need sentence representations to change according to the said order.
    - For example, 'My colleagues work hard' and 'My colleagues hardly work '. Since DAN tries to compute the average over the above sequence representations, the averages tend to be the same, making DAN believe that both sentences represent the same meaning.
    - By using sequential information, GRU gets rid of the above problem.

Section 3: Probing Tasks

- **Probing Performances on Sentiment Task:**

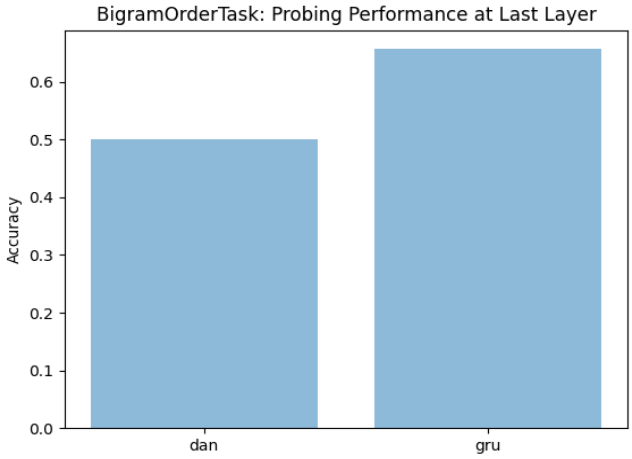




From the above graphs, we can infer the following:

1. The first graph is for DAN. For layer one, the accuracy is 84.6%. There is a slight linear increase to 84.8% for layer two. For layer three, there is a good increase of about 2% at 86.2%. For layer four, there is a negligible increase in accuracy.
2. The second graph is for GRU. For layer one, the accuracy is about 71%. But for layer two, there is a huge improvement of about 14% at 85%. But this same accuracy is maintained for both layers three and four.
3. I attribute the mathematical nature of the functions the word representations are passed through (Average for DAN and Sequential for GRU) for the above results.

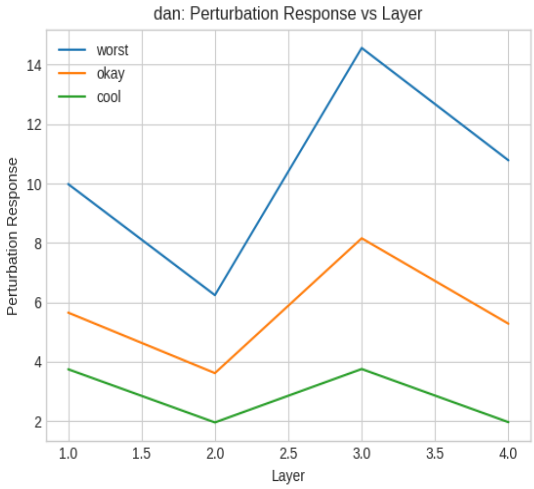
- **Probing Performances on Bigram Order Task**

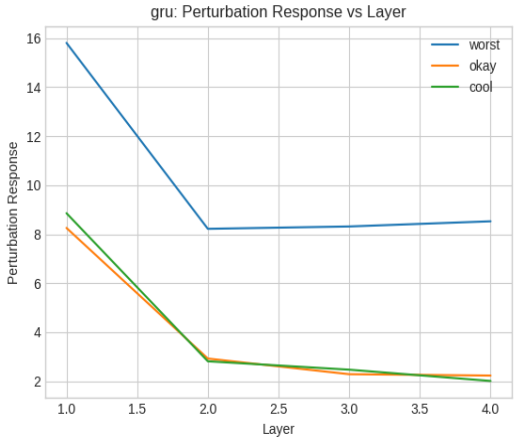


1. For the Bigram order task, there is an almost 15% difference in performance between GRU and DAN.
2. DAN, with just 50% accuracy, is not an effective model in this task as it is a fine-grain task.

3. This can be attributed to the fact that GRU considers word ordering while DAN just computes the average of the representations.

- **Perturbation Analysis:**





1. Perturbation response helps us in understanding how small changes get magnified as the input is passed between different layers.
2. From the above graphs, we can see that DAN is better able to represent the words ‘worst’, ‘okay’, and ‘cool’ than GRU.
3. I think it is due to the following reasons:
    a. The DAN uses ReLU as activation, helping the word divergence to grow farther at each layer.
    b. We use a squishing function in GRU (sigmoid) at each timestep, which smoothens and smashes the value because of exponentiation, keeping it within a bound.