1. Implementation

1.1 DAN

Initialization: Create *nn.ModuleList*, add *nn.Linear(input_dim, input_dim)* and *nn.ReLU()* to it for each layer. *nn.Sequential* is not applied here because it does not allow us to flexibly adding functions. Plain python list is also not applied[1], because the parameters won't be registered properly and they can not be passed to the optimizer.

Dropout: Initialze a *mask* with all one's for every batch. When *training* is True, multiply it by *(1-dropout)* then pass it to *torch.bernoulli* in which every item is independent, which then results in a *[0, 1] mask* according the dropout rate. In order to filter out those padding tokens, simply multiply the mask of batch *i* by *sequence_mask[i]*, which is equivalent to an AND operation. In this way, we can select those vectors for training through *vector_sequence[i][mask > 0]*.

Layer processing: First calculate mean for selected vectors. Then we apply functions in the *ModuleList* sequentially, with all the intermidiate results and final results saved.

1.2 GRU

Intialization: Use *nn.GRU* with config of same input and hidden dimensions,given layer numbers, and also *batch_first=True*.

Layer processing: Calculate valid lengths by *sequence_mask.sum(dim=1).to("cpu")* for every batch(it has to be a 1D CPU tensor thus "cpu" is set) before we pass everything to *pack_padded_sequence*. Pass the packed results to *nn.gru* thus we get the results.

1.3 Probing

Intialization: Simply intialize a linear classification network.

Forward: Choose the layer we want from pretrained model by 0-based index, then apply it with the linear classification we defined above.

---

[1] "PyTorch : How to properly create a list of nn.Linear() - Stack Overflow." 22 May. 2018, https://stackoverflow.com/questions/50463975/pytorch-how-to-properly-create-a-list-of-nn-linear. Accessed 1 Oct. 2021.

2. Analysis

2.1 Learning Curves

Performance with respect to training dataset size:

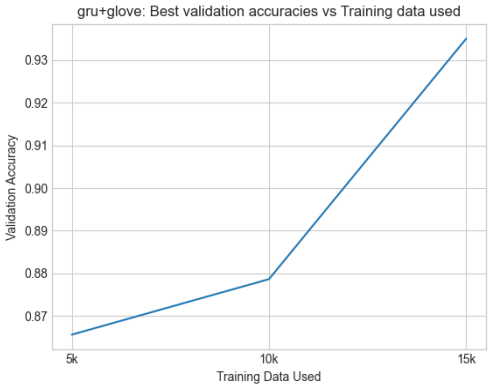


Figure 1: GRU accuracy / training data size

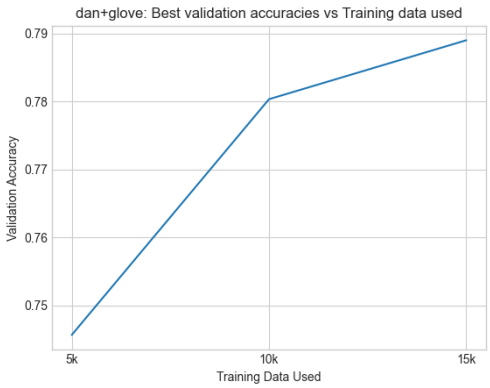


Figure 2: DAN accuracy / training data size

The accuracy against different traning epochs and data size

data size	5k	5k	10k	10k	15k	15k
Model	4-th epoch	8-th epoch	4-th epoch	8-th epoch	4-th epoch	8-th epoch
DAN	0.71	0.74	0.77	0.78	0.78	0.79
GRU	0.87		0.88		0.94	

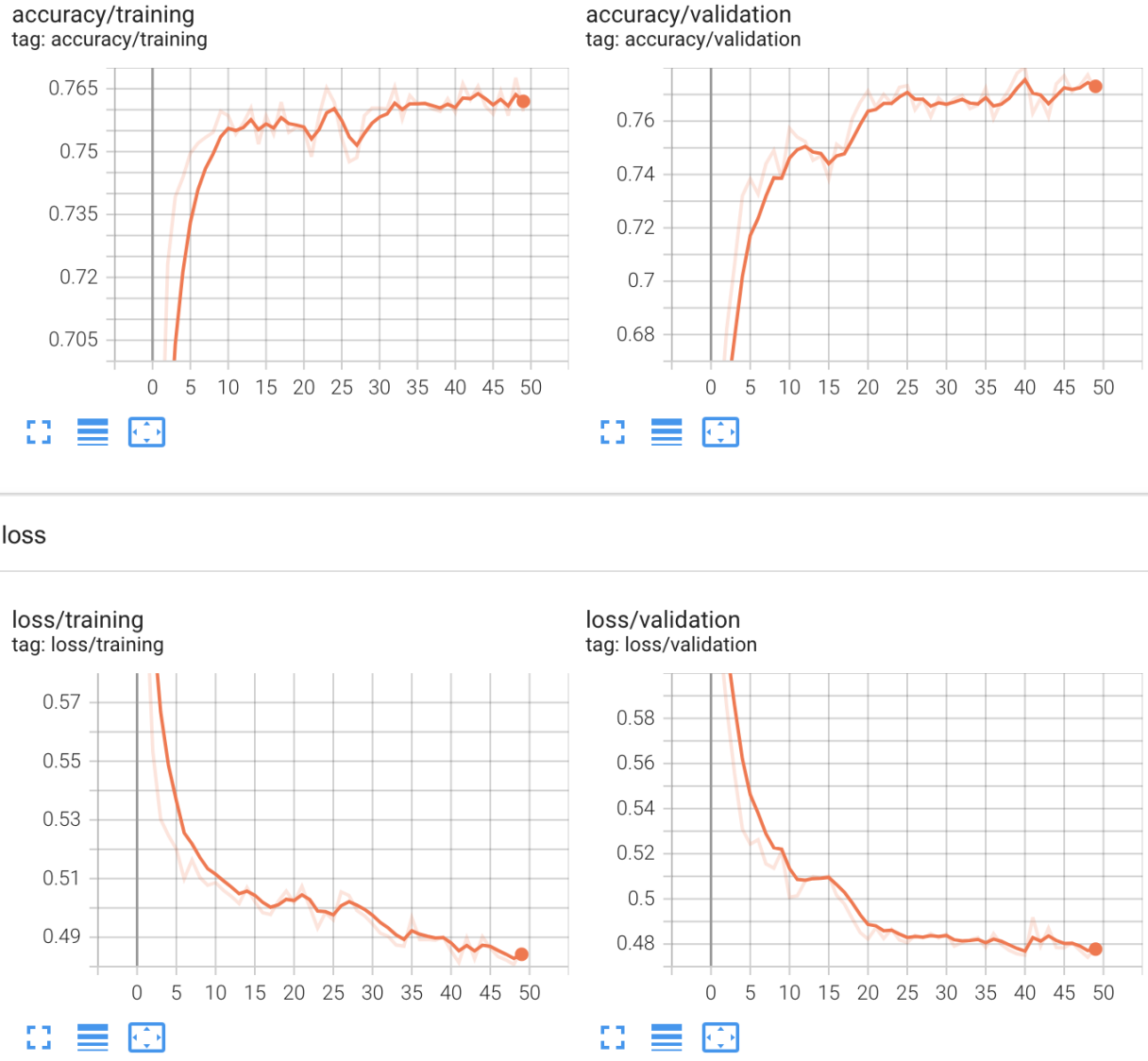
From the figures above, we can gain some insights:

1. The accuracy increases for both GRU and DAN when training data increases.

2. No matter the final or intermediate accuracy, GRU performs better than DAN.
   Possible reason is that, DAN is like bag of words, or a unigram model, which does not take words relation and orders into account, while GRU models the entire sequence.

3. GRU accuracy increases prominently when data size becomes large, however, accuracy increase rate slows down when it comes to DAN.
   The reason could be, for sentiment analysis, there are always lots of irrelevant words inside a sentence, GRU has both input and forget gates that solve such kinds of problem. As a result, GRU works better on large training dataset. Oppositely, DAN takes every word into account, thus its accuracy grows not as fast as GRU.

Performance with respect to training time(for DAN):



From the figures above, we can see the accuracies are growing and losses are decreasing for both training and validation set at around epoch 1 to 10.

However, after that, the loss and accuracy for both training and validation set start to fluctuate. it could be of that the model is learning is to be more "sure" of results. Scores are changing, but none is crossing our threshold so our prediction does not change.

There is an obvious divergence at around epoch 40: loss goes down and accuracy increases for training data, while loss increases and accuracy decreases for validation set, which indicates that we could stop the training as it is a clear signal for overffiting.

2.2 Error Analysis

Explain one advantage of DAN over GRU and one advantage of GRU over DAN.
1. DAN over GRU:
DAN trains fast, while it is slow to compute gradients for GRU thus it is much slower than DAN. Also, DAN is better used in cases when order is not considered important, or every word in a phrase is equally important to the overall meaning.

2. GRU over DAN:
As GRU has cell state, it can infer meaning with word behind while skipping the irrelevant ones. Thus it performs better on an order-sensitive bi-gram task.

Use the above to show and explain failure cases of GRU that DAN could get right and vice-versa. Specifically, run your trained models on example test cases (you are free to choose from the datasets or construct your own) and record the success and failure cases. You will see that DAN is able to predict some cases correctly where GRU fails to and vice-versa. Now use these specific cases to portray the one advantage of DAN and the one advantage of GRU that you mentioned.

The following error cases are extracted by performing the two models trained from 15k dataset on imdb_sentiment_test.jsonl and bigram_order_test.jsonl:

"I don't know how anyone could hate this movie. It is so funny. It took a unique mind to come up with this storyline. It's not your typical alien movie. These aliens are so stupid and confused. You need to rent it at least once." (sentiment analysis)
❌ DAN: 0
✅ GRU: 1
This is a long sentence and the actual useful information are "*It is so funny*" or "*don't know how*"+"*could hate*". But not "*hate this movie*" or "*so stupid and confused*". Thus it is hard for DAN to predict the correct result.

"movie this" (bigram analysis)
❌ DAN: 1
✅ GRU: 0
In this phrase, word order is important. DAN does not consider the order, thus it appears frequently in DAN's error result while GRU gets it all right.

3. Probing Tasks

3.1 Probing Performances on Sentiment Task

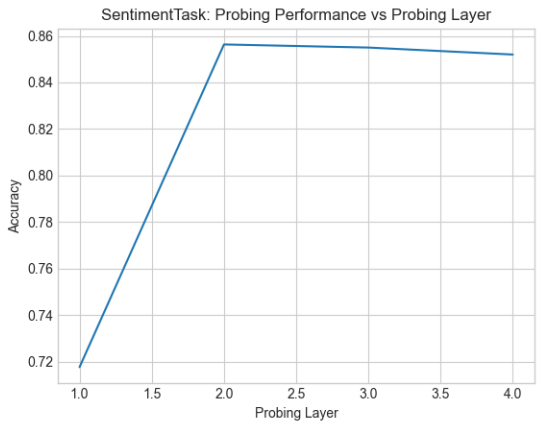


Figure 3: GRU probing performance

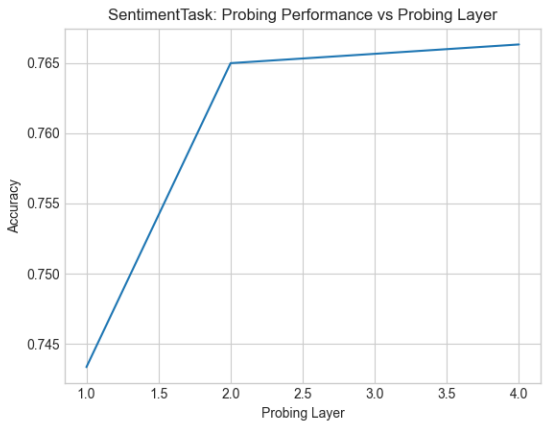
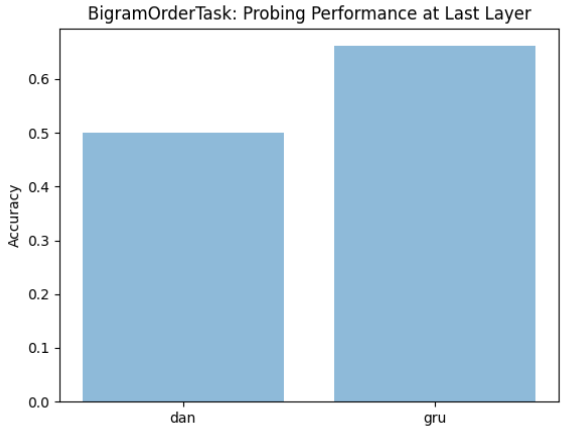


Figure 4: DAN probing performance

We can see that performance increase for as layer number increase from 1 to 2, but slowns down and even becomes worse for GRU. It could be due to that this task prohibits the model from capturing task-specific contextual patterns, and instead only utilizes the information already present in the representations.

3.2 Probing Performances on Bigram Order Task



The probing performance are not good for both the models on bi-gram task, even worse for DAN, which proves that DAN performs bad on order-sensitive tasks, and also GRU performance on bi-gram is not as good as sentiment analysis which inferred from lots of contextual embeddings.

3.3 Perturbation Analysis

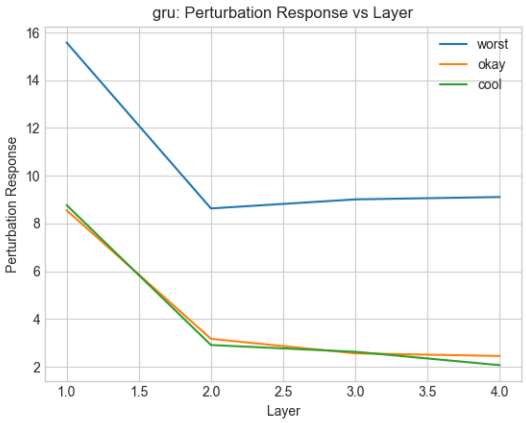


Figure 5: GRU perturbation response

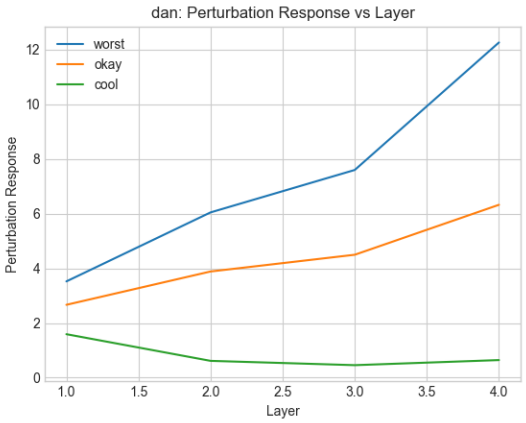


Figure 6: DAN perturbation response

These plots show that DAN is capable of distinguish sentences with slightly differences, as they are more vulnerable to adversarial examples, which can be used to prevent overfitting. However, it becomes hard for GRU to tell the differences especially when layer increases, as it more relys on the contextual meanings.