CSE 354 Assignment 2 Report Spring 2022

JeongYoon Lee (114133199)

April 11, 2022

Part 1. Model Implementation

>>>class DanSequenceToVector(SequenceToVector)

In the __init__ function, I made n-1 of hidden layers using nn.ModuleList to save n-1 nn.Linear layers. And to use dropout, made self.dropout, and for the last nth layer, made one nn.Linear layer to not use with Relu activation function.

In forward function, using self.dropout, turn into 0 values based on the probability of dropout that we got as parameter in init function. And then make sequence_mask to make it to 3D tensor, and then do vector multiplication with vector_sequence which is 3D tensor. After that, using mean function to make it 2D tensor and averaging each embedding dimension.

For the n-1 hidden layers, using for loop to implement Relu function for each layer and got the return value like combined_vector and layer_representations.

After that, we can call self.last_layer without activation function, and then return the combined_vector and layer_representations.

>>>class GruSequenceToVector(SequenceToVector)

In the __init__ function, make nn.GRU layer to use it in forward function.

In the forward function, padding the vector_sequence which is the result of word embedding using pack_padded_sequence. After that, add new_vector_sequence which is the return of padding step in the gru_layers. The second return value for this gru_layer is the list that has all of the output of current layer for each state. We can find the combined_vector in layer_representations since comined_vector is the last state output of it.

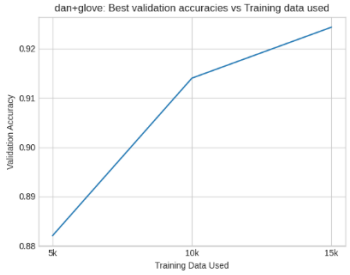
>>>class ProbingClassifier(nn.Module)

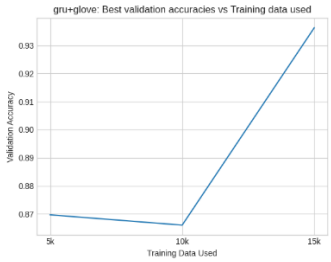
In the __init__ function, I used LazyLinear to simplify the densor layer while probing.

In the forward function, we can send inputs to pre_trained_model and then we can find the layer_representations from those return value. So we get the representation for the particular layer with this return value, which is nth (= pretrained_output[‘layer_representations’].

Part 2. Analysis

1. Learning Curves

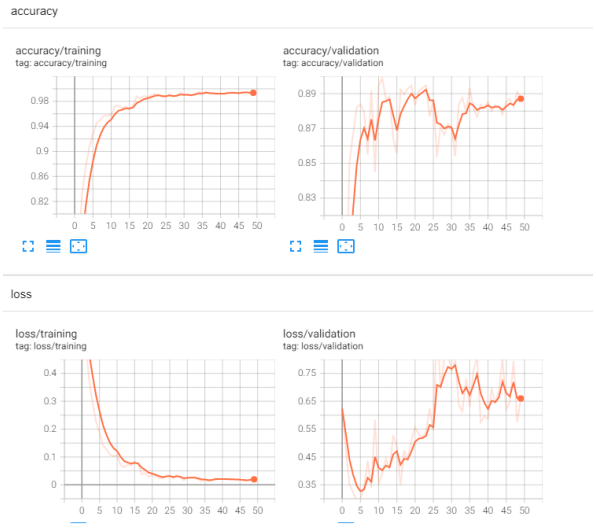




[Performance against data size dan with glove] [Performance against data size gru with glove]

We can find that the more training data, more higher accuracy value for validation. In the dan model, the accuracy increased noticeably even with a small increase in the training set, but the accuracy of the gru increased only after the training set reached 15k. So we can find that the dan model is much accurate even the number of training dataset is not that big.

2. Performance with respect to training time



[Dan model with 50 epochs]

3. Error Analysis

We can see that overfitting occurs from 5 or more layers in the dan model with 50 epochs. If there are too many layers, even the accuracy is high, but if you see the large fluctuations in the validation loss, we can find there's some problem, so we can get help when performing fine-tuning.

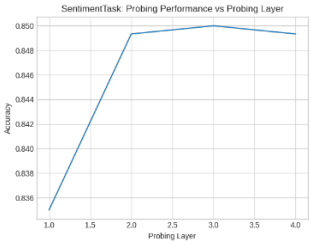
Dan model: For dan, it has an advantage in catching similar relationships in words. Also, similarly, the difference between words with no similarity can be found easily. It also has the advantage that the learning time is very fast.

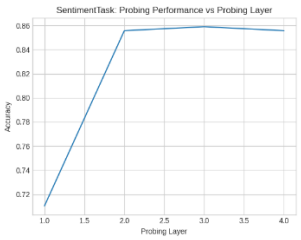
Gru model: In the case of gru, the advantage is that there is no long term memory problem. Unlike dan, it can catch the order of words, which can be checked in the bigram task.

Similarly, since the DAN model does not care about the order of words, it can be vulnerable when different meanings are created with several similar words. Conversely, in the case of gru, this can be possible.

Part 3. Probing Tasks

1. Probing Performances on Sentiment Task





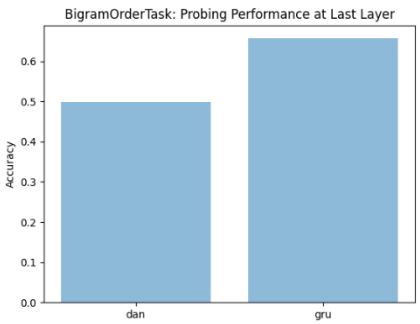
[Probing performance on sentiment task] [Probing performance on sentiment task gru]

The purpose of the probing model is to objectively verify the performance of each sentence embedding. By controlling the structure of the fine-tuning network, it is possible to objectively check how the embedding quality affects downstream tasks. In other words, it is possible to determine what kind of information the output from each layer has been contextualized.

In the case of dan, the accuracy is highest when the probing layer is 3.0, and same for the gru model. In both models, it can be seen that the accuracy increases rapidly from layer 2.

I thought it should have more difference between dan and gru since they have different way to training, but my results are almost same. But the common thing is that each model improves less for the initial feedforward layers, so we can make it simple than higher layer while implementing it.

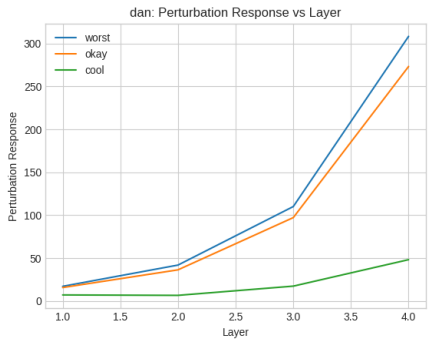
2. Probing Performances on Bigram Order Task

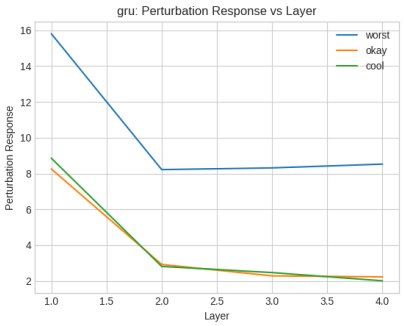


We can see that the gru model performs better than the dan model in the bigram order task. In gru, an update gate and a reset gat are added to determine how to reflect the immediately preceding information or past information. The model that reflects this past state has more strength than dan in the order.

For the dan, we can see that the accuracy is almost half(0.5), so we can guess the order is not affecting in dan model.

3. Perturbation Analysis





In perturbation analysis, like in our paper, DAN showed that words with positive expressions similar to awesome did not increase from a small value of l1 distance, but perturbation response increased in the case of words with negative expressions which is totally different meaning with the original word(awesome). On the other hand, the gru model had a large l1 distance from the initial layer, and there was little change in each result after a certain layer. Like dan, the perturbation response value for the worst word was the largest, but we can see that the value decreased compared to the initial layer and stuck with the same value after second layer.

Since dan averages the combinations of words, if the similarity to the replaced word is low, the distance value will inevitably increase. On the other hand, in the case of similar words, a small perturbation response is produced because they have similar meanings.

In the case of gru, we can see this model cannot find the difference between the words well. So this model treats same with okay and cool in this sentence.