NLP HW3

Tiago Antunes 2020280599

April 2021

1 RNN model description

For this homework I implemented a vanilla RNN model from scratch. Table 1 shows the results obtained in the test dataset. Both Figure 1 and Figure 2 show the accuracy and loss at each training epoch of the best model parameters.

I find relevant to mention the pre-processing that was done on the data. For loading the dataset, I used a python library called *pytreebank* which is designed to load and manipulate the given data. After loading, I just use the entire sentence for each training label, by creating *(sentiment, sentence)* pairs. Each sentence will then be transformed into numbers using the ids from the glove embeddings file, and so each sentence becomes a list of numbers as its representation.

Although RNN is a class of models that allows for variable size input, it is desirable for data to be batched and for simplicity of implementation padding is preferred. A mask is also generated, where it is 1 when the value exists and 0 when it doesn't. Both the data and mask are fed into the model, which will then perform the forward operation. Although this incurs in some extra processing in the forward function, I expect the back-propagation not to be greatly affected by it, and still got very decent run times. Therefore, to update the hidden state $h_{i,j}$, where i is the hidden layer index and j is the input index, is given by:

$$h_{i,j} = tanh(W_{A_i} \cdot h_{i-1,j} + W_{B_i} \cdot h_{i,j-1} + b_i) \cdot m_i + h_{i,j-1} \cdot (1 - m_i)$$

where m_i is the mask at word of index i of the sentence. This works in a batched way and will take fully usage of vectorization, easing the calculation of a batched forward function. The idea of this function is the same as of the Sigmoid function, where it creates a conditional function in a calculation. m_i will either 0 or 1, W_{A_i} and W_{B_i} are trainable matrices. In the implementation, these 2 matrices are concatenated as 1 bigger matrix, which yields the same results.

2 Results

All models followed the same pipelined procedure with early stopping after 20 epochs without improving on the validation dataset and the resulting model

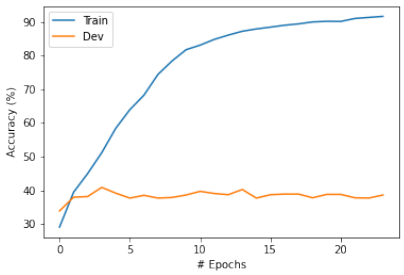

Figure 1: Accuracy per epoch

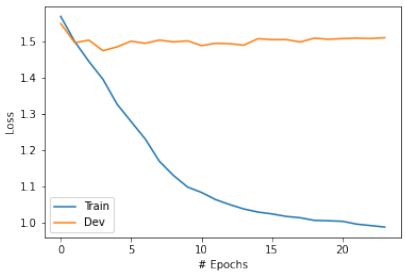

Figure 2: Loss per epoch

Dropout	Hidden Size	#Hidden	Embeddings	Accuracy	Loss	Train acc	Val Acc
0	256	1	glove	0.3928	1.4896	0.5802	0.3924
0	256	1	default	0.3204	1.5622	0.5471	0.3143
0	256	3	glove	0.4018	1.4856	0.5709	0.4051
0	256	3	default	0.2652	1.5807	0.3298	0.2607
0	512	1	glove	0.371	1.5067	0.5218	0.3742
0	512	1	default	0.2733	1.5785	0.336	0.2552
0	512	3	glove	0.3747	1.5247	0.6114	0.3787
0	512	3	default	0.2937	1.5758	0.3	0.2779
0.2	**256**	**1**	**glove**	**0.4131**	**1.4777**	**0.5715**	**0.4087**
0.2	256	1	default	0.324	1.5624	0.7869	0.3079
0.2	256	3	glove	0.3937	1.498	0.4306	0.3896
0.2	256	3	default	0.2914	1.5831	0.3828	0.2888
0.2	512	1	glove	0.3941	1.4842	0.5242	0.3815
0.2	512	1	default	0.3208	1.5707	0.8704	0.3152
0.2	512	3	glove	0.3575	1.5334	0.5138	0.3688
0.2	512	3	default	0.2864	1.5874	0.3663	0.2997

Table 1: Results obtained from all trained model parameters

would be the one that yielded the highest value in it. 20 epochs was used as the models would improve after 10 epochs sometimes. The loss function that was used was CrossEntropyLoss, and an Adam Optimizer with default values was used.

After analyzing the results in Table 1, the best model was **Dropout=0.2, Hidden Size=256, Hidden Layers=1, Embeddings=Glove**. Two conclusions can be taken from the implemented model:

- Bigger models didn't necessarily obtain better results. This has to due with the limited amount of data that is available, that will eventually overfit without generalizing due to its limited amount.
- Pre-trained embeddings not only speed up training but yield much better results. The models that didn't use GloVe had a much higher test accuracy compared to the non pre-trained ones.
  - When training bigger models, it gets even worse results as it needs to learn the relationship between words too, so it should get even more data.

By using dropout the model forced some connections to learn content. In bigger models, this works in a worse way, as the performance will decrease due to lack of sufficient data, but in smaller models yielded very good results. Comparing the results of different number of hidden layers and hidden sizes, it is noticeable that they have a proportional effect on the results and if they're both too high, the data problem will start existing. Overall, they didn't increase the performance of the model, and just use a smaller model worked fine.

3 Conclusion

Overall, the performance is not human level. The accuracy is low even within this dataset and it will quickly overfit the training data. RNN is also a very simple model, and perhaps using other improvements (for example, LSTM or Bidirectional LSTM) would be able to improve its performance. Also, using a dataset with more data would be desirable, since the data used in this case is very basic.

As for the optional task, using a Tree structure RNN would probably also perform well, given that the dataset contains data for that. It is likely that those models perform better in it.