Coding Assignment 2 (Hackation) Report

Jihun Jeung

20211155, jihun@gm.gist.ac.kr

Life Data Mining Lab., School of Life Sciences, GIST

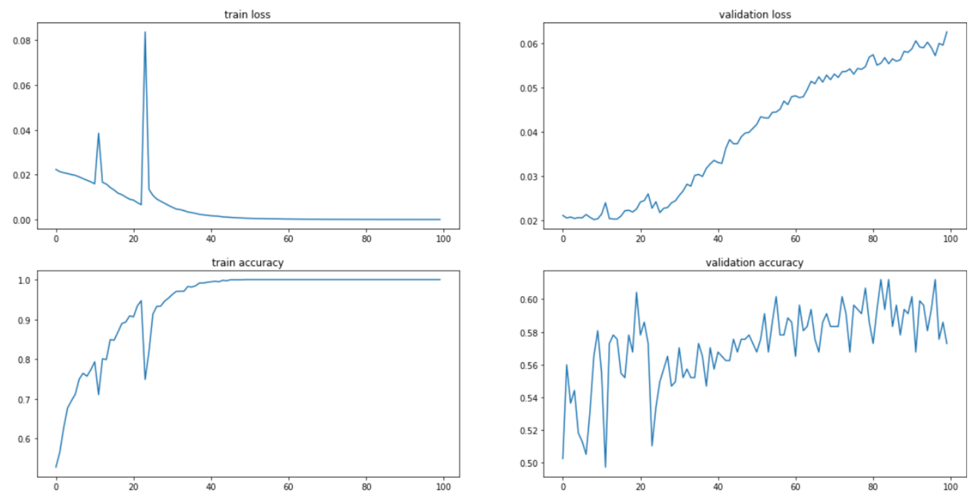
Report1. Construct the dataset

	Value
the total number of unique words in T	24473
the total number of training examples in T	2000
the ratio of positive examples to negative examples in T	1.0 (2000/2000)
the average length of document in T	187.677
the max length of document in T	3816

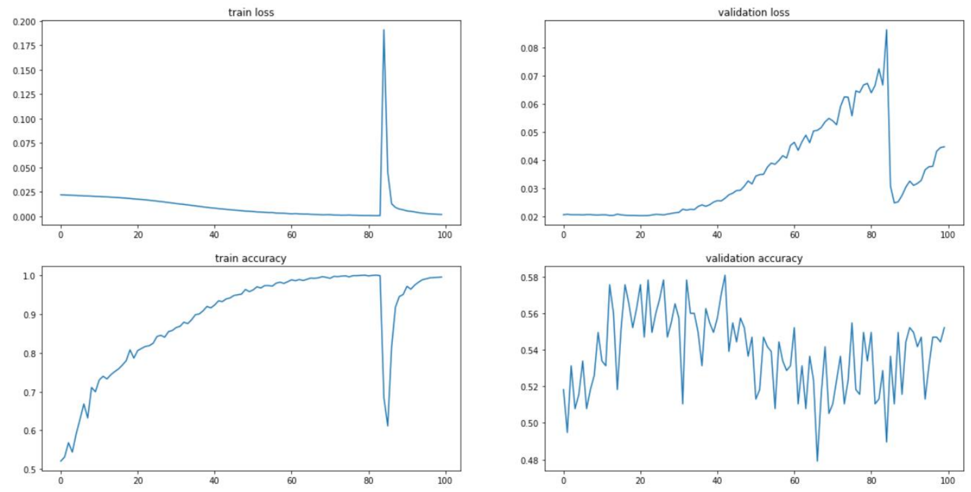
Report 2.

	validation accuracy	Training time (in seconds)
RNN w/o pretrained embedding	0.510417	7.318356990814209
RNN w/ pretrained embedding	0.507812	7.760797500610352
CNN w/o pretrained embedding	0.557292	459.2913086414337
CNN w/ pretrained embedding	0.593750	455.0342073440552
the max length of document in T	200	200

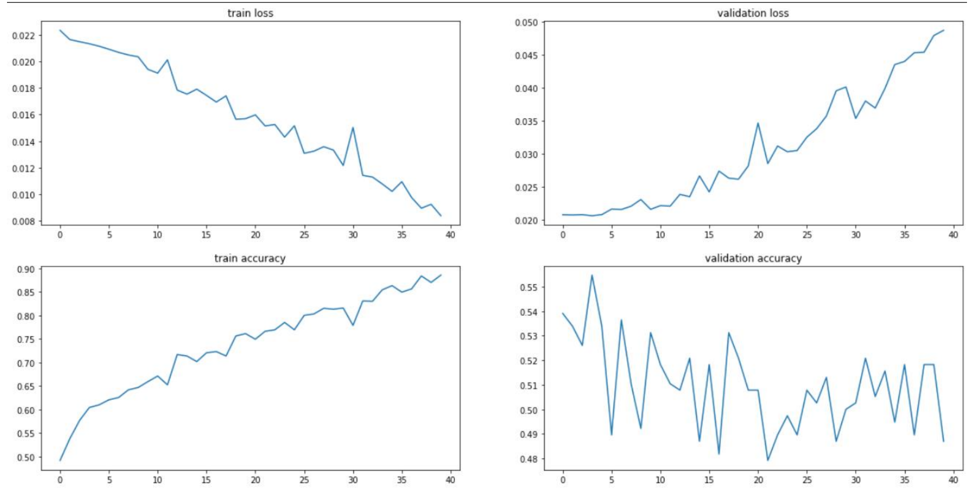
Report 3: RNN w/o pretrained embedding



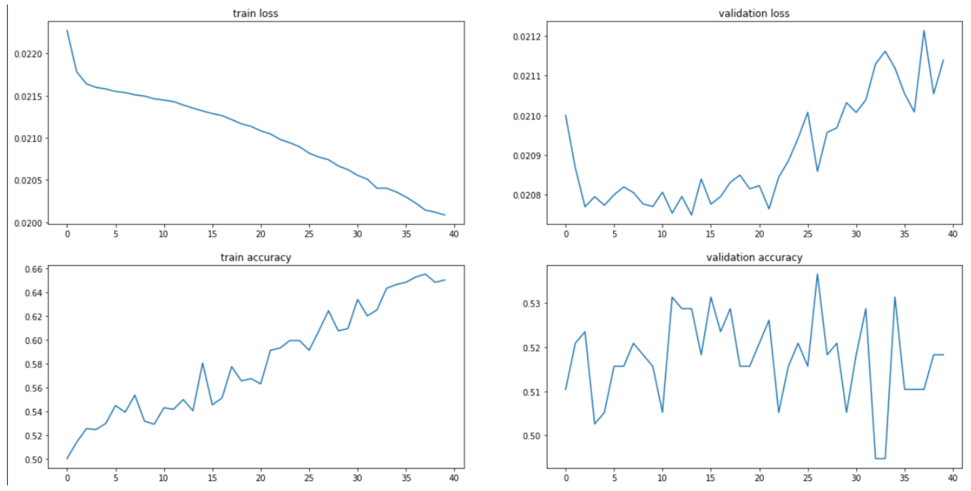
Report 4: RNN w/ pretrained embedding



Report 5: CNN w/o pretrained embedding



Report 6 CNN w/ pretrained embedding



Report 7: Discuss the complete set of experimental results, comparing the algorithms to each other.

- The models with pretraining does not update the weights in an embedding layer by optimizer during training. In contrast, the models without pretraining update the weights in an embedding layer by optimizer during training. In my setting, there was no noteworthy difference between the models with pretraining and the models without pretraining owing to overfitting. Morever, RNN without pretraining shows higher validation accuracy than RNN with pretraining. To observe the difference between them, the model architecture may be required to be changed.
- In my implementation, RNN took the longer time for training than CNN. Also, RNN requires more training (more iteration) to reach just before overfitting (around 80 epoch) than CNN

Report 8 : Discuss your observations about the various algorithms, i.e., differences in how they performed, different parameters, what worked well and didn't, patterns/trends you observed across the set of experiments, etc.

- In my implementation, CNN and RNN shows the overfitting. It requires the model with higher degree of freedom. For example, increase the number of hidden dimension, increase the depth, increase max_length, increase the number of vocabulary.

Report 9

I implemented and compared 4 deep learning model: CNN with pretraining, CNN without pretraining, RNN with pretraining, and RNN without pretraining.

- The model with pretraining used the given pretraining embedding. The model without preraining did not use the given pretraining embedding. Instead, these models randomly initialize the embedding layers and train the weight to find the proper weight.
- CNN use convolution layers for feature extraction that explains the label. To apply convolution layer on time-series data, 1-dimensional convolution layers are used. 1-dimensional convolution means that the column dimension of convolution layers is equal to the corresponding embedding dimension. Thus, 1-dimensional convolution layers can only move along one dimension.
- RNN used LSTM mechanism for feature extraction. LSTM used two hidden states, hidden state and cell state. These two state help RNN to capture the long-term memory by controlling update gate. Note that GRU have only hidden state.

Report 10

The most important thing in my implementation is that I implemented all preprocessing procedures without any automatic preprocessing tools such as torchtext, in order to understand the basic part of preprocessing procedure. Although implementing all preprocessing procedures is not efficient in terms of time and requires a lot of trial-and-errors, this experience teach me a lot.

Report 11 : pandas, numpy, pytorch

- I used the python library, **pandas** for file I/O. Pandaas dataframe and pandas series supports the variety of function that is necessary for preprocessing such as indexing, querying, concatenation, and so on.
- I used the python library, **numpy** in order to convert from pandas dataframe into tensor type.
- I used the python library, **pytorch** in order to use an efficient deep learning framework that has already implemented a basic class for deep learning.

Report 12

Strengths

- All data preprocessing is implemented without a dependency on an external library. It will increase flexibility of my code for future complex data structure.
- My code is dependent on only a few libraries (pandas, numpy, and pytorch).
- My model is simple to understand and easy to run.

Weakness

- Implementing all data preprocessing requires a lot of time and trial-and-errors at first. I could not spend my time for advanced preprocessing technique such as token masking such as ‘,’ and ‘the’.
- My implementation does not contain hyperparameter tunning.
- My models has simple architecture and does require a little training time. I did not implement the code that uses GPU device and GPU parallelization. To scale up and make a complex model, the usage of GPU and GPU parallelization is necessary to reduce running time.