# Deep Learning
# Assignment 5 Report

Tsinghua University

Albert Millan – 2019280366

## Contents

**1 Introduction** **1**

**2 Implementation** **1**

2.1 Pre-processing . . . . . . . . . . 1

2.2 RNN Model . . . . . . . . . . 1

2.3 Results . . . . . . . . . . 1

## 1 Introduction

This report explains the methodology involved to deploy text classification model employing bi-derictional recurrent neural networs (RNN) in Pytorch.

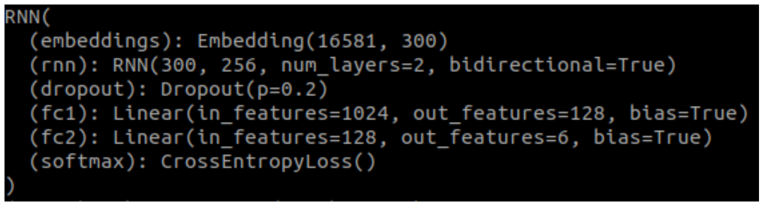
## 2 Implementation

### 2.1 Pre-processing

The provided data was converted into a *torchtext* dataset. Words in the sentence were tokenized and the vocabulary was constructed using pre-trained GloVe word-embeddings. Senteces was sorted into batches with similar token length. A custom data loader was employed to generate the batches.

### 2.2 RNN Model

Am implementation using a bi-directional recurrent neural network architecture was deployed to classify sentiment from sentences. The architecture is defined in figure 1. It includes a stack of recurrent layers moving in both directions with a dynamic size based on the length of the largest sentence in any given batch. It is followed by fully connected layer block, with ReLu activation and dropout (if applicable) and an fully connected output layer. Cross-entropy is then computed to define the loss and identify the prediction.



```
RNN(
  (embeddings): Embedding(16581, 300)
  (rnn): RNN(300, 256, num_layers=2, bidirectional=True)
  (dropout): Dropout(p=0.2)
  (fc1): Linear(in_features=1024, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=6, bias=True)
  (softmax): CrossEntropyLoss()
)
```

Figure 1: RNN architecture for Sentence Sentiment Classification.

### 2.3 Results

Table 1 outline the conditions under which the experiments were conducted, describing the hyper-parameters and results obtained. Tests perform aim to estimate the influence of the number of recurrent layers, the number of features in the hidden state and the use of dropout. Dropout appears to improve the accuracy by 2%. A larger hidden size (512) yielded better performance than the one of size 256, all else equal.

| **Emb.** | **# Hidden** | **Hidden S.** | **Dropout** | **Iters** | **Batch** | **Lrn. R.** | **Test Acc (%)** |
|---|---|---|---|---|---|---|---|
| GloVe | 2 | (256) | 0.0 | 120 | 32 | 0.01 | 39.32 |
| GloVe | 3 | (256) | 0.0 | 120 | 32 | 0.01 | 41.54 |
| GloVe | 4 | (256) | 0.0 | 120 | 32 | 0.01 | 37.82 |
| GloVe | 5 | (256) | 0.0 | 120 | 32 | 0.01 | 39.64 |
| GloVe | 2 | (512) | 0.0 | 120 | 32 | 0.01 | 42.80 |
| GloVe | 2 | (256) | 0.2 | 120 | 32 | 0.01 | 41.99 |

Table 1: Hyperparameters employed to train the RNN model.

The loss and accuracy values across training and validation iterations are shown in figures 2 and 3 respectively. Notice that as the model converges towards an optimal solution on the training data, it moves away from the optimal solution in the validation/test set. In fact, The optimal solution lies within the first 20 iterations, any time after that the performance of the model decreases on the test data as it starts to fit better the training data. Such increase in the loss despite of minimal decrease on the accuracy can only have two possible explanations: (a) the confidence of the model on missclasified predictions as correct labels keeps expanding, increasing the difference relative to the groundtruth label; (b) the confidence of the model on correctly classified predictions tends to decrease over time. That is, while it classifies the input correctly, the difference between the confidence on the top prediction and those that follow might be small.

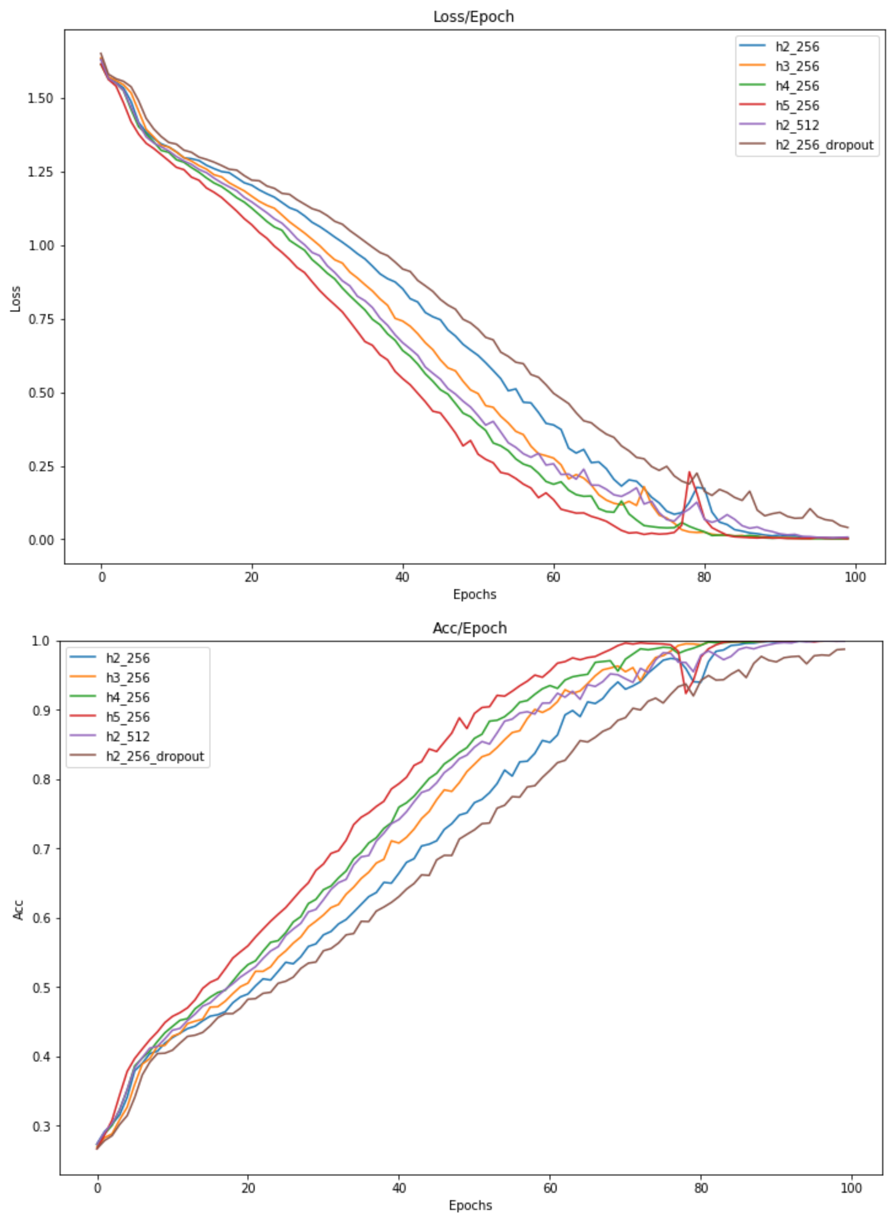


Figure 2: Training Loss and accuracy plots for the model with hidden size of 256 or 512, and with or without dropout.

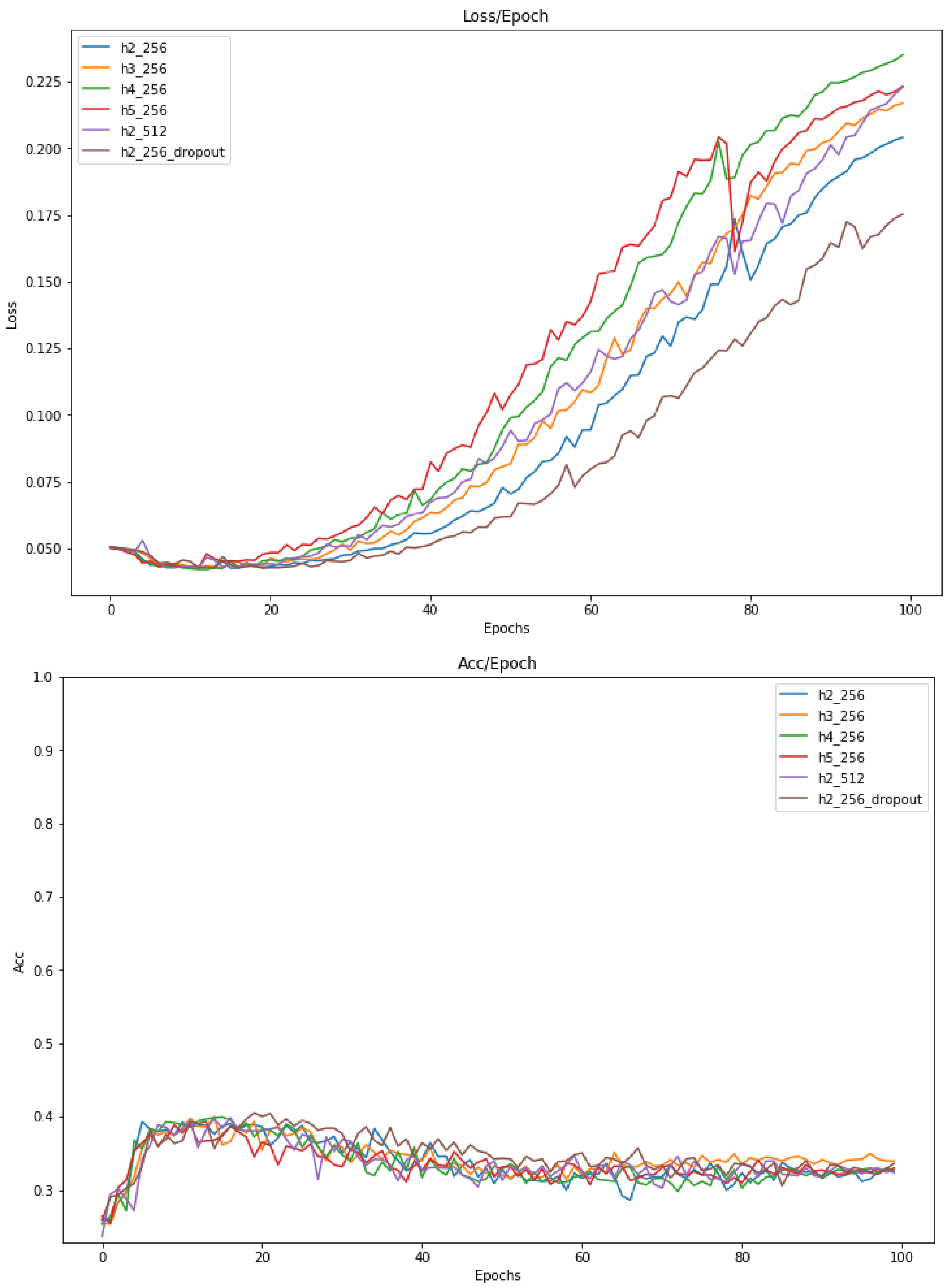


Figure 3: Validation Loss and accuracy plots for the model with hidden size of 256 or 512, and with or without dropout.