# Assignment 1: Text Classification with Neural Nets

Pankaj Kumar Jatav : G01338769

September 10, 2022

**Abstract**

In this assignment, you will implement a binary sentiment classifier based on the feed-forward neural net (FFNN) architecture. This includes implementing the FFNN model and its training procedure. The implementation will be based on PyTorch (https://pytorch.org/). Other generic Python tools such as numpy are also allowed (if you are not sure, please consult the instructor).

## 1 Implementing The Network::FFNN

The network consist of one embedding layer, one hidden layer and one output layer. I used the ReLU activation function for hidden layer and Softmax for converting the output to probability distribution. Here is the embedding layer and hidden layer in my code

```
self.word_embeddings = torch.nn.EmbeddingBag(self.vocab_size, self.emb_dim)
self.fc1 = torch.nn.Linear(self.emb_dim, self.n_hidden_units)
self.relu = torch.nn.ReLU()
self.fc2 = torch.nn.Linear(self.n_hidden_units, self.n_classes)
```

Here is the Softmax which I used in my code.

```
F.softmax(out, dim=1)
```

## 2 Prepossessing Of Dataset

We have give movie review Dataset. I have explore the below techniques for creating the vocab list and I used some of them. I ended with crated vocab of 7143 words.

### 2.1 Removing The Unique Word

I store a frequency dictionary of all the word in dataset. I kept only words which frequency is more than 1. I found this really helped the performance of the model and accuracy increased. So I used this at prepossessing.

### 2.2 Removing The Stop Word

I try with removing all Stop Words from my vocab. But it cause me giving less accuracy on dev dataset. So I decided not to use.

### 2.3 Stemming and Lemmatization

I also explore Stemming and Lemmatization for vocab. But Stemming and Lemmatization giving me less accuracy on dev set. So I decided not to use.

# 3 Training

For Tuning the model I created one python file names tune.py. And this file test the model with different hyper-parameter.
I used the Glove4B300d for testing the model against pre-defined embedding.

I used the following default values for my model training.

```
    'train_path': 'data/train.txt',
    'dev_path': 'data/dev.txt',
    'blind_test_path': 'data/test-blind.txt',
    'test_output_path': 'test-blind.output.txt',
    'glove_path': None,
    'no_run_on_test': True,
    'n_epochs': 10,
    'batch_size': 32,
    'emb_dim': 300,
    'n_hidden_units': 300
```

Below are the value which I explore for hyper-parameters tuning during training model.

| Hyper-Parameters | Values |
|---|---|
| Epochs | 5, 10, 15, 20, 25, 30 |
| Batch Size | 16, 32, 64, 128, 256 |
| Embedding Size | 50, 100, 300, 500, 1000 |
| Optimizer | SGD, Adagrad, AdamW, Adam |
| Learning rate | 1, 0.1, 0.01, 0.001, 0.0001 |

Table 1: hyper-parameter values.

Below are the output while training

```
[pankaj@arch-rp student-distrib]$ python sentiment_classifier.py --n_epochs 10 --batch_size 32
    --emb_dim 300 --n_hidden_units 300
[nltk_data] Downloading package stopwords to /home/pankaj/nltk_data...
[nltk_data]  Package stopwords is already up-to-date!
Namespace(train_path='data/train.txt', dev_path='data/dev.txt', blind_test_path='data/test-blind.txt',
    test_output_path='test-blind.output.txt', glove_path=None, run_on_test=True, n_epochs=10,
    batch_size=32, emb_dim=300, n_hidden_units=300)
6920 / 872 / 1821 train/dev/test examples
Vocab size: 7143
*
Epoch 9
Avg loss: 0.34476
Accuracy: 689 / 872 = 0.790138
Precision (fraction of predicted positives that are correct): 361 / 461 = 0.783080; Recall (fraction of
    true positives predicted correctly): 361 / 444 = 0.813063; F1 (harmonic mean of precision and
    recall): 0.797790
----------

=====Train Accuracy=====
Accuracy: 6519 / 6920 = 0.942052
Precision (fraction of predicted positives that are correct): 3336 / 3463 = 0.963327; Recall (fraction
    of true positives predicted correctly): 3336 / 3610 = 0.924100; F1 (harmonic mean of precision and
    recall): 0.943306
=====Dev Accuracy=====
Accuracy: 698 / 872 = 0.800459
Precision (fraction of predicted positives that are correct): 348 / 426 = 0.816901; Recall (fraction of
    true positives predicted correctly): 348 / 444 = 0.783784; F1 (harmonic mean of precision and
    recall): 0.800000
Time for training and evaluation: 19.26 seconds
```

# 4 Results

Below are my results for part 1 and part 2.

## 4.1 Part 1

I got 94.1185% Accuracy on training dataset and 80.2752% Accuracy on Dev dataset. And below are the output.

```
=====Train Accuracy=====
Accuracy: 6513 / 6920 = 0.941185
Precision (fraction of predicted positives that are correct): 3344 / 3485 = 0.959541; Recall (fraction
    of true positives predicted correctly): 3344 / 3610 = 0.926316; F1 (harmonic mean of precision and
    recall): 0.942636
=====Dev Accuracy=====
Accuracy: 700 / 872 = 0.802752
Precision (fraction of predicted positives that are correct): 348 / 424 = 0.820755; Recall (fraction of
    true positives predicted correctly): 348 / 444 = 0.783784; F1 (harmonic mean of precision and
    recall): 0.801843
Time for training and evaluation: 20.40 seconds
```

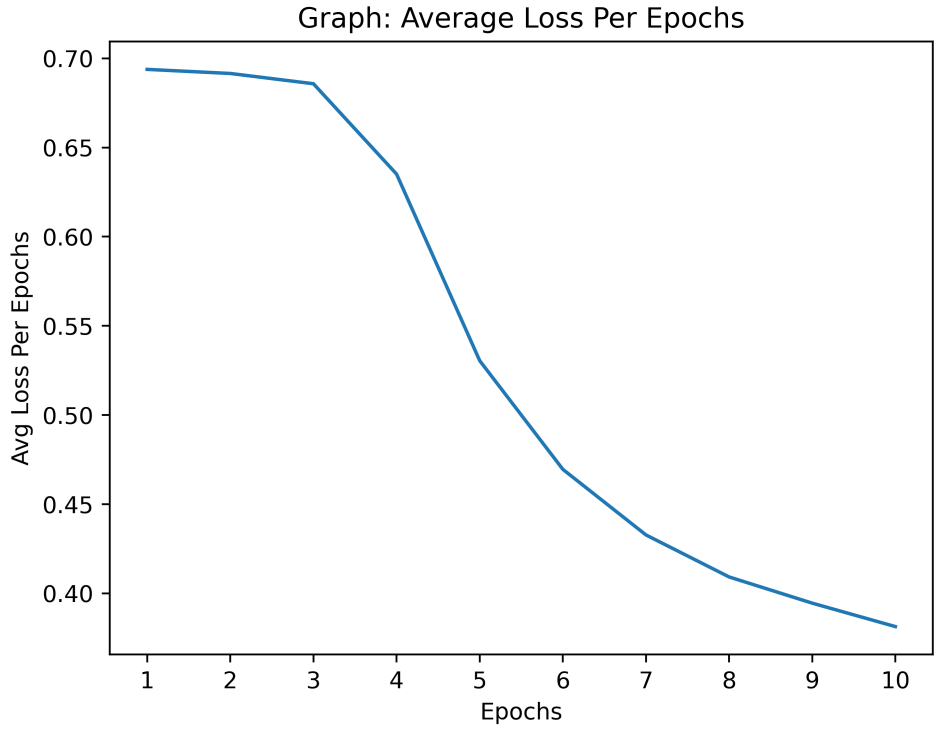


Figure 1: Average Loss Per Epochs

## 4.2 Part 2

During the model parameter tuning I observe the following

- Increasing the number of epochs does not lead to higher accuracy on the dev set. Instead it was giving higher accuracy on training set ie started to over-fitting.
- Increase the number of batch side lead to drop in accuracy as expected.
- Increase the embedding size and number of node in hidden unit did not change much of the accuracy on dev set.
- SGD optimizer does not perform well on traning and dev datset.
- Adam and AdamW perform almost same on traning and test dataset.
- As we know a too big and too small learning rate does not lead to the optimize solution we have notice the same for all optimizer expect SGD.
- Using the predefined embedding does not gave best result as we had different dataset. But using the custom embedding gave less loss for initial average loss as expected because embedding was predefined.
- Updating the glove embedding gave better not updating it in each epochs.

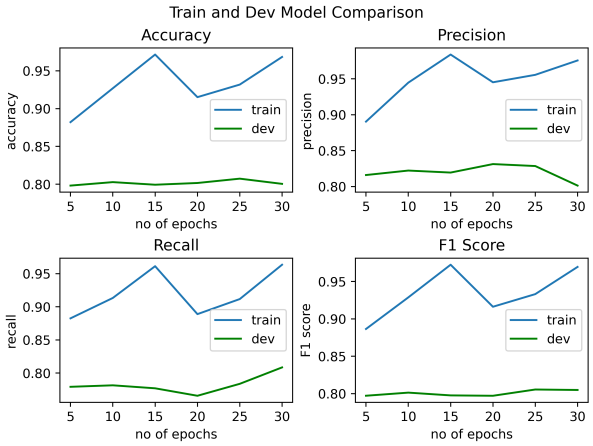


Figure 2: Epochs Values Results

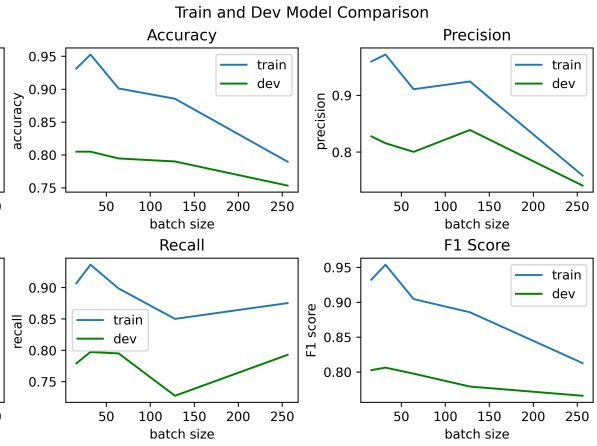


Figure 3: Batch Values Results

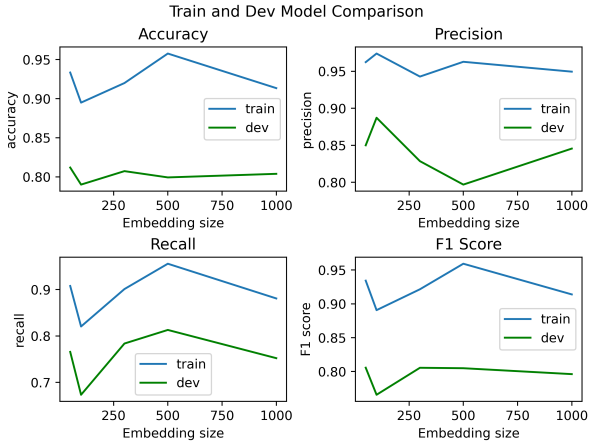


Figure 4: Embedding Values Results

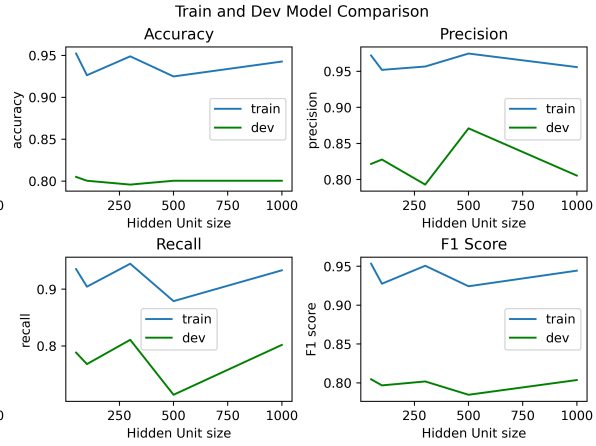


Figure 5: Hidden Layer Values Results

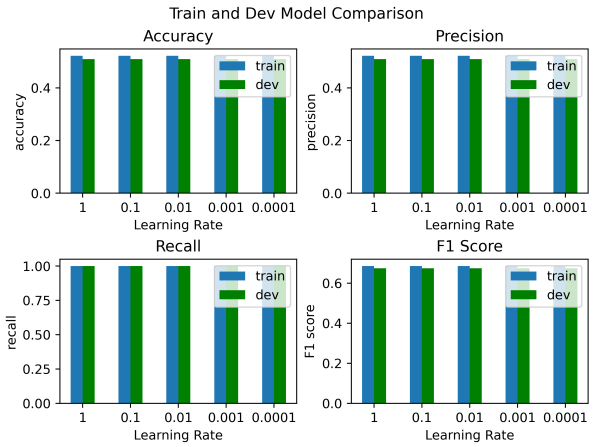


Figure 6: SGD Results

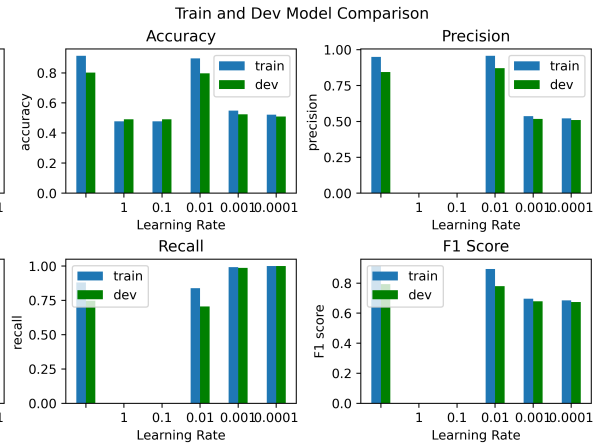


Figure 7: AdaGrad Results

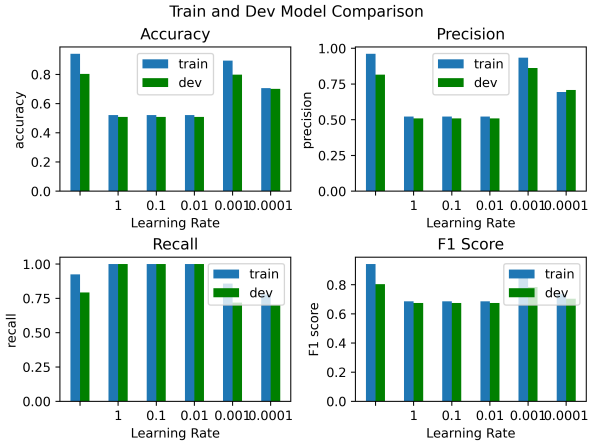


Figure 8: AdamW Values Results

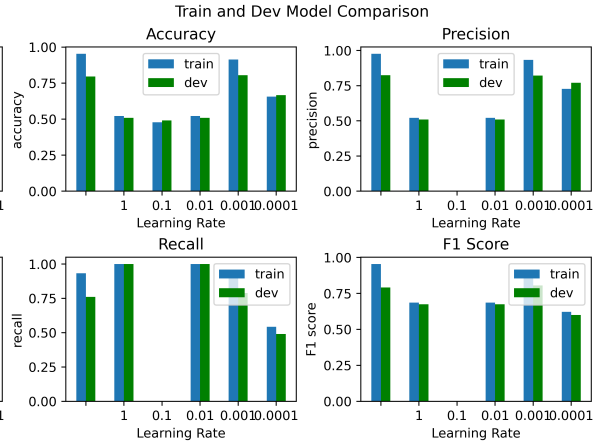


Figure 9: Adam Results

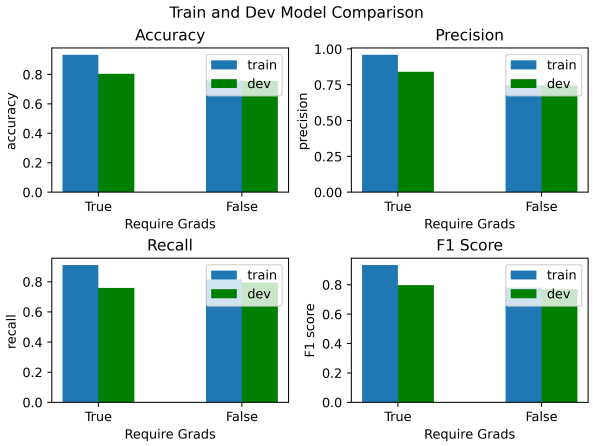


Figure 10: Glove Require Grad Values Results