Assignment 1: Text Classification with Neural Nets
CS678, Fall 2022

Introduction: To Implement a binary sentiment classifier based on the Feed-Forward Neural Net (FFNN) architecture for the movie review dataset of Socher et al. (2013) and classify whether the movie has positive(1) or negative(0) rating. This implementation is based on PyTorch and generic Python tools.

Approach: Dataset: Read the simplified version of movie review dataset of Socher et al.(2013) that consists of review snippets taken from Rotten Tomatoes considering two classes – positive (1) or negative (0).The labels of original data set which are "fine-grained" sentiment labels ranging from 0 to 4: highly negative, negative, neutral, positive, and highly positive and also full parse trees with each constituent phrase of a sentence labeled with sentiment (including the whole sentence) and the data files contain newline-separated sentiment examples, consisting of a label (1 or 0) followed by a tab, followed by the sentence.The data has been split into a train, development (dev), and blind test set.

FFNN Model Implemented the FFNN classifier that is similar to the "Deep Averaging Network" (DAN) model of Iyyer et al. (2015). Formally, given a sentence s = (w1 , w2 , ..., w|s|), where wi is a single word in this sentence and |s| denotes the sentence length, mapping each word to its embedding vector, denoted as ci ∈ Rde , where de is the size of the embedding. The DAN model will then calculate an average embedding vector v∈Rde . for this sentence, and pass it through two feedforward layers:

$$\begin{aligned} av &= \sum_{i=1}^{|s|} \frac{c_i}{|s|}, \\ h &= ReLU(W_h \cdot av + b_h), \\ h_{out} &= W_{out} \cdot h + b_{out}, \\ pred &= softmax(h_{out}). \end{aligned}$$

Here, Wh ∈ Rdh ×de , bh ∈ Rdh are learnable parameters for the first non-linear layer (with activation function *ReLU*);h∈Rdh represents the intermediate hidden units; the second linear layer is parameterised by two learnable parameters Wout ∈ Rncls×dh,bout ∈ Rncls, where ncls is the number of classes in the classification task (e.g., two in a binary classification problem), and it produces the *logits vector* hout ∈ Rncls . and the this last linear layer has projected the hidden units to the class space (and hence has a dimension of Rncls); however, this logits vector is passed hout to a *softmax* function. to get the final prediction probability distribution

Execution command:

```
Python3 sentiment_classifier.py --n_epochs 10 --batch_size 32 --emb_dim 300 --n_hidden_units 300
```

Framework Code:

The framework code consists of four .py scripts, as explained below:

• sentiment_classifier.py: argparse is used to read in several command line arguments where --train_path for training, –-dev_path for development, —-blind_test_path for testing --test_output_path for writing blind test data output , –glove_path to the glove.6B.300d.txt file, --n_epoch for training number of epochs,--batch_size for batch size for training, --emb_dim for dimension of word embeddings,

--n_hidden_units for dimensions for hidden units. It trains the FFNN model, and evaluates it on train, dev, and blind test, and writes the blind test results to a file.

• sentiment_data.py: defines the SentimentExample object, which wraps a list of words with an integer label (0/1) along with word_indices returned by indexing_sentiment_examples based upon given vocabulary. It also includes the SentimentExampleBatchIterator class for processing the training data in mini-batches where for each SentimentExample ,

#TODO implemented batch_inputs (containing list of word_indices),batch_lengths(containing number of word_indices before padding in each example), and batch_labels(containing class labels), added padding for the batch_inputs wherever the max_length is not matched.and are, are returned in the form of tensor.

• models.py: completed the implementation of the FeedForwardNeuralNetClassifier and its training method train feedforward neural net.

#TODO created a randomly initialized embedding matrix, and set padding_idx as 0 by adding nn.Embedding(num_embeddings= vocab_size,embedding_dim= emb_dim) and one linear layers followed by non-linear and linear(output) layers.

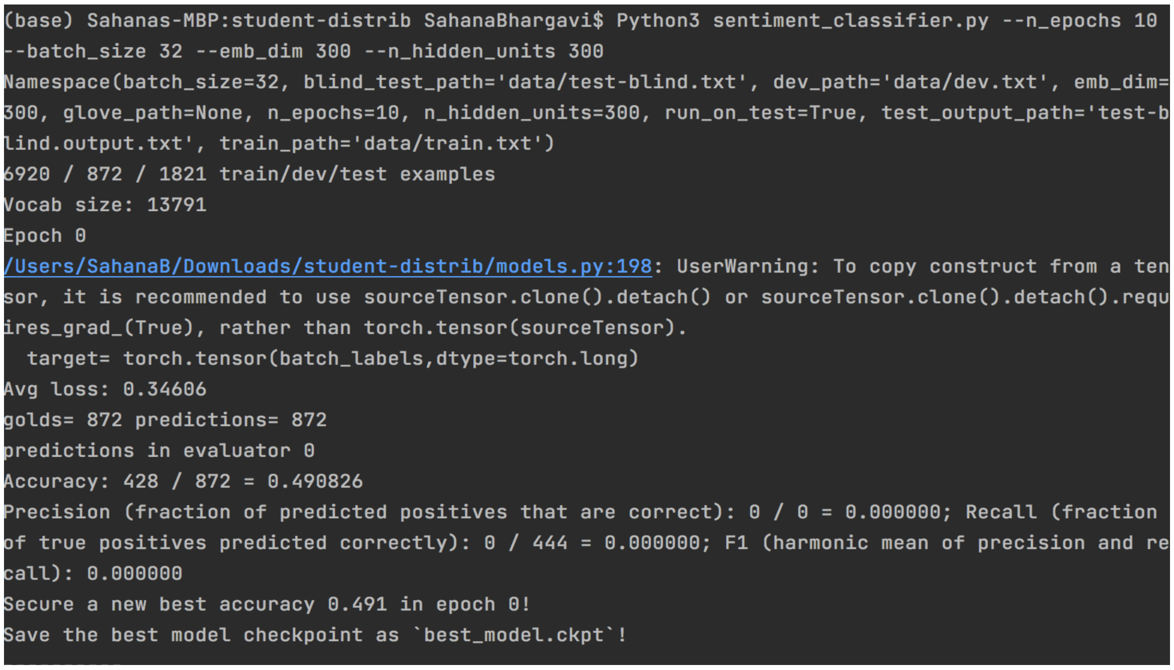
#TODO In forward pass, the batch_inputs are given to word_embedding layer created and then the averaged embeddings are passed into next linear layer nn.Linear(emb_dim ,n_hidden_units) and passed into next non-linear function nn.relu() and then further passed into nn.Linear(n_hidden_units, n_classes) which returns the ‘logits’.

#TODO: In batch_predict, makse predictions for a batch of inputs and the softmax functions utilizes the logits returned by invoking `forward` pass and final probability predictions are done.

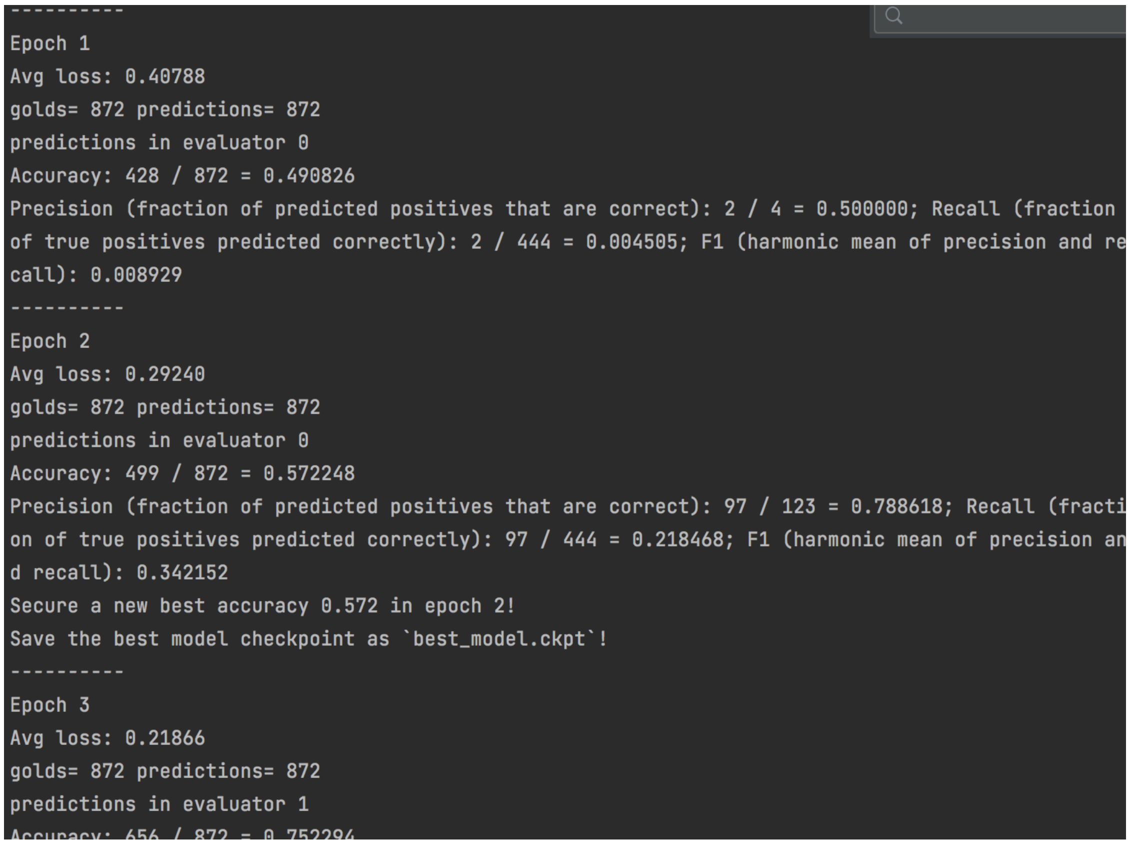
#TODO In vocabulary creation, created a vectorizer object that fits the training set of sentiment-examples that identifies unique words along with their indices and stored in vectorizer.vocabulary and further ["PAD", "UNK"] are appended with vocabulary.
#TODO created an FFNN classifier
#TODO defined an Adam optimizer with rate_learning= 0.00287
#TODO CrossEntropyLoss to which logits and batch_labels are passed and loss is computed.
For each epoch,(total 10), batch_iterator gets a new batch of given batch_size
# TODO: cleaned up the gradients for this batch optimizer.zero_grad()
# TODO: called the model to get the logits = model.forward(batch_inputs,batch_lengths)
# TODO: calculated the loss loss = criterion(logits,target)
# TODO: backpropagation (backward and step) loss.backward() ,optimizer.step()

• evaluator.py: evaluation code, which calculates accuracy, precision, recall, and F1 of the tested examples.

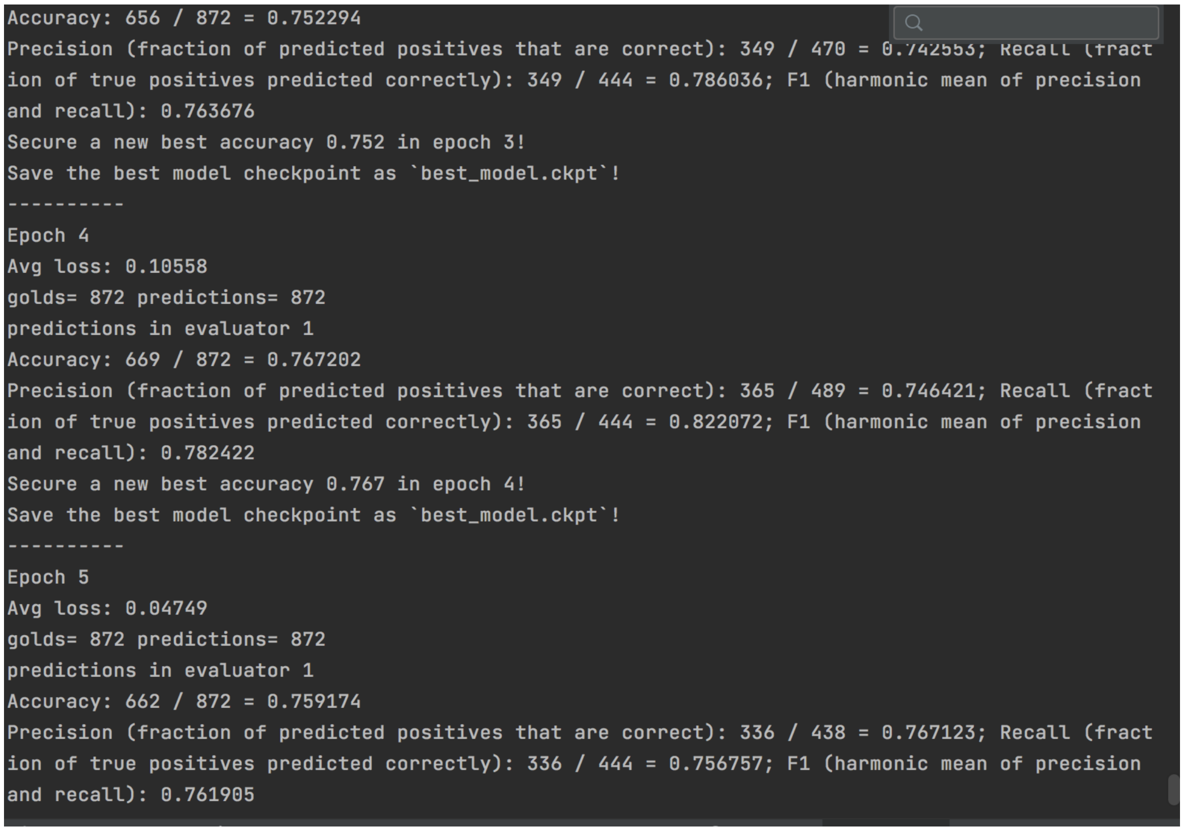
Part 1: The final training and dev set performance of the trained FFNN model.



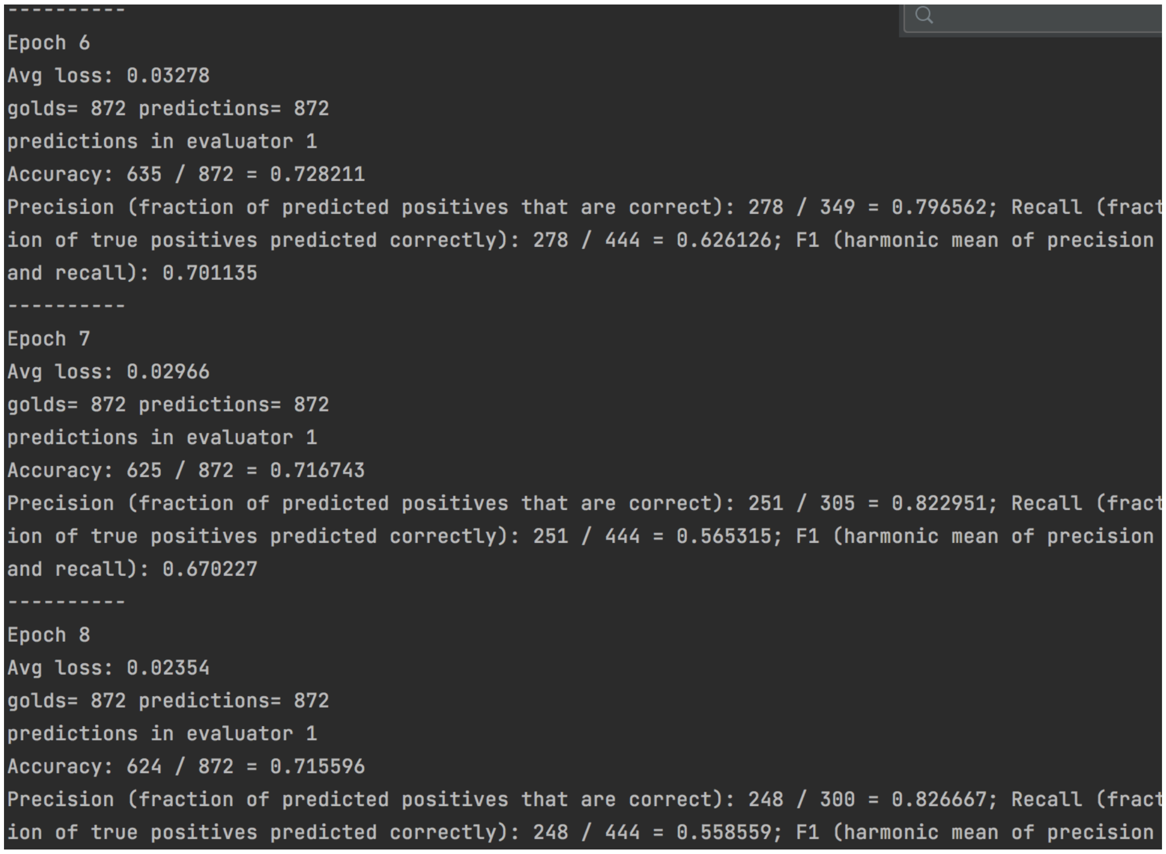
```
(base) Sahanas-MBP:student-distrib SahanaBhargavi$ Python3 sentiment_classifier.py --n_epochs 10
--batch_size 32 --emb_dim 300 --n_hidden_units 300
Namespace(batch_size=32, blind_test_path='data/test-blind.txt', dev_path='data/dev.txt', emb_dim=
300, glove_path=None, n_epochs=10, n_hidden_units=300, run_on_test=True, test_output_path='test-b
lind.output.txt', train_path='data/train.txt')
6920 / 872 / 1821 train/dev/test examples
Vocab size: 13791
Epoch 0
/Users/SahanaB/Downloads/student-distrib/models.py:198: UserWarning: To copy construct from a ten
sor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requ
ires_grad_(True), rather than torch.tensor(sourceTensor).
  target= torch.tensor(batch_labels,dtype=torch.long)
Avg loss: 0.34606
golds= 872 predictions= 872
predictions in evaluator 0
Accuracy: 428 / 872 = 0.490826
Precision (fraction of predicted positives that are correct): 0 / 0 = 0.000000; Recall (fraction
of true positives predicted correctly): 0 / 444 = 0.000000; F1 (harmonic mean of precision and re
call): 0.000000
Secure a new best accuracy 0.491 in epoch 0!
Save the best model checkpoint as `best_model.ckpt`!
```



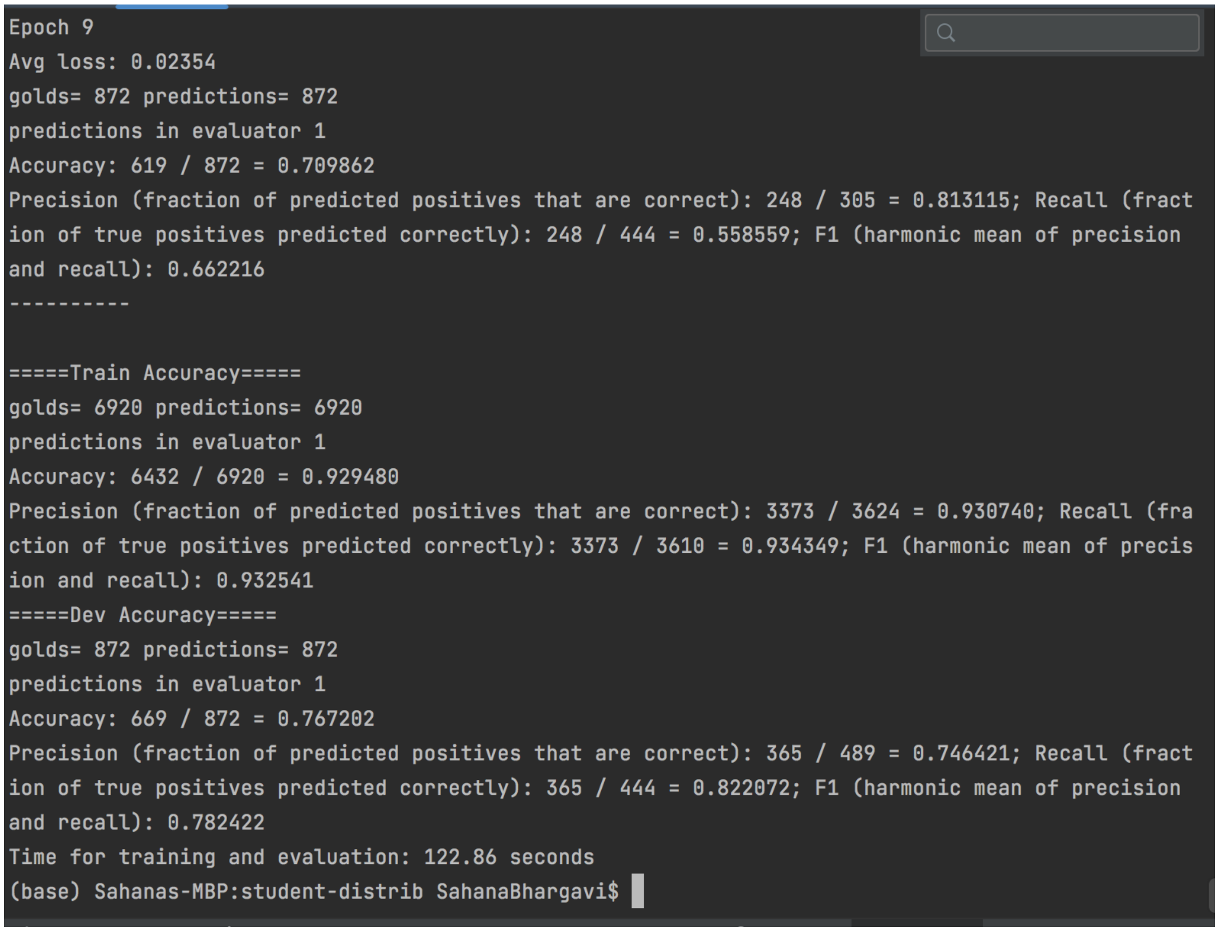
```
----------
Epoch 1
Avg loss: 0.40788
golds= 872 predictions= 872
predictions in evaluator 0
Accuracy: 428 / 872 = 0.490826
Precision (fraction of predicted positives that are correct): 2 / 4 = 0.500000; Recall (fraction
of true positives predicted correctly): 2 / 444 = 0.004505; F1 (harmonic mean of precision and re
call): 0.008929
----------
Epoch 2
Avg loss: 0.29240
golds= 872 predictions= 872
predictions in evaluator 0
Accuracy: 499 / 872 = 0.572248
Precision (fraction of predicted positives that are correct): 97 / 123 = 0.788618; Recall (fracti
on of true positives predicted correctly): 97 / 444 = 0.218468; F1 (harmonic mean of precision an
d recall): 0.342152
Secure a new best accuracy 0.572 in epoch 2!
Save the best model checkpoint as `best_model.ckpt`!
----------
Epoch 3
Avg loss: 0.21866
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 656 / 872 = 0.752294
```



```
Accuracy: 656 / 872 = 0.752294
Precision (fraction of predicted positives that are correct): 349 / 470 = 0.742553; Recall (fract
ion of true positives predicted correctly): 349 / 444 = 0.786036; F1 (harmonic mean of precision
and recall): 0.763676
Secure a new best accuracy 0.752 in epoch 3!
Save the best model checkpoint as `best_model.ckpt`!
----------
Epoch 4
Avg loss: 0.10558
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 669 / 872 = 0.767202
Precision (fraction of predicted positives that are correct): 365 / 489 = 0.746421; Recall (fract
ion of true positives predicted correctly): 365 / 444 = 0.822072; F1 (harmonic mean of precision
and recall): 0.782422
Secure a new best accuracy 0.767 in epoch 4!
Save the best model checkpoint as `best_model.ckpt`!
----------
Epoch 5
Avg loss: 0.04749
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 662 / 872 = 0.759174
Precision (fraction of predicted positives that are correct): 336 / 438 = 0.767123; Recall (fract
ion of true positives predicted correctly): 336 / 444 = 0.756757; F1 (harmonic mean of precision
and recall): 0.761905
```



```
----------
Epoch 6
Avg loss: 0.03278
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 635 / 872 = 0.728211
Precision (fraction of predicted positives that are correct): 278 / 349 = 0.796562; Recall (fract
ion of true positives predicted correctly): 278 / 444 = 0.626126; F1 (harmonic mean of precision
and recall): 0.701135
----------
Epoch 7
Avg loss: 0.02966
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 625 / 872 = 0.716743
Precision (fraction of predicted positives that are correct): 251 / 305 = 0.822951; Recall (fract
ion of true positives predicted correctly): 251 / 444 = 0.565315; F1 (harmonic mean of precision
and recall): 0.670227
----------
Epoch 8
Avg loss: 0.02354
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 624 / 872 = 0.715596
Precision (fraction of predicted positives that are correct): 248 / 300 = 0.826667; Recall (fract
ion of true positives predicted correctly): 248 / 444 = 0.558559; F1 (harmonic mean of precision
```



```
Epoch 9
Avg loss: 0.02354
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 619 / 872 = 0.709862
Precision (fraction of predicted positives that are correct): 248 / 305 = 0.813115; Recall (fract
ion of true positives predicted correctly): 248 / 444 = 0.558559; F1 (harmonic mean of precision
and recall): 0.662216
----------

=====Train Accuracy=====
golds= 6920 predictions= 6920
predictions in evaluator 1
Accuracy: 6432 / 6920 = 0.929480
Precision (fraction of predicted positives that are correct): 3373 / 3624 = 0.930740; Recall (fra
ction of true positives predicted correctly): 3373 / 3610 = 0.934349; F1 (harmonic mean of precis
ion and recall): 0.932541
=====Dev Accuracy=====
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 669 / 872 = 0.767202
Precision (fraction of predicted positives that are correct): 365 / 489 = 0.746421; Recall (fract
ion of true positives predicted correctly): 365 / 444 = 0.822072; F1 (harmonic mean of precision
and recall): 0.782422
Time for training and evaluation: 122.86 seconds
(base) Sahanas-MBP:student-distrib SahanaBhargavi$
```

Part 2: results and analyses.

Vector dimensions

Embedding size	Hidden Size	Accuracy (dev)
50	100	0.738532
100	100	0.751147
500	100	0.755734
100	500	0.745413
500	500	0.761468

After exploring different choices for the embedding and hidden dimensions, it can be observed from the table that on increasing both the dimensions, the performance of the model is improved whereas on having less embedding size compared to hidden size the performance decreases and having more embedding size comparatively the accuracy increased slightly.

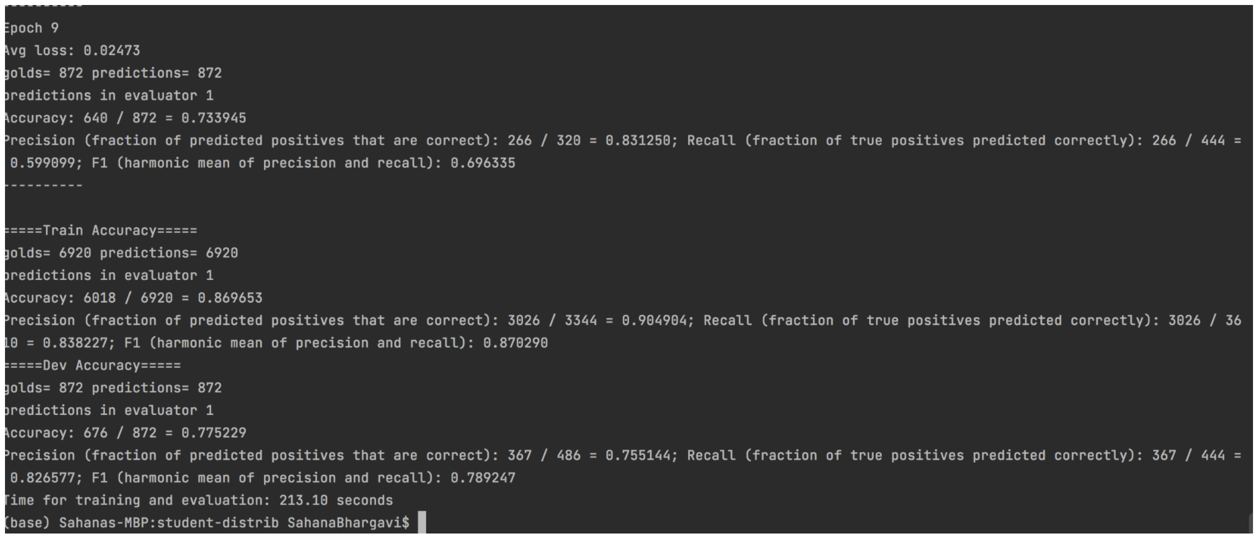
Optimization methods with different learning rates

Learning rates	0.00185	0.00287	0.0032	0.004	0.5	1
Adam	0.747	0.767	0.759	0.747	0.51	0.49
Adagrad	0.717	0.70	0.708	0.7201	0.7305	0.696
Adamax	0.51	0.575	0.5435	0.5435	0.4908	0.511
SGD(Stochastic Gradient Descent)	0.48967	0.49083	0.49083	0.49083	0.49083	0.49083

On applying Adam optimiser, the model achieved around 0.77 accuracy at learning rate 0.00287 while the value is around 0.75 with remaining slightly higher and larger learning rates .With the other optimisers like Adagrad the accuracy is around 0.73 for higher learning rates and is 0.70 with lower ones, Adamax and SGD has shown consistent low performance having 0.51 for higher learning rates and 0.57 for lower ones and 0.48 for SGD for all the learning rates.On considering the performances mentioned, Adam is better optimization method compared to others and also model performance is better for smaller learning rates compared to larger ones.

Embedding initialization: Implemented loading and preprocessing the open-source GloVe embedding (https://nlp.stanford.edu/projects/glove/) as well as using it to initialize the word embeddings of FeedForwardNeuralNetClassifier. The frame-work code has set up an optional argument --glove path in sentiment_classifier.py such that the GloVe embedding would be invoked by executing:

python sentiment_classifier.py --n_epochs 10 --batch_size 32 --emb_dim 300 --n hidden_units 300 --glove_path path/to/GloVe.



```
----------
Epoch 9
Avg loss: 0.02473
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 640 / 872 = 0.733945
Precision (fraction of predicted positives that are correct): 266 / 320 = 0.831250; Recall (fraction of true positives predicted correctly): 266 / 444 =
 0.599099; F1 (harmonic mean of precision and recall): 0.696335
----------

=====Train Accuracy=====
golds= 6920 predictions= 6920
predictions in evaluator 1
Accuracy: 6018 / 6920 = 0.869653
Precision (fraction of predicted positives that are correct): 3026 / 3344 = 0.904904; Recall (fraction of true positives predicted correctly): 3026 / 36
10 = 0.838227; F1 (harmonic mean of precision and recall): 0.870290
=====Dev Accuracy=====
golds= 872 predictions= 872
predictions in evaluator 1
Accuracy: 676 / 872 = 0.775229
Precision (fraction of predicted positives that are correct): 367 / 486 = 0.755144; Recall (fraction of true positives predicted correctly): 367 / 444 =
 0.826577; F1 (harmonic mean of precision and recall): 0.789247
Time for training and evaluation: 213.10 seconds
(base) Sahanas-MBP:student-distrib SahanaBhargavi$ 
```

GloVe Pre-trained word embedding consists of word vectors that are trained on different massive web datasets The file glove.6B.300d.txt have vectors with 300 dimension, trained on a corpus of 6 billion tokens and contains a vocabulary of 400 thousand tokens. As it can be observed that there is slight increase in performance i.e accuracy 0.7753 using the pre -trained embeddings as initialization weights replacing the embedding layer trained on own word vectors from scratch that is rather than initializing our neural network weights randomly.

Conclusion: I have learnt how to implement Feed forward neural network as binary sentiment classifier using different embedding,hidden dimensions, optimisers, learning rates,vocabulary creation,batchifying the datasets and usage of Glove pre-trained embedding layers.

References:

https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neural network/#model-c-1-hidden-layer-feedforward-neural-network-relu-activation

https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/

https://medium.com/@martinpella/