Project 1: Sentiment Classification Writeup

The project is based on the binary sentiment classification of the textual data using logistic regression and neural networks.

Logistic regression is implemented from scratch by calculating the gradient and then updating the weights after each iteration. For Part 1 of the coding assignment, a total of 20 epochs are used with a learning rate of 0.01 which resulted in the training set accuracy of 0.98 and development set accuracy of 0.787 during a run time of 12 seconds. The training samples are shuffled after each iteration to disregard any inherent ordering within the examples.

Neural network model is developed using PyTorch library and a simple network is designed having an embedding layer, 2 fully connected layers and an output layer. Adam optimizer is used for gradient descent and categorical cross entropy is used as the loss measure during the training. For a total of 20 epochs and training batch size of 1, a training accuracy of 0.807 and development accuracy of 0.778 is achieved with a 2.5 minutes run-time for the training.

Exploration 1:

For this task, three learning rates (0.1, 0.01 and 0.001) are used for different no. of epochs and training/dev accuracy is recorded which resulted in Tables 1-3 and Figure 1. From the tables and figure, it is evident that if we use a low learning rate, we need to have a greater number of iterations to reach the same level of accuracy. For higher learning rates, the optimal point may be reached faster but it may lead to overfitting such as in the case of learning rate of 0.1. For low learning rates, optimal point may be reached with high number of iterations and having a smaller number of iterations may lead to underfitting as is the case with 0.001. So, we need to find a perfect balance of learning rate which neither overfits nor underfits. Besides, the training procedure should be completed in an optimal number of iterations, training time and computer resources, which is the case with 0.01.

Exploration 2:

The bigram feature extractor combines the adjacent words of the sentence as the features in the index dictionary. Although it gets more context aware by combining two words, its performance is below par as per the unigram model. Bigram only gives a max development accuracy of 0.73, albeit tried with different epochs and learning rates. This extractor was run for 20/40 epochs and learning rate as 0.01 and 0.001 thereby displaying a train accuracy of 0.98 and development accuracy of 0.729. This depicts that bigram model is overfitting on the training data rather than generalizing well on the unseen data.

Better features extractor combines unigram, bigram and trigram features along with the following data pre-processing steps apart from lower casing which happens to be in the unigram/bigram implementation as well:

- Remove stop words.
- Remove all the individual punctuation characters from each word
- Consider the words having more than three characters

Such pre-processing and combined features lead to a comparable performance to that of unigram approach but not necessarily better than that:

- Training Accuracy: 0.92, Dev Accuracy: 0.777 (Iterations: 20, learning rate=0.01)
- Training Accuracy: 0.94, Dev Accuracy: 0.772 (Iterations: 40, learning rate =0.01)

This shows that unigram with a simplistic approach still takes priority over the Bigram feature extractor as well as the combination of data pre-processing and feature combination of unigram/bigram/trigram.

Exploration 3:

The sub-class of **nn.Module** designed for sentiment classification was based on batching inputs. The input sentences were either padded or truncated to maximum token length of 40 during the training/testing. Following are the results with different batch sizes:

- **Batch 1:** Epochs 20, Train Acc: 0.80, Dev Accuracy: 0.772
- **Batch 8:**
    - Epochs 20, Train Acc: 0.74, Dev Accuracy: 0.75
    - Epochs 40, Train Acc: 0.80, Dev Accuracy: 0.780
- **Batch 16:**
    - Epochs 40, Train Acc: 0.797, Dev Accuracy: 0.770
    - Epochs 80, Train Acc: 0.807, Dev Accuracy: 0.778

From the results, it is evident that increasing the batch size should accompany an increment of epochs as well if an optimal performance is to be reached in terms of train/dev accuracy. A larger batch size tends to make less updates to the weights as compared to stochastic approach, so an increase of epochs caters to that disadvantage which avoids underfitting of the neural network model.

Exploration 4:

An LSTM layer was added after the embedding layer with embedding size=300, hidden size=128, number of layers=1 and batch_first = True (due to batching based class implementation). This is shown using FFNN_LSTM class in the codebase. This architecture showed the best performance out of all the other tried without using the LSTM layer such as only linear layers, or linear layers with Tanh()/ReLU() non-linear transformations:

- LSTM(): Epochs 20, Train Acc: 0.998, Dev Accuracy: 0.817
- Tanh(): Epochs 20, Train Acc: 0.839, Dev Accuracy: 0.777
- ReLU(): Epochs 10, Train Acc: 0.844, Dev Accuracy: 0.769

Appendix:

(1) Learning Rate: 0.1

Epochs	5	10	20	30	40
Training Acc	0.944	0.975	0.989	0.994	0.997
Development Acc	0.791	0.791	0.787	0.787	0.786

Table 1: Accuracy of Logistic Regression Classifier for Learning Rate = 0.1

(2) Learning Rate: 0.01

Epochs	5	10	20	30	40
Training Acc	0.836	0.880	0.918	0.939	0.951
Development Acc	0.775	0.778	0.782	0.787	0.787

Table 2: Accuracy of Logistic Regression Classifier for Learning Rate = 0.01

(3) Learning Rate: 0.001

Epochs	5	10	20	30	40
Training Acc	0.693	0.728	0.770	0.799	0.821
Development Acc	0.693	0.714	0.745	0.751	0.763

Table 3: Accuracy of Logistic Regression Classifier for Learning Rate = 0.001

(4) Plot for all Learning Rates

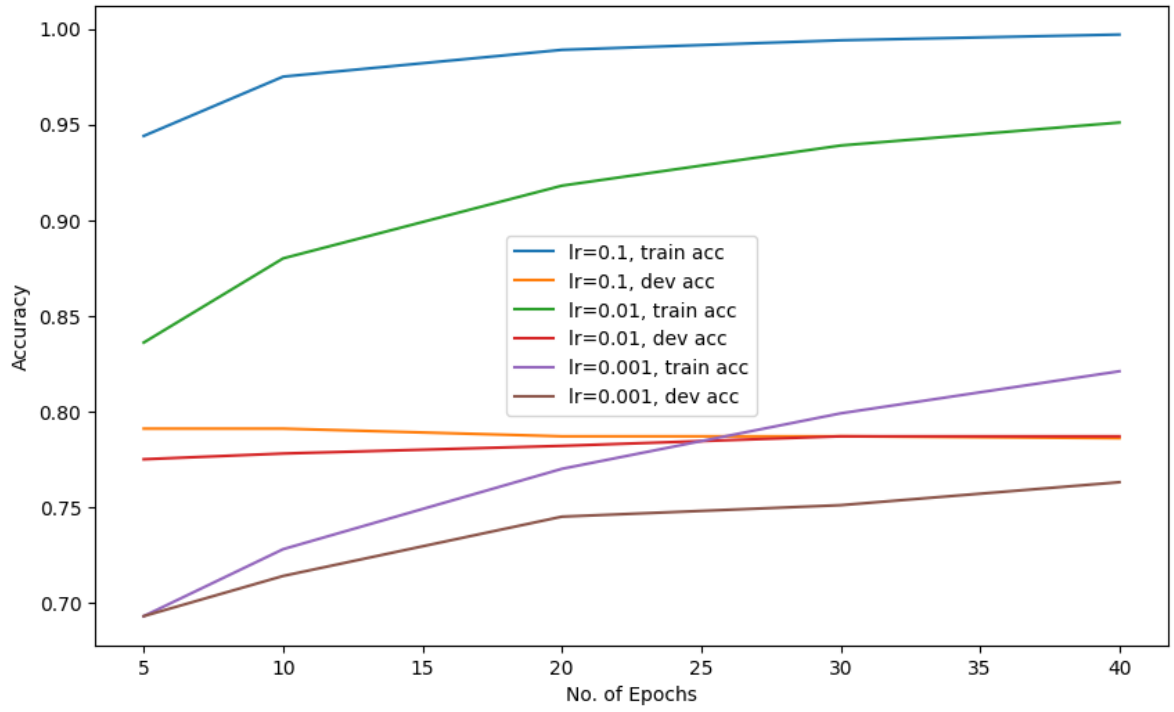

Figure 1: Plot between Accuracy and Epochs for train/dev accuracy (Different Learning Rates)