COMP 4901I: Assignment #2

I-Tsun Cheng
20576079

Hong Kong University of Science and Technology — March 23, 2019

1 Dataset

In *MyData* class, the vocabulary is constructed by the most frequent *vocab_size-1* words and the "UNK" token, which is used to represent all the words in the training data that are not in the vocabulary. Size of vocabulary is calculated and returned by the *__len()__* function. The index-to-word list and word-to-index dictionary are created using the words in the dictionary, with higher occurring words having lower indices. "UNK" is given an index of 0 for convenience and because there are usually so many out-of-vocab words in the training dataset.

The *context_target* variable stores a list of tuples containing every possible combination of the context words and the corresponding target word in the training dataset. Window size is set to 2 for faster training, so in each tuple, there are a list of 4 context words and 1 target word. In the *__getitem__()* function, the selected context and target words are converted into their indices and then returned as torch tensors.

For faster instantiation of dataset, the *vocab_size* is set to 10k words for training the embeddings.

2 Continuous Bag-of-Words (CBOW)

The *CBOW* class consists of an Embedding layer, a hidden layer, and an output layer. Here are five hyper-parameter configurations and their resulting losses:

Hyperparameter / Case	Batch Size	Hidden Size	Learning Rate	Loss
1	32	120	0.005	2.1216
2	32	100	0.01	1.3772
3	32	150	0.01	1.2320
4	64	150	0.007	2.6762
5	64	120	0.01	2.1228

Table 1: Hyperparameter Configurations and Losses

We take the best hyperparameter configuration for training the word embeddings: the third case where batch size is set as 32, hidden size as 150, and learning rate as 0.01. The Embedding layer has input size equal to the vocab size, which is 10k, and the embedding dimension is set to 300 for more meaningful representation of the words. During each forward propagation, the inputs with shape [*bsz*, *context_size*] = [32, 4] are converted into embeddings resulting in shape [*bsz*, *context_size*, *embed_dim*] = [32, 4, 300]. Then, the embeddings for each word are added together to represent the whole context with a single vector, therefore giving the shape [*bsz*, *embed_dim*] = [32, 300]. After that, the context embeddings are passed into a hidden layer with hidden size 150 resulting in [*bsz*, *hidden_size*] = [32, 150] and then output layer finally resulting in [*bsz*, *vocab_size*] = [32, 10000]. This final output is passed through the *log_softmax()* function which calculates the log_softmax probabilities of the likelihood of each vocab word.

The CBOW model uses Cross Entropy to calculate loss and Stochastic Gradient Descent with momentum to update its weights with the learning rate of 0.01. It is trained for 40 epochs and the final average loss is **1.2320**. The model is trained using Google Colaboratory notebook with a Nvidia Tesla K80 GPU. The file is named as *loader_CBOW_similarity.ipynb*, and the pdf version *loader_CBOW_similarity.pdf* is provided too in case there are formatting issues.

3 Evaluation

3.1 Word Similarity

The *get_similarity()* function takes two words, converts them into embeddings, and computes their cosine similarity, given by:

$$\text{similarity}(\mathbf{w_1}, \mathbf{w_2}) = \frac{\mathbf{w_1} \cdot \mathbf{w_2}}{\|\mathbf{w_1}\| \|\mathbf{w_2}\|} \tag{1}$$

The higher the value of their similarity, the more closely related the two words are. The following table lists the similarity values of 5 sample pairs of words:

Word Pairs / Case	Word 1	Word 2	Similarity
1	man	woman	0.09862
2	brother	sister	0.08965
3	dog	cat	0.07777
4	chair	throw	0.01732
5	you	i	-0.09269

Table 2: Word Pairs and their Similarities

The similarity function actually gives an accurate depiction of the relationship of the words shown in the table. The word pairs between "man", "woman" and "brother", "sister" are both pretty high, indicating that they have similar relationships. "Dog" and "cat" are both household animals and are shown to be quite related too. On the other hand, because the words "chair" and "throw" do not have similar meanings or a clear correlation, their similarity value is calculated to be relatively low. Also, "you" and "i" are considered to be opposite pronouns, and this concept is captured by the model as their similarity is shown to be very low. From the sample word pairs, we can see that the trained word embeddings can actually predict the relation of two words with an acceptable level of accuracy.

3.2 Sentiment Classifier

The word-to-index dictionary from *MyData* dataset class and the trained word embeddings from CBOW are saved to local drive as "*word2index.npy*" and "*w_emb_mat.txt*" respectively. They are then loaded into the python files "*preprocess.py*" and "*logistic_regression.py*" for sentiment classification from assignment 1. The only difference between the previous implementation and this implementation is that the current one extracts embedding features from tweet sentences instead of frequency-based bag-of-words. The current approach loops over each word in the sentence, converts them into word embeddings using the loaded variables, and sum them up to form as inputs to the logistic regression model. Words are checked against word-to-index; if words are not in word-to-index, they are treated as "UNK".

For the experiment, the model performs string cleaning to preprocess the data. It computes Cross Entropy Loss and performs Gradient descent with momentum to update the weights with a learning rate of 0.02. The model is trained for 200 epochs. The best validation accuracy achieved is **0.6723**%. The prediction results of the twitter sentiment test set is shown in the file "*train_final.csv*".

Well-trained learned embeddings do help with performance for sentiment analysis. Since the embeddings represent their corresponding words as numerical vectors, they actually try to encode their meaning more so than the bag-of-word model. This is because bag-of-word solely looks at the frequency of each vocabulary word in the sentences, but often times, high frequency does not mean that the word is more important. Also, bag-of-word does not accurately encode the meaning of the sentence. Even though the validation accuracy here is shown to be slightly lower than that of Assignment 1, the learned embeddings is still a great input feature for the model. The reason why the accuracy is not as high as expected is likely due to the hyperparameters and the shallow depth of the network. If the model uses an optimal set of hyperparameters with more hidden layers and is trained for a longer period of time, the validation accuracy will increase by a fair amount.

4 Bonus 1: Pretrained Word Embeddings

GloVe is an unsupervised learning algorithm for vector representations of words, and its training is performed on word-to-word co-occurrence statistics from various corpus.

The pretrained GloVe used for this experiment contains 400k words, each represented by a vector of size 300. The algorithm is trained on Wikipedia and Gigaword corpus, altogether 6 billion tokens. It can be downloaded from the official website here and the file name is "*glove.6B.300d.txt*".

This logistic regression model using GloVe performs string cleaning for preprocessing data. It calculates loss using Cross Entropy and updates its weights and parameters through Gradient Descent with momentum, using the learning rate of 0.02. It is trained for 200 epochs. The model achieved a best validation accuracy of **0.73025**%. The code is in "*preprocess_glove.py*" and "*logistic_regression_glove.py*" and the test set predictions are shown in file "*glove_final.csv*".

Using GloVe word embeddings result in relatively high accuracy, because they are trained from large amount of texts. The word embeddings, which are represented in high dimensions, encode the meanings of the words well. Also since GloVe has 400k vocab words as opposed to the previous 10k, training words are less likely to be turned into "UNK," and thus contributing in pretty good overall performance.

5 Bonus 2: Neural Network for Sentiment Analysis

The neural network contains an input layer, hidden layer, RELU layer, and finally an output layer. The Embedding layer in the *Tweets* Dataset Class is initialized using the embeddings trained previously by the CBOW. Whether to continue update the embeddings during the process of sentiment analysis can be controlled using the argument “freeze” provided by the Pytorch Embedding layer itself. The code is provided in the dataset class "*tweets_update.py*" and "*neural_network_update.py*". The default mode does not update the embeddings; to update the embeddings, add the flag "*-ue*" when running the python file.

For both cases 1) fix the word embeddings while training, and 2) keep updating the word embeddings, the neural network performs string cleaning, and utilizes Cross Entropy Loss to calculate loss while relying on Stochastic Gradient Descent with momentum to update its weights and embeddings. The learning rate is set to an optimal value of 0.005. In both occasions, the network is trained for 20 epochs.

The best validation accuracy achieved when embeddings are not updated is **0.6835**% while the best validation accuracy achieved when embeddings are keep being updated during training is **0.7020**%. Their test set predictions are saved in files "*neural_network_final.csv*" and "*neural_network_final_ue.csv*", respectively.

The neural network is generally better than the approach used in purely logistic regression since logistic regression just does simple forward propagation and then backpropagation to optimize the weights. The process is all linear. On the other hand, our neural network has one additional hidden layer which means it has more parameters than logistic regression classifier to train and update. It also has a RELU layer which prevents overfitting on the training data. Both the additional hidden layer and RELU layer contributes to better performance. However, the results actually show similar performance due to the lack of depth of the neural network and the short amount of time it has been trained. If it is trained for longer periods of time to further lower its loss, it will achieve higher validation accuracy than the logistic regression classifier.

The neural network that continuously updates the word embeddings with new words from another dataset increases performance since it trains on extra words. This is similar to transfer learning, where word embeddings are already pretrained on reviews, and now also trained on another dataset of tweets for sentiment classification. As seen from the experiment results, the validation accuracy for this model is slightly higher than its counterpart with freezed, non-updating embeddings.