COMP 4901I: Assignment #3

I-Tsun Cheng
20576079

Hong Kong University of Science and Technology — April 14, 2019

1 Data

1.1 Data Statistics

The following table shows the important information regarding data statistics of the dataset involved:

Data	Statistics
Number of sentences	10000
Number of words	1164819
Vocabulary size	77430
Top 10 Most Frequent Words	'the', 'to', 'I', 'a', 'and', 'is', 'it', 'of', 'for', 'that'
Max Sentence Length	2079
Average Sentence Length	116.4819
Sentence Length Standard Deviation	133.1331
Class Distribution	0:4000, 1:2000, 2:4000

Table 1: Data Statistics

1.2 Preprocessing Methods

The model uses the following methods to process and clean the strings to remove any noisy sentences with unknown words, misspelled characters, etc. for better dataset:

- Remove anything that is not alphanumeric and common punctuations like "(),!?´ from the strings
- Add space before contractions to separate the previous word with contractions so that they get identified as two different words
- Add spaces before and after common punctuations

2 ConvNet with PyTorch

2.1 WordCNN model

The WordCNN model consists of an embedding layer, three convolutional layers, a dropout layer, and a fully-connected layer, which acts as the softmax layer. The model takes in the following arguments to initialize the layers: embedding dimension, number of kernels, kernel sizes (different width and height values of kernels), dropout value (probability of dropout), number of classes, and the vocab size.

Each convolutional layer includes one channel and results in a width value of kernel_num after kernel convolution. The three convolutional layers are convolved using three different kernels with different sizes with the same embedding dimension. The dropout layer zeroes out each neuron with a probability of dropout value. The final fully-connected layer maps a vector with dimension number of kernels for each kernel size multiplied by the number of different kernel size, to the number of classes.

For each input of feature vector of a sentence, each word is converted to its corresponding word embedding. Then it is unsqueezed to increase an extra dimension to account for the convolution afterwards. Convolution by the kernels and RELU nonlinear activation function are applied to the inputs. Squeeze is applied again to reduce one of the dimensions, and then max pooling takes only the maximum value of each vector in the last dimension, followed by another squeeze. The whole convolution, RELU, and max pooling are repeated for two additional times. The final result is finally concatenated and fed to the dropout layer to dropout some values for the sake of better performance on the validation set and avoiding overfitting on the training data. Finally, the processed vector is passed to the fully-connected layer to convert the original input sentence vector into one of the three classes.

2.2 Using Word Embeddings

```
self.embedding = nn.Embedding(vocab_size, embed_dim)
```

This line randomly initializes the embedding layer with *vocab_size* number of embeddings and each embedding of length *embed_dim*.

```
self.embedding.weight.requires_grad = True
```

The above line updates the weights of the embeddings in this layer so they are trained and updated along with the Amazon reviews. The word embeddings are updated per training epoch. This method of input is more effective and results in better performance than simply just by using the sentence Bag of Words feature vector. The embedding lookup updates the embeddings, which attempt to represent semantic and grammatical meaning of the words instead of just purely frequency of the words. The embedding lookup table is thus much more effective than its counterpart and is implemented in this model.

2.3 Hyperparameters

The following table displays each hyperparameter configuration and its corresponding best validation accuracy.

Case	Learning Rate	Dropout Probability	Kernel Number	Kernel Sizes	Embedding Dim	Best Val Acc
1	0.1	0.3	100	3,4,5	100	0.5652
2	0.01	0.3	100	3,4,5	100	0.6316
3	0.1	0	100	3,4,5	100	0.5740
4	0.1	0.1	100	3,4,5	100	0.5012
5	0.1	0.5	100	3,4,5	100	0.5208
6	0.1	0.3	50	3,4,5	100	0.5788
7	0.1	0.3	150	3,4,5	100	0.5616
8	0.1	0.3	100	2,3,4	100	0.5300
9	0.1	0.3	100	4,5,6	100	0.5244
10	0.1	0.3	100	3,4,5	50	0.5652
11	0.1	0.3	100	3,4,5	200	0.5172

Table 2: Hyperparameter Configurations and their Validation Accuracies

In order to evaluate **Average Pooling** as another method to reduce the size of the training data while preserving their content, it is implemented and tested for accuracy. Under the default hyperparameter configuration of the learning rate of **0.1**, dropout probability of **0.3**, kernel number of **100**, kernel sizes of **[3,4,5]**, and the embedding dimension of **100**, using Average Pooling results in the best validation accuracy of **0.5856%**

3 Results and Analysis

The best validation accuracy is **0.6316%**, trained with the following set of hyperparameters: learning rate of **0.01**, dropout probability of **0.3**, kernel number of **100**, kernel sizes of **[3,4,5]**, and the embedding dimension of **100**. The test set predictions using this particular configuration are stored in *submission.csv* file.

3.1 Analysis for hyperparameter set

- Effect of Learning rate:
  Learning rate is one of the most essential parameters that heavily affect the performance of the neural network. Slower learning rate results in slower convergence but consume more time to train. Larger learning rate speeds up the rate the network trains but may cause gradients explosion if it is too high and therefore leading to totally unexpected results. Thus, it is very important to tune the right learning rate for the task at hand. In this case, reducing the learning rate from 0.1 to 0.01 increases the validation accuracy by almost 0.07%, a noticeable difference. This suggests that the network could perhaps increase its performance by setting lower learning rates.

- Effect of Dropout Probability: As shown from the experiment, higher dropout does not increase the performance of the network, as comparing the dropout scenarios with the default configuration. This indicates that dropout in this task is unsuitable and causes lower accuracy unlike most other cases, where dropout helps to prevent overfitting on training data and actually increases validation accuracy.

- Effect of Kernel Number: The experimnent shows that the neural network performs better with lower number of kernels: 50 kernels perform better than 100 kernels, and 100 kernels perform better than 150 kernels. This suggests that having less kernels per kernel size results in higher performance, likely due to too much kernel convolution on the input features. The number of kernels can perhaps be lowered to further increase its performance.

- Effect of Kernel Sizes: The default configuration of kernel sizes has the best validation accuracy. This shows that kernels with sizes of 3,4,5 are more effective than others. In convolution, the kernel sizes are essential in contributing to the final output size, so optimal kernel sizes must be used to prevent any numerical dimension mismatch after convolution.

- Max Pooling vs. Average Pooling: From the results mentioned earlier, Average Pooling with default configuration achieved **0.5856%**, while the Max Pooling achieved a little lower accuracy of **0.5652%**. In image, max pooling has always been shown to perform more consistent and better than average pooling; however, in this scenario, average pooling is shown to be better. One of the probable reasons is that it is not a bad metric to represent a part of a sentence with the average of all its words, and it will likely be better than just taking the most important word. Therefore, average pooling is more effective at reducing down the training size of the data and should be used when conducting similar tasks.

4 Bonus 1: Dynamic padding for each batch

The bonus part is written in the file *preprocess_dpad.py*. First, I uncommented some of the lines in *preprocess* function, which adjusts each sentence to length *max_len*. I wrote a function called *my_collate* which takes each batch as input and calculates the maximum length of the sentences within the batch. Afterwards, the function readjusts the length of each sentence by adding the pad index to the end of the sentences which are shorter than maximum length. Finally, the function returns the rating and the dynamic padded sentences.

This *my_collate* function is then passed to the *collate_fn* parameter of the three dataloaders in *get_dataloaders* function. The dataloaders are for creating batches of data using the function passed to *collate_fn*, for train, validation, and test data.

The *main.py* file has a command-file flag of "-dpad," which when used when running the script, the file will import *get_dataloaders* function from *preprocess_dpad.py* file instead of *preprocess.py*. Thus, it will use dynamic padding if the flag is activated and static padding if it is not activated.