**CS5590 APS - Deep Learning Programming**

**ASSIGNMENT 2**

**Introduction**

In class, we learned about Artificial Neural Networks (ANN) and why we use them. We talked about several different types of ANN including, specifically, feedforward neural networks. These are the simplest type of NN because the information is moving in only one direction from input to output; no cycles or loops are used for FNN. We zeroed in on a more specific type of FNN, the convolutional neural network (CNN), which is the topic of this assignment.

**Objectives**

The objective of this lab is to illustrate how to implement text classification with the CNN model. The fashion\_7000.txt and the lifestyle\_7000.txt files were used as the input data for this assignment (these files can be found on [Github](https://github.com/eldq5d/UMKC-490-Python/tree/master/Lab5)).

**Approaches/Methods**

CNNs are biologically-inspired variants of multilayer preceptrons (MLPs) and there are four main operations involved:

1. Convolution: this is used primarily to extract features from input images or text while maintaining the relationship between these features.
2. Non-linearity: this is used to make the solution applicable to real world problems by using nonlinear mapping called activation function.
3. Pooling, or sub sampling: this is used to reduce the dimensionality of features while maintaining the important information, which may be determined in a variety of different ways (max, average, sum, etc.).
4. Classification: the activation function is applied here to produce the fully connected output layer.

**Workflow**

For this assignment, we used tensorflow, learn from tf.contrib, TextCNN from text\_cnn, and a few other standard libraries (details can be found in the code on [Github](https://github.com/eldq5d/UMKC-490-Python/tree/master/Lab5)).

*PARAMETERS*

After the library installation, we setup the parameters for this model. There are several different steps involved here: we need to define the data loading parameters, the model hyper parameters, the training parameters, and a few other miscellaneous parameters.

For the data loading parameters, we define the percentage of the training data to be used for validation and then we define which data should be use for positive/negative data.

There are several aspects involved in defining the model hyper parameters. We look at character embedding dimensions, coma-separated filter sizes, the number of filters per filter size, the dropout keep probability, and a regulation parameter.

We also have several values to consider for the training parameters: the batch size, the number of training epochs, how frequently the model should be evaluated and saved, and the number of checkpoints that should be saved.

The remaining miscellaneous parameters allow us to decide whether or not we want to allow for soft device placement or if we want to log the placement of ops on devices.

After the parameters have all been set, we then move on to the data preparation steps. These steps include loading the data (using our pre-defined parameters), building the vocabulary of the model, randomly shuffling the data around, and then splitting the data into the training and testing sets.

We begin the training process by initiating a session in tensorflow for our graphs and calling a few of our parameters. We then define the training procedure, including declaring what optimizer we will be using; in this case we used the AdamOptimizer. The next step is optional, but we chose to include it, and is an opportunity to keep track of the gradient values and sparsity summaries. We also define our summaries for loss and accuracy, the training steps, and development, including formatting for their output as well.

We must create the checkpoint directory, the place where the checkpoints will be saved, because tensorflow already assumes the location exists. We then write the vocabulary created and initialize all variables.

The remainder of the code works through the function definitions that we call throughout the training steps and include the single training step, the evaluation of the model on a dev set, and the training loop for each batch which is also created at this time.

**Datasets**

The two datasets we used are purely text files, with several paragraphs each and many sentences composing them. The fashion file is a piece about the fashion industry in England and the lifestyle file is about two different takes on the same concept surrounding the holiday/Christmas season.

**Parameters**

*See ‘PARAMETERS’ under Workflow.*

**Evaluation & Discussion**

When running the program, we compare the value of loss and the value of accuracy after 25 steps (the program is set to do many more than 25 steps but it takes an incredibly lengthy amount of time to run through the entire thing so in order to evaluate more efficiently, we stopped at every 25th step).

I first tried manipulating the dev\_sample\_percentage parameter. From increasing and decreasing this value and running the program each time, I found that the value did not have much effect on the accuracy, but a smaller percentage did lead to a smaller loss.

I next manipulated the droupout\_keep\_prob by decreasing this value 10-fold. The results were almost immediately apparent; the smaller the dropout\_keep, the larger the loss, and also a lower accuracy, even if only slightly.

Then I played with embedding\_dim by cutting this value in half. By doing so, the loss was reduced and the accuracy was increased.

After I had obtained these results, I began using combinations of the different parameters to try to reach the lowest loss with the highest accuracy. This occurred when the dev\_sample\_precentage was 0.2 and the dropout\_keep\_prob was 0.9; at this point I obtained values of 0.617595 for loss, and 0.75 for accuracy.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Step** | **dev\_sample\_precentage** | **dropout\_keep\_prob** | **embedding\_dim** | **Loss** | **Accuracy** |
| 25 | 0.5 | 0.5 | 128 | 1.933 | 0.578 |
| 25 | 0.2 | 0.5 | 128 | 2.423 | 0.531 |
| 25 | 0.1 | 0.5 | 128 | 1.645 | 0.547 |
| 25 | 0.1 | 0.1 | 128 | 5.801 | 0.453 |
| 25 | 0.1 | 0.1 | 60 | 0.990 | 0.641 |
| 25 | 0.5 | 0.9 | 128 | 1.128 | 0.531 |
| 25 | 0.1 | 0.9 | 128 | 0.743 | 0.688 |
| 25 | 0.2 | 0.9 | 128 | 0.618 | 0.750 |

Table 1

**Conclusion**

In conclusion, this exercise proved to be extremely helpful and educational for learning how to utilize Artificial Neural Networks, specifically the feedforward type of Convolutional Neural Network.