CSCI544: Homework Assignment №3

Due on March 1, 2023 (before class)

This assignment is an extension to HW assignment 1. Please follow the instructions and submit a zipped folder containing:

1. A PDF containing a Jupyter Notebook response to the assignment with sufficient comments and explanations. Your Jupyter Notebook should contain both code and text cells with sufficient comments such that the reader can understand your solution as well as your responses for some of the questions. On the Jupyter notebook, please print the requested values, too. If it is more convenient, you can also submit a PDF similar to assignment 1, i.e., initially explaining your solution and then merge a Jupyter notebook.

2. You also need to submit your Jupyter Notebook separately in .ipynb format such that it can be easily executed. Please include the version of required dependencies. You can consider that data.tsv is a raw dataset in the current directory that your notebook should read and perform all the required steps and generate the desired outputs.

You can use PyTorch or TensorFlow to implement the neural network models in this assignment. Please name your zipped file "HW3-YourFirstName-YourLastName-L.zip", where L should be either "PT" or "TF" which denotes whether you have used PyTorch or TensorFlow. You can also use publicly available implementations in portions of your solution but you need to include proper reference to your resources, e.g., url to the page that you used as a reference, books, etc.

1. Dataset Generation

We will use the Amazon reviews dataset used in HW1. Load the dataset and build a balanced dataset of 60K reviews along with their ratings to create labels through random selection similar to HW1. You can store your dataset after generation and reuse it to reduce the computational load. For your experiments consider a 80%/20% training/testing split.

2. Word Embedding (25 points)

In this part the of the assignment, you will generate Word2Vec features for the dataset you generated. You can use Gensim library for this purpose. A helpful tutorial is available in the following link:

`https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html`

(a) (5 points)

Load the pretrained "word2vec-google-news-300" Word2Vec model and learn how to extract word embeddings for your dataset. Try to check semantic similarities of the generated vectors using three examples of your own, e.g., $King - Man + Woman = Queen$ or $excellent \sim outstanding$.

(b) (20 points)

Train a Word2Vec model using your own dataset. You will use these extracted features in the subsequent questions of this assignment. Set the embedding size to be 300 and the window size to be 13. You can also consider a minimum word count of 9. Check the semantic similarities for the same two examples in part (a). What do you conclude from comparing vectors generated by yourself and the pretrained model? Which of the Word2Vec models seems to encode semantic similarities between words better?

For the rest of this assignment, use the pretrained "word2vec-google-news-300" Word2Ve features.

3. Simple models (20 points)

Using the Google pre-trained Word2Vec features, train a single perceptron and an SVM model for the classification problem. For this purpose, use the average Word2Vec vectors for each review as the input feature ($x = \frac{1}{N}\sum_{i=1}^{N} W_i$ for a review with N words). Report your accuracy values on the testing split for these models similar to HW1, i.e., for each of perceptron and SVM models, report two accuracy values Word2Vec and TF-IDF features.

What do you conclude from comparing performances for the models trained using the two different feature types (TF-IDF and your trained

Word2Vec features)?

4. Feedforward Neural Networks (25 points)

Using the Word2Vec features, train a feedforward multilayer perceptron network for classification. Consider a network with two hidden layers, each with 100 and 10 nodes, respectively. You can use cross entropy loss and your own choice for other hyperparamters, e.g., nonlinearity, number of epochs, etc. Part of getting good results is to select suitable values for these hyperparamters.

You can also refer to the following tutorial to familiarize yourself:

`https://www.kaggle.com/mishra1993/pytorch-multi-layer-perceptron-mnist`

Although the above tutorial is for image data but the concept of training an MLP is very similar to what we want to do.

(a) (10 points)

To generate the input features, use the average Word2Vec vectors similar to the "Simple models" section and train the neural network. Report accuracy values on the testing split for your MLP.

(b) (15 points)

To generate the input features, concatenate the first 10 Word2Vec vectors for each review as the input feature ($x = [W_1^T, ..., W_{10}^T]$) and train the neural network. Report the accuracy value on the testing split for your MLP model.

What do you conclude by comparing accuracy values you obtain with those obtained in the "'Simple Models" section.

5. Recurrent Neural Networks (30 points)

Using the Word2Vec features, train a recurrent neural network (RNN) for classification. You can refer to the following tutorial to familiarize yourself:

`https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html`

(a) (10 points)

Train a simple RNN for sentiment analysis. You can consider an RNN cell with the hidden state size of 20. To feed your data into our RNN, limit the maximum review length to 20 by truncating longer reviews and padding shorter reviews with a null value (0). Report accuracy values on the testing split for your RNN model.

What do you conclude by comparing accuracy values you obtain with those obtained with feedforward neural network models.

(b) (10 points)

Repeat part (a) by considering a gated recurrent unit cell.

(c) (10 points)

Repeat part (a) by considering an LSTM unit cell.

What do you conclude by comparing accuracy values you obtain by GRU, LSTM, and simple RNN.

Note: In total, you need to report accuracy values for:
2 (Perceptron + SVM) + 2 (FNN) + 3 (RNN) = 7 cases.