CSE582 NLP: Homework #2

Due on April 9, 2023 at 11:00pm

Professor Yin Spring 23

Collin Beaudoin 956850763 cpb5867@psu.edu

Pre-processing

The implementation of the CNN and LSTM models required an identical processing, which only diverged on the model definition. To begin, the first 100k samples from the yelp data set were loaded. The data set was originally stored using the JSON format. To use pandas for additional processing commas were inserted, and the trailing new line characters were removed. To reduce future run-times, I decided to add a check which would see if a saved version of the processed data existed before performing the processing. Once this data has been loaded into python using pandas, the reviews had to be converted for sentiment analysis. Each review had a given count of stars, in cases where there were 2 or less stars it was considered a negative review. In cases with 3 stars, it was considered a neutral review and in cases with more than 3 stars it was considered a positive review. Unfortunately, the data set contained a bias towards positive reviews. To mitigate the bias the top 10k samples from each of the positive, negative, and neutral reviews were taken. After removing the bias of the data set, the text of the reviews was tokenized using simple_preprocess. This removes words shorter than 2 characters, longer than 15 characters, accent marks on characters, and converts all characters to lowercase. The tokenized text was then stemmed using the Porter stemming method. This removes a word's prefix and suffix to leave only the root word. Once the pre-processing was completed, the training and testing data was made using a 70-30 split. The stemmed and tokenized data was also used to create a word2vec embedding that is used for training both models.

CNN

For the CNN model, each input is embedded using the word2vec embedding that was generated. This is then fed to multiple 2D convolution layers. Each convolution layer used a kernel of a varying window size ranging from: 1,2,3 and 5. Padding was used to ensure each resulting vector would match the input size. Each of the resulting convolution outputs were then concatenated together and fed to a feed-forward network, which shrunk the dimension of each vector to the number of classes. The softmax of this was then taken to get the predicted probability of each class.

LSTM

For the LSTM model, each input is embedded using the word2vec embedding that was generated. This is then fed to the LSTM. The LSTM model uses a hidden dimension that matches the embedding size used for the input. This was in hopes of allowing for the largest possible search space within the LSTM. Once the result of the LSTM is produced the final state is taken, as this should contain all required information for the given sentence. The state vector is fed to a fully connected layer that reduces the dimension to the number of classes. The softmax of this was then taken to get the predicted probability of each class.

Comparison of CNN to LSTM

While running the models the LSTM took roughly 7 hours to complete training while on a GPU. In comparison the CNN model was able to complete, using the same number of epochs, in less than an hour on the same GPU. Examining the parameters of the 2 models, CNN used only roughly 55k parameters while the LSTM required over 2M. In the case of the CNN these parameters are divided across 4 different convolution layers, while the LSTM only has a single layer being used.

Comparison of Activation Functions

I focused my testing of activation functions on the CNN model. I opted to only use the CNN due to the quicker run-time. While performing my testing I tried switching between tanh, ReLU, and LeakyReLU. I found the ReLU and LeakyReLU to be working with near identical results, while tanh would consistently obtain 1-2% less testing accuracy than the ReLU options. The lower performance is likely due to the saturation of tanh. In the case of the LeakyReLU, I believe there is potential to obtain higher results with a greater alpha value.