Machine Learning Engineer Nanodegree

Capstone Proposal

Domain Background

The project we will work on is performing the sentiment classification for a large movie review dataset from IMDB. We will classify/predict for the positive or negative reviews based on the review in the training set. This kind of task belong to the domain of Nature Language Processing (NLP). What makes this problem difficult is that the sequences can vary in length, be comprised of a very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols in the input sequence. In this project, we will develop several LSTM recurrent neural network models for sequence classification problem specified, compare the performance of each model, and conclude the most suitable model for this particular problem.

Problem Statement

The dataset is the Large Movie Review Dataset often referred to as the IMDB dataset.

The Large Movie Review Dataset contains 25,000 highly polar moving reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given moving review has a positive or negative sentiment.

The data was also used as the basis for a Kaggle competition titled "Bag of Words Meets Bags of Popcorn" in late 2014 to early 2015.

In this project, multiple solutions will be proposed – the predicting result/performance on test set will be measured by the model prediction metrics "accuracy" in Keras.

Text classification/sentiment analysis problem can be easily found in our daily life – Yelp review, movie review, survey for something, etc. The techniques suggested in this project can be easily applied to these applications.

Datasets and Inputs

The dataset used in this project is from the link:
http://ai.stanford.edu/~amaas/data/sentiment/

This is a publicly available dataset, and is used in one of the Kaggle competition. The data (IMDB movie reviews) are written in 50,000 text files, 25,000 files are for training data, 25,000 files are for testing data – among these, 50% are positive reviews and 50% are negative reviews – 12,500 positive reviews, and 12,500 negative reviews in each of training and testing dataset. The contents of the text files contain the review (text) part. The directory/file structure of the dataset (after untar and unzip) looks like below: **aclImdb** (root)/**train**(or **test**)/**pos**(or **neg**)/**file**(**x_y**). File is named in the format: **file_id** (**x**)– unique id for each review entry, **number**(**y**) (integer 1~4 means negative review, integer 7~10 means positive review).

One input file we will use: "glove.6B.50d.txt". This is the Global Vectors for Word Representation (6B tokens, 400K vocab, uncased, 50-dimensions vector) file we will use for word embedding layers in Keras. Each line of the file is in this format: "word word_vec" – a word followed by 50 float point numbers (which are the 50-dimensions word vector).

Regarding word embedding layer in Keras: in order to perform text classification using Deep Neural networks, we may need to convert the words in reviews into real valued vectors – this is a popular technique when working with text. This is a technique where words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.

Something specific for Glove: GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

More descriptions about Glove can be found in the following 2 links:

1) https://nlp.stanford.edu/pubs/glove.pdf

2) https://nlp.stanford.edu/projects/glove/

Citing some descriptions from the first link: most word vector methods rely on the distance or angle between pairs of word vectors as the primary method for evaluating the intrinsic quality of such a set of word representations. Recently, Mikolov et al. (2013c) introduced a new evaluation scheme based on word analogies that probes the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation king − queen = man − woman. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations (Bengio, 2009). The two main model families for learning word vectors are: **1) global matrix factorization methods**, such as latent semantic analysis (LSA) (Deerwester et al., 1990) and **2) local context window methods**, such as the skip-gram model of Mikolov et al. (2013c). Currently, both families suffer significant drawbacks. While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts. In this work, we analyze the model properties necessary to produce linear directions of meaning and argue that global log-bilinear regression models are appropriate for doing so. We propose a specific weighted least squares model that trains on global word-word co-occurrence counts and thus makes efficient use of statistics. The model produces a word vector space with meaningful substructure, as evidenced by its state-of-the-art performance of 75% accuracy on the word analogy dataset.

Other references for the problem and dataset explanations can be found at:

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/

Solution Statement

At least 2 different methods will be used as solutions - each method will have at least one model for solution. One method uses pre-trained Glove (Global Vectors for Word Representation) glove.6B.50d (6B tokens, 400K vocab, uncased, 50 dimension vector) as word embedding layer, the other one use vocabulary/tokens generated from the training set (with proper text cleaning by ourselves, and data preprocessing by using tools in Keras for tokenizing the words in the review) as word embedding layer. The models for solutions will include the following combinations of neural network layers – embedding layer, (optional convolutional layer+maxpooling layer for maximizing performance), LSTM layer, dropout layer, and Dense layer. We will also add models, which use vocabulary generated/loaded by using imdb.load_data API provided by Keras, to compare with our results. Imdb.load_data API returns the training/testing data set which have been converted to real-valued vectors, which can be easily used later for embedding layer.

Note: the vocabulary generated by Tokenizor class in Keras is ordered by the word frequency – the word embedding generated by this technique is more related to model 1) mentioned in previous section (global co-occurrence counts), while the pre-trained Glove word embedding relates to both model 1) and model 2) mentioned above.

Regarding RNN: The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that’s a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far. Because of the nature that the output of current time step is depending on the output from previous time step, RNN is a good choice in predicting trend in sequence (time-series) data like stock marketing price, and many NLP tasks such as text classification (sentiment analysis), machine translation, speech recognition, etc. Our model will be based on RNN networks.

Regarding LSTM: Long short-term memory (LSTM) units are units of a recurrent neural (RNN). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying and making predictions based on time-series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problems that can be encountered when training traditional RNNs.

Regarding using convolutional neural network in sentiment classification: convolutional neural networks excel at learning the spatial structure in input data. The IMDB review data does have a one-dimensional spatial structure in the sequence of words in reviews and the CNN may be able to pick out invariant features for good and bad sentiment. This learned spatial features may then be learned as sequences by an LSTM layer. Another advantage of using CNN is: it will greatly reduce the training time – especially when we have a large vocabulary and large dataset.

Reference for RNN:

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Reference for LSTM network:

https://en.wikipedia.org/wiki/Long_short-term_memory.

Reference for using CNN in sentiment analysis:

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Benchmark Model

The benchmark model we will use in this project is: Naïve Bayes with Bag of Words. We will use CountVectorizer class in scikit-learn to build a vocabulary, and tokenizing the training text. We then use MultinomialNB classifier (a Naive Bayes classifier for multinomial models) to fit the training set and predict on the test set. Naïve Bayes plus Bag of Words is the simplest but almost the most common model for NLP tasks. This combination should be suitable for our benchmark model.

Evaluation Metrics

The predicting result/performance on test set will be measured by the model prediction metrics "accuracy" in Keras. This is widely used by solutions using Deep Neural Network on Keras programming framework. We just need to specify the evaluation metrics in model.compile API call and collecting the output from model.evaluate API call.

Project Design

The workflow for approaching the solutions likes below:

1) Import necessary libraries.
2) Data preprocessing: the data (IMDB movie reviews) are written in 50,000 text files, 25,000 files are for training data, 25,000 files are for testing data – among these, 50% are positive reviews and 50% are negative reviews – 12,500 positive reviews, and 12,500 negative reviews in each of training and testing dataset.
    2.1) Read out each review, perform text cleaning - removing punctuations and numbers from the words, split words in a line to a list of words, removing stop words (to, at, the, etc), removing words with length equals 1.
    2.2) Save words in a Counter (dictionary: word vs the count for occurrences) for data analysis purpose.
    2.3) Save each cleaned review data (text, and label) in a row of pandas Dataframe – we will save cleaned training and testing review in separate Dataframes.
3) Data analysis and data visualization: for word embedding, we need to determine the size of vocabulary and maximum words from each review. Perform data analysis for determine these numbers. Note: the vocabulary size for Glove is fixed, only maximum words for the review will be used for models using Glove.
4) Build benchmark model - implement Bag of Words/tokenize training set plus use MultinomialNB classifier to predict on the test set. Get the baseline score for the benchmark model.
5) Implement routines for word embedding using Glove:
    5.1) Read out words and vectors for words from glove.60B.50d.txt and created 3 dictionaries: word_to_vec, word_to_index, index_to_word.
    5.2) Convert words (text) to vector for words, padding 0 the input review < maximum words for each review.

6) Build 2 models for word embedding using Glove. One model only uses word embedding layer, LSTM, and Dense layer, the other model uses word embedding layer, Convolutional Neural Network, MaxPooling Layer, LSTM and Dense layer. Train the models and evaluating performance on test set.
7) Implement routines for embedding using Keras Tokenizer:
   7.1) Create a Tokenizer object, and fit it with the review (text) in the training set. This will create and load a vocabulary of size X determined by us in 3) – vocabulary contains the top X most common words in the training set.
   7.2) Convert the texts in the review of training set into sequences of integers – these are the indexes into the vocabulary. The indexes are ordered according to the appearing frequency of the words in the training set. Pad the sequences with 0 if the sequences are shorter than the max word chosen.
8) Build 2 models for word embedding using Keras Tokenizer. One model only uses word embedding layer, LSTM, and Dense layer, the other model uses word embedding layer, Convolutional Neural Network, MaxPooling Layer, LSTM and Dense layer. Train the models and evaluating performance on test set.
9) Use imdb.load_data in Keras to load the vocabulary (built from top X common words in the dataset, indexes of word is according to term (word) frequency) and X_train, y_train, X_test, y_test. Note: X_train, y_train, X_test, y_test have been converted to integers (indexes to the vocabulary). Pad the training/test data with 0 if the sequences are shorter than the max word chosen.
10) Built 2 models – similiar to 6) and 8) and evaluating the performance using test set.
11) Comparing the performance of each model, and pick most suitable model for our problem.