Machine Learning Engineer Nanodegree

Capstone Project

Xiaodong TAN
31 March 2016

I. Definition

1.1 Project Overview

Sentiment analysis, sometimes known as opinion mining, is one of the main applications of natural language processing techniques. Its main aim is to determine if an individual's attitude/reaction towards a certain topic or event is positive or negative by analyzing his or her writings, voices or something else. Sentiment analysis is widely used in market research and customer analysis, but can also be used in other areas such as analyzing patients' feelings in the clinic settings and analyzing public opinion.

This aim of the project is to determine if the sentiment of a given review of a movie on IMDB is positive or negative. The project is inspired by Kaggle competition "Bag of words meets bags of popcorn".

1.2 Problem Statement

The task of this project is a binary classification task. The following models are trained or designed to complete the task.

- Models that convert a single word to a numerical expression and/or convert a paragraph/review to one or a sequence of numerical expression(s). The models explored in this project include bag-of-words and word2vec.
- Classification models that take the numerical representation(s) of text as input and sentiment (positive or negative) as output. The models explored in this project include random forest, linear SVM, logistic regression and recurrent neural network with long-short-term-memory structure.

1.3 Metrics

Accuracy is used as the main evaluation matrix of this project. It is defined as the proportion of correctly classified reviews (true positive and true negative) in all the reviews. It is chosen as the dataset used for the project is balanced, i.e., there are equal numbers of positive and negative reviews. Accuracy is used as the evaluation matrix in most of the other research on sentiment analysis using a similar dataset (e.g., Socker etc., 2013).

II. Analysis

2.1 Data Exploration

The datasets used in this project is the labeled movies reviews from IMBD, used in Learning Word Vectors for Sentiment Analysis (Mass et al, 2011). The dataset contains 50,000 unique labeled reviews (25,000 in the training folder and 25,000 in the testing folder) and an additional 50,000 unlabeled reviews. In the entire collection, no more than 30 reviews are allowed for each movie because reviews for the same movie tend to have correlated ratings.

According to the description of the datasets, in the labeled datasets, a negative review has a score <= 4 out of 10, and a positive review has a score >= 7 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets, which makes the binary classification task easier. The numbers of positive and negative reviews are balanced in each dataset. In the unlabeled set, reviews of any rating are included and there are an even number of reviews > 5 and <= 5.

2.2 Algorithms and Techniques

Represent a word with numerical value(s)

Bag-of-words is a simple way to convert texts to numerical values. The method learns a vocabulary from all the labeled data, and represents each datapoint (i.e., a review) by counting the number of times each word in the vocabulary appears in the review. The length of the vocabulary should be limited to a reasonable number if it is too large.

Another way to numerically represent texts is to transform each word of the text to a vector (“word embedding”). This transformation should preserve the semantics of words. That is, if the meanings of two words are close, their vectors should be “close” too. This method is usually not task specific and can take non-labeled dataset as the corpora. An example of word embedding methods is Google’s word2vec model. The model takes sentences as input, learns a vocabulary and converts each word in the vocabulary into a vector with a fixed number of features. It provides two architectures (skip-gram or continuous bag of words), and two training algorithm (hierarchical softmax or negative sampling). Skip-gram architecture and hierarchical softmax are better for infrequent words while negative sampling is better for frequent words.

Represent paragraphs with numerical values

After a vocabulary is learnt by the word2vec model and each word is represented by a vector, a paragraph can be represented by simply averaging the vectors of all its words (“averaging”). Another way would be using clustering algorithms, such as K-means algorithm to form clusters of words in the vocabulary based on the similarities between the words, and then representing a paragraph by a vector of items, each item representing the number of words in a certain cluster (“clustering”).

However, the representations of paragraphs using bag-of-words or word2vec do not take the orders of the words into consideration. So some semantics are lost during the conversion. Recurrent Neural Networks (RNNs) described below can overcome the issue by taking ordered sequence of words as input. To fit into the model, each review is represented by a sequence of vectors of each word in the sentence, and the length of the sequence should be the same for all the reviews.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have proven to be successful in many natural language tasks, including in learning language models that outperform n-grams (Mikolov, Karafiat et al., 2011), and in achieving the state-of-the-art in speech recognition (Hannun et al., 2014). RNNs are a type of neural network that contains directed loops. These loops represent the propagation of activations to future inputs in a sequence. Once trained, instead of accepting a single vector input x as a testing example, an RNN can accept a sequence of vector inputs (x1, . . . ,xT) for arbitrary, variable values of T, where each xt has the same dimension. In our specific application, each xt is a vector representation of a word, and (x1, . . . ,xT) is a sequence of words in a movie review. As shown in Figure 1, we can imagine “unrolling” an RNN so that each of these inputs xt has a copy of a network that shares the same weights as the other copies. For every looping edge in the network, we connect the edge to the corresponding node in xt+1’s network, thus creating a chain, which breaks any loops and allows us to use the standard backpropagation techniques of feedforward neural networks.

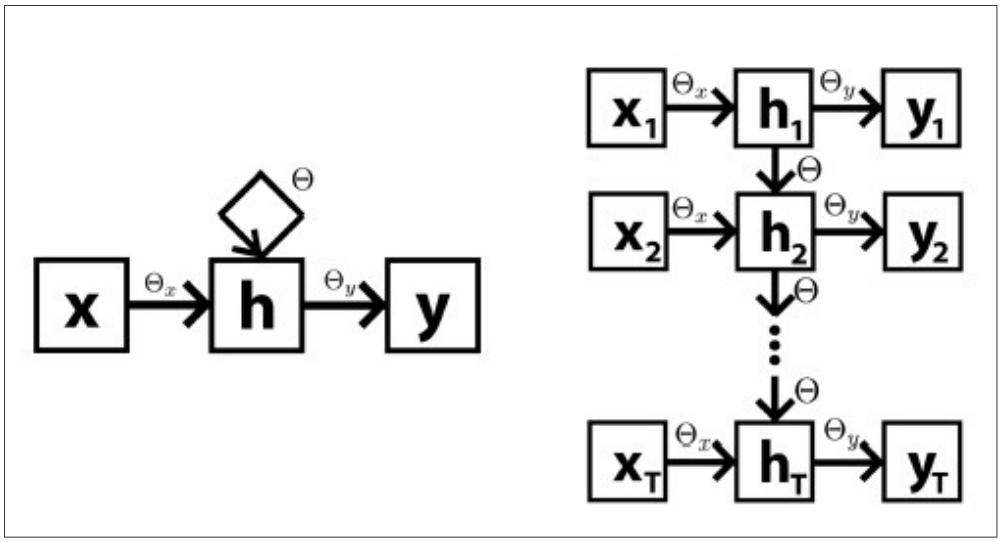

Figure 1: An RNN as a looping graph (left), and as a sequence of time steps (right)

One of the instabilities of the standard RNN is the vanishing gradient problem. The problem involves the quick decrease in the amount of activation passed into subsequent time steps. This limits how far into the past the network can remember. In order to alleviate this problem, Long Short Term Memory (LSTM) units, referred to as “memory cell”, were created to replace normal recurrent nodes. These units introduce a variety of gates that regulate the propagation of activations along the network. This, in turn, allows a network to learn when to ignore a new input, when to remember a past hidden state, and when to emit a nonzero output.

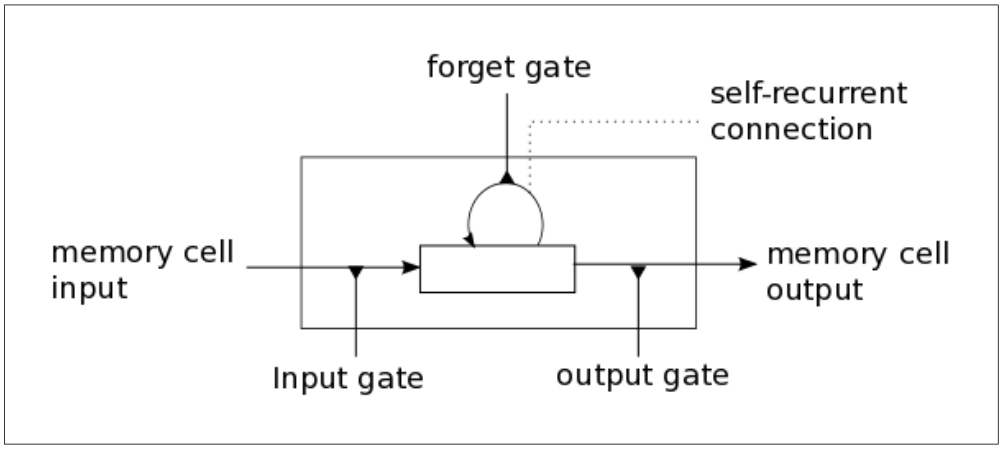

Figure 2: LSTM with memory cell

2.3 Solutions and Benchmark

In this project, word2vec model and LSTM network are used as the main solution.

The benchmark models used in this project include combinations of different models to represent text with numerical values and different classification models. Examples of these benchmark models include bag of words + random forest/SVM/logistic regression and words to vectors + averaging/clustering + random forest/SVM/logistic regression.

III. Methodology

3.1 Creating Datasets

The following four datasets were created for the project, each containing equal number of positive and negative reviews.

- The unlabeled set which contains 75,000 unlabeled reviews, 25,000 from the “training folder” and 25,000 from the “unlabeled folder”. The set was used to train the word2vec model(s).
- The training set which contains the 25,000 labeled reviews from the “training folder”. The set was used to train bag-of-words model and all the classification models.
- The validation set which contains 12,500 labeled reviews, a random half-split from the “testing folder”. The set was used to evaluate and tune the classification models.
- The test set which contains 12,500 labeled reviews, i.e., another half of the data from the “testing folder”. The set was used only to validate the finalized model.

3.2 Cleaning and Preprocessing Data

The reviews in the datasets contain HTML tags as they were directly extracted from the website of IMDB. These tags were removed by using the BeatifulSoup library.

To prepare data to train the bag-of-words model, punctuation, numbers, and stop-words that carry no meanings were removed from the reviews. After preprocessing, each review was converted to unicode containing small-case words separated by space.

To prepare data to train the word2vec model, each review was converted to a list of sentences using nlkt, and each sentence is represented by a list cleaned words, that is, the punctuation numbers and other non-letter were removed, and the all the words were converted to lower cases. The stop_words were not removed as they might carry meanings in the context for the word2vec model to learn. After preprocessing, the whole unlabeled set was converted a list of 808,700 sentences, each represented by a list of cleaned words.

To prepare the data for transformations before training the classification models, each review was converted to a list of cleaned words with punctuation, numbers and stop_words removed.

Please see notebook 1clean_preprocess_data.ipynb for the details.

3.3 Implementation

Train a Bag-of-words model and classification models

The feature_extraction module from scikit-learn was used to train a bag-of-words model using the training set. The maximum features were set to be 5,000. Each review is then represented as an array of the shape (, 5,000) and each item in the array represents the number of times a word in the vocabulary appears in the given review.

The training, validation and test sets were then processed using the bag-of-words model, to vectors of the shape (25,000, 5,000), (12,500, 5,000) and (12,500, 5,000) respectively. The training data was then used as input to train a random forest classifier (with 100 trees), a linear SV classifier with default setting, and a logistic regression classifier with default setting respectively. The performances of the models were tested on the validation set.

Please see notebook 2bw_clf.ipynb for the details.

Train a Word2vec model

Google's word2vec model was used to train a word2vec model using the unlabeled set. Initially, as suggested by the Kaggle tutorial and google documentation, the default architecture (skip-gram) and training algorithm (Hierarchical softmax) were used and other parameters were set as follows.

- Word vector dimensionality = 300
- Minimum word count = 40
- Number of threads to run in parallel = 4
- Context window size = 10
- Downsample setting for frequent words = 1e-3

The word2vec model generated a vocabulary of 16,490, and each word included in the vocabulary is represented as a vector of 300 features. The model can identify words with similar meanings, as they are represented with "similar vectors". Using the method of TSNE, the relationship between the most frequent 150 words in the vocabulary is displayed as follows (Figure 3).

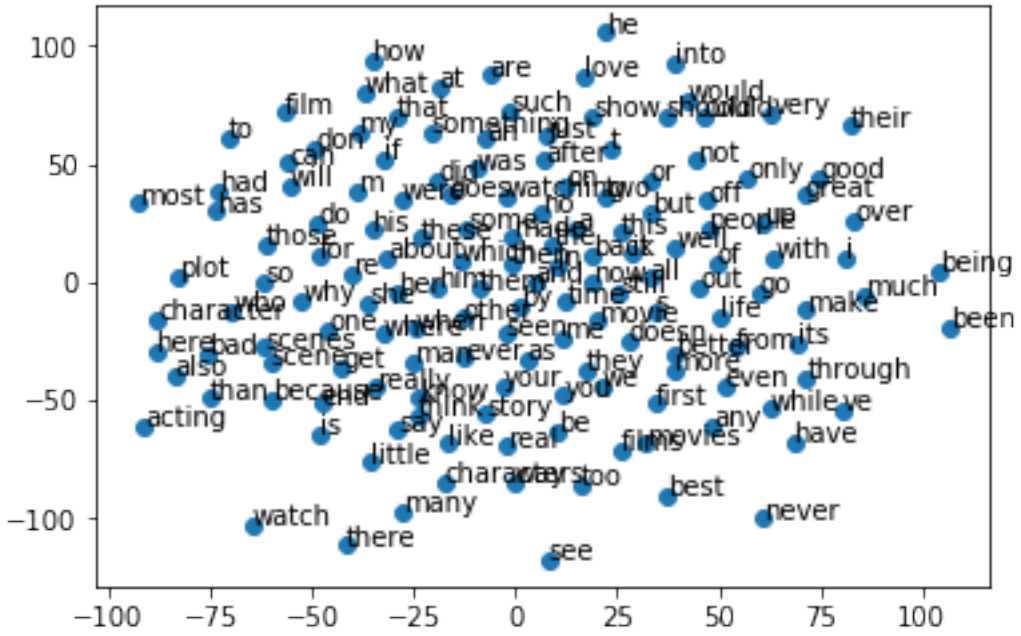

Figure 3: The relationship between the 150 most frequent words in the vocabulary

The model identified words with similar meaning as “awful” (Figure 4), which can be very important in determining the sentiment of a review.

```
model_w2v1.most_similar("awful")
[(u'terrible', 0.7557451725006104),
 (u'atrocious', 0.7374610900878906),
 (u'dreadful', 0.732129693031311),
 (u'horrible', 0.7258363366127014),
 (u'abysmal', 0.7127856016159058),
 (u'horrendous', 0.664879560470581),
 (u'appalling', 0.6576082110404968),
 (u'lousy', 0.6411954164505005),
 (u'amateurish', 0.6183844804763794),
 (u'horrid', 0.6154941320419312)]
```

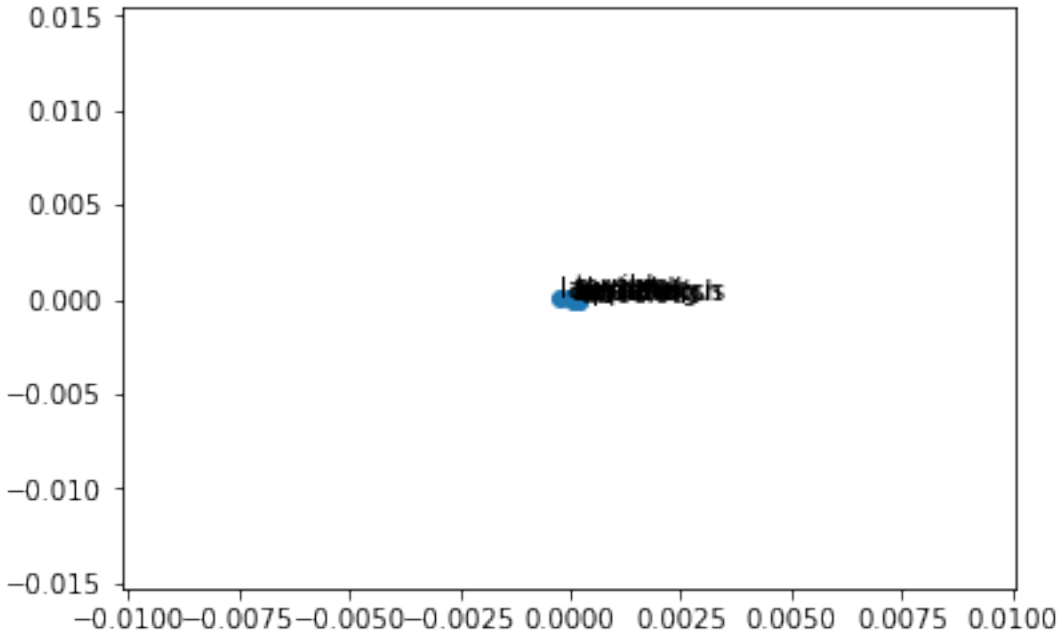

Figure 4: The relationship between the words with similar meanings as “awful”

Please see notebook 3w2v.ipynb for the details.

Represent each review with a single array and train classification models

Averaging: Following the model of word2vec, each word in a given review is now represented by a vector of the shape (,300) and the entire review can be represented by averaging the vectors of all the words. After averaging, each review still has the shape (, 300).

Clustering: Following the vocabulary built by the word2vec model, k-means clustering was implemented to the vocabulary, so that 1) similar words that carry the simialar meanings belong to the same cluster and 2) each cluster contains only 5 words, which is suggested by Kaggle’s tutorial, as trials and error show that using small clusters with 5 words performs better than using larger clusters. 3298 clusters were created from the implementation. Then a map containing each word from the vocabulary and the index of the cluster it belongs to was formed. Each review is then represented by an array of the shape (, 3298), with each item in the array representing the number of words belonging to that cluster.

The datasets were then processed using the word2vec model and averaging/clustering method. The training data was used as input to train a random forest classifier (with 100 trees), a linear SV classifier

with default setting, and a logistic regression classifier with default setting respectively. The performances of the models were tested on the validation set. Please see notebooks 4average_clf.ipynb and 4cluster_clf.ipynb for the details.

Represent each review with a sequence of arrays

100 was chosen as the length of the sequence for each review, as the median of the number of words the cleaned reviews contained is around 100. The distribution of the length of the cleaned reviews is shown as follows.

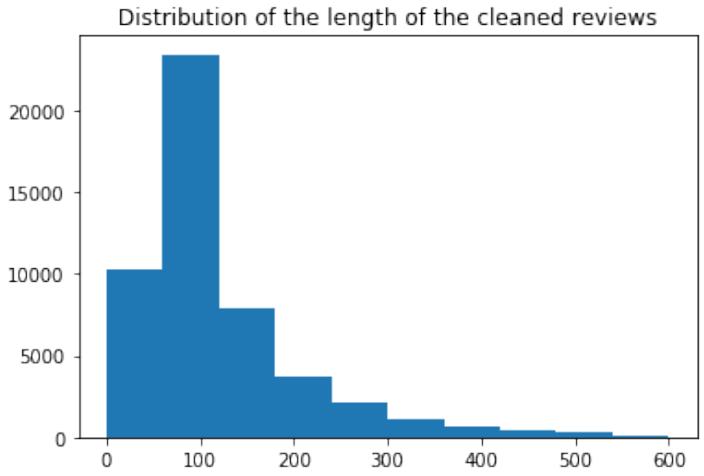

Figure 5: Distribution of the length of the cleaned reviews

Each review is then represented by an array of shape (100, 300), of which 300 is the number of features chosen for the word2vec model. The representations are used as input to train LSTM network as follows.

Train a LSTM network model

Keras was used to train LSTM network models. The initial model includes a LSTM layer, a dropout layer and an output layer with softmax activation. The batch size is 256 and the epoch number is 40.

3.4 Refinement

The following hyper-parameters were explored for the word2vec model and the performance was tested using averaging method and the random forest, logistic regression and SVM classification models. Eventually the original hyper-parameters were kept.

- Number of features. Documentation suggested the more dimension included, the better the results usually are, but not always. When the number of features was adjusted to 600, the models got slight improvement with big cost of the increasing transformation time, and the doubled size of dataset after transformation. As a result, I decided that 300 is a reasonable choice for this project.
- Sub-sampling size: the suggested size by google documentation is 1e5 – 1e3. When 1e5 was chosen, the accuracy of the models dropped significantly.
- Structure and training algorithms. Except for the default architecture (skip-gram) and training algorithm (Hierarchical softmax), negative sampling was tried with a smaller number of features (100) as suggested. The performances worsen significantly after the adjustment.

Based on some online tutorial and literature research,[1] the following factors were explored for the architecture of LSTM.

- regularization: adding more dropout is a way to deal with over-fitting and improve performance. In the learning process of a neural network, dropout randomly "switches off" neurons alternatively, so the neural network has to learn multiple independent representations of the same data. By doing so, the network is prevented from co-adapting too much, which makes overfitting less likely. Some literature suggested dropout should not be added to the recurrent layers.[2]
- optimizer/Learning rate: except for rmsprop, adagrad is also considered to be a good choice for LSTM structure.
- stack layers: stack layers can deepen the network, which might help improve performance

---

1 https://deeplearning4j.org/lstm

2https://www.quora.com/How-do-I-tune-the-parameters-for-the-LSTM-RNN-using-Keras-for-time-series-modeling

https://arxiv.org/pdf/1409.2329.pdf

https://arxiv.org/abs/1512.05287

- initial LSTM layer: softsign is said to be a good activation function

The following four LSTM architectures were implemented and tested on the validation test to find the best model.

	optimizer	Initial LSTM layers	No. of LSTM layers	dropout between layers
Architecture1 (initial)	*rmsprop*	no activation no dropout	1	0.2
Architecture2	*adagrad*	activation = “softsign” dropout_W =0.5	1	0.5
Architecture3	*adagrad*	activation = “softsign” dropout_W =0.5	3	0.5

Table 1: Architectures for LSTM models

Please see notebooks 4sequence_lstm for the details.

IV. Results

The performances of the all the models trained in this project are summarized as follows.

	Text models	**Classification model**	**Validation set**	**Test set**
Baseline1	bag of words	Random Forest	0.8464	0.8450
		Linear SVC	0.8142	0.8510
		Logistic Regression	0.8434	0.8254
Baseline2	word2vec model + averaging	Random Forest	0.8231	0.8296
		Linear SVC	0.8732	0.8718
		Logistic Regression	0.8738	0.8724
Baseline3	word2vec model + clustering	Random Forest	0.8473	0.8472
		Linear SVC	0.8564	0.8600
		Logistic Regression	0.8462	0.8530
Model 1	word2vec model	LSTM architecture1	0.8544	0.8510
Model 2	word2vec model	LSTM architecture2	0.8693	0.8670
Model 3 (Final model)	word2vec model	LSTM architecture3	0.8718	0.8732

Table 2: Accuracy of different models

The results show that models using word2vec have better performance than models using bag-of-words. Models using LSTM reached accuracy of over 0.85 and even 0.87 on the validation set, which are generally better than other models (although surprisingly, baseline 2 models with linear SVC and logistic regression classification models had accuracy of over 0.87). The final model (model 3), which generates over 0.87 on the validation set, also generates the best result in the test set.

Although the simplest LSTM architecture 1 had an accuracy of over 0.85, it suffered from strong over-fitting. As is shown in the charts below, with more iterations, the accuracy reached over 0.99 on the training set but did not improve much on the validation set, and the loss on the validation set was not even decreasing. By adding more dropout, Model 2 and 3 improved significantly from Model1 with gradually increasing accuracy, decreasing loss on the validation set and reached an reasonable loss of 0.3 or below in the end.

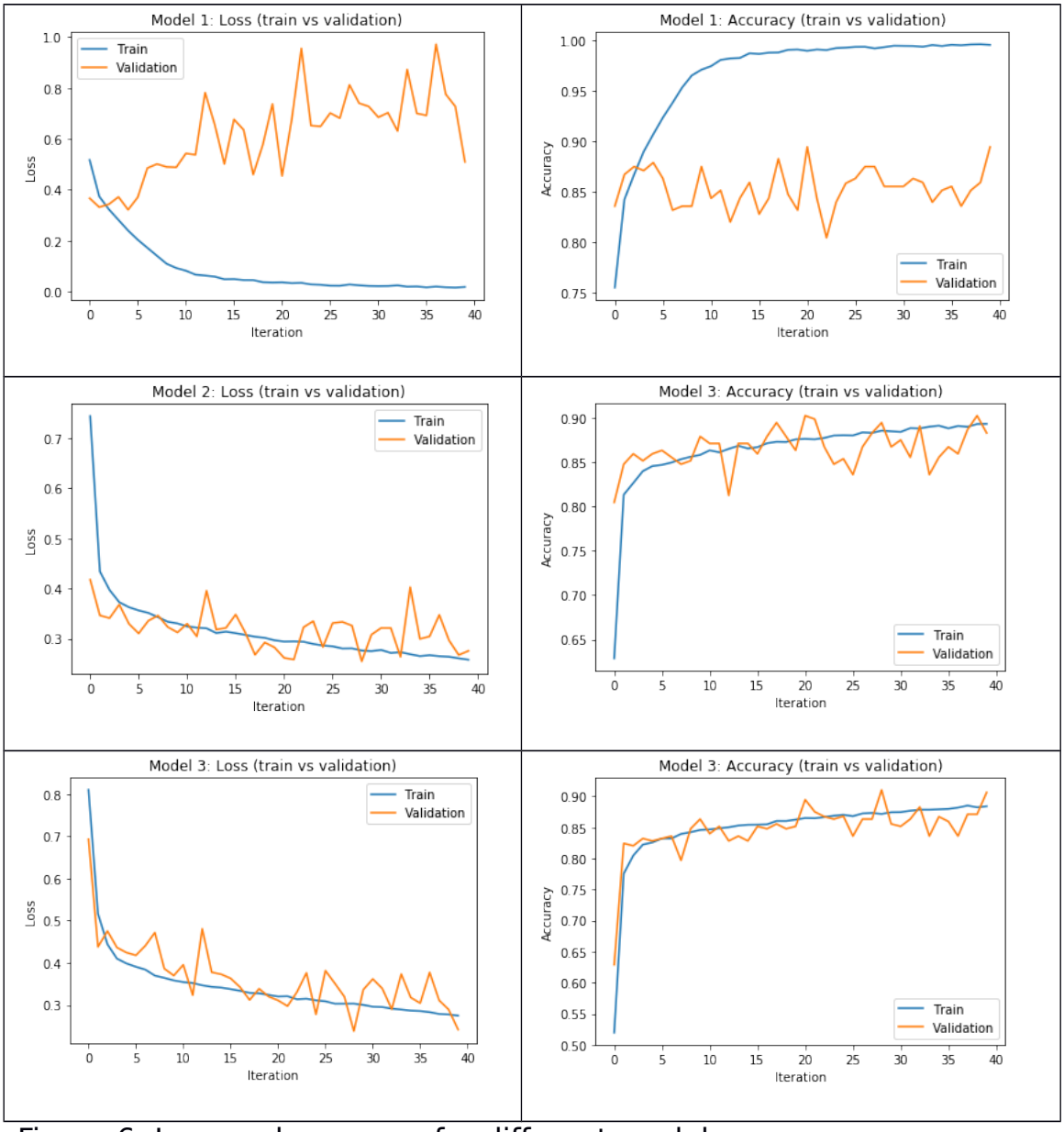

Figure 6: Loss and accuracy for different models

V. Conclusion

Summary

The aim of this project is to determine if the sentiment of a given movie review is positive or negative. This is basically a binary classification problem that can be solved by supervised classification models such as logistic regression, SVM and random forest, as well as deep learning techniques such as recurrent neural network. However, before that, the text needs to be represented by numerical values. In this project, the major solution is to train a word2vec model so each word in the review can be represented by a vector, each review is represented by an ordered sequence of vectors with fixed length and then use as the input to train a RNN with LSTM structure.

The finalized model reached an accuracy of 0.8732 on the test set, which outperformed all the other baseline models and LSTM models, and at the same level of the other literatures.

Reflection

The project is my first exposure to NLP problem. There are many different models and transformations involved, and each process requires different forms of data and has multiple hyper-parameters to tune, which is both exciting and frustrating. For this project, I followed some practical advices, such as separating a big function to small ones so they can be tested one by one, and naming the functions and making documentations properly. I tuned the parameters manually in this project, but in fact some fancy methods such as packaged to do bayes optimization were recommended.

Some processes of this project are computationally demanding. Although the original datasets with text are relatively small, after multiple transformations were conducted, the datasets became bigger and bigger and eventually crashed the system. In order to tackle the issue, data generator was created to train the LSTM model instead of feeding the whole dataset into the model all at once.

Of note, due to the random processes involved in training LSTM, the results of the three LSTM models might not be reproducible. I ran four trials on the validation set for each LSTM model and the results reported are from the last trial. For model 1, the performance is slightly above 0.85 in two trials and approaching 0.855 in the other two trials.

For model2, the performance is around 0.865 for two trials and above 0.869 in the other two trials. For model 3, the performance is around 0.86 for one trial, around 0.865 for two trials and above 0.87 in the last trial. These results made me question if the deeper Model3 is really better than the simpler Model 2. The final tests show that the performance of the finalized Model3 on the test set is indeed better than that of Model2.

I explored suggestions online regarding the reproducibility of LSTM using Keras, such as setting numpy seed at the right place, setting keras flag, and updating theano package, but none of them worked. Due to the limited resources, I haven't try other suggestions, such as using CPU instead of GPU, switching to some slower libraries that can handle the deterministic issue, and a relevant new stateful LSTM model introduced by keras.

Improvement

One way to improve the model is to use a word2vec model pre-trained with a larger dataset instead of training a word2vec using the current smaller dataset. An example of the pre-trained model would be the model from Google Word2Vec trained on 100 bln words from Google News dataset. Usually a model trained on a larger dataset is more robust and have less over-fitting issues. For example, an implementation using Google's pre-trained word2vec model with a very simple LSTM structure with low dropout achieved loss /accuracy = 0.1978 / 0.8195 on the testing dataset.[3]

One issue that was not dealt with in this project is data normalization. Although most literature suggested that data normalization is very necessary to improve the performance of LSTM model, there are not many discussions on its influence on the word embeddings generated by word2vec model. Regarding if the word vectors generated by word2vec model should be normalized before fitting to other models for different purposes, there are different opinions but not may practical work. Some opinions are that it depends on the downstream task. As vector normalization loses the information about vector length, downstream tasks that do not require that information would be benefited from it. [4] Further exploration can be conducted on the influence of different types of normalization on the performance of the task.

3 http://www.volodenkov.com/post/keras-lstm-sentiment-p2/
4 http://stats.stackexchange.com/questions/177905/should-i-normalize-word2vecs-word-vectors-before-using-them

References

https://www.kaggle.com/c/word2vec-nlp-tutorial
https://code.google.com/archive/p/word2vec/
https://keras.io
http://web.stanford.edu/class/cs224n/
http://ai.stanford.edu/~amaas/data/sentiment/
http://deeplearning.net/tutorial/lstm.html
https://deeplearning4j.org/lstm
https://deeplearning4j.org/visualization
http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
https://github.com/fchollet/keras/issues/
https://www.quora.com/Should-I-do-normalization-to-word-embeddings-from-word2vec-if-I-want-to-do-semantic-tasks
http://stats.stackexchange.com/questions/177905/should-i-normalize-word2vecs-word-vectors-before-using-them
https://www.quora.com/How-do-I-tune-the-parameters-for-the-LSTM-RNN-using-Keras-for-time-series-modeling
https://arxiv.org/pdf/1409.2329.pdf
https://arxiv.org/abs/1512.05287
https://www.quora.com/How-does-the-dropout-method-work-in-deep-learning

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). *Learning Word Vectors for Sentiment Analysis*. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2012.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.

Graves, Alex. Supervised sequence labelling with recurrent neural networks. Vol. 385. Springer, 2012.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Richard Socher, Alex Perelygin, Jean YWu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. *Recursive deep models for semantic compositionality over a sentiment treebank*. In Proceedings of the conference on empirical methods in natural language processing (EMNLP), 2013.