

Sentiment Analysis of Movie Reviews using a Deep Learning Convolutional Neural Network

Ahrim Han, Ph.D.

July 23, 2019

Contents

1	Introduction	3
2	Data Preprocessing	5
2.1	Environment and Data	5
2.2	Cleaning Documents	6
2.3	Multiprocessing	8
2.4	Data Encoding	9
3	Building Deep Learning Models	9
3.1	Embedding Layers with Word Embeddings	10
3.1.1	Pre-Trained Word Embedding: Glove	11
3.1.2	Training New Word Embedding	11
3.2	1-Dimensional Convolutional Neural Network (CNN) and Pooling	12
3.3	Activation functions and Dropout	12
4	Training Deep Learning Models	13
4.1	Model Configurations	13
4.2	Results of Models	14
4.2.1	Effect of the Use of Word Embeddings	15
4.2.2	Effect of the Number of Convolutional Network Layers and Filters	16
4.2.3	Effect of the Number of Units in Dense Layer	17
4.2.4	Effect of Cleaning Documents	18
4.2.5	Effect of Dropouts	18
4.3	Preventing Overfitting	18
4.4	Limitation	21
5	Conclusion and Future Work	23

1 Introduction

Problem Statment. Sentiment analysis (or opinion mining) is the task of identifying and classifying the sentiment expressed in a piece of text as being positive or negative. Given a bunch of text, sentiment analysis classifies peoples opinions, appraisals, attitudes, and emotions toward products, issues, and topics. The sentiment analysis has a wide range of applications in industry from forecasting market movements based on sentiment expressed in news and blogs, identifying customer satisfaction and dissatisfaction from their reviews and social media posts. It also forms the basis for other applications like recommender systems.

In the past years of studies, a review text was converted to fixed-length vector using bag-of-words and these vectors were later used to train the classifier such as Naive Bayes or Support Vector Machine. The major problem of the bag-of-words are the 1) lack of consideration of semantic relationship between words 2) data sparsity and high dimensionality. Moreover, as there are tons of reviews and posts on the web, there is a strong need for the more accurate and automated sentiment analysis technique.

In recent years, there has been a remarkable performance improvement to Natural Language Processing (NLP) problems by applying deep learning neural networks. Therefore, in this project, I will build deep learning models using various parameters to classify the positive and negative movie reviews using the high-edge deep learning techniques. I will compare models and observe the parameters affecting the performance in accuracy.

Expected Beneficiaries. Automated and accurate sentiment analysis techniques can be used to detect fake reviews, news, or blogs and are becoming more and more important due to the huge impact on the business markets [1] [2]. We provide the potential beneficiaries of this work.

- Businesses can find consumer opinions and emotions about their products and services.
- E-commerce companies, such as Amazon and Yelp, can identify fake reviews. There are cases

when the results are not consistent from the actual looks and the sentiment analysis (automated analysis). For example, if majority reviews are positive, but the sentiment analysis determines that reviews should not be positive. Then the administrators should inspect the reviews manually if those are fake or not. Fake reviews are not only damaging both competing companies and customers, but they also lead to reduced trust in e-commerce companies and lower sales.

- Potential customers also can know the opinions and emotions of existing users. Customers also can use the system based on the sentiment analysis [2] and check if the reviews are reliable and trustable before they make the decisions on buying products or services.

Approach. We prepare the movie review text data for classification with deep learning methods. We obtain the large data set of the movie reviews [3] [4]. We clean the documents of text reviews by removing punctuations, stopwords, stemming and removing non-frequent words to prevent a model from overfitting. In this pre-processing of documents, we use the more sophisticated methods in the **NLTK python package**.

To build a deep learning Convolutional Neural Network (CNN) model, we basically use the sequential model of Keras.

1. First, the Embedding layer is located. There are two ways of setting the embedding layer: using the pre-trained word embedding or training new embedding from scratch. For a pre-trained word embedding, we use the GloVe (Global Vectors for Word Representation) embeddings [5].
2. Second, a series of convolution 1D Neural Network and pooling layers are added according to typical CNN for text analysis. Then, after flattening layer, fully connected Dense layers are added. Since this is a binary classification problem, we use the Sigmoid function as an activation function for the final Dense layer. To prevent overfitting, we add Dropouts to deactivate neurons randomly, which forces the network to learn a more balanced representation.

3. Finally, we will make the different deep learning models by adjusting the parameters and will find the best accurate model. We later will investigate the features affecting the accuracy.

Results In the paper [3], the performance of machine learning models are in range of 67.42% to 88.89%. The model performed best in the cross validated Support Vector Machine (SVM) when concatenated with bag of words representation. In this project, we generate the preliminary results, and the best accuracy of our deep learning models based on CNN is 90.14%.

2 Data Preprocessing

In this section, we perform cleaning documents. More details of the process for cleaning documents can be found in [this IPython notebook](#).

2.1 Environment and Data

Environment. For this project, we used **my own Linux machine having AMD Ryzen 7 2700X, 16GB Memory, Geforce RTX 2070**.

In addition, **Keras with Tensorflow backend** is used for making a deep learning model. Keras is a high-level API, written in Python and capable of running on top of TensorFlow, Theano, or CNTK deep learning frameworks. Keras provides a simple and modular API to create and train Neural Networks, hiding most of the complicated details under the hood. By default, Keras is configured to use Tensorflow as the backend since it is the most popular choice. Keras is becoming super popular recently because of its simplicity.

Data. There is a large dataset for binary sentiment classification of movie reviews [3] [4]. It contains substantially more data than previous benchmark datasets. This data set contains 50,000 reviews which is evenly split into two groups: 25,000 highly polar movie reviews for training and 25,000 for testing. **The train and test sets contain a disjoint set of movies**, so no significant performance is obtained by

memorizing movie-unique terms and their associated with observed labels. Therefore, we can assume that the validation result with testing data set can be applicable for other movie reviews.

Each group has the same number of positive and negative reviews: a positive review has a score from 7 to 10 while a negative review has a score from 1 to 4. The reviews having score 5 or 6 are excluded to avoid vagueness.

For building deep learning models, all the documents are loaded as in Figure 1. The data sets for training and testing are stored in `data/train` and `data/test`, respectively. For each data set, positive and negative reviews are stored in `pos` and `neg` sub-directories. I have attached the progress bars using the `tqdm library`, which is useful in dealing with large data by allowing us to estimate each time of the stages.

2.2 Cleaning Documents

In most of NLP related works, documents are normally pre-processed to get better performance. We tried to apply several techniques which are well-known as follows:

1. **Removing punctuations.** Normally punctuations do not have any meaning, but they exist for understandability. Therefore, such punctuations should be removed. But, we did not remove the apostrophe mark (') since such removing caused the incorrect stemming.
2. **Removing stopwords.** We filtered out the stopwords. The stop words are those words that do not contribute to the deeper meaning of the phrase. They are the most common words such as: “the”, “a”, and “is”. NLTK python package provides a list of commonly agreed upon stop words for a variety of languages.
3. **Stemming.** The `PorterStemmer` is provided in NLTK python package. We made the words into lowercases and used the stemming method in order to both reduce the vocabulary and to focus on the sense or sentiment of a document rather than deeper meaning.

```
In [3]: # load all docs in a directory
def load_docs(directory):
    documents = list()
    # walk through all files in the folder
    for filename in tqdm(os.listdir(directory)):
        # create the full path of the file to open
        path = directory + '/' + filename
        with open(path, 'r') as f:
            # load the doc
            doc = f.read()
            # add to list
            documents.append(doc)
    return documents
# load all training reviews
print("Loading training-positive-docs")
global_train_positive_docs = load_docs('data/train/pos')
print("Loading training-negative-docs")
global_train_negative_docs = load_docs('data/train/neg')
# load all test reviews
print("Loading test-positive-docs")
global_test_positive_docs = load_docs('data/test/pos')
print("Loading test-negative-docs")
global_test_negative_docs = load_docs('data/test/neg')
```

Loading training-positive-docs

 100% 12500/12500 [00:01<00:00, 6822.51it/s]

Loading training-negative-docs

 100% 12500/12500 [00:03<00:00, 3314.74it/s]

Loading test-positive-docs

 100% 12500/12500 [00:03<00:00, 3479.76it/s]

Loading test-negative-docs

 100% 12500/12500 [00:00<00:00, 19914.30it/s]

Figure 1: Data Loading.

4. **Defining a vocabulary dictionary of preferred words and removing non-frequent words.** It is important to define a vocabulary of known words when using a bag-of-words or embedding model. The more words, the larger the representation of documents, therefore it is important to constrain the words to only those believed to be predictive.

In this project, we set up the vocabulary dictionary and remove the non-frequent words to

prevent a model from overfitting. Building code of the vocabulary dictionary is implemented in [vocab.ipynb notebook](#). The process can be summarized as follows.

- First, based on only reviews in the training dataset, we count the words' occurrences using `Counter` function. This is saved in [vocab_counter.txt](#) file and contains 52,826 words. Figure 2 shows the most frequent 30 words extracted from training data set.

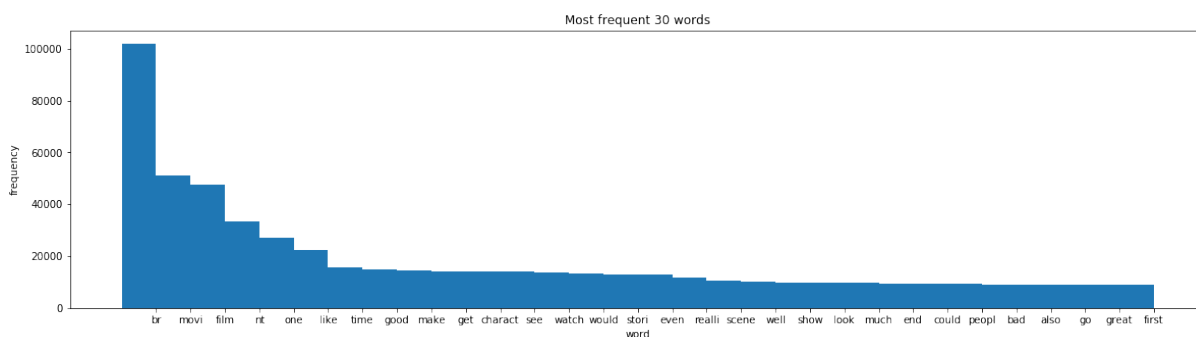


Figure 2: Most Frequent 30 Words in Training Data Set.

- Then, we filter out the words that have low occurrences, such as only being used once or none and set up the vocabulary dictionary. Thus, the vocabularies have the two or more occurrences. There are 30,819 number of words in the vocabulary dictionary and those are saved in the [vocab.txt](#) file.
- Finally, using the vocabulary dictionary, we remove the non-frequent words for cleaning documents. This is done by filtering out words that are not in the vocabulary dictionary and removing all words that have a length ≤ 1 character.

2.3 Multiprocessing

Pre-processing mentioned above requires heavy computation. To improve the speed, we parallelized the pre-processing using the **Pool module** in **multiprocessing package**. Since we use a CPU having 8 cores, the size of Pool is set as 8. By using this technique, **we could achieve 6 to 7 times speed up**. Using the

single thread, it takes 10 to 12 minutes for cleaning up 12,500 documents, whereas, using the multiple threads, it takes only 1 minute and 20 to 40 seconds.

2.4 Data Encoding

To use documents as an input of a model, each document is encoded as a sequence object of Keras.

First, we encode the documents as sequence objects. We pad all reviews to the length of the longest review in the training dataset. We can find the longest review using the `max()` function on the training dataset and take its length. We can then call the Keras function `pad_sequences()` to pad the sequences to the maximum length by adding 0 values on the end.

We create a list of labels: '0' for negative reviews and '1' for positive reviews. We do not need the one-hot-encoding process (a function called `to_categorical()` in Keras) because there is only two classes of positive and negative.

3 Building Deep Learning Models

To build a deep learning model, we basically use the sequential model of Keras. The model is built in the following steps as in Figure 3.

1. First, the Embedding layer is located. There are two options of setting embedding layers: using the pre-trained word embedding or training new embedding from scratch.
2. Second, a series of **convolution 1D** and **pooling layers** are added according to typical CNN for text analysis. In order to check the effects of the number of convolution layers, we made the function below configurable to set the number of additional convolution layers.
3. Then, after flattening layer, fully connected Dense layers are added. Since this is a binary classification problem to output a value between 0 and 1 for the negative and positive sentiment in the review, we use the Sigmoid function as an activation function for the final output Dense layer. If

you try to predict a score of a review, it would be better to use 'softmax' function as the activation function.

```
def build_model(word_index, max_word_length, number_of_additional_conv_layers,
                use_pre_trained_embedding, trainable_for_embedding, number_of_filters, use_dropout, num_units):
    # define model
    model = Sequential()
    embedding_func = new_embedding
    if use_pre_trained_embedding:
        embedding_func = load_pre_trained_embedding
    model.add(embedding_func(word_index, max_word_length, trainable_for_embedding))
    if use_dropout:
        model.add(Dropout(0.5))

    for i in range(number_of_additional_conv_layers):
        model.add(Conv1D(filters=number_of_filters, kernel_size=5, activation='relu'))
        model.add(MaxPooling1D(pool_size=4))
        if use_dropout:
            model.add(Dropout(0.5))

    model.add(Conv1D(filters=number_of_filters, kernel_size=5, activation='relu'))
    model.add(MaxPooling1D(pool_size=10))
    if use_dropout:
        model.add(Dropout(0.5))

    model.add(Flatten())
    #model.add(Dense(64, activation='relu'))
    model.add(Dense(units=num_units, activation='relu')) #num_units= [128,32]
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    # compile network
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 3: Building a Deep Learning Model.

3.1 Embedding Layers with Word Embeddings

A word embedding is a way of representing text where each word in the vocabulary is represented by a real valued vector in a high-dimensional space. The vectors are learned in such a way that words that have similar meanings will have similar representation in the vector space (close in the vector space). This is a more expressive representation for text than more classical methods like bag-of-words, where relationships between words or tokens are ignored, or forced in bigram and trigram approaches.

We have two options for using word embeddings on natural language processing projects: **learning an embedding or reusing an embedding**. If choosing to learn own embedding, this requires a large amount of text data to ensure that useful embeddings are learned. Therefore, **it is common to use pre-trained word embeddings, such as Word2Vec and Glove** available for free download. These can be

used instead of training from scratch.

Learning a word embedding while training a neural network is a common technique to deal with texts in Deep Learning. To investigate the effectiveness of the existence of pre-trained word embedding, each of the 1) pre-trained word embedding and the 2) new (not-trained) embedding, and the 3) updated pre-trained embedding (i.e., pre-trained embedding is used to seed the model, but the embedding is updated jointly during the training of the model) are used for building models and the accuracy of those models are compared.

Keras offers an Embedding layer that can be used for neural networks on text data. It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras. This is well explained in [this blog](#).

3.1.1 Pre-Trained Word Embedding: Glove

In this project, we use the [GloVe](#). GloVe stands for “Global Vectors for Word Representation” and is an unsupervised learning algorithm for obtaining vector representations for words. It is a somewhat popular embedding technique based on factorizing a matrix of word co-occurrence statistics. Specifically, we use the 200-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia. You can download these data [here](#). In addition, to check whether the pre-trained word embedding needs to be trained or not, we made the function below configurable for the *trainable* parameter of Embedding object.

3.1.2 Training New Word Embedding

We can perform by not using pre-trained word embeddings, but instead initializing our Embedding layer from scratch and learning its weights during training. **New word embedding is created with no pre-trained weights**, so the *trainable* parameter should always be “True”. Figure 4 shows the python code to return the Embedding layer that is used to load a pre-trained word embedding or to learn a new word

embedding.

```
1 EMBEDDING_DIM = 200
2 def load_pre_trained_embedding(word_index, max_word_length, trainable_for_embedding):
3     embeddings_index = {}
4     with open('./glove/glove.6B.200d.txt') as f:
5         for line in f:
6             values = line.split()
7             word = values[0]
8             coefs = np.asarray(values[1:], dtype='float32')
9             embeddings_index[word] = coefs
10
11     embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
12     for word, i in word_index.items():
13         embedding_vector = embeddings_index.get(word)
14         if embedding_vector is not None:
15             # words not found in embedding index will be all-zeros.
16             embedding_matrix[i] = embedding_vector
17     return Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=max_word_length, weights=[embedding_matrix],
1
1 def new_embedding(word_index, max_word_length, trainable_for_embedding):
2     # This is new embedding layer, so trainable must be True regardless the value of trainable_for_embedding
3     return Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=max_word_length, trainable=True)
```

Figure 4: Embedding Layers with Word Embeddings.

3.2 1-Dimensional Convolutional Neural Network (CNN) and Pooling

We use a CNN as they have proven to be successful at document classification problems. A conservative CNN configuration is used with 32 filters (parallel fields for processing words) and a kernel size of 8. We used the 96 filters and 5 kernels by default with a rectified linear ('ReLU') activation function. We will adjust the number of filters to investigate if they affect the accuracy of models. This extracted set of features are then passed to MaxPooling layer, which filters the dominating terms from the extracted features.

3.3 Activation functions and Dropout

Activation Function. The activation function is used as a decision making body at the output of a neuron. The neuron learns Linear or Non-linear decision boundaries based on the activation function. It also has a normalizing effect on the neuron output which prevents the output of neurons after several layers to become very large, due to the cascading effect.

There are three most widely used activation functions. We used ReLU and Sigmoid.

- **Sigmoid:** It maps the input (x axis) to values between 0 and 1 (which may later results in the **vanishing gradient problem**).
- **Tanh:** It is similar to the Sigmoid function but maps the input to values between -1 and 1.
- **Rectified Linear Unit (ReLU):** - It allows only positive values to pass through it. The negative values are mapped to zero.

Dropout. During training, when dropout is applied to a layer, some percentage of its neurons (a hyperparameter, with common values being between 20% and 50%) are randomly deactivated or “dropped out” along with their connections. Which neurons are dropped out are constantly shuffled randomly during training. This forces the network to learn a more balanced representation, and **dropouts are known to help prevent overfitting.**

4 Training Deep Learning Models

4.1 Model Configurations

All the functions defined above are integrated into the `build_and_train_model` function. Each model is trained by different combinations of six parameters, and we compare the accuracy of those models.

The meanings of parameters can be explained as follows:

1. **use_cleaned_docs:** Cleaning review documents or not.
2. **number_of_additional_conv_layers:** Number of additional convolution layers (0,2). Basically, one convolution layer is used (`'number_of_additional_conv_layers'=0`). If you want to add more convolution layers, we can adjust this parameter to a higher number. For experiment, we adjust the `'number_of_additional_conv_layers'=2`, so the total convolution layers becomes 3.

3. **use_pre-trained_embedding**: Using pre-trained embedding or not. If True, the GloVe embedding will be used as mentioned above.
4. **trainable_for_embedding**: Training the embedding layer with training data set or freezing. Note, when using the new embedding layer, then 'trainable_for_embedding' should be True.
5. **number_of_filters**: Number of filters in the convolution layers (96, 48, and 24).
6. **use_dropout**: Using Dropout or not. In the experiment, 50% percentage of its neurons are randomly deactivated.
7. **num_units**: Number of units in Dense layer. This reduces the capacity of network (128, 64, 32, 8).

When setting up model configurations by combination of features, there are no cases of no embedding layers ('use_pre-trained_embedding'= False and 'trainable_for_embedding' = False), so these should be eliminated.

We use the callbacks functions of EarlyStopping and ModelCheckpoint. One way to avoid overfitting is to terminate the process early. We used the **EarlyStopping** function and set the arguments 'monitor'= val_acc (test accuracy) and 'patience'=2. The 'patience' indicates the number of epochs with no improvement after which training will be stopped. The **ModelCheckpoint** callback saves the model after every epoch.

4.2 Results of Models

Table 1 shows the accuracy of classification results for the deep learning CNN models and configurations. In those models, the Model 11 has the best accuracy of 90.14% (highlighted in red), and the Model 5 has the second best accuracy of 89.73% (highlighted in green).

The abbreviated terms in Table 1 are as follows:

- use_cleaned_docs: clean_docs

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 0	FALSE	0	TRUE	FALSE	96	FALSE	128	86.62
Model 1	FALSE	0	FALSE	TRUE	96	FALSE	128	87.50
Model 2	FALSE	0	TRUE	TRUE	96	FALSE	128	88.82
Model 3	FALSE	2	TRUE	FALSE	96	FALSE	128	88.63
Model 4	FALSE	2	FALSE	TRUE	96	FALSE	128	89.11
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 6	FALSE	0	TRUE	TRUE	24	FALSE	32	88.39
Model 7	FALSE	0	TRUE	TRUE	24	TRUE	32	89.49
Model 8	FALSE	2	TRUE	TRUE	48	FALSE	128	89.62
Model 9	FALSE	2	TRUE	TRUE	24	FALSE	128	89.51
Model 10	FALSE	2	TRUE	TRUE	96	FALSE	32	89.42
Model 11	FALSE	2	TRUE	TRUE	96	FALSE	64	90.14
Model 12	FALSE	2	TRUE	TRUE	96	FALSE	8	89.20
Model 13	FALSE	2	TRUE	TRUE	24	FALSE	32	89.12
Model 14	FALSE	2	TRUE	TRUE	96	TRUE	128	89.18
Model 15	FALSE	2	TRUE	TRUE	48	TRUE	128	88.99
Model 16	TRUE	2	TRUE	TRUE	96	TRUE	128	86.98
Model 17	TRUE	2	TRUE	TRUE	96	FALSE	128	87.36

Table 1: Results of the accuracy of classification for the deep learning models and configurations.

- number_of_additional_conv_layers: # conv.
- use_pre_trained_embedding: pre-trained
- trainable_for_embedding: trainable
- number_of_filters: # filters
- use_dropout: dropout
- num_units: # units

Figure 5 represents the model configuration and training progress of Model 11.

4.2.1 Effect of the Use of Word Embeddings

For the word embedding, the **use of sole pre-trained word embedding produces the least performance**. This can be interpreted that Embedding model cannot be generalized and should be trained into their contexts. Rather, the **use of the new training word embedding was the better choice**.


As highlighted in Table 2, most of all, **models that use the pre-trained word embedding and updating weights during training provide the best performance..** Using the pre-trained word embedding

```

Fitted tokenizer on 25000 documents
Top 5 most common words are: [('the', 336148), ('and', 164097), ('a', 163040), ('of', 145847), ('to', 135708)]

Layer (type)                 Output Shape                 Param #
=====
embedding_1 (Embedding)      (None, 2470, 200)           17716600
conv1d_1 (Conv1D)            (None, 2466, 96)            96096
max_pooling1d_1 (MaxPooling1 (None, 616, 96)            0
conv1d_2 (Conv1D)            (None, 612, 96)             46176
max_pooling1d_2 (MaxPooling1 (None, 153, 96)            0
conv1d_3 (Conv1D)            (None, 149, 96)             46176
max_pooling1d_3 (MaxPooling1 (None, 14, 96)            0
flatten_1 (Flatten)          (None, 1344)                0
dense_1 (Dense)              (None, 64)                  86080
dense_2 (Dense)              (None, 1)                   65
=====
Total params: 17,991,193
Trainable params: 17,991,193
Non-trainable params: 0

Train on 25000 samples, validate on 25000 samples

Training  25% 4/16 [01:15<03:46, 18.83s/it]

Epoch 1/16
- 19s - loss: 0.4433 - acc: 0.7740 - val_loss: 0.2678 - val_acc: 0.8904
Epoch 2/16
- 19s - loss: 0.2023 - acc: 0.9223 - val_loss: 0.2468 - val_acc: 0.9014
Epoch 3/16
- 19s - loss: 0.1006 - acc: 0.9651 - val_loss: 0.3010 - val_acc: 0.8925
Epoch 4/16
- 19s - loss: 0.0432 - acc: 0.9864 - val_loss: 0.4467 - val_acc: 0.8818

```

Figure 5: Model 11: model configuration and training progress.

produces the better performance takes advantages of leveraging massive datasets with a vast corpus of language capturing word meanings in a statistically robust manner. It is difficult to train from scratch, therefore, we need to use the pre-defined word embedding and customize it into our dataset at the same time.

4.2.2 Effect of the Number of Convolutional Network Layers and Filters

Number of CNN. As highlighted in Table 3, the model with the additional 2 layers works better than the model with only one layer, but the difference is marginal.

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 0	FALSE	0	TRUE	FALSE	96	FALSE	128	86.62
Model 1	FALSE	0	FALSE	TRUE	96	FALSE	128	87.50
Model 2	FALSE	0	TRUE	TRUE	96	FALSE	128	88.82
Model 3	FALSE	2	TRUE	FALSE	96	FALSE	128	88.63
Model 4	FALSE	2	FALSE	TRUE	96	FALSE	128	89.11
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73

Table 2: Accuracy results comparing of the use of word embeddings.

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 6	FALSE	0	TRUE	TRUE	24	FALSE	32	88.39
Model 13	FALSE	2	TRUE	TRUE	24	FALSE	32	89.12
Model 2	FALSE	0	TRUE	TRUE	96	FALSE	128	88.82
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 0	FALSE	0	TRUE	FALSE	96	FALSE	128	86.62
Model 3	FALSE	2	TRUE	FALSE	96	FALSE	128	88.63
Model 1	FALSE	0	FALSE	TRUE	96	FALSE	128	87.50
Model 4	FALSE	2	FALSE	TRUE	96	FALSE	128	89.11

Table 3: Number of CNN layers (0,2): Results of the accuracy.

Number of Filters in CNN. Model 5, 8 and 9 represent the results of accuracy for the different number of filters 98, 48, and 24. As highlighted in Table 4, the largest number of filters of 96 in convolutional neural network gives best performance (accuracy of 89.73%). But it does not guarantee that the larger number of filters always produces the better performance.

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 8	FALSE	2	TRUE	TRUE	48	FALSE	128	89.62
Model 9	FALSE	2	TRUE	TRUE	24	FALSE	128	89.51

Table 4: Number of filters (98, 48, 24) in CNN: Results of the accuracy.

4.2.3 Effect of the Number of Units in Dense Layer

As highlighted in Table 5, the model with the 64 number of units in the Dense layer (Model 11) has the highest accuracy. Exception of that model, the larger number of units in the Dense layer has, the

better performance. In short, models have the higher accuracy in the order of the number of units of 128 (Model 5), 32 (Model 10), and 8 (Model 12).

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 10	FALSE	2	TRUE	TRUE	96	FALSE	32	89.42
Model 11	FALSE	2	TRUE	TRUE	96	FALSE	64	90.14
Model 12	FALSE	2	TRUE	TRUE	96	FALSE	8	89.20

Table 5: Number of units (128, 64, 32, 8) in Dense Layer: Results of the accuracy.

4.2.4 Effect of Cleaning Documents

As highlighted in Table 6, it seems that the cleaning degrades or does not improve the accuracy in our project.

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 17	TRUE	2	TRUE	TRUE	96	FALSE	128	87.36
Model 14	FALSE	2	TRUE	TRUE	96	TRUE	128	89.18
Model 16	TRUE	2	TRUE	TRUE	96	TRUE	128	86.98

Table 6: Cleaning documents: Results of the accuracy.

4.2.5 Effect of Dropouts

As highlighted in Table 7, Dropouts in most of models do not improve accuracy improvements. Dropouts lead improvements (i.e., Model 6 when dropout=false to Model 7 when dropout=true) only when the network size is small (i.e., number of filters are 24 and number of units are 32) (see in Figure 8d). We interpret that if the network size is large, too much data is missing so the dropouts may not be effective.

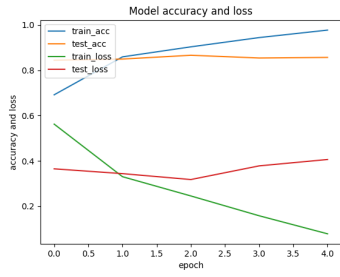
Dropouts are known to help avoid overfitting. This will be discussed in Section 4.3.

4.3 Preventing Overfitting

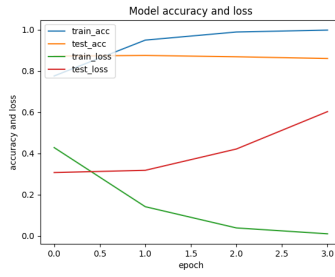
Figure 6 and Figure 7 show the plots of accuracy and loss (y-axis) on every epoch (x-axis) for train and test data for deep learning models trained above.

Model #	clean_docs	# conv.	pre-trained	trainable	# filters	dropout	# units	Accuracy
Model 8	FALSE	2	TRUE	TRUE	48	FALSE	128	89.62
Model 15	FALSE	2	TRUE	TRUE	48	TRUE	128	88.99
Model 5	FALSE	2	TRUE	TRUE	96	FALSE	128	89.73
Model 14	FALSE	2	TRUE	TRUE	96	TRUE	128	89.18
Model 17	TRUE	2	TRUE	TRUE	96	FALSE	128	87.36
Model 16	TRUE	2	TRUE	TRUE	96	TRUE	128	86.98
Model 6	FALSE	0	TRUE	TRUE	24	FALSE	32	88.39
Model 7	FALSE	0	TRUE	TRUE	24	TRUE	32	89.49

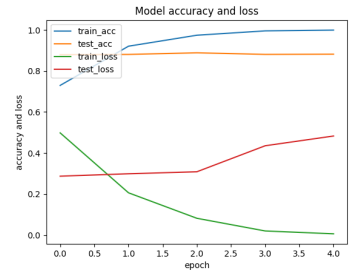
Table 7: Dropouts: Results of the accuracy.



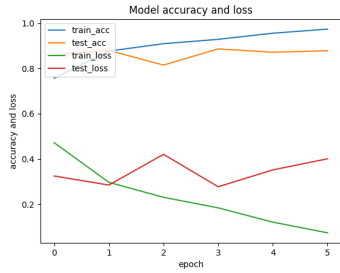
(a) Model 0.



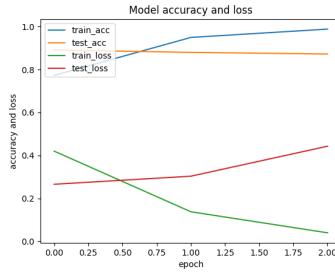
(b) Model 1.



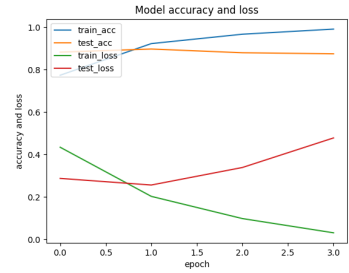
(c) Model 2.



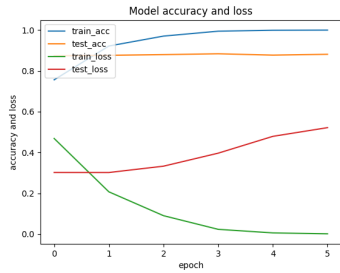
(d) Model 3.



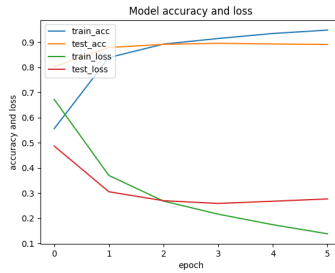
(e) Model 4.



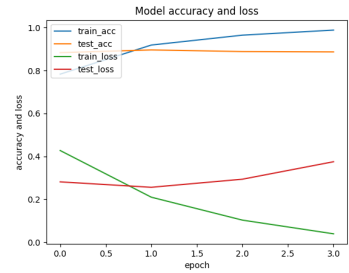
(f) Model 5 (# filters = 96).



(g) Model 6.

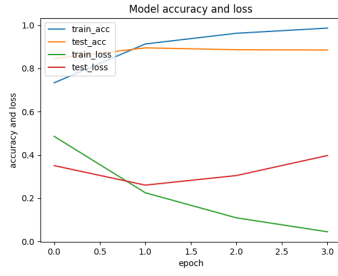


(h) Model 7.

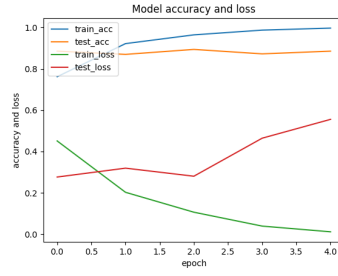


(i) Model 8 (# filters = 48).

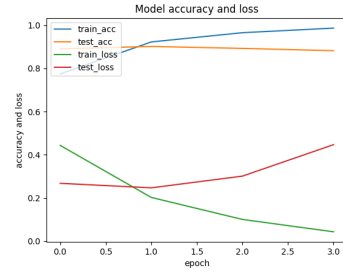
Figure 6: Results of models on train and test dataset (X-axis: Epoch, Y-axis: accuracy and loss).



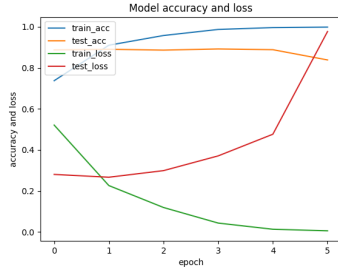
(a) Model 9 (# filters = 24).



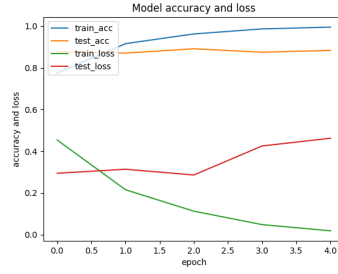
(b) Model 10 (# units = 32).



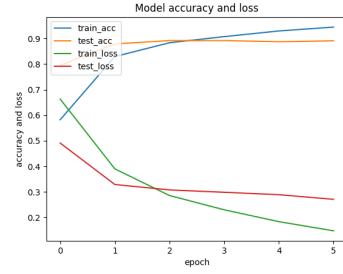
(c) Model 11 (# units = 64).



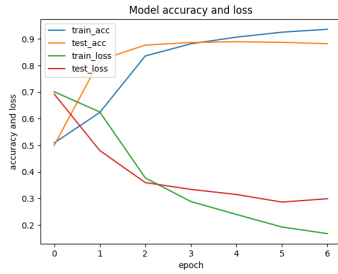
(d) Model 12 (# units = 8).



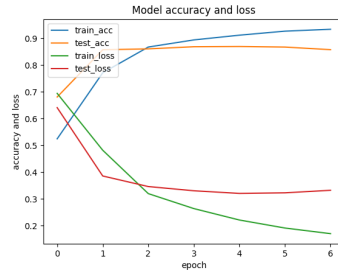
(e) Model 13.



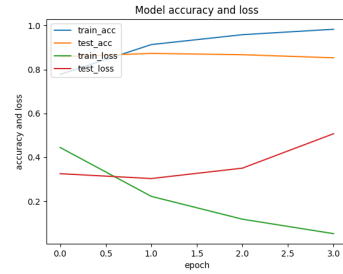
(f) Model 14.



(g) Model 15.



(h) Model 16.



(i) Model 17.

Figure 7: (Continued to Figure 6.) Results of models on train and test dataset (X-axis: Epoch, Y-axis: accuracy and loss).

Some of models seem to have the overfitting. For example, in Model 12 in Figure 7d, we can see that the test loss (red line) starts to increase as from epoch 4. The training loss (green line) continues to lower, which is normal as the model is trained to fit the train data as good as possible. Just as with the test loss, the test accuracy peaks at the 4th epoch. After that, it goes down slightly. So to conclude, we can say that the model starts overfitting as from epoch 4.

We applied **reducing the capacity of the network**—i.e., lowering the number of convolutional layers, filters in convolutional layers, and hidden elements in the remaining layers (the units in

Dense layers)—but it did not really help to avoid overfitting.

It turned out that **adding Dropout layers help to prevent overfitting as in Figure 8**. We can see that it takes more epochs to start overfitting than the no dropouts applied models. For instance, in Figure 8a, the model with no dropouts starts to increase the test loss from the epoch 1, while the dropout applied model starts at 5 epoch. Moreover, the test loss increases much slower in the dropout applied model compared to the model with no dropouts (Figure 8b). In short, the loss increases much slower afterwards. This is because adding Dropout layers randomly remove certain features by setting them to zero.

As mentioned in Section 4.2.5, Dropouts lead the accuracy improvements only in the case of Figure 8d (i.e., Model 6 when dropout=false to Model 7 when dropout=true) where the network size is small (i.e., number of filters are 24 and number of units are 32). It is apparent that the use of Dropouts continuously lowers the test loss while contributes to improve the test accuracy so that avoiding overfitting to find better solutions.

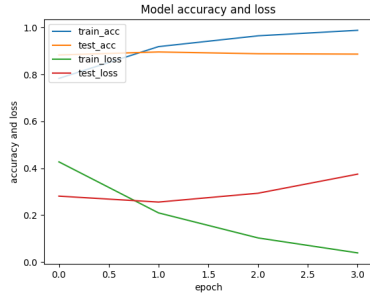
However, we decide that it is better to choose the model having the higher accuracy. **We do not sacrifice the accuracy but, to prevent overfitting, we used the EarlyStopping function to stop the training if the test accuracy dropped successively in two iterations (epochs).**

4.4 Limitation

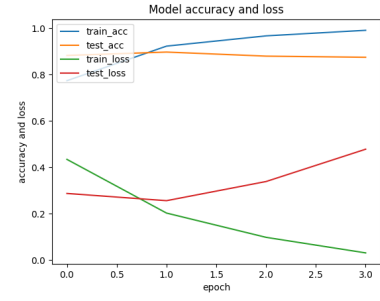
To improve the performance of accuracy, we can use the other techniques other than reducing network size and adding dropouts to prevent overfitting in neural networks.

1. Get more training data.
2. Add weight regularization.
3. Data-augmentation.
4. Batch normalization.

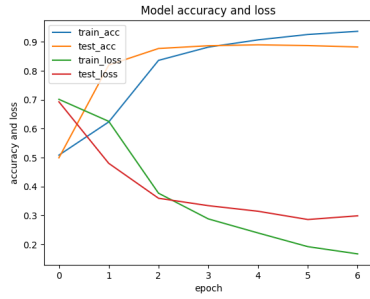
In the previous study [3], the performance of machine learning models are in range of 67.42% to



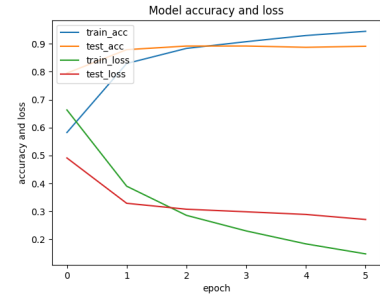
(a) Model 8 (Dropout=False) to Model 15 (Dropout = True).



(b) Model 5 (Dropout = False) to Model 14 (Dropout = True).



(c) Model 17 (Dropout = False) to Model 16 (Dropout = True).



(d) Model 6 (Dropout = False) to Model 7 (Dropout = True).

Figure 8: Dropout Effects on Overfitting.

88.89%. This model performed best in the cross validated Support Vector Machine (SVM) when concatenated with bag of words representation.

In this project, the best model reaches 90.14% classification accuracy on the test (validation) set at the first epoch. We could probably get to an even higher accuracy by 1) training longer with more overfitting preventing methods (such as dropout or regularization) or by 2) fine-tuning the Embedding layer. However, this is the preliminary project for the classification of reviews using deep learning techniques. We think this accuracy is enough to observe the advantages of using deep learning models on NLP problems. We will work further to improve the accuracy more of the deep learning models.

5 Conclusion and Future Work

In this project, we build deep learning models to classify the positive and negative movie reviews using the high-edge deep learning techniques.

The process of building deep learning models is as follows. We obtain the large data set of the movie reviews. We clean the documents of text reviews by removing punctuations, stopwords, stemming and removing non-frequent words to prevent a model from overfitting. To build a deep learning Convolutional Neural Network (CNN) model, we basically use the sequential model of Keras. We used the three ways of setting the embedding layer: 1) using the pre-trained word embedding (GloVe embeddings), 2) training new embedding from scratch, and 3) both. Then, a series of convolution 1D Neural Network and pooling layers are added according to typical CNN for text analysis. After flattening layer, fully connected Dense layers are added. To prevent overfitting, we add Dropouts to deactivate neurons randomly, which forces the network to learn a more balanced representation. We made different deep learning models by adjusting the parameters.

The best accuracy of our deep learning models based on CNN is 90.14%. The model produces the best accuracy when documents are not cleaned, the number of CNN is 3 (number of additional convolution layer is 2) with 96 filters, pre-trained word embedding and training the Embedding layer are

used, and 64 Dense layer is added without Dropouts. Dropouts help to train data in better ways so that the loss of test starts to increase later or slower after using the Dropouts. But Dropouts do not guarantee the accuracy improvements. Therefore, **the model with the highest accuracy is considered the best**. In our project, to prevent overfitting, the training is stopped (EarlyStopping) when there is no more accuracy improvements, thus it is enough to use the models generated so far.

We summarize the findings as follows. For the Word Embeddings, models that use the pre-trained word embedding and updating weights during training provide the best performance. For the CNN and filters, the model with 3 layers works better than the model with only one layer, but the difference is marginal. Likewise, the largest number of filters in CNN gives best performance, but it does not guarantee that the larger number of filters always produces the better performance. Regarding to the units in Dense layers, the model with the 64 number of units in the Dense layer has the highest accuracy. Exception of that model, the larger number of units in the Dense layer has, the better performance. For the use of the cleaned documents, it seems that the cleaning degrades or does not improve the accuracy in our project.

For the future work, I will investigate another interesting projects using deep learning techniques. I have special interests in a photo caption generator. In this project, using the photo and text data for training, I aim to design and train a deep learning caption generation model, evaluate a train caption generation model, and use it to caption entirely new photographs.

For the final comment, I give my special thanks to Tony Paek who has been my mentor for completing this project.

References and Notes

- [1] “Deep Learning Sentiment Analysis of Amazon.com Reviews and Ratings.”, Shrestha, Nishit, and Fatma Nasoz, *arXiv preprint arXiv:1904.04096* (2019).
- [2] [Fake spot finding website](https://www.fakespot.com/), <https://www.fakespot.com/>.
- [3] “Learning Word Vectors for Sentiment Analysis.”, Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts, *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*.
- [4] [Stanford Large Movie Review Dataset](https://ai.stanford.edu/~amaas/data/sentiment/), <https://ai.stanford.edu/~amaas/data/sentiment/>.
- [5] [GloVe: Global Vectors for Word Representation](https://nlp.stanford.edu/projects/glove/), <https://nlp.stanford.edu/projects/glove/>.