Text Classification and Summarization with Deep Learning using RNN-LSTM and CNN

Arshnoor Jandu, Inela Beqaj and Xin He
Data Science and Analytics, Ryerson University
{ajandu, inela.beqaj, xin.he}@ryerson.ca

Abstract

We present deep learning approach towards text classification and summarization using RNN-LSTM and CNN. We compared RNN-LSTM and CNN for text classification and determined that CNN performed significantly better than RNN-LSTM for our dataset of BBC news. For text summarization, we implemented RNN-LSTM and compared it's performance with human generated summary of news article using ROUGE-2. Where we calculated common percentage of bigrams between human vs. model generated summaries and we found on average 38% overlap between the two.

1 Introduction

Distributed representation of words have been widely used in many natural language processing (NLP) tasks. Due to its success, there is a substantial interest in learning distributed representation of the continuous words, such as phrases, sentences, paragraphs and documents [1]. Several methods have been developed to learn distributed representation of words for different tasks, such as Naive Bayes for classification, Text Rank for summarization, Hidden Markov models for part of speech tagging, so forth and so on. However, artificial neural networks are gaining popularity these days and are increasingly used for NLP tasks. These networks are computational nonlinear models that consists of artificial neurons or processing units that are generally organised in three main layers: input, hidden and output. With the use of multiple hidden layers, modern neural networks are often called deep learning. This project focuses on implementing neural network based deep learning models for text classification and summarization. The motivation of implementing only these two tasks is to learn "behind the scenes" computation of news websites in terms of how they know which articles are related to each other and second, how article summaries are generated. Deep learning approach is used to implement this idea with focus on recurrent neural networks (RNN) and convolutional neural networks (CNN).

In this project, we implemented both RNN and CNN for text classification. We compared the performance of these models in terms of accuracy and time to train. However, we only used RNN for text summarization due the complexity of implementing encoder-decoder. We measured the accuracy of text summarization with ROUGE-2, which finds common percentage of bigrams between human vs. model generated text summaries.

2 Related Work

RNN approach is quite popular with text classification task due to it's speciality of remembering the past. (Duyu Tang *et al.,* 2015) states that standard recurrent neural networks can map vector of sentences of variable length to fixed length vector by recursively transforming current sentence vector with the output vector of the previous step. Thus, making RNN favourable choice for time series task such as text classification where information from pervious steps is carried over. This paper also shows that gated recurrent neural network models dramatically outperforms standard recurrent neural network models for classification task. (Pengfei Liu *et al*., 2016*)* discusses the concern of training deep neural networks. Usually these networks need large corpus due to the large number of parameters and it is hard to train a model that generalizes well with limited data. These models are trained with supervised pre-training phase with gradient based optimization. Due to data limitations, this paper suggests that multi-task learning utilizes the correlation between related tasks to improve classification by learning tasks in parallel. Besides, this paper explains the mechanism of RNN with classification in great detail. Article [3] and [4] discusses the implementation of RNN-LSTM on real world applications using TensorFlow and Keras library.

(Yoon Kim, 2014) talks about using CNN for text classification where CNN utilizes layers with convolving filters that are applied to local features. Several CNN models are discussed such as CNN-rand, CNN-static, CNN-non-static and CNN-multichannel. In CNN-rand, all words are randomly initialized and then modified during training. CNN-static uses pre-trained word2Vec embeddings and only the other parameters of the model are randomly initialized. CNN-non static uses same idea are the previous model but fine tunes pre-trained vectors. In case of CNN-multichannel a model with two sets of word vectors is used, where each word is treated as a channel and filter is applied to both channels, but gradients are back propagated through only one channel. Hence, the model fine-tunes one set of vectors while keeping the other static. This paper purposes that simple CNN with little hyper parameter tuning and static vectors achieves excellent results on multiple benchmarks out of other CNN models such as CNN-rand, CNN-non-static and CNN-multichannel. However, models that are randomly initialized can gain surprised magnitude of performance by fine-tuning parameters. Article, [6] and [7] shows how to effectively build CNN model in TensorFlow and also indicates on selecting the hyper parameters for the model.

For text summarization, we consulted three papers:

First paper, (Alexander M. Rush *et al.*, 2015) discusses about using a neural attention model for sentence summarization with key points:

1. Extractive method summarizes that uses only words from the original text. Whereas, abstractive summarization may generate new words and phrases not present in the source text. This method of summarization is similar to the human-written summaries.

2. Type of Encoders:

1. Bag of words Encoder: This is the basic model that simply uses the bag-of-words of the input sentence embedded down to size

H, while ignoring properties of the original order or relationships between neighboring words. This model can capture the relative importance of words to distinguish content words from stop words or embellishments. Potentially, the model can also learn to combine words; although it is inherently limited in representing contiguous phrases.

2. Convolutional Encoder: This architecture improves on the bag-of-words model by allowing local interactions between words while also not requiring the context while encoding the input.

3. Attention-Based Encoder: While the convolutional encoder has richer capacity than bag-of-words, it still is required to produce a single representation for the entire input sentence. A similar issue in machine translation inspired (Bahdanau *et al*., 2014) to instead utilize an attention-based contextual encoder that constructs a representation based on the generation context. Here we note that if we exploit this context, we can actually use a rather simple model similar to bag-of-words. Informally, we can think of this model as simply replacing the uniform distribution in bag-of-words with a learned soft alignment, P, between the input and the summary.

Second paper, (Ramesh Nallapati *et al*., 2016) focuses on abstractive text summarization using sequence-to-sequence RNNs with key points:

1. Encoder-Decoder RNN with Attention and Large Vocabulary Trick: This model is based on the neural machine translation model used in (Bahdanau *et al.*, 2014). The encoder consists of a bidirectional GRU-RNN (Chung *et al*., 2014), while the decoder consists of a uni-directional GRU-RNN with the same hidden-state size as that of the encoder, and an attention mechanism over the source-hidden states and a soft-max layer over target vocabulary to generate words. In addition to the basic model, it is also adapted to the summarization problem, the large vocabulary 'trick' (LVT) described in (Jean *et al.*, 2014). In this approach, the decoder-vocabulary of each mini-batch is restricted to words in the source documents of that batch. In addition, the most frequent words in the target dictionary are added until the vocabulary reaches a fixed size. The aim of this technique is to reduce the size of the softmax layer of the decoder which is the main computational bottleneck. In addition, this technique also speeds up convergence by focusing the modeling effort only on the words that are essential to a given example.

2. Capturing Keywords using Feature-rich Encoder: In this method it is important to identify the key concepts and key entities in the document. This the idea of word-embeddings-based representation of the input document and capture additional linguistic features such as parts-of-speech tags, named-entity tags, and TF and IDF statistics of the words. It is created additional look-up based embedding matrices for the vocabulary of each tag-type, similar to the embeddings for words. For continuous features such as TF and IDF, they are converted into categorical values by discretizing them into a fixed number of bins, and use one-hot representations to indicate the bin number they fall into. This allows us to map them into an embeddings matrix like any other tag-type. Finally, for each word in the source document, it is looked-up its embeddings from all of its associated tags and concatenate them into a single long vector. On the target side, it is

used only word-based embeddings as the representation.

3. Modeling Rare/Unseen Words using Switching Generator-Pointer: Often-times in summarization, the keywords or named-entities in a test document that are central to the summary may actually be unseen or rare with respect to training data. Since the vocabulary of the decoder is fixed at training time, it cannot emit these unseen words. In summarization, an intuitive way to handle such OOV words is to simply point to their location in the source document instead. In this model it is used a novel switching decoder/pointer architecture. In this model, the decoder is equipped with a 'switch' that decides between using the generator or a pointer at every time-step. If the switch is turned on, the decoder produces a word from its target vocabulary in the normal fashion. However, if the switch is turned off, the decoder instead generates a pointer to one of the word-positions in the source. The word at the pointer-location is then copied into the summary. The switch is modeled as a sigmoid activation function over a linear layer based on the entire available context at each timestep

4. Capturing Hierarchical Document Structure with Hierarchical Attention: In datasets where the source document is very long, in addition to identifying the keywords in the document, it is also important to identify the key sentences from which the summary can be drawn. This model aims to capture this notion of two levels of importance using two bi-directional RNNs on the source side, one at the word level and the other at the sentence level. The attention mechanism operates at both levels simultaneously. The word-level attention is further re-weighted by the corresponding sentence-level attention and renormalized. The re-scaled attention is then used to compute the attention weighted context vector that goes as input to the hidden state of the decoder. Further, it is also concatenated additional positional embeddings to the hidden state of the sentence-level RNN to model positional importance of sentences in the document. This architecture therefore models key sentences as well as keywords within those sentences jointly.

Lastly, third paper (Abigail See *et al., 2017*) discusses performing summarization with pointer-generator networks with key points:

1. Multi-sentence summary: In this model the abstractive work has focused on headline generation tasks (reducing one or two sentences to a single headline), the longer-text summarization is both more challenging (requiring higher levels of abstraction while avoiding repetition) and ultimately more useful.

2. Coverage Vector: Repetition is a common problem for sequence to- sequence models (Tu *et al.*, 2016; Mi *et al.*, 2016; Sankaran *et al.*, 2016; Suzuki and Nagata, 2016), and is especially pronounced when generating multi-sentence text. In this model it is adapted the coverage model of (Tu *et al.*, 2016) to solve the problem. In this coverage model, it is maintained a coverage vector, which is the sum of attention distributions over all previous decoder timesteps. This ensures that the attention mechanism's current decision (choosing where to attend next) is informed by a reminder of its previous decisions. This should make it easier for the attention mechanism to avoid repeatedly attending to the same locations, and thus avoid generating repetitive text.

3 Dataset

The data we chose was from the BBC news from year 2004 to 2005 available at (http://mlg.ucd.ie/datasets/bbc.html). News articles are divided into five categories: business, entertainment, technology, sports, and politics. Each category contains almost about 500 news. We used 70% data as a training set and 30% as a test set for all models.

4 Methodology

4.1 RNN LSTM for Text Classification

RNN in it's most fundamental level, is simply a densely connected neural network that has a memory. Output of the hidden layer is then fed back into the next hidden layer with its own input making it remember the past. There are many different structures of RNN based on number of input and output neurons. For text classification, many-to-one configuration will be used, where each news article with multiple words will be an input sequence and the labeled class will be an output. There is a problem with standard neural network i.e. vanishing or exploding gradients. For RNN, ideally, we would want it to have memories, so that the network can connect data relationships at significant distances in time [3]. However, more time steps we have, the more chance we have of vanishing or exploding gradients while back propagation. This problem makes it difficult for the RNN to learn long-distance correlations in a sequence.

Long short term memory (LSTM) networks proposed by [Hochreiter and Schmidhuber, 1997] to specifically address this issue of learning long-term dependencies [1]. LSTM cells uses internal memory state which is simply added to the processed input, greatly reduces the multiplicative effect of small gradients. The time dependencies and effects of the previous inputs are controlled by an interesting concept called a forget gate, which determines which state are remembered or forgotten [3]. Input gate and output gate are also part of LSTM cells. Input gate uses a *tanh* activation function and determines which inputs are switched on or off. Output gate which is implemented at the final stage of LSTM cell uses to two components, another activation *tanh* function and an output *sigmoid* function. This gate determines which states from the LSTM cell are the output. For our text classification model, RNN-LSTM was used to learn the sequence of words used for classification and Keras library was used for implementation. Diagram 1 shows the architecture of LSTM cell.

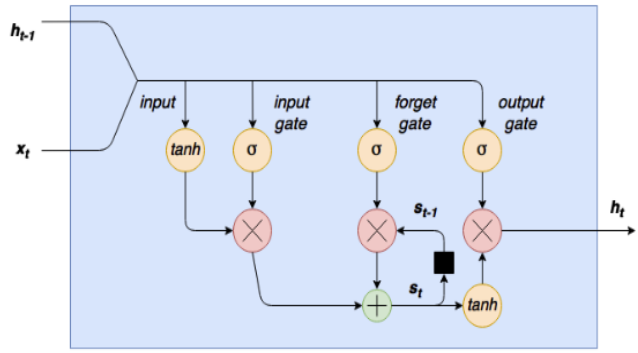

Diagram 1: LSTM cell

For the input layer of our model, it is required to represent each news article in a vector representation. Vocabulary size of our training dataset is 26,108 which is good size and therefore, instead of using pre-trained word embeddings such as GLoVe or word2Vec, we preferred to use our own embeddings. At first, data preprocessing was done with tokenization, removal of stop words and converting the text into lower

case for each news article. Then, we truncated the news articles to length of 3000 words and used 'post' padding for shorter news articles to make all input same length as same length vectors are required to perform computation in Keras. After that, we mapped each news article into real vector domain, where words are encoded as real-valued vectors in a high dimensional space, in our case 300 and the similarity between the words is seen by their closeness in the vector space. We limited the total number of words that we are interested in modeling to top 10,000 and zeroed the rest.

Next, we implemented hidden layer with 100 LSTM cells, where the word sequence of an article is learnt and the results are forwards to the output layer. As there are five categories for news, we used five neurons with *sigmoid* activation function. RNN-LSTM has a problem of over fitting, therefore, we used two dropout layers between input-hidden and hidden-output with 10% dropout.

For training our model, we used binary cross entropy as an error function. In training, 3 epochs were used with 80% accuracy, which was the highest accuracy achieved with different hyper parameters. Same data preprocessing steps were followed for test dataset. Results of RNN-LSTM will be discussed in the results section.

4.2 CNN for Text Classification

CNN was another model we used for text classification. CNN models are mostly used with image related tasks, however, they are gaining popularity with NLP tasks as well. Diagram 2 shows the use of CNN model for sentence classification. In this diagram, three filter region sizes 2,3 and 4, each of which has two filters is used. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus, a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states [7]. Similar approach is used for text classification.

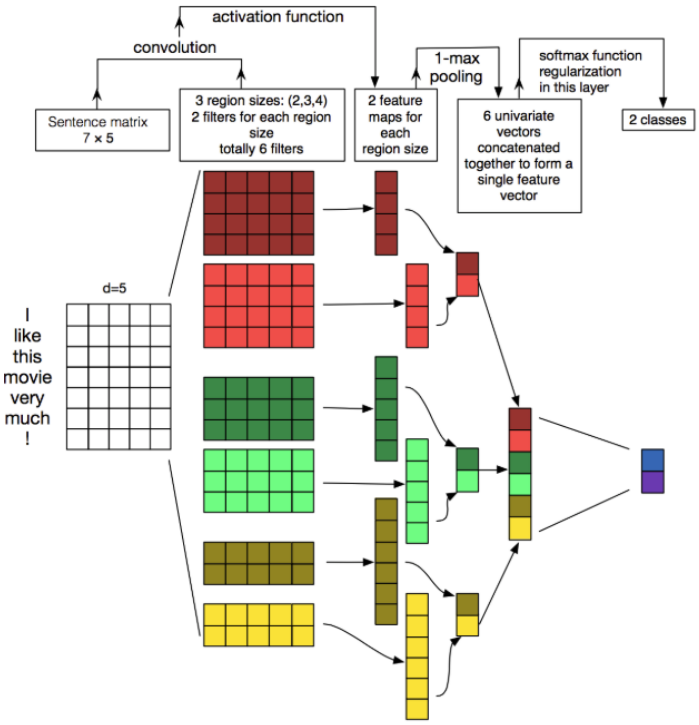

Diagram 2: CNN model architecture for sentence classification

For our CNN model, we used same preprocessed data as the RNN-LSTM due to same training dataset. Inputs are processed

with embedding and dropout layers which is the same as RNN-LSTM model. Then these embedding vectors are passed to two parallel convolution layer with different filter sizes.

These filters slide over full rows of the matrix (words). Thus, the width of our filters is same as the width of the input matrix, however, height, or region size may vary. Different filter size represents different features. In our model, we use filter size 1 and 2 which is same as unigram and bigram. These filters are capturing features quite similar to (but not limited) to n-grams, but represent them in a more compact way. In order to pick the most obvious features, we used max pooling. Then we merged two layers together and follow the same process with RNN, including, dropout and dense. Dropout of 10% was used to prevent over fitting. Additionally, in the convolution layer, 'ReLU' is used as activation, but, in the dense layer, 'sigmoid' is used as an activation. For training, 3 epochs were used with training accuracy 97.05%.

4.3 RNN LSTM for Text Summarization

The first step of the model is data preparation. The data processing step include tokenization, removing stopwords, substituting contractions with the full form of them and prepare word-embeddings.

The RNN-LSTM model that is used to train and produce the text summaries is a two-layered bidirectional RNN with LSTMs on the input data and a two layers RNN , each with an LSTM using Bahdanau attention on the target data. 100 epochs were used for training on the entire dataset that took 8 hours.

5 Results

For text classification, RNN-LSTM and CNN were used. Training accuracy was 80% and 97.05% when trained with same hyper parameters. We also measured the training time, RMM-LSTM took 21 minutes and 5 seconds to train on a 8 GB RAM and 7 core processor machine. However CNN model was faster to train and only took 5 minutes 26 seconds on same machine.

Testing accuracy for RNN LSTM was 80% and CNN was 99.07%. Same dataset was used for both models to make them comparable. Table 1 shows result summary.

METRICS	RNN LSTM	CNN
Training Accuracy	80.00%	97.05%
Time to Train	21mins 05 secs	5 mins 26 secs
Testing Accuracy	80.00%	99.07%

Table 1: Result summary for text classification

Performance of text summarization model using RNN-LSTM was measured using method called Recall Oriented Understudy for Gisting Evaluation (ROUGE). For a document D and automatic summary X, we get N human generated summaries for document D and calculate common percentage of bigrams between X and N reference summaries using the following formula [15]:

$$\text{ROUGE-2}=\frac{\sum_{s\epsilon\{RefSummaries\}}\sum_{bigrams\ i\epsilon S}\min(count(i,X),count(i,S))}{\sum_{s\epsilon\{RefSummaries\}}\sum_{bigrams\ i\epsilon S}count(i,S)}$$

We created human summaries for first 100 articles and compared it with our model. The mean value of ROUGE-2 is 0.38.

6 Conclusions

Based on the results, one can conclude that CNN performed better than RNN-LSTM for text classifications for BBC news dataset. RNN-LSTM is widely used for NLP tasks and is expected to perform well for text classification. Whereas, CNN that is commonly preferred for image related tasks showed very high accuracy in text classification. Instead of using own word embeddings, we could use pre-trained ones such as word2Vec or GloVe. Another potential for improvement is to use more vocabulary, in our project we only used top 10000 words, which may not be enough for RNN-LSTM model to distinguish document text between categories. CNN is known for its faster computations and it was seen during training the text classification models. CNN took almost 1/4th of time to train the model despite same parameters. For text summarization RNN LSTM is a good method but it is still far away from the human generated summary. This is supervised learning and the success of it depends on the data. The more data are used for training the better it is. But the future of deep learning is unsupervised. In the future development of types of neural networks unsupervised would bring an improvement of performance .

References

[1] Pengfei Liu, Xipeng Qiu, Xuanjing Huang, 2016 "Recurrent Neural Network for Text Classification with Multi-Task Learning"

[2] Duyu Tang, Bing Qin , Ting Liu, "Document Modeling with Gated Recurrent Neural Network for Sentiment Classification"

[3] "Recurrent neural networks and LSTM tutorial in Python and TensorFlow - Adventures in Machine Learning", Adventures in Machine Learning, 2018. [Online]. Available: http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/

[4] J. Brownlee, "Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras", Machine Learning Mastery, 2018. [Online]. Available: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

[5] Yoon Kim, "Convolutional Neural Networks for Sentence Classification", New York University

[6] "Implementing a CNN for Text Classification in TensorFlow", WildML, 2018. [Online]. Available: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

[7] "Understanding Convolutional Neural Networks for NLP", WildML, 2018. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[8] Alexander M. Rush, Sumit Chopra, Jason Weston . “A Neural Attention Model for Abstractive Sentence Summarization” arXiv:1509.00685

[9] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, Bing Xiang. “Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond” arXiv:1602.06023

[10] Abigail See, Peter J. Liu, Christopher D. Manning. ”Get To The Point: Summarization with Pointer-Generator Networks” arXiv:1704.04368

[11] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio “Neural Machine Translation by Jointly Learning to Align and Translate” arXiv:1409.0473

[12]"Text Summarization with Amazon Reviews – Towards Data Science", *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/text-summarization-with-amazon-reviews-41801c2210b.

[13] Brownlee, "Encoder-Decoder Deep Learning Models for Text Summarization", *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/encoder-decoder-deep-learning-models-text-summarization/

[14] [2]"facebookarchive/NAMAS", *GitHub*, 2018. [Online]. Available: https://github.com/facebookarchive/NAMAS. [Accessed: 24- Apr- 2018]

[15] Don Jufrasky Lectures on NLP: http://web.stanford.edu/class/cs124/