Key-sentence extraction with Neural Network

Jie Tang(jietang@stanford.edu)

Abstract

In this report, we explore neural networks to extract key sentence from document. By feeding each sentence with a fixed context window into feedforward neural network, over the testing set, we could achieve 80.56% accuracy, and negative predictive value(NPV) 94%, True negative rate(TNR) 83.53%, Positive predictive value(PPV, Precision) 29.31%, True positive rate(TPR, Recall) 56.16%, F1-score: 0.385. To improve this model, we add convolutional layer, so that we could capture word/sentence local/invariant features, and also train word/sentence representation. Finally we could achieve 85% accuracy, and NPV 94.5%, TNR 88.3%,PPV(precision) 37.7%,TPR(recall) 58.1%, F1-score: 0.457.

Introduction

Document summarization is an important topic in NLP, there are two mainstream methods: extractive and abstractive. The extractive method extracts key sentences from document and forms summary of the document, we then could consider summary as a concise representation of original document. In this report, our initial goal is: given a document, extract keyphrase and average these keyphrases' embedding vector as document embedding vector. But since we decide to choose supervised learning algorithm, it turns out it's not easy to find a labeled document set(in milestone, we describe some way to get key-phrase from summary sentence as label, but it turns out the quality is not good). At another side, we could easily find large document summarization data set which has high quality summary sentences for each document, so it would be easy for us to generate labels based on summary sentence. The final goal of this project right now is: Given a document as input, extract key sentences from document. Unfortunately, we have no time to finish the document embedding part, although it's trivial part comparing to key sentence extraction, we will continue it after this course project.

Related work

Unsupervised learning, e.g. TextRank[1] and LexRank[2], are very popular methods to extract key sentence or score sentences in document. These methods are usually based on some handcrafted features/scoring functions(e.g. tf/idf, PageRank model). Some researchers also explored supervised learning methods, e.g. [3], [4], which are also based on handcrafted features like sentence length, position, word pairs, etc, but they train with some model like Naive Bayesian/ Maxent. Recently, deep learning(neural network) is widely applied in NLP, especially word2vec[5] efficiently represents each word/phrase as an embedding vector, which is proved to be very useful feature in many NLP prediction tasks. At sentence/document level, similar work is also explored by [6][7][8][9], especially [6] applied convolutional neural network to sentence classification problem, improved state of art on most tasks. On the sentence extraction task, [10] proposed a novel algorithm to extract key sentence using convolutional neural network, the structure is very similar with [6][7], but they add one extra layer to model document. In this project, we would also apply neural network to key sentence extraction problem, instead of

handcrafted features, we will take advantage of embedding vector and convolutional neural network, let the neural network to learn useful features by itself.

Dataset

We will use data published in [11] as our training data, which contains both CNN and Daily Mail news articles. With each news, there is also “story highlights”, which is summary of the news. In total, we have 311,672 cnn/daily mail news articles, we will use 286,817 articles as training, 13,368 articles as validation, remaining 11,847 as testing.
Since the summary sentences would be slightly or even totally different from sentences in document, we can’t use summary sentence as our labels directly. So as a preprocessing step, we generate our labels in following steps:

1. Break down document into sentences(by the symbol . ! ?)
2. Break each sentence into phrases according to an embedding vector vocabulary(Detailed algorithm would be described in “Methods” section).
3. Generate sentence embedding vector by averaging each phrases’ embedding vector with tf/idf weight.
4. In the same way, we generate sentence embedding vector for summary sentence.
5. Calculate similarity between each document sentence and summary sentences, pick up the highest similarity as document sentence score.
6. For document sentences whose scores are higher than some threshold, label them as key sentences. In our case, we find when the threshold is set as 0.9, most of the labeled key sentences are very close to summary sentences.

Feature

In this project, we implemented two models : feedforward network, and convolutional network, in both model, the features are embedding vectors of each sentence and its context window, e.g. given a sentence S_i, we consider surrounded sentences $\{S_{i-n}, S_{i-n+1}, ...S_{i-1}, S_{i+1}, ..., S_{i+n}\}$ as its context window. Also for each sentence S_i, we could construct an embedding vector $E_i \in R^m$. For feedforward network, for each sentence, we concat and flatten $\{E_{i-n}, ...E_{i,}....,E_{i+n}\}$ as $F_i \in R^{(2n+1)m\times 1}$ as input feature which is single column matrix. But for convolutional network, we concat $\{E_{i-n}, ...E_{i,}....,E_{i+n}\}$ as $F_i \in R^{(2n+1)\times m}$ which is a 2D matrix.

Methods

Sentence embedding

We use pre-trained word embedding vector in our project(these embedding vectors are trained from Google search queries). There are also a few ways[8][9] to generate sentence embedding vector, but to be simple, here we simply average embedding vectors of phrases in sentence(by tf/idf weight) as sentence embedding vector for feedforward network, and contact word embedding vector as sentence embedding vector for convolutional network. In both case, we need split sentence into phrases according to embedding vector vocabulary. This is done by a simple algorithm: Generally, the problem is stated as

Given a dictionary D, break down document S so that S = $\bigcup_{i=1}^{n} phrase_i$ *where* $phrase_i \in D$ *and* $phrase_i \cap phrase_{i+1} = empty$, If we assume segments are independent, we could define likelihood as $Likelihood = \prod_{i=1}^{n} P(phrase_i)$ where $P(phrase_i) = \frac{NumberOfDocumentContains(phrase_i)}{TotalNumberOfDocuments} = \frac{1}{IDF(phrase_i)}$

By solving $max\ Likelihood$, we could construct an optimal segmentation of each document, this could be solved by dynamic programming:

$$log\ Likelihood = \sum_{i=1}^{n} logP(phrase_i) = -\sum_{i=1}^{n} log(IDF(phrase_i))$$

So it's equal to $min\ L = \sum_{i=1}^{T} log(IDF(phrase(w_{i-m+1}...w_i)))$ where $m = 1, ..., PhraseSizeLimit$, $phrase(w_{i-m+1}...w_i) \in D$, D is embedding vector vocabulary, IDF of each phrase is also stored in the vocabulary. $phrase(w_{i-m+1}...w_i)$ is a phrase which is composed of words $w_{i-m+1}...w_i$.

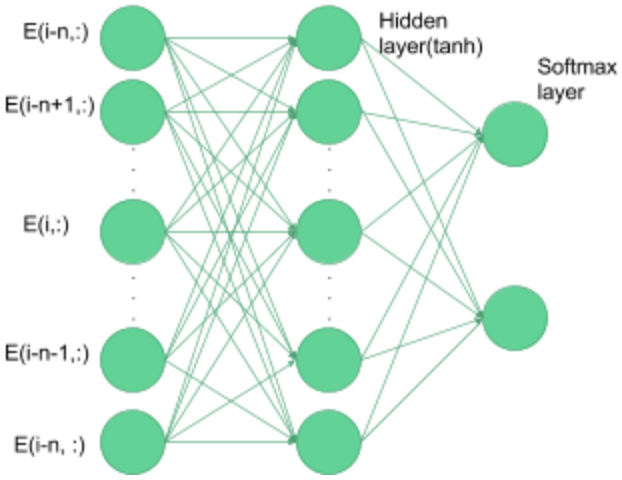
Let $L(j)$ as minimal negative log likelihood ending at j-th word, we have

$$L(j) = min\ \{L(j - m) + log(IDF(phrase(w_{j-m+1}...w_j))$$

By solving this DP, we could construct an optimal segmentation of document, and in our current implementation, it's simple, and the result looks OK.

Feedforward network

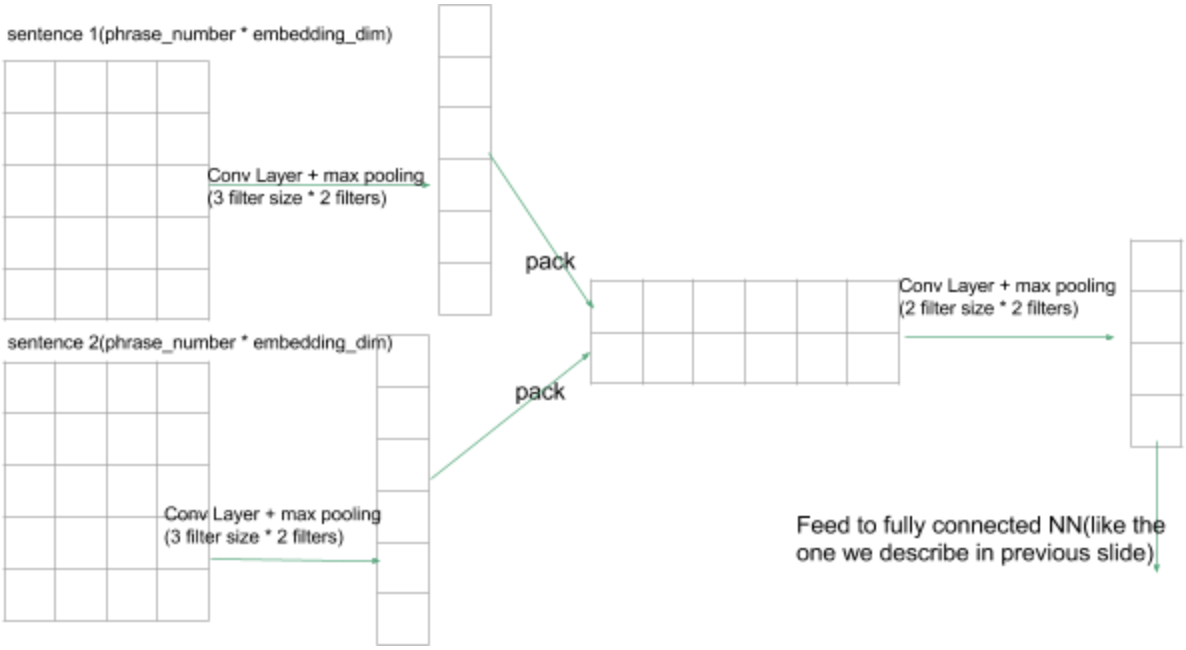
As we describe in "Feature" section we feed embedding vector of each sentence and its context window into a single layer feedforward neural network, the label would be 1(it's a key sentence) or 0(it's not a key sentence). Alternatively, the label means whether the sentence could represent its contextual sentences, if it could, it's a key sentence. So what we are predicting is $p(S_i\ is\ key\ sentence|C_i)$ where $C_i = \{S_{i-n}, S_{i-n+1}, ...S_{i-1}, S_{i+1}, ..., S_{i+n}\}$



Convolutional network

One drawback of our feedforward network is the sentence embedding is averaged by phrase embedding using tf/idf weight, it may be not the best way to construct a sentence embedding.

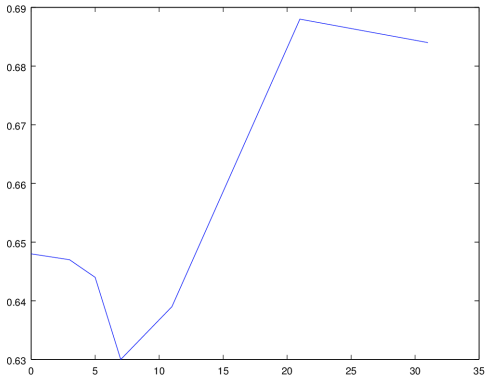
As [6][7][10] suggested, convolutional network would be a better way to learn sentence(even document) representation. The network is like this(a tiny version):



we concat $\{E_{i-n}, ...E_{i,}....,E_{i+n}\}$ as $F_i \in R^{(2n+1)\times m}$ which is a 2D matrix, the convolutional layer apply filters of different size over the 2D embedding matrix, this is done by a sliding window from top to down with step 1, and calculate convolution between embedding matrix and filter. Unlike image application, we don't calculate convolution in horizontal direction, since it doesn't make sense for word/phrase embedding, which should be considered as an inseparable representation of word. So the filter size is $R^{l\times n}$, where n is dimension of word/phrase embedding and is fixed. l is filter size, which could be varied to capture different level of features. By doing this and following MaxPool operation, we could construct embedding vector for each sentence, and with one more convolutional layer, we finally generate representation for this sentence and its context window, and feed them into a fully connected feedforward network as we described before.

Experiments

Context window explicitly is an important factor in our model, if it's too big, it's not easy for our model to capture the central topic in the context, if it's too small, the model may not have enough contextual information to make decision. So we explore this part at first, we fix other parameters, only change the window size, and plot the results as loss at dev set V.S. window size(right image).



It's clear that when we increase window size, the loss would be dropped at first, then increased as window size is bigger than 7. This is somehow consistent with our intuition, since at average, 7 sentences could be a section in the document. In our following experiments, we will fix our window size as 7.

It's obvious key sentences are much less than non-key sentences(roughly 1: 10), this may mislead our model to classify all sentences as non-key sentences, since the accuracy/loss would be low. To solve this issue, in our each

training batch, we pick up 32 key sentences, and randomly sample another 32 non-key sentences around those key sentences, to make sure training data is balanced in each batch.

By dosing a few iterations of hyper-parameter searching, we choose the best model over dev set, and evaluate over testing set, we could achieve 80.56% accuracy, and negative predictive value(NPV) 94%, True negative rate(TNR) 83.53%, Positive predictive value(PPV, Precision) 29.31%, True positive rate(TPR, Recall) 56.16%, F1-score: 0.385.
And confusion matrix(NOTE: sample is sentence instead of document) is
[[173782 34269]
[11093 14208]]
For feed-forward nn, some parameters are learning rate : 0.1, L2 regularization: 0.001, batch size 64, embedding dimension 200, hidden layer : 256.
With convolutional network, we could improve performance a little bit, but it took much longer to train: we could finish feedforward network training in 3 hours with 1M+ sentence, for convolutional network, it took me 16 hours to finish one epoch with same training data. Finally we could achieve 85% accuracy, and NPV 94.5%,TNR 88.3%,PPV(precision) 37.7%,TPR(recall) 58.1%, F1-score: 0.457, and confusion matrix:
[[183772 24279]
[10600 14701]]
The convolutional layer setup is 3 filter sizes [3, 4, 5], 128 filters. All sentences share the same convolutional layer weight.
By looking at the testing results, we find the model learned that for news articles, first 2~3 sentences are usually salient sentences, so the model tend to label these sentences as key sentences, but this would not be always true even for news articles, a big part of errors come from this. And in other cases, our preprocessing step failed to label some sentences as key sentences, but the neural network successfully labels them by learning from training data, this is not a small part of “errors”. It’s also a trade-off for us, we want to get high quality training labels, so we have to set high threshold(0.9) in preprocessing step, but we have to lose some recall. At another side, lower down the threshold would introduce too much noises into training data. Another finding is our evaluation metrics may be too strict, with precision/recall, we will exclude the case that some sentence is not labeled as key sentence in preprocessing step, but it’s still somehow similar to some summary sentence. Rouge score would be a perfect metric to avoid this kind of issue, but we don’t have time to set up and calculate it, will do this after this project.

Conclusions and Future work

In this project, we successfully applied neural network on key sentence extraction problem, and shows convolutional network does give us some gain although the training time is much longer comparing to a single layer feedforward neural network. There are still a few things we need to consider as future improvements:

1. Instead of classifying each sentence as important or not, we may directly generate label for the whole document, e.g. a binary label sequence for each document, label is 0/1 for each sentence. Although in our experiment, it shows big context window doesn’t help, but in that case we simply concat embedding vectors. We hope convolutional layer could help extract more useful features at document level.
2. Our pre-trained embedding vectors are trained from search query log, for this specific problem, we may try embedding vector from news articles.

References

[1]Rada Mihalcea, Paul Tarau, TextRank: Bringing Order into Texts, Department of Computer Science University of North Texas
[2]Güneş Erkan and Dragomir R. Radev: LexRank: Graph-based Lexical Centrality as Salience in Text Summarization
[3]Miles Osborne, Using Maximum Entropy for Sentence Extraction, Proceedings of the Workshop on Automatic Summarization (including DUC 2002), Philadelphia, July 2002, pp. 1-8. Association for Computational Linguistics.
[4]Peter D. Turney,Learning Algorithms for Keyphrase Extraction,Information Retrieval
May 2000, Volume 2, Issue 4, pp 303–336
[5]Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Distributed Representations of Words and Phrases and their Compositionality, NIPS 2013
[6]Yoon Kim, Convolutional Neural Networks for Sentence Classification, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751, October 25-29, 2014, Doha, Qatar
[7] Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom, A Convolutional Neural Network for Modelling Sentences,Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 655–665, Baltimore, Maryland, USA, June 23-25 2014
[8] Quoc Le, Tomas Mikolov, Distributed Representations of Sentences and Documents, Proceedings of the 31 st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32
[9]Ryan Kiros, Yukun Zhu, etc, Skip-Thought Vectors, Advances in Neural Information Processing Systems 28 (NIPS 2015)
[10] Misha Denil, Alban Demiraj, Nando de Freitas, Extraction of Salient Sentences from Labelled Documents, https://arxiv.org/pdf/1412.6815v2.pdf
[11]Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, Bing Xiang, Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond,arXiv preprint arXiv:1602.06023