Automatic Summarization of Document and Document Collection with Attention Mechanism

Faculté Informatique et Comminications

École Polytechnique Fédérale de Lausanne

Chen LIU

Supervised By:

Dr. Claudiu Musat, Swisscom
Prof. Martin Jaggi, EPFL

Lausanne, 2017

Acknowledgements

As internship credited with Master Project, I am very grateful for Swisscom to provide a pleasant working environment, sufficient computing resources and hardwares for me to complete this project. Colleagues in Swisscom are very nice and share common interests with me, both in academic field and daily life.

Especially, I would like to thank Dr. Claudiu Musat and Prof. Martin Jaggi for supervising me throughout the whole project. They have given me much help in finding relevant work and spiring my intuition. The logic chain of this project becomes clearer and clearer after several discussions with them.

Lausanne, 14th, Auguest, 2017 Chen Liu.

Abstract

Generating readable, comprehensive, concise and consistent summaries on large-scale dataset is a challenging task with broad applications. Recently, there are many data-driven neural network models to learn the document-summary mapping. The most popular one is Sequence-to-Sequence Model which contains a LSTM-based encoder and decoder. While learning a LSTM-based model is not an easy job, specially for long sequences, so Attention Mechanism is introduced to capture the long time dependency by shortcuts.

Sequence-to-Sequence Model with Attention Mechanism has achieved great success in summarization tasks. However, to the best of our knowledge, few previous research works have incorporated semantic or linguistic information into this model, such as entity name detection and sentence embeddings. In this project, we will explore the possibility to improve Sequence-to-Sequence Model and their variants by external information.

Key words: Deep Neural Network, Text Summarization, Attention Mechanism

Contents

List of Figures

List of Tables

1 Introduction

Text summarization is a key problem in the field of natural language processing (NLP). It compresses the documents to generate summaries which only contain the most informative part of the original texts. In the era of big data, summarization tools are very beneficial for many tasks such as information retrieval, data mining and knowledge discovery. [Carbonell and Goldstein, 1998]

There are two kinds of techniques to generate summaries: extraction and abstraction. Extractive models directly select a subset of the original documents to generate summaries, while abstractive models are capable of synonyms replacement and paraphrasing. On one hand, extractive model can better guarantee the grammar strcuture of the text and some key words; on the other hand, abstractive models are more flexible and tend to generate even more compressive sentences.

Many traditional summarization models rely heavily on human-engineered features or rules [Hu and Liu, 2004, Maskey and Hirschberg, 2005], which make models difficult to generalize and easy to be fooled because of the high flexibility of texts. Recently, neural network has become a very powerful tool to train end-to-end models without manual features. Deep neural network, especially recurrent neural network and its variants such as Long Short Term Memory (LSTM), can better describe the language model compared with traditional classifiers. [Mikolov et al., 2010, Sundermeyer et al., 2012]

Relationship between two distant words or other language unites is a common phenomena in natural languages. However, even some complicated structures like LSTM can not capture this long time dependency very well. Attention Mechanism is proposed to solve this problem. It uses shortcuts to force the current states to "look back on" the old ones to boost their correlation. Recent experiments show that Attention Mechanism improves the performance of long time dependency detection significantly. [Rush et al., 2015]

Text summarization is a very subjective task. It is not an easy job even for human to evaluate the generated summaries in a quantitative way. Traditionally, researchers use summaries written by humans as gold summaries and compare them with machine generated ones based on their statistical features. ROUGE score is one popular metric of this category. [Lin, 2004] Recently, researchers have taken advantages of neural networks to obtain semantic representation of language units like word embeddings [Mikolov et al., 2013a] and sentence embeddings [Pagliardini et al., 2017]. In this project, in addition to ROUGE score, we also calculate the embedding distances between input documents and output summaries. We use it as one additional metric for evaluation. This metric can even be incorporated for training neural summarization models.

In this project, we mainly pay attention to neural network models to generate qualified summaries efficiently. We implement difficult extractive, abstractive and hybrid models to compare their performances. The structure of this report is organized as follows:

- The chapter 2 introduces some background knowledge, which is the building blocks or frameworks of the following models.
- The chapter 3 demonstrates some neural network models for summary generation, including extractive models and abstractive models.
- The chapter 4 mainly shows the results of the experiments. We will discuss about the experiment results as well as the evaluation metric.
- A brief conclusion will be drawn in the final chapter and we will also point out some possible future directions to improve the performance of current models.

2 Model Components

2.1 Overview

In language model, a sentence or a document $\mathbf{D}$ is considered as a sequence of tokens $\{\mathbf{d}_i\}_{0<i<N}$. Given the context $\mathbf{C}$, the possibility of this sequence's appearance is $P(\mathbf{D}|\mathbf{C}) = \Pi_{i=1}^{N} P(\mathbf{d}_i|\mathbf{C}, \mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_{i-1})$. This formula can be modeled by a recurrent neural network, where a predicted distribution $P(\mathbf{d}_i|\mathbf{C}, \mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_{i-1})$ is computed in each step. If we use negative log-likelihood as the loss function, the model will be trained to maximize the conditional probability $P(\mathbf{D}|\mathbf{C})$. [Bengio et al., 2003]

Text summarization is a generative problem. We need to take advantage of neural language model to make sure the output sequences have large probability to appear. That is to say, the output sequences are readable and make some sense. In most models, we feed the original documents in sequences to learn representations of the whole documents. Then we run another recurrent neural network for generating summaries in another sequences. This encoder-decoder framework is the cornerstone of all neural summarization models in this project.

The content of this chapter expands around neural language models based on encoder-decoder networks. In section 2.2, we foucus on the input part. We will introduce CBOW and Sent2Vec to convert words or sentences to vectors, which can be recognizable by neural networks. We begin neural language model with its building blocks, recurrent cells, in section 2.3. In section 2.4 and 2.6, we introduce two kinds of encoder-decoder networks: Sequence-to-Sequence Network (Seq2Seq Network) and Pointer Network. For Seq2Seq Network, we demonstrate a trick called Attention to boost its performance in section 2.5.

In this and the following chapters, we use $\mathbf{X}^e = \{\mathbf{x}_1^e, \mathbf{x}_2^e, ..., \mathbf{x}_M^e\}$ and $\mathbf{X}^d = \{\mathbf{x}_1^d, \mathbf{x}_2^d, ..., \mathbf{x}_N^d\}$ to denote the input sequence of encoder and decoder, of which the length are M and N respectively. Similarly, $\mathbf{H}^e = \{\mathbf{h}_1^e, \mathbf{h}_2^e, ..., \mathbf{h}_M^e\}$ and $\mathbf{H}^d = \{\mathbf{h}_1^d, \mathbf{h}_2^d, ..., \mathbf{h}_N^d\}$ are used to represent the hidden states of encoder and decoder. $\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_N\}$ and $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\}$ represent the predicted output sequence and ground truth sequence respectively.

2.2 Word Embedding and Sent2Vec

Word embedding is a common technique for word representation in the field of natural language processing, because we must encode words before feeding it to the neural network. The most simple and naive way of word embedding is to use one-hot vector. However, the dimension of one-hot representation is the vocabulary size and this consumes a lot of memory. In addition, one-hot vector neglects the semantic relations between words since it has no context information.

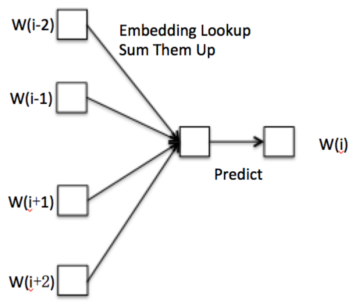
As a result, we need compressive word representation with semantic meanings to encode words for further processing. Continuous bag of words (CBOW) is a popular context-based structure to obtain such word embeddings. [Bengio et al., 2003, Mikolov et al., 2013a] As figure 2.1a shows, it trains a network consisting of an embedding matrix and a classifier, both of which are randomly initialized. For a sequence of words $\{\mathbf{w}_i\}_{1\le i\le N}$ and a window size t, the model takes $t+1$ consecutive words except the central one $\{\mathbf{w}_i\}_{j-\frac{t}{2}\le i\le j+\frac{t}{2}, i\neq j}$, sums up their word embeddings as input and predicts the missing central word $\mathbf{w}_j$ using a softmax classifier.

CBOW is a totally unsupervised model but it needs big dataset to get word embeddings of good quality. One computation bottleneck is the softmax classifier part, because the output dimension is the large vocabulary size. Hierarchical softmax [Mikolov et al., 2013b] is designed here to solve this problem. It arranges words in a Hoffman tree, a special binary tree, with each of them as a leaf. The local softmax of each node only assigns a probability to visit its left or right child. As a result, a global probability is assigned to a word by a random walk.

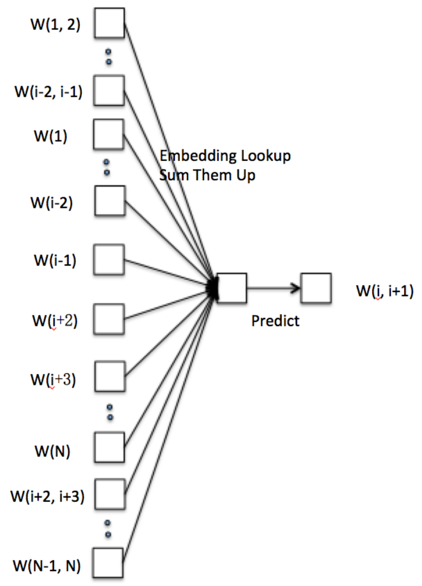
Another popular trick to train CBOW more effectively is negative sampling. [Mikolov et al., 2013b] Since different words have variant frequencies in the whole corpus, the bag-of-words based CBOW model has tendency to pay more attention to the most frequent words. This is not beneficial because most frequent words such 'a' and 'the' usually contain little information while the frequency of key words like entity names are not very big. To ease the model's bias to word frequency, negative sampling randomly removes some most frequent words by a possibility of $1-\sqrt{\frac{p}{f_w}}$ (p is a hyper-parameter and f_w is the word's frequency). Negative sampling is mainly used to improve the quality of rare words' embeddings.

In addition to CBOW, [Mikolov et al., 2013a] also introduced another popular model called Skip-gram. In contrast to CBOW, Skip-gram uses the central word to predict the joint distribution of its context. Hierarchical softmax and negative sampling can also be applied to Skip-gram. In this report, we do not describe Skip-gram in detail, because it is not used in the following models.

Word embedding concentrates on the meaning of single words, but sometimes we need embeddings of larger language units such as phrases, sentence or even documents. One simple and fast way is to use the average embeddings of words they contain. However, this method does not take the word order into consideration. In order to improve the embedding quality, we extend CBOW model by inputting embeddings of not only words but also N-



(a) Continuous bag of word model (CBOW), the window size is 4.



(b) Sent2Vec model which takes only unigram and bigram into consideration.

Figure 2.1 – CBOW and Sent2Vec Models

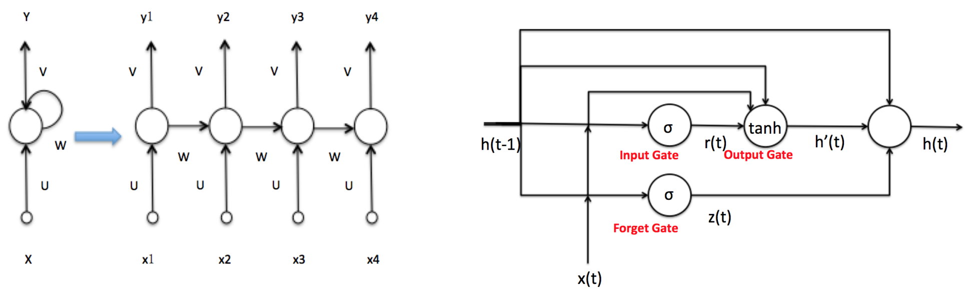
gram tokens to make it better understand the word order. The embedding of a sentence are the average embedding of all N-gram tokens it has. The new model is called Sent2Vec [Pagliardini et al., 2017] and Figure 2.1b shows the bigram version of Sent2Vec.

Hierarchical softmax is also applicable to Sent2Vec to accelerate training and test process. The dictionary of Sent2Vec model also includes N-gram tokens, so hierarchical softmax is even more necessary. However, negative sampling is not suitable here, especially when N is big. It is because that negative sampling will remove some frequent words randomly and will probably break the structure of N-gram tokens.

2.3 Recurrent and LSTM Cells

In this part, we will demonstrate the building blocks of all the following neural networks: Recurrent Neural Network Cell (RNN Cell, figure 2.2a) and Long Short-Term Memory Cell (LSTM Cell, figure 2.2b). They are very powerful to process sequential inputs. In summarization task, it is a sequence of words or sentences in the original document.

As figure 2.2a shows, a recurrent connection $\mathbf{W}$ is added to a RNN Cell to enable it to capture time dependencies. The hidden states $\mathbf{h}_t$ of RNN Cell depend not only on the current input $\mathbf{x}_t$ but also on the previous hidden states $\mathbf{h}_{t-1}$. Mathematically, given the input matrix $\mathbf{U}$ and activation function f, the hidden states of RNN Cell is calculated as follows: $\mathbf{h}_t = f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1})$.



(a) Recurrent Neural Network Cell. (b) Long-Short Term Memory Cell.

Figure 2.2 – Structure of RNN and LSTM cells.

RNN Cell is an iterative model since its current status depends on the previous one. Compared with feedforward neural networks, recurrent neural network (RNN) is more difficult to train. There are some tricks to train RNN. Firstly, $tanh$ instead of rectified linear (RELU, [Krizhevsky et al., 2012]) and sigmoid is used in RNN. [Elman, 1990] The sigmoid function suffers from gradient vanishing problem since its maximum gradient is 0.25. By contrast, RELU suffers from an explosion problem because RELU is not a saturated function. It is very easy for it to explode after a few iterations. Secondly, the biggest singular value of recurrent matrix **W** can not be too big, otherwise the direction of hidden state vector will be driven to corresponding singular vector and ignore the others by iterations. [Pascanu, 2015] To prevent the recurrent matrix from being too big, regularization is usually applied.

We can unfold the RNN via time dimension, like the right part of figure 2.2a. Information flows not only in the spatial dimension but also in the temporal dimension. In the same time, the gradient back propagates also in two dimensions. This method to calculate gradients is called Back-Propagation Through Time (BPTT, [Werbos, 1990]).

Sometimes, the length of input sequence is very long, RNN is not capable of catching long time dependencies. Long Short-Term Memory Cell (LSTM Cell) is desgined to solve this problem. [Hochreiter and Schmidhuber, 1997] The structure of LSTM Cell is shown in figure 2.2b, including an input gate, a forget gate and an output gate. The input - output gate is similar to RNN Cell, they mix input $x(t)$ and previous hidden states $h(t-1)$ to get a 'updated state' $h'(t)$. Forget gate receive the same input but output a forget rate vector, which is used to coordinate the 'updated states' $h'(t)$ and previous states $h(t-1)$ and obtain the next hidden states $h(t)$. The mathematical formula is below:

$$\begin{aligned}
\text{InputGate}:\mathbf{r}(t) &= \sigma(\mathbf{W}_r[\mathbf{h}(t-1),\mathbf{x}(t)]) \\
\text{ForgetGate}:\mathbf{z}(t) &= \sigma(\mathbf{W}_z[\mathbf{h}(t-1),\mathbf{x}(t)]) \\
\text{OutputGate}:\mathbf{h}'(t) &= tanh(\mathbf{W}[\mathbf{r}(t)\mathbf{h}(t-1),\mathbf{x}(t)]) \\
\text{Update}:\mathbf{h}(t) &= (1-\mathbf{z}(t))\mathbf{h}(t-1)+\mathbf{z}(t)\mathbf{h}'(t)
\end{aligned}$$

Forget gate is the most important part of LSTM Cell. [Greff et al., 2016] Actually LSTM Cell learns whether or not it should forget the previous sequence by forget gate. This flexibility makes LSTM able to capture long time dependencies. In many speech recognition tasks and natural language processing tasks, LSTM outperforms standard RNN [Gers et al., 1999, Gers and Schmidhuber, 2001, Sak et al., 2014] by a big margin.

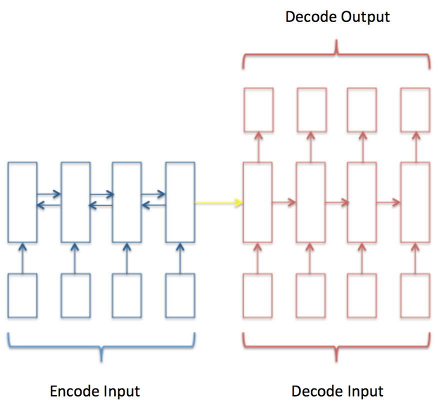
Traditional LSTM is uni-directional, it only pays attention to the previous inputs. Sometimes, the future inputs also can give us some hints about current outputs. Bidirectional LSTM is such an extension by integrating two LSTMs. [Graves and Schmidhuber, 2005] It consists of one forward LSTM and one backward LSTM, which scan input sequences in two different directions. The output of each time stamp is generated from the hidden states of both component LSTMs.

2.4 Sequence-to-Sequence Network

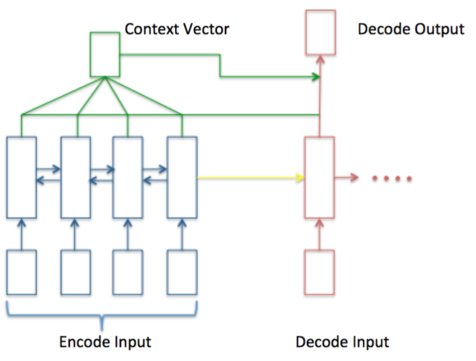
As figure 2.3a shows, Sequence-to-Sequence Network (Seq2Seq Network, [Sutskever et al., 2014]) is a generative model consisting of an encoder and a decoder. Usually, the encoder and the decoder are two different LSTMs. The encoder learns a series of representations of input sequences. Sometimes, Bidirectional LSTM is used in this part to learn better context representations. In classical Seq2Seq Network, the decoder generates the output sequences based on its hidden states. The only connection between the encoder and the decoder is a projection layer from the encoder's last hidden states to the decoder's first hidden states.

In summarization task, the input of the encoder is a sequence of word embeddings or sentence embeddings in the original documents. In the training phrase, where the ground truth output sequences are available, the input of the decoder at time stamp t is the ground truth token at time stamp $t-1$. The first decoder input token is always a special token '[START]'. In validation and test phrase, where there is no available ground truth, we input the lastest predicted token instead.

Seq2Seq Network is a good interpretation of language model. If we parameterize the network by θ, then the output distribution at decoding time t represents $P(\mathbf{y}_t|\theta,\mathbf{X}^e,\mathbf{y}_1,\mathbf{y}_2,...,\mathbf{y}_{t-1})$. By minimizing the negative log-likelihood $-\sum_{t=1}^{N}\log P(\mathbf{y}_t|\theta,\mathbf{X}^e,\mathbf{y}_1,\mathbf{y}_2,...,\mathbf{y}_{t-1})$, the Seq2Seq Network is trained to maximize the probability $P(\mathbf{Y}|\mathbf{X}^e,\theta) = \Pi_{t=1}^{N}P(\mathbf{y}_t|\theta,\mathbf{X}^e,\mathbf{y}_1,\mathbf{y}_2,...,\mathbf{y}_{t-1})$, which is consistent with language model.



(a) Classic Sequence-to-Sequence Network



(b) Sequence-to-Sequence Network with Encoding Attention

Figure 2.3 – Sequence-to-Sequence Network and Attention Mechanism

In the test phrase, we need to find a sequence $\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_N\}$ to maximize the $P(\hat{\mathbf{Y}}|\mathbf{X}^e, \theta)$. Usually, beam search is used here to avoid exponential complexity.

Beam Search

Beam search is an exploratory algorithm that builds a route by expanding the most promising node within a limited candidate set. [Reddy et al., 1977] If the beam size is K, it means that every time we only keep the K most promising routes we explored. In our summarization task, at time t, we have K summary prefix candidates $C_t = \{\mathbf{seq}_i | 1 \leq i \leq K\}$. We expand each candidate sequence by appending the top K most probable next words: $C'_t = \{\mathbf{seq}_i + \mathbf{w}_i[k] | 1 \leq i, k \leq K\}$. Here $\mathbf{w}_i$ is the sorted list of most probable next words given the prefix $\mathbf{seq}_i$. C'_t is a set of size K^2, we only select the K most probable candidate sequences to form the set C_{t+1} for next iteration.

Beam search is an efficient algorithm for searching optimal routes. It explores larger spaces than greedy search and the complexity is $O(N * K)$ given the route length N.

2.5 Attention Mechanism

Unlike classic RNN, which outputs a token each time when receiving a token, Seq2Seq Network's input and output part are separate. It first receives all the input tokens of the encoder to obtain a compressive representation and then generates outputs in the decoding phrase from this representation. If the input sequence is very long, it would be very difficult for the decoder to "remember" some information in the very early encoding steps and even the early decoding steps. In addition, in Seq2Seq Network, predictions are generated only directly from the decoders' hidden states. These compressive hidden states also constrain the information flow originated from the encoder.

Essentially, these problems arise from the fact that Seq2Seq Network has only one long route from the input to the output part. For a long sequence, it is equivalent to a deep network with shared forward weight of each layer if we unfold the Seq2Seq Network like figure 2.3a. Training a very deep neural network is not an easy job [Larochelle et al., 2009], and the shared weight of forward connections constrains the model's flexibility and leads to information loss.

For feedforward networks, researchers use shortcuts to skip one or more layers to avoid gradient vanishing/exploding and improve the training efficiency, such as deep residual networks. [He et al., 2016] Similarly, we use shortcuts between hidden states of encoder and decoder to evoke current decoder's "attention" to previous encoding or decoding states. This extension for Seq2Seq Network is called Attention Mechanism, including Encoding Attention and Decoding Attention. [Rush et al., 2015, Paulus et al., 2017]

Encoding Attention

As figure 2.3b shows, Encoding Attention adds some shortcuts between current decoding state $\mathbf{h}_t^d$ and all hidden states in the encoder $\mathbf{H}^e$. Such shortcuts can be any kinds of layer, such as linear projection layer or inner dot operator, which outputs a scalar meaning the attention that the decoder should pay to a specific encoding state. Mathematically, we use $\mathbf{a}_{it} = F_{att}(\mathbf{h}_i^e, \mathbf{h}_t^d)$, $\alpha_{it} = softmax(\mathbf{a}_{it})$ to represent the unnormalized and normalized attention from decoding state $\mathbf{h}_t^d$ to encoding state $\mathbf{h}_i^e$.

With attention vector α, we can calculate the context vector **cv** by a linear combination of all encoder's hidden states: $\mathbf{cv}_t = \sum_{i=0}^{M} \alpha_{it}\mathbf{h}_i^e$. The final predicted distribution is made by both decoder's current hidden state and the context vector: $P(\mathbf{y}_t|\mathbf{X}^e, \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{t-1}, \theta) = F_{out}(\mathbf{h}_t^d, \mathbf{cv}_t)$.

Encoding Attention is widely used in the Seq2Seq model and can greatly boost its performance. By shortcuts, it actually provides alternative data flow from the encoder's hidden states to the final outputs. In addition, it enriches the information directly linked to the output, which make the output distribution more comprehensive. What's more, the current decoded token is more relevant to the input token with high attention. As a result, the normalized attention vector α can also be regarded as content coverage of current decoded token over the input sequence. This property can be used to get rid of repetition in the output sequence. [See et al., 2017]

Decoding Attention

Encoding Attention boosts the connections between the encoder and the decoder, but this is not enough. Sometimes, the decoder should also pay attention to the previous decoded tokens to avoid repetition outputs, especially when the decoding sequence is very long.

Similar to Encoding Attention, Decoding Attention is the mechanism which introduces several shortcuts between current and previous decoder's hidden states. [Paulus et al., 2017] The decoding attention vector $\mathbf{a}^d_{:,t}$ at t-th decoding step is a t-dimensional vector calculated by $\mathbf{a}^d_{it} = F^d(\mathbf{h}^d_i, \mathbf{h}^d_t), 1 \leq i \leq t$. The normalized attention vector is obtained by softmax and the decoding context vector $\mathbf{cv}^d_t$ is also the linear combination of attention and decoder's hidden states: $\mathbf{cv}^d_t = \sum_{i=0}^{t} \mathbf{a}^d_{it}\mathbf{h}^d_t$. It is clear that $\mathbf{cv}^d_t$ has the same dimension of decoder's hidden states and is independent of decoding time t. Usually, decoding context vector $\mathbf{cv}^d_t$ together with encoding context vector $\mathbf{cv}_t$ and the current decoder's hidden state $\mathbf{h}^d_t$ jointly determine the predicted distribution.

Decoding Attention keeps the model reminded of what has already been decoded. As a result, this mechanism will prevent the model from decoding duplicate sequences, which is a common problem for abstractive models.

2.6 Pointer Network

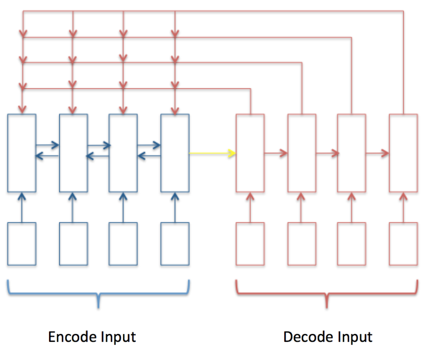
Seq2Seq Network is very powerful to solve many generative problems and Attention Mechanism can improve its performance, however it still has some limitations and disadvantages to solve our problems.

Generally, Seq2Seq Network relies on a fixed output dictionary. It can only output a probability distribution over words in a fixed and known dictionary. This property limits the model's ability to deal with out-of-vocabulary words in input documents. In addition, Seq2Seq Network can not model problems where the output dictionary depends on the input sequences, such as Salesman Problem. [Vinyals et al., 2015]

For our summarization task, although abstractive model encourages some advanced tricks like paraphrasing and replacement to generate a summary, it is still true that words that appear in the original documents should have higher probability to appear in the summary. What's more, some factual words, such as names and locations, cannot be paraphrased or replaced by other words. A wrong factual name will even change the meaning of the whole sentence, although it is still grammatically right and has a high appearing probability in language model. Seq2Seq model does not explictly favors the words in the input documents. [See et al., 2017]

Pointer Network is designed to solve the limitations of Seq2Seq model. [Vinyals et al., 2015] In figure 2.4a, we can see that the structure of Pointer Network is very similar to Seq2Seq Network with attention except that the output distribution is directly drawn from the input sequence. The output distribution of Pointer Network is calculated in almost the same way of attention vector in Seq2Seq Network: $P(\mathbf{y}_t = \mathbf{x}^e_i) = softmax(F_{ptr}(\mathbf{h}^e_i, \mathbf{h}^d_t))$. Instead of generating a sequence from a dictionary, Pointer Network directly copies the tokens from the input sequence.

The Pointer Network does not rely on a big dictionary, so its output layer has much fewer parameters than Seq2Seq Network. As a result, it is much faster to train and test a Pointer



(a) Pointer Network

Figure 2.4 – Pointer Network

Network.

2.7 Model Comparison and Analysis

In all, Seq2Seq Network and Pointer Network are two popular generative models based on LSTM. They both have advantages and disadvantages. Seq2Seq Network is capable of paraphrasing and replacement but is not good at handling out-of-vocabulary words. Pointer Network is the opposite, it solves the problems of Seq2Seq Network and runs faster, but it can not generate new words, let alone paraphrasing.

Actually, Seq2Seq Network and Pointer Networks are two very different models in spite of the similar structures. Pointer Network is an extractive model because all its outputs are directly copied from the input. By contrast, Seq2Seq Network is an abstractive model, it generates tokens from a dictionary instead of the input sequences.

In this project, we will use both Seq2Seq Network and Pointer Network to construct neural summarization models. In the following chapters, we will demonstrate that these two models are complementary. If we combine them together, we will have advantages of both and get rid of their disadvantages at the same time.

3 Neural Summarization Models

3.1 Overview

In this chapter, we will introduce many kinds of neural summarization models. They are classified into two categories: extractive models and abstractive models.

Extractive models in this project are all on the sentence level and many of them are regression or classification models. The regression models predict an important index for each sentence, while the classification models divide sentences into important and unimportant ones. When generating summaries, we sort the sentences by their importance index or probabilities of being important, we directly pick the top ranked sentences and concatenate them to form the summary by their order in the original documents.

In the extractive model part (section 3.2), we first show some naive binary classifiers which do not consider the relationship between sentences. Then, a Seq2Seq Network-based model will be introduced to improve performance. Finally, we demonstrate how sentence embeddings (see section 2.2) can be applied to select sentences directly.

In section 3.3, we illustrate some abstractive models. Firstly, we will show a straightforward application of Seq2Seq Network for abstractive model. Then we add some 'extractive elements', Pointer Network, into this model to ease the drawback of Seq2Seq Network. Afterwards, we construct a mixed model by incorporating the extractive model into the abstractive one. Finally, we incorporate sentence embedding in abstractive model's training.

3.2 Extractive Models

3.2.1 Some Naive Extractive Models

In this part, we will briefly introduce fully-connected neural network and convolutional neural networks for sentence classification. [Joulin et al., 2016, Kim, 2014]

For fully-connected neural network (noted as FCNetExt), we use 1-layer network followed by a softmax classifier to classify sentences. The input features of this network is the average word embeddings of all words in input sentence. This structure is the same as Facebook's fasttext tool. [Joulin et al., 2016] It is simple and fast. It can also be used to train word embeddings. [Bojanowski et al., 2016]

For convolutional network, we use the same model as the one which has achieved great success in semantic analysis. (noted as ConvNetExt, [Kim, 2014]) The structure of this model is shown in figure 3.1a. We use a matrix to describe input sentence by word embedding. Then several convolutional filters are applied to this matrix. One dimension of the filters is aligned with the dimension of word embeddings, while another one is various, representing different N-gram features to be captured. After convolutional operator, we obtain several feature vectors of different dimensions. Afterwards, global maximum pooling is applied to each of these vectors and all of them are compressed into a scalar. We concatenate all scalars to form one global feature vector and then feed it into a softmax classifier.

First, the dimension of word embeddings and filters must be aligned because the different dimensions of word embeddings are independent. It is meaningless to extract regional features in word embedding dimension. Actually, a dimension aligned filter is equivalent to a full connection. Second, in this model, we use filters of different sizes to capture different N-gram features. The global maximum pooling makes sure the output dimension does not depend on the filters' size. What's more, the feature vectors from convolution have some regions corresponding to the padding. The global maximum pooling also minimize the effect of paddings and makes the model work for sentences of different length.

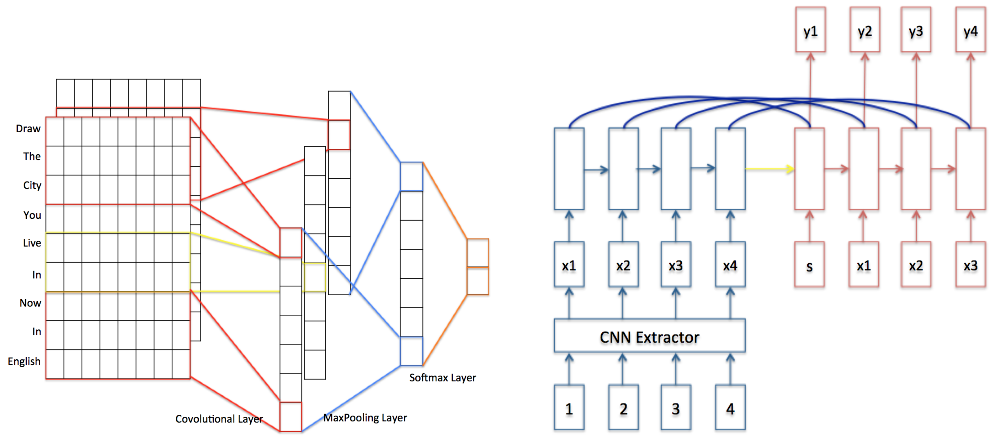
Compared with fully-connected neural network, convolutional network takes advantage of several N-gram features. However, both models are pure sentence-level classifiers and do not consider the relationship between sentences. This point can be solved by a Seq2Seq Network introduced in the following section.

3.2.2 Encoder-Decoder Extractive Model

The Encoder-Decoder Extractive Model (noted as EnDeExt Model) is shown in figure 3.1b and is based on the Seq2Seq Network (see section 2.4). [Cheng and Lapata, 2016] Both encoder and decoder of this model use uni-directional LSTM to learn the representation of the document or its prefix.

The input of the encoder in this model is a sequence of sentence embeddings $\{\mathbf{x}_i\}_{1\leq i\leq N}$. These sentence embeddings are obtained by a convolutional feature extractor, whose structure is the same as the convolutional model in section 3.2.1 but without a softmax classifier. The convolutional feature extractor directly use the global feature vector in ConvNetExt model as sentence embeddings.

The input of the decoder is the same as the encoder's except two points. First, they are



(a) Convolutional network for sentence classification

(b) Seq2Seq Model with attention to extract sentences

Figure 3.1 – Two extractive models

dislocated by one time stamp and begin with a special start token. Second, the decoder's input sentence embeddings are multiplied by their importance, which are the output of the previous timestamp. Mathematically, if the decoding timestamp is t, then the input is $\hat{y}_{t-1}\mathbf{x}_{t-1}$. Here, $\hat{y}_{t-1}$ is the predicted output at decoding time $t-1$, meaning the probability that $(t-1)$-th sentence is important. This mechanism can not only filter the information of unimportant sentence in the decoding sequence but also prevent the model from extracting redundant sentences.

EnDeExt Model also uses the Encoding Attention (see section 2.5) to reinforce the connection between the encoder and the decoder. The case here is much simpler because there is a very obvious relationship mapping between the encoding hidden states and decoding hidden states. For the i-th decoding hidden state, we directly use the i-th encoding hidden state as the context vector under the assumption that the attention is 1.0 for this state and 0.0 for the rest. We concatenate decoder hidden state with the context vector and feed them to a multilayer perceptron with a scalar as output. We apply sigmoid function to get the normalized output $\hat{y}_t \in (0,1)$ as the probability of current sentence's being important.

In the early steps of training phrase, the whole model is in a mess and the predicted sentence importance is not reliable at all. As a result, it is not a good idea to use the predicted importance $\hat{y}_{t-1}$ to calculate the decoder's next input. In this case, we rely on the ground truth label $y_{t-1} \in \{0,1\}$ instead to 'boot' the model. In practise, we use a coefficient driving from y_{t-1} to $\hat{y}_{t-1}$ by $\alpha(n)y_{t-1} + (1-\alpha(n))\hat{y}_{t-1}$, here $\alpha(n)$ is a variable decaying from 1.0 to 0.0 during the iteration n. In the later step of training phrase and in the test phrase, the model is fine-tuned or the ground truth label is not available. In that case, we directly use the prediction $\hat{y}_t$ to calculate the next input for the decoder.

Entity Name Highlight

Entity names, also known as factual names, are very important for summarization tasks, because it is usually the subjects or objects of the summarized sentences. However, entity names are often specific to a topic and few documents, so they are not frequent words. This phenomenon makes the entity names lack of highlight in the dictionary i.e word embedding matrix and not sufficiently trained.

To solve this problem, we can reverse part of the dictionary for entity names. That is to say, we sort the frequency of ordinary words and entity names in two different lists. We fill the most frequent words and entity names into the embedding matrix and the rest words are compressed into a [UNKNOWN] token.

In addition to increasing the proportion of entity names in the dictionary, we also append the word embeddings by an additional 'entity bit'. The 'entity bit' is 1 if the token is an entity name and 0 otherwise. Unlike the other part of word embeddings, the 'entity bit' is not trainable and not adjusted during training phrase.

3.2.3 Sentence Embedding Extractive Model

In addition to neural network based classification model, we can directly extract sentences according to their embeddings. By method described in section 2.2, we can obtain semantic representations of each sentence and the whole document. Then the sentences whose embeddings are most similar to the document embedding will be extracted. The similarity between embeddings is based on either cosine or Euclidean distance. We call this model Sentence Embedding Extractive Model (noted as SEExt).

Besides text summarization, similar methods are also applied for key phrase extraction tasks. [Wang et al., 2014] Sometimes, instead of directly choosing the K most similar sentences, we also calculate the similarity between extracted sentences. We try to minimize such similarity to prevent the extracted summaries from being redundant.

Mathematically, given document $\mathbf{d}$ consisting of N sentences $\{\mathbf{s}_i\}_{1\le i\le N}$ and similarity function sim, we try to optimize the selection vector $\{\gamma_i\}_{1\le i\le N}$, which means whether or not the corresponding sentence is selected, to maximize the gap between extracted sentences' similarity with the whole document and their internal similarity. So it can be formalized as a constrained 0-1 programming problem.

$$\begin{aligned}
& max_{\gamma} \sum_{i=1}^{N} \gamma_i sim(\mathbf{d}, \mathbf{s}_i) - \lambda \sum_{i,j=1;i\neq j}^{N} \gamma_i \gamma_j sim(\mathbf{s}_i, \mathbf{s}_j) \\
& s.t. \sum_{i=1}^{N} \gamma_i = K; \gamma_i \in \{0,1\}, i = 1,2,...,N.
\end{aligned}$$

In this formula, λ is a combination hyper-parameter and K is the number of sentences to be selected. Traditionally, we can use branch and bound algorithm to solve such 0-1 programming problem. In addition, Gibbs sampling can also be applied to find an approximately optimal solution. Gibbs sampling is easy to implement but needs many iterations to converge.

In this project, for simplicity, we use the naive SEExt Model i.e $\lambda = 0$. Sentences are chosen only based on its similarity to the whole document.

3.3 Abstractive Models

3.3.1 Encoder-Decoder Abstractive Model

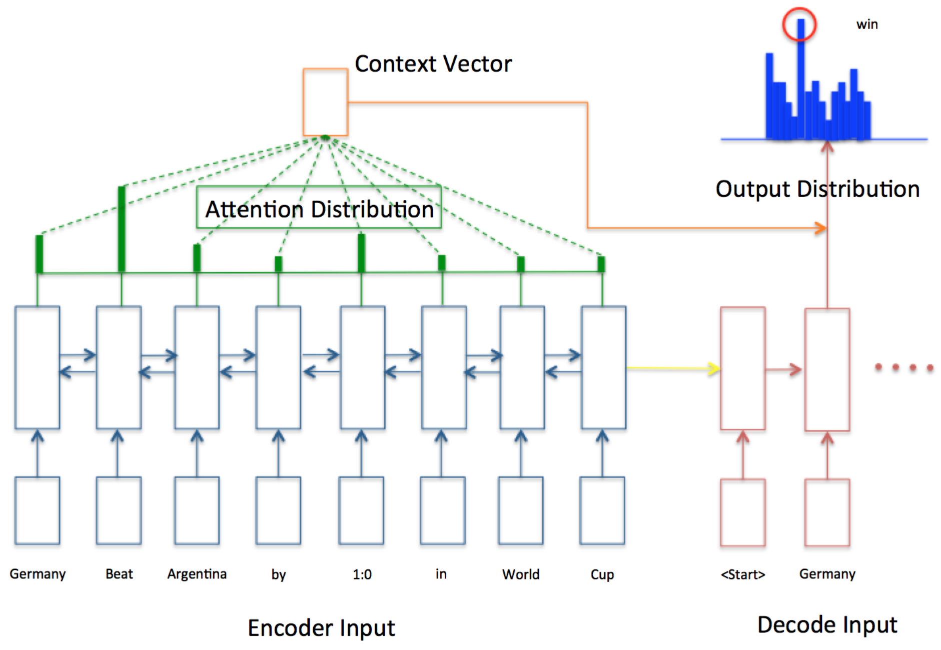
As figure 3.2a shows, Encoder-Decoder Abstractive Model (noted as EnDeAbs, [Nallapati et al., 2016]) is essentially a Seq2Seq Network. Unlike extractive model (figure 3.1b), we input the original documents word by word and the model also outputs summaries word by word according to a distribution over a pre-defined dictionary. Attention Mechanism is a necessary for EnDeAbs Model because the encoding and decoding sequences are much longer than EnDeExt Model.

Compared with extractive models, abstractive models are able to do word-level compression. On one hand, this makes the model capable of more flexible and powerful functions such as paraphrasing. On the other hand, such flexibility also brings risks of inaccuracy in both grammatical and semantic respects. For example, EnDeAbs Model is able to generate word that does not exist in the input documents, but the generated word is not always right since it is picked from a very large vocabulary. A wrong word, especially a wrong factual word, will lead to a expression of totally different meaning. In addition, EnDeAbs Model has a much longer decoding sequence than extractive models, which makes it much possible to decode duplicate sequences. It is because the long decoding sequence leads to the very weak dependencies between distant hidden states. Only Decode Attention introduced in section 2.5 can help to ease this problem, since it reinforces the connections between decoder's hidden states.

3.3.2 Pointer-Generator Network

In section 3.3.1, we demonstrate the two main problems of pure Seq2Seq Network-based abstractive model: wrong factual names and repeated sequences. This part, we will introduce the Pointer-Generator Network and some tricks to solve both problems. [See et al., 2017]

Wrong factual names usually arise from model's picking up a word which does not appear in the original text. This is because EnDeAbs Model uses a global and fixed dictionary for word generation. However, vocabularies used in documents of different topics are rather different. In some way, EnDeAbs Model is too free to generate words. We should constrain new word generation and favor the words that appear in the original document. In section 2.6, we know that Pointer Network can generate a sequence of tokens from the input sequence, which is a good feature to solve this problem.



(a) Encoder-Decoder Abstractive Model with Attention

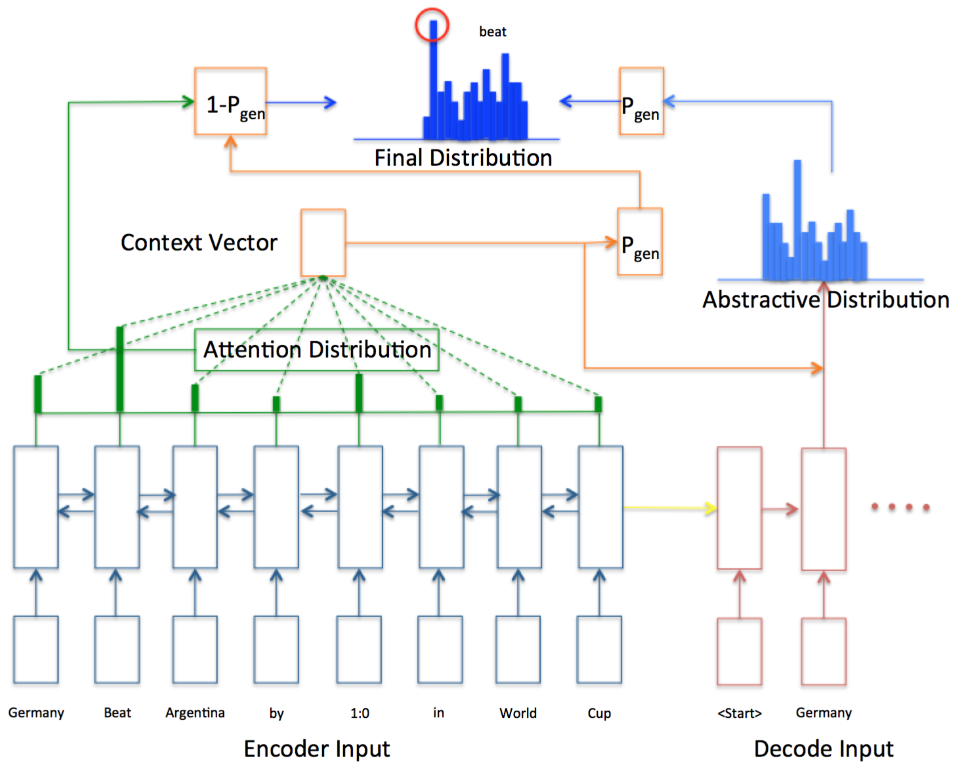
Figure 3.2 – Encoder-Decoder Abstractive Model with Attention

In figure 3.3a, we combine Seq2Seq Network together with Pointer Nework to contruct Pointer-Generator Network (noted as PGNet). The output distribution of PGNet is a combination of Seq2Seq Network and Pointer Network. That is to say the final distribution is the sum of one over a global, fixed dictionary and one over the words in the input document: $P_{final} = p_{gen}P_{seq2seq} + (1 - p_{gen})P_{pointer}$. The combination coefficient p_{gen}, which indicates how willing the model is to generate new words, is determined by context vector $\mathbf{cv}_t$, current decoding hidden state $\mathbf{h}_t^d$ and decoder's input word embedding $\mathbf{x}_t^d$: $p_{gen_t} = \sigma(\mathbf{v}_c^T\mathbf{cv}_t + \mathbf{v}_h^T\mathbf{h}_t^d + \mathbf{v}_x^T\mathbf{x}_t^d + b_{gen})$.

Unlike pure abstractive models, PGNet is a combination of extractive and abstractive model. In detail, it combines the advantages of Seq2Seq Network and Pointer Network. The Seq2Seq Network part makes it capable of paraphrasing and word-level compression. While the Pointer Network part forces the output summary to be similar to the input document literally.

PGNet solves the problem of wrong factual names. In order to avoid the redundant sequences, Coverage Vector and Intra-Attention is introduced below. Both tricks can be incorporated into the PGNet.

The intuition behind these two methods are roughly the same. In section 2.5, we know that the normalized encoding attention vector is a good indicator of decoded word's coverage over the input document. An effective way to avoid redundant sequences is to prevent duplicated highlight in attention vector. Coverage Vector [See et al., 2017] manages to do this by adding a



(a) Pointer-Generator Network

Figure 3.3 – Pointer-Generator Network

punishment term in the loss function, while Intra-Attention [Paulus et al., 2017] uses a history normalization term in the calculation of attention vector.

Coverage Vector

In the decoding phrase, we use a M-dimensional coverage vector $\mathbf{c}$ to represent the generated sequence's coverage over input document. It is element-wise sum of all previous attention vector: $\mathbf{c}_{it} = \sum_{k=0}^{t-1} \alpha_{ik}$. Since the coverage of current decoded word is α_{it}, the attention overlap of current word and previous words is $min(\alpha_{it}, \mathbf{c}_{it})$ for the i-th word in the original document.

To prevent the repeated sequence, we add the attention overlap as the punishment term in the loss function, multiplied by a hyper-parameter λ: $\lambda \sum_{t=1}^{N} \sum_{i=1}^{M} min(\alpha_{it}, \mathbf{c}_{it})$.

Intra-Attention

The consideration behind Intra-Attention is similar to Adagrad or Adadelta optimizer [Zeiler, 2012], which normalize the gradients according to their histories to update the trainable parameters. Intra-Attention records the element-wise natural exponent of previous attention vectors as the history vector $\mathbf{hist}_{it} = \sum_{k=1}^{t-1} exp(\mathbf{a}_{ik})$. Then the history vector is used to normalize the attention: $\mathbf{a}'_{it} = \frac{\mathbf{a}_{it}}{\mathbf{hist}_{it}}$ and $\alpha_{it} = \frac{exp(\mathbf{a}'_{it})}{\sum_{j=1}^{M} exp(\mathbf{a}'_{jt})}$. Entries of the attention vector which has peak values in the past will be punished in the future attention calculation.

From the formula above, we can see that the history-based normalization term always keeps increasing, which usually strongly discourages even the single words from reappearing. Actually, reappearing words are quite common in the summaries. Similar to the modification from Adagrad to Adadelta, we can introduce another hyper-parameter γ to the history vector update formula $\mathbf{hist}_{i(t+1)} = (1-\gamma)\mathbf{hist}_{it} + \gamma exp(\mathbf{a}_{it})$. Same as the parameter in Adadelta, γ is the forgetting rate.

3.3.3 Mixed Model

In PGNet, if a document's length is bigger than the model's encoding length, the model will automatically cut the later tokens, in which there might be some useful information. For news, editors tend to put the most important information in the first paragraph as well as the first sentence of each following paragraph. Usually from the second paragraph, there might be a big chunk of useless information and it is a waste of encoding iteration.

Mixed Model is designed to solve this problem, it is an abstractive model (PGNet) followed by an extractive model (EnDeExt). The extractive model first picks up several sentences and concatenates them by their order in the original document. Then the concatenated sentences are fed into an abstractive model to generate summaries.

Mixed Model is a typical two-step model and consists of two independent components. The extractive model produces a semantically dense sequence to improve the quality of the abstractive model's input and narrows down the gap between original document and summary desired.

The Mixed Model can also be considered as the reverse processing of how editors write news. Most probably, the editors list the outline of the news and put these contents in the first few sentences and the first sentence of each paragraph. Then they expand each paragraph by putting more details after the first sentence. On the contrary, Mixed Model first removes the unimportant sentences by extractive part. Then the key sentences are further compressed by abstractive part.

In Mixed Model, the connection between extractive and abstractive part is a top-K selection and concatenation operator, which is not differentiable. In addition, we want to take advantage of the extractive label for sentences to better train the extractive part. Therefore, for mixed model, I do not train it in an end-to-end way. I train extractive and abstractive part separately, then combine them together in the decoding phrase.

3.3.4 Abstractive Model with Sent2Vec-based Loss

In section 2.2, we introduce a Sent2Vec method to map sentences or documents to vectors with semantic meanings. In summarization task, we hope our generated summaries reflect the contents of the original documents to the biggest extent. So we hope the embeddings of the original documents and summaries should be as close as possible.

In this section, We introduce an abstractive model whose loss is the distance between the embeddings of input and output sequences. The rest parts are the same as PGNet. We expect this structure is able to generator qualified summaries. On one hand, the limited decoding steps constrain the length of generated sequences. On the other hand, the meaning of output summaries are driven to the original documents in a semantic space by minimizing their embedding distances.

Unlike models in the previous sections, the model with pure Sent2Vec-based Loss is totally unsupervised. It does not need any gold summaries as supervised signals, but instead model is adjusted by the similarity between input and output sequences. This is a cool property, but relevant research and experiments show that the output sequence generated by this model is not readable although it indeed contains some key words. [Paulus et al., 2017] This is because sentence embeddings are mainly based on CBOW and CBOW does not care about the order of words. The situation is no better even if we use Sent2Vec introduced in 2.2 to take N-gram into consideration, because Sent2Vec is still not able to capture relations between distant words. As a compromise, the loss of the final model (noted as Sent2VecAbs) is a linear combination of language model loss and Sent2Vec loss by a hyper-parameter λ: $Loss = \lambda Loss_{languagemodel} + (1-\lambda) Loss_{Sent2Vec}$

For input sequences, we use the embedded matrix to feed Sent2Vec model. For output sequences, PGNet gives a distribution $\{\mathbf{dist}_i\}_{1\le i\le V}$ over a fixed dictionary $\{\mathbf{w}_i\}_{1\le i\le V}$. Instead of using the embedding of most probable word, the embedding matirx to feed the Sent2Vec model is the weighted combination of embeddings of all words in the dictionary: $\sum_{i=1}^{V} \mathbf{dist}_i \mathbf{w}_i$. The advantages of this is to make the whole network differentiable and prevent the gradient vanishing problems.

4 Experiments

In this chapter, we will show all the experiments we have done in this project. We will first introduce and analyze the dataset we use in section 4.1. Brief information about implementation and the open source softwares we use are discussed in section 4.2. In section 4.3, we will discuss the metric for evaluation of summaries, since text summarization task is very subjective. Finally, in section 4.4 and 4.5, we will compare the performance of different extractive and abstractive models from different aspects. We will give some explanation after each experiment.

All the codes in this project are in python and all neural network implementations are in Tensorflow. [Abadi et al., 2016] We will publish all the codes on github https://github.com/liuchen11.

In addition, several summarization model demos are available on https://aidemo.mcp.ai/summary, powered by Swisscom server. Users only need to type in a URL, the backend server will parse the URL and show the original article and the proposed summary on the webpage in just a few seconds.

4.1 Dataset

4.1.1 Overview

All our experiments are based on Dailymail dataset. [Chen et al., 2016] It consists of 193986 training documents, 12147 validation documents and 10350 testing documents. In each document, there are 119 sentences at most.

The documents in dataset is well-formatted and contains four parts: 1) source(URL), 2) sentences and corresponding labels, 3) gold summary, 4) entity name list. Entity names are detected in the original documents. All entity names are anonymized and replaced with tokens like '@entity1'. The purpose of this design is to encourage the model to focus more on the structural features instead of the details of entity names. In our experiments, we do not use the anonymous entity name but use the original words directly except when we use entity highlight.

The labels of sentences are used in extractive models to learn a classifier. There are three kinds of labels for sentrences: 0 means this sentence is unimportant for summaries, while 1 means important and 2 means partially important. In practise, we treat label 2 same as label 0 to remove the ambiguous. For each document, a gold summary is provided only for the training of abstractive models.

Word embeddings of high quality are very important for us to train a fine-tuned model. In this project, we use pre-trained word embeddings from *Glove*, which is trained from Wikipedia 2014 and gigawords 5. [Pennington et al., 2014] It has a vocabulary of 40000 with word embeddings of different dimensions. In experiments, we use the 100-dimensional embeddings. For Sent2Vec model (in section 2.2), it is also trained based on Wikipedia corpus and outputs a 750 dimensional vector given a word or a sequence of words.

4.1.2 Dataset Analysis

The Dailymail has more than 200000 documents and there might be some 'wrong cases' inside, so it is very necessary to look into some statistical properties of this dataset.

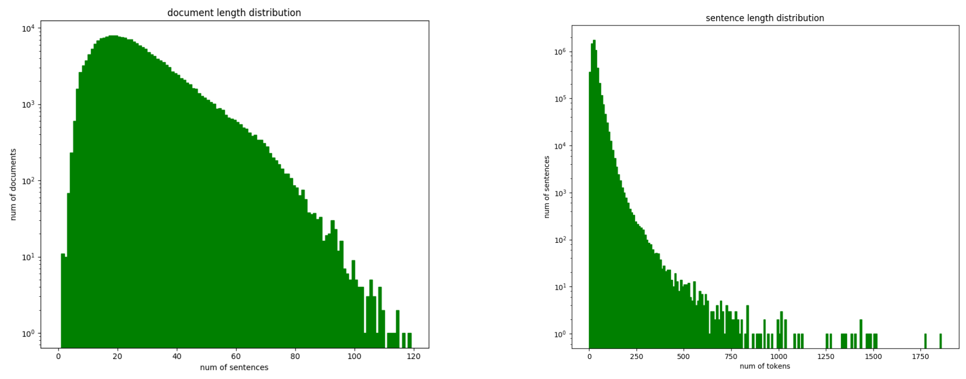
The whole dataset has a vocabulary of more than 211k words, only 20% of which have a minimum frequency of 10. There are much more enity names, 860k, than ordinary words. The frequency of entity names are lower than ordinary words on the whole. Less than 10% of all entity names have a frequency over 10.

Extractive Features

In figure 4.1a and figure 4.1b, we plot the distribution of documents' length in sentences and sentences' length in tokens. It is clear that the length of both documents and sentences are very variant. It is estimated that less than 1% of all documents which have more than 60 sentences and less then 1% of all sentences which have more than 100 tokens. As a result, we set 60 and 100 as the maximum length for documents and sentences in extractive models. For ones that exceed this threshold, we use random clipping to pick a segmentation to feed the model.

Figure 4.2a and figure 4.2b show the distribution of labels regarding the position and length of sentences.

In figure 4.2a, it is clear that the first few sentences have much higher possibility to be marked as important than the later sentences. It is reasonable because the newspaper editors tend to put the most important information in the first to catch readers' attention. It is also why some very naive and intuitive methods like simply extracting the first 3 sentences have competitive performance.



(a) The distribution of documents' length in sentences

(b) The distribution of sentences' length in tokens

Figure 4.1 – Analysis about the length of documents and sentences

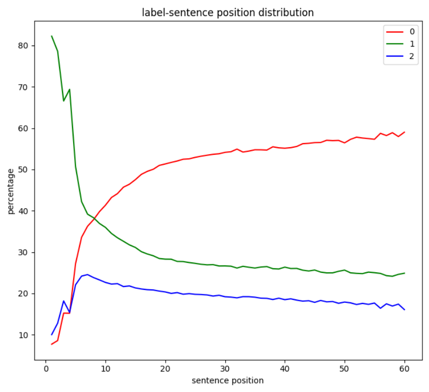
Figure 4.2b demonstrates the relationship between the label and the length of sentences. In our dataset, longer the sentence is, more probable it is to be marked as important. This is partly because longer sentences contain more information which can better make the reader know the content of the whole documents. However, training an extractive model based on data labeled this way will drive the model far away from what we expect. We do not want the selected sentences in the output summary to be too long. In these cases, compression inside the sentences becomes extremely necessary. This can in some way explain the main disadvantages of extractive models and why constructing abstractive models is necessary.

Abstractive Features

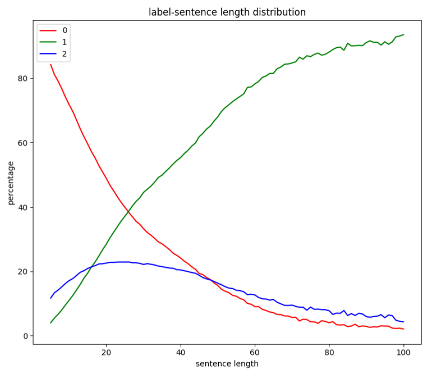
Same as extractive model, encoding and decoding length is one of the most important hyper-parameters for abstractive models. In figure 4.3a and 4.3b, we show the distribution of original documents' length and summaries' length in tokens respectively.

The length of encoding steps is a trade-off between the coverage of important information and training efficiency. A long encoding sequence means more iterations and paddings, meaning longer training and test time, but a short encoding sequence will cause the information loss because of input sequence clipping. In figure 4.3a, we not only show the document length distribution of input documents but also plot the curve of information coverage. We mark the token as important token if it is in a sentence whose extractive label is 1 i.e important. The point (x, y) on the curve means the percentage of important tokens in the first x tokens of each document over all important tokens is y. Finally, we set the encoding length to be 400 because over 60% important tokens are within the first 400 tokens in the documents. Larger encoding length will lead to a lower training efficiency.

For the decoding steps, things are simpler. As figure 4.3b shows, over 90% of gold summaries



(a) Proportion of all labels over the position of sentences



(b) Proportion of all labels over the length of sentences

Figure 4.2 – Analysis about extractive labels' disgtribution

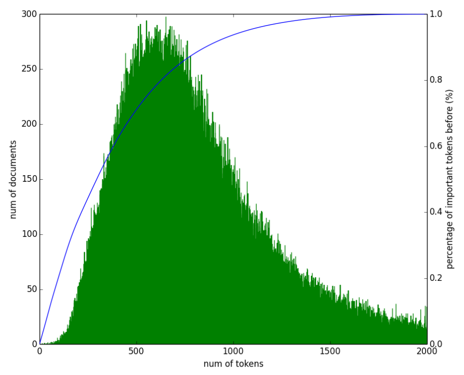
have less than 100 tokens, so we set decoding length to be 100. We have a special [STOP] token in the dictionary, which means the termination of generated sequence, so the summaries we generated by an abstractive model have 100 or less tokens.

4.2 Implementation and External Open Source Software

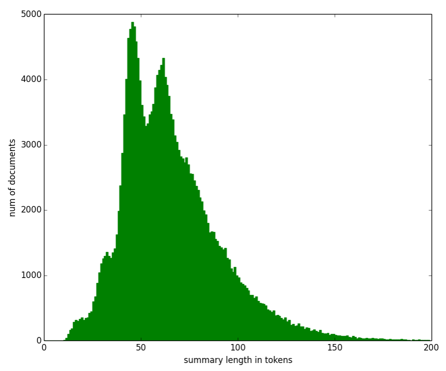
All codes in this project are in Python and will be available on github. All deep neural network models are implemented in Tensorflow. All running time report or running speed comparison, unless specified, in the following sections are based on experiments on a virtual machine with 12 cores, 64 GB main memory and one NVIDIA Tesla P100 GPU. There are two external open source softwares we rely on for this project. The rest codes are all written by the writer independently.

The first external software we use is Sent2Vec library developed by EPFL MLO lab (codes available on https://github.com/epfml/sent2vec). This library is based on Facebook's fasttext library (codes available on https://github.com/facebookresearch/fastText) and implements the Sent2Vec model introduced in section 2.2. In experiment part, it is only used to analyze some statistical features such as results in section 4.1.2 and implements SEExt Model in section 3.2.3. However, for Abstractive Model with Sent2Vec Loss (Sent2VecAbs model) in section 3.3.4, we do not use this library due to compatibility issues. In that part, we use CNN feature extractor introduced in section 3.2.2 instead.

The other external source code software is PG Net framework implemented by Abi See et al (codes available on https://github.com/abisee/pointer-generator). This repository contains a well-written implementation of PG Net which does not consume a lot of memory. We use this library to construct all abstractive model introduced in section 3.3. The framework



(a) Distribution of document length in tokens (left axis); Proportion of important tokens (right axis).



(b) Distribution of summary length in tokens.

Figure 4.3 – Analysis about the length of documents and summaries.

has an excellent preprocessing module which can handle out-of-vocabulary words very well by extending vocabulary dictionary temporarily. It also contains Encoding Attention and Coverage Vector but Decoding Attention and Intra-Attention is not implemented in this framework.

By comparison, we write another version of PGNet by ourselves in tensorflow. We implement all the tricks decribe above except that the preprocessing module is as simple as extractive models. It turns out that this implementation runs at least three times faster that the PG Net library but suffers from out-of-vobulary words. PG Net of this version uses a fixed dictionary, so it usually outputs an [UNK] token most time when facing an out-of-vocabulary word. It losses a lot of information to project all out-of-vocabulary words into a single [UNK] token and will affect the following decoding states as well.

Experiments show that results of the PG Net framework is better than our self-written codes. So for most abstractive introduced in section 3.3, we implement them based on the PG Net library.

4.3 Evaluation Metrics

Summarization is a very subjective task. Different people may have different views on what are key points of one document and produce different summaries. Even for ones that have similar ideas, it is highly possible that they write very different summaries literally. As a result, it is not an easy job to find an objective metric to measure how good a summary is. So far, researchers mainly rely on some statistical metrics such as ROUGE score to evaluate the generated summaries.

For extractive models, things are much simpler, because it is much less ambiguous to tell if or not a sentence is informative for summary. We have very reliable supervision to train our supervised models. For classification model, we evaluate model's accuracy in test set. For models that select some sentences out of a document, we evaluate the proportion of selected sentences that are marked as 'important'.

For abstractive models, the most popular evaluation metric are ROUGE-N and ROUGE-L scores [Lin, 2004] between the generated summary and 'gold summary', which is written by human. ROUGE-N and ROUGE-L is defined below:

$$
\begin{aligned}
ROUGE-N &= \frac{\sum_{S\in S_H}\sum_{g_N\in S} C_m(g_N)}{\sum_{S\in S_H}\sum_{g_N\in S} C(g_N)} \\
ROUGE-L &= \frac{1}{\|S_H\|}\sum_{S\in S_H}\frac{\#\text{LCS}}{\#\text{gold\_summary}}
\end{aligned}
$$

In the first equation, S_H, S mean the gold summary set and an individual gold summary. g_N, $C(g_N)$ and $C_m(g_N)$) mean N-gram tokens, number of N-grams in gold summary and number of co-appearing N-grams in gold summary and generated summary. In the second equation, ROUGE-L score is the proportion of longest common sequence (LCS) over the length of gold summary.

ROUGE is a statical, deterministic and easy-to-calculate metric. In this project, we still mainly use ROUGE score for evaluation of abstractive models. However, the drawback of ROUGE score is very obvious. As said before, the gold summary is not always the best nor unique. At least, paraphrasing or synonym replacement can be applied to make the alternated summary semantically almost the same while statistically quite different. Considering the drawbacks of ROUGE score, some previous research works also use human to evaluate generated summaries, for example scoring the generated summaries from 1 to 10 and then calculate the average scores by different humans. This method is more reliable but very time consuming and not feasible for big dataset.

Basically, summarization is a task to compress the document while preserving its major meaning. In section 2.2, we introduce Sent2Vec to map sentences or documents to vectors. The Euclidean distance or the cosine value between two vectors are two good indicators of their semantic similarity. In this way, we can compare the embedding-based similarity of the original document and generated summary. This intuition is confirmed by dataset analysis. Given original document embeddings $\{\mathbf{d}_i\}_{1\le i\le N}$ and summary embeddings $\{\mathbf{s}_i\}_{1\le i\le N}$ from Sent2Vec model, the average cosine value $\frac{1}{N}\sum_{i=1}^{N}\frac{<\mathbf{d}_i,\mathbf{s}_i>}{\|\mathbf{d}_i\|\|\mathbf{s}_i\|}$ in the whole dataset is 0.73. This value will drop more than 0.1 even if we only random shuffle the tokens of the summaries.

In section 3.3.4, we directly incorporate the embedding similarity into the training of abstractive model. For other models, we can also use this metric for evaluation.

4.4 Extractive Summarization Models

In this section, we will give more details about the implementation for extractive models in section 3.2 and compare the performance of them.

Among four models, FCNetExt, ConvNetExt and EnDeExt are classification models. We evaluate them according to their accuracy on test set. They are also able to sort the sentences by their predicted probability of being informative. So we also calculate the proportion of 'true important' sentences out of the top-K ranked ones selected by models. For SEExt model, which can only sort sentences according to their relative importance, we only evaluate it in the second way.

The word embedding of the first three models are all of 100 dimensions and trainable. They are either randomly intialized or loaded from pretrained source. We always take advantage of Adam optimizer [Kingma and Ba, 2014] to update the parameters in the neural network. It is a easy-to-compute first-order optimizer with adaptable step size. It handles the problem of local optima and is invariant to the scaling of gradients' different dimensions.

FCNetExt model directly takes the average of the word embeddings in the sentence and feed it directly to the softmax classifier. For ConvNetExt, we use filters of size from 1 to 7 and each of these filters have 300 feature maps in the convolutional layer. (See figure 3.1a) After global max-pooling, a 300 dimensional feature vector is obtained and is fed to the softmax classifier. EnDeExt model has the same kind of feature extractor as ConvNetExt. (See figure 3.1b) The 300 dimensional feature vector of each sentence is fed to a LSTM cell as a sequence. The LSTM cell has 750 dimensional hidden states in both encoding and decoding phrase. In the decoding phrase, the 750 dimensional hidden states will be fed to a 3-layer fully connected network, with 300, 60 and 1 neuron in each layer. Sigmoid function is applied to the output to get the final prediction.

Table 4.1 shows the test set accuracy of different models and different configurations. All models are trained for 10 epochs. It is clear that the pretrained word embeddings and entity highlight are benificial to help the models improve performance. Regarding model structures, EnDeExt significantly outperforms the rest models, although it takes much more time to train(12 hours v.s less than 1 hour). This result is consistent with our expectation, because EnDeExt is the only model that take the relationship between sentences into consideration.

Table 4.2 confirms the superiority of EnDeExt model. It shows the proportion of important sentences out of selected sentences of each model. In addition to EnDeExt Model, another highlight in this table is that SEExt model works better than FCNetExt and has approximately the same performance as ConvNetExt which has a much more complicated structure. In section 2.2, we know the sentence embeddings of SEExt is obtained by 1-layer fully connected neural network, which is similar to FCNetExt. The main different between FCNetExt and SEExt is that the later takes advantage of the global features i.e embeddings of the whole document. The performance of these two models shows that global features such as ones from the original

Type	Embedding	Entity Highlight	Accuracy (%)
FCNetExt	Random	No	72.4
FCNetExt	Pretrain	No	72.9
ConvNetExt	Random	No	74.3
ConvNetExt	Pretrain	No	74.5
ConvNetExt	Random	Yes	74.4
ConvNetExt	Pretrain	Yes	74.7
EnDeExt	Random	No	79.0
EnDeExt	Pretrain	No	80.1
EnDeExt	Random	Yes	79.8
EnDeExt	Random	Yes	80.3

Table 4.1 – Prediction Accuracy on Test Set

Model	FCNetExt	ConvNetExt	EnDeExt	SEExt
Top1 (%)	83.0	90.8	95.3	90.7
Top2 (%)	80.8	87.0	92.8	86.2
Top3 (%)	78.0	83.6	89.8	81.6
Top5 (%)	72.4	76.8	82.7	74.1
Top10 (%)	60.5	62.8	66.2	60.1

Table 4.2 – Proportion of true cases among selected sentences.

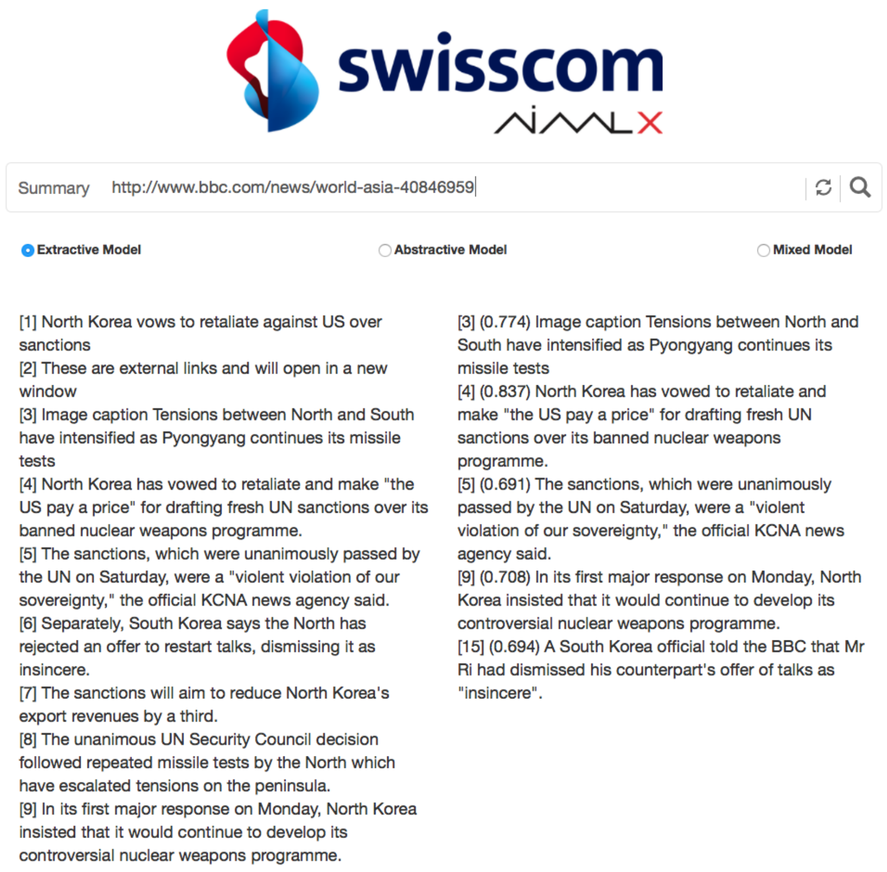
documents can help to boost the performance of extractive models.

In conclusion, EnDeExt Model with entity name highlight and pretrain word embeddings outperforms the rest models, so we use it as the online extractive model available on https://aidemo.mcp.ai/summary. Like figure 4.4, the webpage shows the original document in the left column and extracted summary in the right. The extractive sentences are sorted according to their order in text and the importance index is also shown in front of the sentence. So far, this online tool only support English news. For most cases, its response time after a query is no more than 5 seconds.

4.5 Abstractive Summarization Models

In this section, we will show the implementation details of abstractive models and compare their performance. Generally speaking, the strcuture of abstractive models is much more complicated than extractive models. Firstly, they have much longer encoding and decoding sequence. Secondly, they have much larger output dimension, which is a distribution over a large dictionary instead of one label out of two or a importance index. Finally, in the decoding phrase, we need to search for the optimal sequence, which is also time consuming.

As a result, training a fine-tuned abstractive model usually takes up to 3 weeks. Because of time limits, we only fine-tune the EnDeAbs, PG Net and Mixed Model. We have also tried



http://www.bbc.com/news/world-asia-40846959

[1] North Korea vows to retaliate against US over sanctions
[2] These are external links and will open in a new window
[3] Image caption Tensions between North and South have intensified as Pyongyang continues its missile tests
[4] North Korea has vowed to retaliate and make "the US pay a price" for drafting fresh UN sanctions over its banned nuclear weapons programme.
[5] The sanctions, which were unanimously passed by the UN on Saturday, were a "violent violation of our sovereignty," the official KCNA news agency said.
[6] Separately, South Korea says the North has rejected an offer to restart talks, dismissing it as insincere.
[7] The sanctions will aim to reduce North Korea's export revenues by a third.
[8] The unanimous UN Security Council decision followed repeated missile tests by the North which have escalated tensions on the peninsula.
[9] In its first major response on Monday, North Korea insisted that it would continue to develop its controversial nuclear weapons programme.

[3] (0.774) Image caption Tensions between North and South have intensified as Pyongyang continues its missile tests
[4] (0.837) North Korea has vowed to retaliate and make "the US pay a price" for drafting fresh UN sanctions over its banned nuclear weapons programme.
[5] (0.691) The sanctions, which were unanimously passed by the UN on Saturday, were a "violent violation of our sovereignty," the official KCNA news agency said.
[9] (0.708) In its first major response on Monday, North Korea insisted that it would continue to develop its controversial nuclear weapons programme.
[15] (0.694) A South Korea official told the BBC that Mr Ri had dismissed his counterpart's offer of talks as "insincere".

Figure 4.4 – A snapshot of online extractive model demo

many hyper-parameters for them, each of the model and configuration has been trained for at least 10 epochs. For Sent2VecAbs model, we have no time to fine-tune this model, we have only trained it for 2 epochs. When we compare this model with the others, we also use the corresponding model which is trained for only 2 epochs.

EnDeAbs Model, PGNet Model and Mixed Model

The structure of these three models are progressive. EnDeAbs Model is the baseline; PG Net adds Pointer Network to calculate a hybrid output distribution; Mixed Network adds an extractive model to compress the original document first before generating summaries. The encoder-decoder structure is the core part of all three models. In experiment of this section, we set encoding and decoding steps to be 400 and 100 respectively. LSTM cell of both encoder and decoder have 256-dimensional hidden states. Word embeddings are 128-dimensional vector randomly initialized for these three models.

There are 6 special tokens in the dictionary. They are start and end token of the document, start and end token of the sentence, token representing unknown words and token representing

Name	ROUGE-1 (%)	ROUGE-2 (%)	ROUGE-L (%)
EnDeAbs	16.0	1.1	12.4
PG Net	31.8	9.8	18.9
Mix-5	31.2	9.6	18.5
Mix-10	31.4	9.7	18.6
Mix-20	31.4	9.6	18.7

Table 4.3 – Rouge score comparison among EnDeAbs, PG Net and Mixed Model

paddings. In the decoding phrase, when the model decodes an 'end of document' token, the model will ignore the following decoded tokens. To prevent the summaries from being too short, we drop the summaries which have less than 35 tokens in the beam search process.

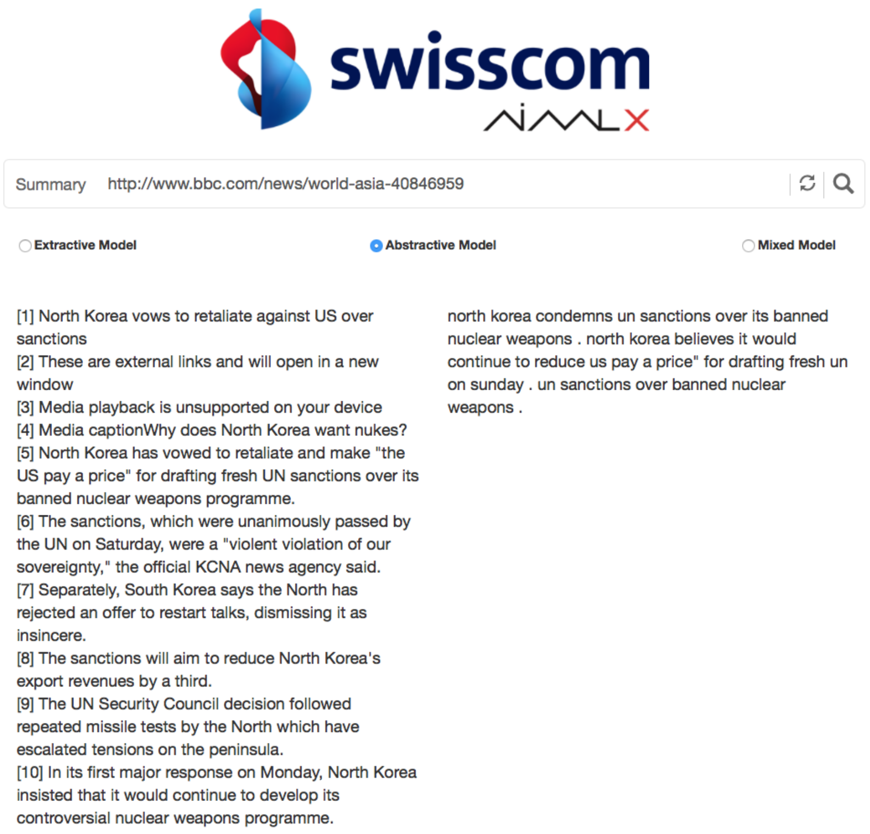
We trained the PG Net and Mixed Model for about 20 epochs. For baseline Model, which converges faster, we trained it for 10 epochs. The table 4.3 shows the ROUGE-1, ROUGE-2 and ROUGE-L score of these three models. Mix-K means the extractive part of Mixed Model selects top-K sentences out of original documents and feed them into the abstractive part.

From table 4.3, it is clear that PG Network outperforms baseline on all metrics by a big margin. However, Mixed Model does not achieve better performance than PG Net, no matter how many sentences the extractive part selects.

It is easy to explain PG Net's superiority over baseline EnDeAbs Model, since PG Net incorporates Pointer Network to solve the wrong factual name problem of Seq2Seq model. In addition, we find that over 80% unigrams and nearly 40% bigrams in gold summaries have exact the same expressions in the original documents. This indicates that actually the gold summaries in the dataset seldom use paraphrase and word replacement. Meanwhile, the Pointer Network part of PGNet makes the decoder prefer the ones appearing in the original document. This is why the Pointer Network has increased ROUGE scores significantly compared with baseline.

However, we do not expect Mixed Model to be defeated by PG Net, because the structure of Mixed Model is an extension of PG Network. One possible explanation is that the two parts of Mixed Model are trained separately. The abstractive part is trained on the whole original document while it is fed selected sentences in the decoding phrase. Such inconsistence between training and decoding phrases might lead to the failure of Mixed Model compared with PG Network.

In figure 4.5, we show the online interface of abstractive model and use the same input query as what figure 4.4 shows. This figure shows the result of PG Net, a interface of Mixed Model is also provided by changing the select box. From this example, we can see that the first sentence of the decoded summary is perfect, while the following sentences begin to go out-of-shape. Although the later sentences can tell us something but suffer from problems like grammar incorrectness and sentence incompleteness. This phenomenom is quite common in the test set, which indicates that the model is still in lack of the ability to capture long time



http://www.bbc.com/news/world-asia-40846959

( ) Extractive Model [✓ Abstractive Model] ( ) Mixed Model

[1] North Korea vows to retaliate against US over sanctions
[2] These are external links and will open in a new window
[3] Media playback is unsupported on your device
[4] Media captionWhy does North Korea want nukes?
[5] North Korea has vowed to retaliate and make "the US pay a price" for drafting fresh UN sanctions over its banned nuclear weapons programme.
[6] The sanctions, which were unanimously passed by the UN on Saturday, were a "violent violation of our sovereignty," the official KCNA news agency said.
[7] Separately, South Korea says the North has rejected an offer to restart talks, dismissing it as insincere.
[8] The sanctions will aim to reduce North Korea's export revenues by a third.
[9] The UN Security Council decision followed repeated missile tests by the North which have escalated tensions on the peninsula.
[10] In its first major response on Monday, North Korea insisted that it would continue to develop its controversial nuclear weapons programme.

north korea condemns un sanctions over its banned nuclear weapons . north korea believes it would continue to reduce us pay a price" for drafting fresh un on sunday . un sanctions over banned nuclear weapons .

Figure 4.5 – A snapshot of online abstractive model demo

dependencies.

Sent2VecAbs Model

In table 4.4, we compare the performance of EnDeAbs, PGNet and Sent2VecAbs. Due to time limits, we only compare these models trained after 2 epochs. Here, the Sent2VecAbs model are trained based on the combination of traditional language model loss and the cosine similarity of embeddings between the input and output sequences. We not only calculate the different ROUGE scores, but also the cosine value and distance between embeddings from decoded sequences and original documents (Cos-Document, Dist-Document), as well as ones from decoded sequences and gold summaries (Cos-Reference, Dist-Reference).

It is obvious that EnDeAbs Model performs much poorer than the other two models, because EnDeAbs Model predicts words in a big dictionary without highlight, it is quite common for it to generate the most frequent words. Compare the summaries generated by three models, we find that EnDeAbs Models generates much more [UNK] tokens than the other two.

Another interesting finding is that PG Net performs better than Sent2VecAbs model on ROUGE

Name	EnDeAbs	PGNet	Sent2VecAbs
ROUGE-1 (%)	11.8	29.7	27.9
ROUGE-2 (%)	0.4	8.5	6.7
ROUGE-L (%)	8.6	18.2	16.5
Dist-Document	8.61	4.45	4.50
Cos-Document	0.36	0.99	0.98
Dist-Reference	6.75	2.88	2.97
Cos-Reference	0.36	0.99	0.98

Table 4.4 – Rouge score and embedding similarity comparison among EnDeAbs, PGNet and Sent2VecAbs models

scores. On embedding-based metrics, these two models have similar performance. It seems that the PG Net has been trained implicitly to generate sequences whose embeddings are close to the original documents. We can explain this phenomenon in three different aspects.

Firstly, PG Net contains a Pointer Network part which generates words from the original documents. The language model trained by Seq2Seq Network part encourages these words in a similar pattern as in the original document. Since most sentence embedding methods introduced in section 2.2 are based on bag of words or at most short N-grams, the summaries generated by PG Net with the same words and similar token patterns will highly probably have close embeddings with the orignal documents

Secondly, the embeddings of the original documents and gold summaries are almost collinear in our dataset, using the convolutional feature extractor. This is why 'Cos-Document' value and 'Cos-Reference' value are almost the same for three models. As a result, in the loss function of Sent2VecAbs model, the language model part and sentence embdding part actually share approximately the same optimal position.

Thirdly, we notice that the cosine similarity is over 0.98 even only after two epochs' training for both models. Inside the loss function of Sent2VecAbs model, the Sent2Vec part is almost zero while the language model part is still very large. That is to say, the magenitute of both parts do not scale very well. Compared with language model, Sent2Vec loss is almost ignored! At this point training Sent2VecAbs Model is approximately equivalent with PG Net, but the gradient is multiplied by coefficient λ. As a result, Sent2VecAbs will converge slower than PG Net.

5 Conclusion and Future Work

Text summarization is a generative model, a neural language model is trained to model the probability of output sequences. In this project, Seq2Seq Network is employed to learn the language model. All other neural network models are based on Seq2Seq Network.

In the previous chapters, we have demonstrated several neural network models for text summarization. We have also tried to add some extra linguistic or semantic information, including pretrained word embeddings, entity name detection, embeddings of original document and output summary, to improve their performance.

The results show that the pretrained embeddings and entity name detection can help to improve the performance. However, the incorporation of embeddings of input and output sequence does not help to improve the performance. Experiments show that its advantages overlap with the Pointer Network. In addition, the loss based on input-output embedding comparison does not scale well with the original loss from language model.

Another main discussion in this project is the evaluation metric for summaries we generated. For extractive models, things are simpler, because they are classificaion or regression model and the sentence labels are reliable. However, it is not an easy job to find an objective metric for evaluation of abstractive models. Traditionally, statistical metrics such as ROUGE scores are used despite some shortcomings. In this project, we propose a metric based on the similarity of sentence embeddings. It is semantics based and more robust to paraphrasing and synonyms replacement. It is consistent with ROUGE scores for model comparison in experiments.

Possible Directions

Text summarization is a big, open and challenging problem. There is always some space for our models to improve. According to the experiments and analysis in this project, we points out several directions for model improvement.

- The Mixed Model in section 3.3.3 does not perform as well as PG Net because of the inconsistency in training and decoding phrase. We also mention that training the extractive and abstractive part jointly may waste the extractive labels of sentences. However, there still exists one possible way to train them jointly with both extractive and abstractive labels. That is to first pretrain the extractive part alone with extractive labels, then train the abstractive part with the pretrained extractive part jointly with abstractive labels. We think this training method will probably improve the performance of Mixed Model.

- We mentioned in section 4.5 that the Sent2VecAbs model does not obtain the expected performance because the loss of sentence embedding part does not scale well with the language model part. This is because we use cosine-based embedding similarity instead of the distance-based one, the later has approximately the same scale as the language model one. It is worth a try to use the distance-based embedding similarity to train the Sent2VecAbs Model.

Bibliography

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Bojanowski et al., 2016] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

[Carbonell and Goldstein, 1998] Carbonell, J. and Goldstein, J. (1998). The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM.

[Chen et al., 2016] Chen, V., Montaño, E. T., and Puzon, L. (2016). An examination of the cnn/dailymail neural summarization task. *arXiv preprint arXiv:1606.02858*.

[Cheng and Lapata, 2016] Cheng, J. and Lapata, M. (2016). Neural summarization by extracting sentences and words. *arXiv preprint arXiv:1603.07252*.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[Gers and Schmidhuber, 2001] Gers, F. A. and Schmidhuber, E. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

[Gers et al., 1999] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

[Greff et al., 2016] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hu and Liu, 2004] Hu, M. and Liu, B. (2004). Mining opinion features in customer reviews. In *AAAI*, volume 4, pages 755–760.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Larochelle et al., 2009] Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40.

[Lin, 2004] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

[Maskey and Hirschberg, 2005] Maskey, S. and Hirschberg, J. (2005). Comparing lexical, acoustic/prosodic, structural and discourse features for speech summarization. In *Ninth European Conference on Speech Communication and Technology*.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Nallapati et al., 2016] Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

[Pagliardini et al., 2017] Pagliardini, M., Gupta, P., and Jaggi, M. (2017). Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*.

[Pascanu, 2015] Pascanu, R. (2015). On recurrent and deep neural networks.

[Paulus et al., 2017] Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

[Reddy et al., 1977] Reddy, D. R. et al. (1977). Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Camegie-Mell University, Pittsburgh, PA*.

[Rush et al., 2015] Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*.

[Sak et al., 2014] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*.

[See et al., 2017] See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

[Sundermeyer et al., 2012] Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Vinyals et al., 2015] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

[Wang et al., 2014] Wang, R., Liu, W., and McDonald, C. (2014). Corpus-independent generic keyphrase extraction using word embedding vectors. In *Software Engineering Research Conference*, volume 39.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.