# Variations In Seq2seq Models For Abstractive Summarisation

**Matthew Lo**
Department of Computer Science
Royal Holloway University of London
Egham TW20 0EX
matthewhx.lo@gmail.com

## Abstract

Automatic text summarisation has been one of the hottest subject in Natural Language Generation today. Instead of selecting key phrases/sentences (extractive summarisation) from the original text, an abstractive summariser produces word sequences from a given set of vocabularies, after interpretation and preserving the important information. This project shows an experiment on a variety of Seq2seq models (sequence-to-sequence), in order to determine different abstractive approaches in summarisation and their pros/cons respectively. [1]

## 1 Key concept

Abstractive summarisation is heavily based on the Neural Machine Translation (NMT) tasks. They are both trained on Seq2seq, while NMT reproduces word sequences in a different language from the original text, abstractive summarisation generates word sequences in the same language with shorter lengths.

### 1.1 Seq2seq

Seq2seq model has found a great success in NMTin recent years. Sentences lengths in typical NMT tasks are usually not known in prior. For normal DNNs (deep neural networks), they can only be used if input and output data are encoded into vectors with known and fixed dimension. The Seq2seq model which invented by Sutskever, et al. (2014), is an approach to tackle such tasks and a solid foundation on which the future advancements are to be invented.

The model is made up of 2 components, an Encoder and an Decoder. After converting and embedding the input words (source), the encoder encodes the source to a context vector in a fixed size (hidden dimension). The decoder then decodes it to an output sequence (predicted target) by generating a vocabulary distribution at each step. The predicted target word is then chosen by the corresponding vocabulary distribution.

Both encoder and decoder are RNNs (recurrent neural networks) at this stage. The most common choices of RNNs are GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) networks. The encoder emits the last encoder hidden state as the context vector, and this will be used as the first decoder hidden state to the decoder.

## 2 Training details and limitations

Everything in this project are coded in Python. The models are built with the machine learning framework PyTorch and executed in the Google Colaboratory environment, trained by the provided 12GB NVIDIA Tesla K80 GPU, up to 12

[1] Submitted for the Degree of MSc Machine Learning ($2^{nd}$ edition)

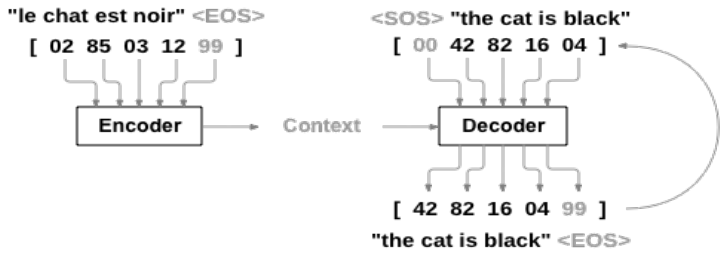


Figure 1: A simple Seq2seq model - `https://pytorch.org/tutorials/_images/seq2seq.png`

hours continuously. Due to the limited RAM size, the batch size for a larger dataset will also be limited, hence longer training time is required. Given the 12 hours running time, the only solution is to train with a subset of the given dataset. It seems that the model performance will also be limited as well.

The aim of this project is to compare and contrast the performances among different model architectures, instead of training the best summariser / achieving the highest score. Therefore it is normal to have a relative lower performance to the works described in the key papers, as the model's convergence is not optimised.

The first model development is based on Ben Trevett's work[2], he provided a very standard and clear tutorial in building a basic Seq2seq model.

## 3 Datasets and evaluation metrics

### 3.1 Gigaword

The Gigaword dataset is one of the largest English news datasets for summarisation tasks, with almost 4 millions example pairs, first used in the work of Rush, et al. (2015). The version of Gigaword used here is collected from the Tensorflow Datasets API [3], mainly used for news headline generation, each example pair consists of a document and a summary. Due to the enormous size of the dataset, only 300k of the example pairs will be used for training, 10k for validation and 1951 for testing. There are several things have to bear in mind before using this dataset. Here is an example pair:

| **document** |
| --- |
| five people were killed , and a woman gravely wounded , following a lethal shootout at a nightclub in cali , colombia 's third largest city , local authorities said monday . |

| **summary** |
| --- |
| colombian nightclub shootout leaves five dead |

**Numbers.** Numerical details in sentences are being encoded as "#", and the number of "#" represents the digits of the corresponding numerical data. To simplify it in our settings, "#" of any length will be converted to a newly created token `<NUM>`. Here is an other example pair:

| **document** |
| --- |
| the number of u.s. service members who have died in iraq since the war began last march reached ### on saturday after a roadside bomb detonated north of baghdad , killing three u.s. soldiers and two iraqi civil defense troopers . |

| **summary** |
| --- |
| three u.s. soldiers killed north of baghdad military says bringing total to ### since war began |

**Sentence length.** The maximum sentence length of the documents and summaries for training are 98 and 44 respectively, which are the outliers in the data, and such sentences lengths will be extra burden to the model during training, especially with large batch size, leaving excess padding within the batch. We set the 5th percentile as the

[2] `https://github.com/bentrevett/pytorch-seq2seq`

[3] `https://www.tensorflow.org/datasets/catalog/gigaword?hl=es`

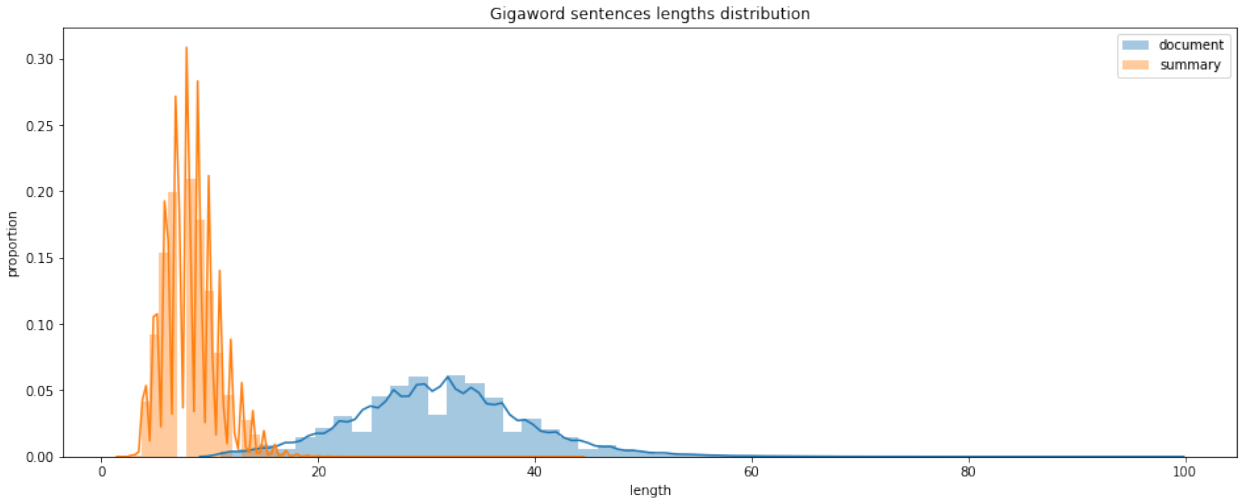


Figure 2: Gigaword sentence length distribution

| Percentile | Document | Summary |
|---|---|---|
| 5% | 18 | 5 |
| 95% | 45 | 13 |

Table 1: Gigaword sentence length percentiles

minimum length and the 95th percentile as the maximum length for each. Example pairs with sentence length out of the ranges will be filtered.

**Default UNK (unknown).** Some of the example pairs have been found containing the tokens <UNK>, which represent unknown words. A typical way to resolve this is to treat them as OOV words with a pointer-generator model. However this will not be used in the early stage of the model development, so example pairs with default <UNK> tokens will also be filtered.

**document**
kazakh racer andrei $< UNK >$ has left credit agricole to sign a two-year deal with spanish team liberty , his agent jacinto $< UNK >$ told afp on saturday .

**summary**
liberty bell sounds for $< UNK >$

### 3.2 CNN/Dailymail

Another data used here is the CNN/Dailymail non-anonymised news summarisation dataset. Similar to Gigaword, each example contains 2 entities: article and highlights. The dataset used here is a tokenised version processed by Jaffer Wilson [4].

[4] https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail

| Split | Original | Chosen | Filtered |
|---|---|---|---|
| Train | 3,803,957 | 300,000 | 208,647 |
| Validation | 189,651 | 10,000 | 6,644 |
| Test | 1,951 | 1,951 | 909 |

Table 2: Number of examples pairs in each split of the Gigaword: After filtering and text cleaning, 64,514 vocabularies were extracted from this dataset and now can be used.

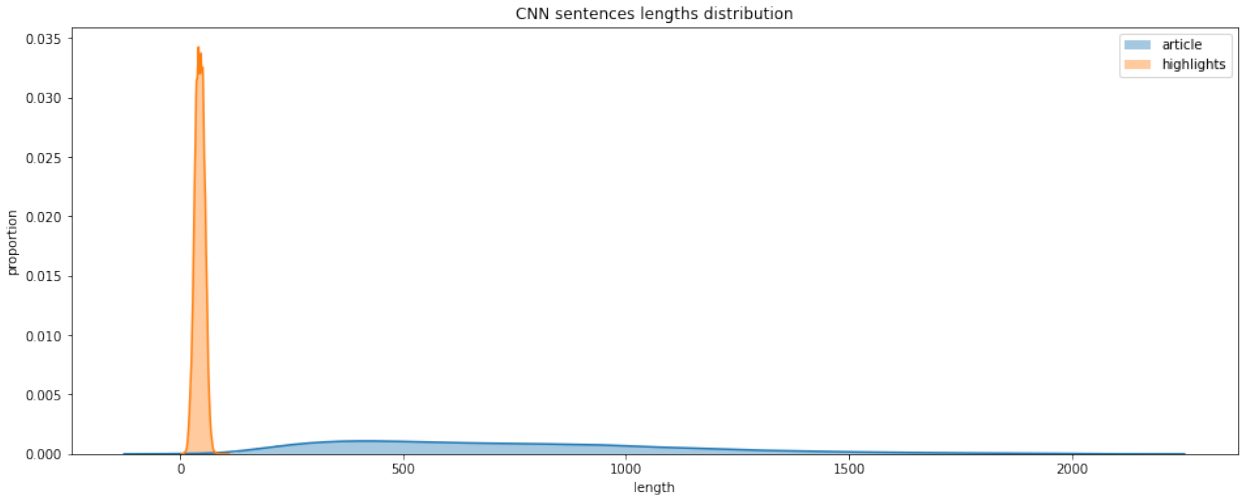


Figure 3: CNN sentence length distribution

**Sentence length.** Despite of the relative small data size, examples have much longer sentences lengths, so only CNN news examples are used here. The length distribution of the source sentences is flat and most of them have exceeded 500 words, filtering them by lengths seems useless. Therefore, the source sentences are truncated 300 words max. A similar approach can also be applied on the target sentences. However, truncating them by words count may deteriorate the sentence structure after training. Each example has 1-4 target sentences, they would be truncated up to 2 sentences instead.

**Reducing vocab.** After truncating the sentences, the vocab size of the training set is still very large (197k). Such large vocabulary size is a burden to the output layer of the model. Reducing the vocabulary size can speed up training and test time, which will be performed in the following steps:
1. Sort the vocabulary list by word frequencies
2. Truncate the list by the parameter `max_size`
3. Filter the list by the parameter `min_freq`
The above process can also preserve the most useful vocabularies. A vocabulary size of 80k here with `max_size` = 80k and `min_freq` = 4. Words that are filtered will be mapped into `<UNK>`.

### 3.3 ROUGE score

In order to evaluate how well is the summary generated quantiatively, ROUGE (Recall-Oriented Underlying for Gisting Evaluation) developed by C.Y. Lin (2004) is one of the best metrics among all.

**ROUGE-N.** This is basically an n-gram recall between a generated summary and a set of reference summaries. As there exists only 1 reference summary for each example in the dataset, the score calculation will be simplified to this:

$$score = \frac{\sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{gram_n \in S} Count(gram_n)} \tag{1}$$

where $n$ stands for the length of the n-gram, $gram_n$, and $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a generated summary and a reference summary. As n-gram increases, we can expect a decline in the ROUGE-N score. Usually, we do only interested in $n$ up to 3.

**ROUGE-L.** The words in the LCS (longest common sub-sequence) do not necessarily to be consecutive and no n-gram has to be predefined. This calculates the LCS by the following:

$$R_{lcs} = \frac{LCS(X,Y)}{m} \tag{2}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{3}$$

$$score = F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \tag{4}$$

# 4 Models

Each sub-sections in this section corresponds to a model implemented, with experiments results and error analysis. As mentioned before, the test results here are significant lower then the one from some well-known research papers due to the limitation. Some of them are aimed to resolve certain issues.

## 4.1 GRU seq2seq (Baseline)

This is the baseline model in this project, with the use of the Gigaword dataset. Both encoder and decoder use a single-layered unidirectional GRU network.

| Notation | Description | Value |
|---|---|---|
| $\|V\|$ | Vocabulary dimension | 64,514 |
| $n$ | Hidden dimension | 300 |
| $m$ | Embedding dimension | 300 |
| $b$ | Batch size | 128 |
| $T_x$ | Batch source length | varies |
| $T_y$ | Batch target length | varies |

Table 3: The list of hyper-parameters used the model, altering the number of model trainable parameters, where $V$ is the vocabulary set.

| Notation | Description |
|---|---|
| $t$ | Time step |
| $x_t$ | Embedded input source at time $t$ |
| $y_t$ | Embedded input target at time $t$ |
| $h_t^e$ | Encoder hidden state at time $t$ |
| $h_t^d$ | Decoder hidden state at time $t$ |
| $r_t$ | Reset gate at time $t$ |
| $z_t$ | Update gate at time $t$ |
| $n_t$ | New gate at time $t$ |
| $\sigma$ | Sigmoid function |
| $\hat{y}_t$ | Predicted target word at time $t$ |

Table 4: A list of common notations in the models.

**Embedding layer.** Here is an explanation of how an embedding layer works. Before putting the source and target batches into the encoder and decoder, they will be mapped into embedded tensors by $E : \{0, ..., |V| - 1\}^{T_x} \rightarrow \mathbb{R}^{T_x \times m}$. Or in other words, each index in each sentence will be mapped into a vector $\mathbb{R}^m$ by an embedding matrix, which serves as a lookup table. The size of the matrix is the number of trainable parameters in the embedding layer: $m|V|$.

**Encoder.** The encoder receives the embedded source $\mathbf{x} = (x_1, ..., x_{T_x})$ and produces the encoder output $[h_1^e, \ldots, h_{T_x}^e] \in \mathbb{R}^{T_x \times n}$ and a context vector $h_{T_x}^e = c^*$, which are the concatenated encoder hidden states and the last encoder hidden state respectively. ($h_1^e$ is a zero tensor, the initial hidden state input to the encoder.)

$$h_t^e = \begin{cases} GRU(x_t, h_{t-1}^e) & 0 < t \leq T_x \\ \mathbf{0} & t = 0 \end{cases} \in \mathbb{R}^n \tag{5}$$

where the encoder hidden state is being updated by the GRU below:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1}^e + b_{hr}) \tag{6}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h^e_{t-1} + b_{hz}) \tag{7}$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h^e_{t-1}) + b_{hn})) \tag{8}$$

$$h^e_t = (1 - z_t) \odot n_t + z_t \odot h^e_{t-1} \tag{9}$$

where $W_{ir}, W_{iz}, W_{in} \in \mathbb{R}^{n\times m}$, $W_{hr}, W_{hz}, W_{hn} \in \mathbb{R}^{n\times n}$ and $b_{ir}, b_{iz}, b_{in}, b_{hr}, b_{hz}, b_{hn} \in \mathbb{R}^n$ are the trainable parameters. The total number of trainable parameters in the encoder is $3(n^2 + nm + 2n)$ [5].

**Decoder.** The decoder receives a hidden state and an embedded target word at each decoding step, producing the next hidden state, then mapped into a matrix by a fully-connected feed-forward network $f$: $\mathbb{R}^n \to \mathbb{R}^{|V|}$. (The decoder GRU works the same as the encoder GRU). The predicted target word will then be chosen after the values in the matrix are transformed to probabilistic values (vocabulary distribution) by a softmax layer. At the first decoding step, the context vector will be used as the first input hidden state. The number of steps is the target length $T_y$. The predicted target word will be chosen by greedy search, the word with the highest probability along the vocabulary dimension. The number of trainable parameters in the decoder is: $3(n^2 + nm + 2n) + n|V|$.

$$\hat{y}_{t+1} = argmax(softmax(f(h^d_t))) \in \{0, ..., |V| - 1\} \tag{10}$$

**Teacher Forcing.** There are 2 choices for inputting the next embedded target word, the previous predicted target word or the ground truth target word. The predictions of the model are poor at the beginning of training, if the previous predicted word is chosen, the accumulated error will then continuously deteriorate the prediction. If the ground truth word is chosen, the model will then be updated by a correct target word. This can also lead to a faster convergence of the model. However, exposure bias will be our next concern, where the result generated will be different between training and test time, the model may perform badly. This problem will not be discussed here since the main concern is to observe the improvement in performance between models under the same condition [6].

**Cross-Entropy Loss.** The loss for each decoding step is calculated by the negative log-likelihood of the predicted target word: $loss_t = -\log(p(\hat{y}_t))$, The overall loss is then $loss = \frac{1}{T_y}\sum_{t=0}^{T_y} loss_t$, which is known as the cross entropy loss.

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 19,354,200 |
| Encoder | $3(n^2 + nm + 2n)$ | 541,800 |
| Decoder | $3(n^2 + nm + 2n) + n\|V\|$ | 19,896,000 |

Table 5: The number of trainable parameters in each part of the model. The total number is 39,792,000.

The model is being trained for 5 epochs, the decline in training loss means the it is successfully trained. The validation loss stops decreasing at epoch 4, achieved the best lowest validation loss, hence the model state at epoch 4 will be used for testing. ROUGE will be used to evaluate the test set instead of cross-entropy loss.

| Epoch | Validation loss |
|---|---|
| 1 | 5.1341 |
| 2 | 4.7152 |
| 3 | 4.5923 |
| 4 | **4.5824** |
| 5 | 4.6236 |

Table 6: The validation losses

Apart from using evaluation metrics, error analysis is also important, readability issues can be discovered. In our case, using a simple GRU Seq2seq model directly may cause lots of problems.

[5] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`

[6] `https://towardsdatascience.com/what-is-teacher-forcing-3da6217fed1c`

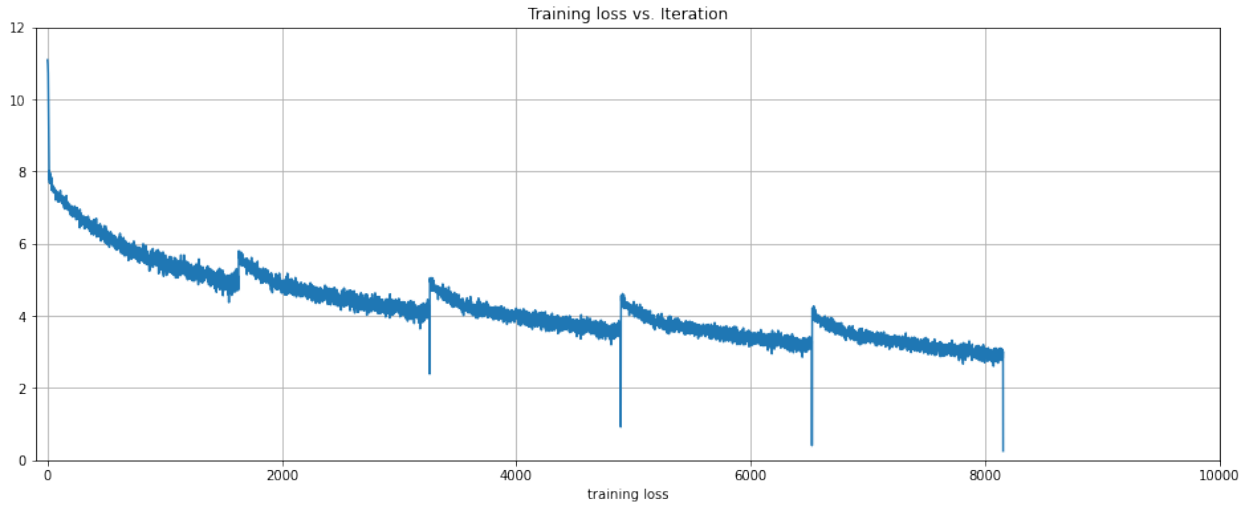


Figure 4: Training loss of each batch (batch order is not shuffled)

| | |
|---|---|
| ROUGE-1 | 19.9038 |
| ROUGE-2 | 5.4203 |
| ROUGE-3 | 2.1796 |
| ROUGE-L | 18.0634 |

Table 7: ROUGE scores

**Inevitable Error.** As below, error occurs when the name entities "israelis" and "lebanese" are different. Although the word "israelis" is in the vocabulary, it does not appear in the source sentence. We cannot expect the model to generate words that are not in the source text, hence such name entity will be replaced by another one with similar meaning, this mismatch is inevitable.

**document**
somali pirates holding a huge oil-laden saudi tanker on saturday vowed to fight back should any assault be attempted to free the ship and urged its owners to pay up a ## million dollars ransom .

**reference summary**
two israelis hurt in hezbollah rocket attacks

**generated summary**
two lebanese soldiers killed in rocket attack

**Factual Error.** However, content mismatch can still occur when every word in the reference summary are in the vocabulary. The generated summary can be produced with unrelated content, which can also deduced from the low ROUGE score. This may due to the model is not well converged. Later, the model will be improved and showing a better convergence, with a lower validation loss.

**document**
the moroccan royal palace says a doctor has ordered king mohammed vi placed in convalescence for five days for a digestive infection .

**reference summary**
morocco king convalescing with digestive virus

**generated summary**
spanish king tut for divorce

### 4.2 + Attention

The attention mechanism proposed by Bahdanau, et al. (2014). is implemented on top of the baseline model.

Despite of the successful result produced by the seq2seq model, Cho, et al. (2014b) found that the model performance rapidly degrades as the sequence length increases, as the context vector carries the encoder's information and is only passed to the first decoder RNN cell. The decoder is then highly relied on the first decoder hidden state. The information carried in the context vector will then diminishes while being transmitted along the decoder RNN cells.

Different words/phrases in a sentence have different importance. The attention mechanism proposed by Bahdanau, et al. (2014) has significantly improved the performance, which allows the decoder to observe the entire source at each decoding step, instead of taking a single context vector. The importance of words/phrases in source will also be converted into a weight/attention distribution vector, forming a better context vector for each step.

**Alignment Model.** As opposed to the baseline model, the context vector will be updated at each decoding step, by the encoder output and the previous decoder hidden state. The alignment model has shown that, for each sentence, we can form a score matrix (energies) (with dimension $T_y \times T_x$) at each step:

$$e_{tj} = a(h^d_{t-1}, h^e_j) = v^T \tanh(W h^d_{t-1} + U h^e_j) \tag{11}$$

where $t$ and $j$ represents positions along the target and source sentences receptively, $v \in \mathbb{R}^n$, $U, W \in \mathbb{R}^{n \times n}$ are the trainable parameters. The context vector can then be calculated from the energies after taking softmax along the $T_x$ axis.

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})} \tag{12}$$

$$c^*_t = \sum_{j=1}^{T_x} \alpha_{tj} h^e_j \tag{13}$$

**Attention Decoder.** In the baseline model, the vocabulary distribution is solely depends on a single decoder hidden state, but in the case of attention decoder, the context vector created in the alignment model and the embedding target word are included. A simplified version by Ben Trevett is that: the embedded target word and the context vector will be concatenated and generate the hidden state by the GRU ($[y_t, c^*_t]$) instead of the only the embedded target word. Then, the hidden state, the context vector and the encoder output will be concatenated and pass through a fully-connected feed-forward network $f$: $\mathbb{R}^{2n+m} \to \mathbb{R}^{|V|}$, get mapped into vocabulary distribution by a softmax layer, then the predicted word will then be chosen.

$$r_t = \sigma(W_{ir}[y_t, c^*_t] + b_{ir} + W_{sr} h^d_{t-1} + b_{hr}) \tag{14}$$

$$z_t = \sigma(W_{iz}[y_t, c^*_t] + b_{iz} + W_{sz} h^d_{t-1} + b_{hz}) \tag{15}$$

$$n_t = \tanh(W_{in}[y_t, c^*_t] + b_{in} + r_t \odot (W_{sn} h^d_{t-1}) + b_{hn})) \tag{16}$$

$$h^d_t = (1 - z_t) \odot n_t + z_t \odot h^d_{t-1} \tag{17}$$

$$\hat{y}_{t+1} = argmax(softmax(f([h^d_t, y_t, c^*_t]))) \in \{0, ..., |V| - 1\} \tag{18}$$

where $W_{ir}, W_{iz}, W_{in} \in \mathbb{R}^{n \times (n+m)}$, $W_{sr}, W_{sz}, W_{sn} \in \mathbb{R}^{n \times n}$ and $b_{ir}, b_{iz}, b_{in}, b_{sr}, b_{sz}, b_{sn} \in \mathbb{R}^n$ are the trainable parameters. The total number of trainable parameters in the attention decoder is then: $2n^2 + n + 3(2n^2 + nm + 2n) + (2n + m)|V|$.

The model is trained for 3 epochs. The validation loss stops decreasing at epoch 2, achieved the best lowest validation loss, hence the model state at epoch 2 will be used for testing. Here the validation loss is lower then the one in the baseline model, indicating a better and faster convergence. There is also a significant rise in ROUGE scores.

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 19,354,200 |
| Encoder | $3(n^2 + nm + 2n)$ | 541,800 |
| Decoder | $2n^2 + n + 3(2n^2 + nm + 2n) + (2n + m)\|V\|$ | 59,054,700 |

Table 8: The number of trainable parameters in each part of the model. The total number is 78,950,700.

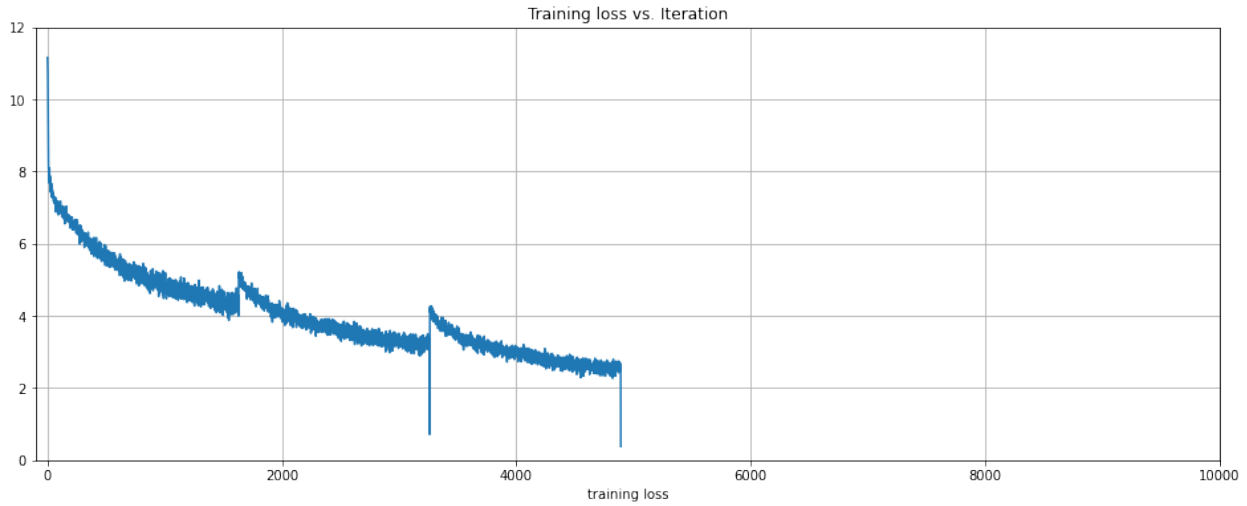


Figure 5: Training loss of each batch (the order is not shuffled)

To show the improvement of the model with the attention mechanism, the same example pairs will be used here.

**Getting the correct subject.** We can see that although the meaning of the sentence is still twisted, the subject of the sentence "morocco king" has been successfully captured, rather than generating words that is not appeared in the source. Several content mismatch can also be seen in the test set, but at least it got some keywords right, leading to a rise in ROUGE scores.

**reference summary**
morocco king convalescing with digestive virus

**generated summary (w/o attention)**
spanish king tut for divorce

**generated summary (w/ attention)**
moroccan king orders provisional release of the right

### 4.3 Bidirectional LSTM + Attention

In this section, two small modifications have been made, changing from unidirectional RNN to bidirectional RNN and from GRU to LSTM, while the attention mechanism remains.

**Encoder.** The main difference between GRU and LSTM is that extra cell states $c_t^e$ and $c_t^d$ are being updated with the hidden states. The issue with the hidden state is abut complicated. The model will run the embedded input in 2 directions, forward and backward. Therefore, 2 sets of hidden states and cell states can be obtained. At the end, hidden states and cell states from 2 directions will be concatenated and pass to the decoder, 2 fully-connected feed-forward network and a tanh activation will be required to reduce their dimensions. It is expected to improve the performance

| Epoch | Validation loss |
|---|---|
| 1 | 4.6014 |
| 2 | **4.3396** |
| 3 | 4.3520 |

Table 9: The validation losses

| | |
|---|---|
| ROUGE-1 | 23.5790 |
| ROUGE-2 | 8.1803 |
| ROUGE-3 | 3.3208 |
| ROUGE-L | 22.4925 |

Table 10: ROUGE scores

with the use of Bi-LSTM, given that it can understand the sentence better, reading the sentence from both directions. Take a look at the forward direction:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}\overrightarrow{h^e}_{t-1} + b_{hi}) \tag{19}$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}\overrightarrow{h^e}_{t-1} + b_{hf}) \tag{20}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}\overrightarrow{h^e}_{t-1} + b_{ho}) \tag{21}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}\overrightarrow{h^e}_{t-1} + b_{hg})) \tag{22}$$

$$\overrightarrow{c^e_t} = i_t \odot g_t + f_t \odot \overrightarrow{c^e}_{t-1} \tag{23}$$

$$\overrightarrow{h^e_t} = o_t \odot \tanh(\overrightarrow{c^e_t}) \tag{24}$$

where $W_{ii}, W_{if}, W_{io}, W_{ig} \in \mathbb{R}^{n\times m}$, $W_{hi}, W_{hi}, W_{ho}, W_{hg} \in \mathbb{R}^{n\times n}$ and $b_{ii}, b_{if}, b_{io}, b_{ig}, b_{hi}, b_{hf}, b_{ho}, b_{hg} \in \mathbb{R}^n$ are the trainable parameters. The number of trainable parameters in one direction is: $4(n^2 + nm + 2n)$.

$$h^e_{T_x} = \tanh(f_h([\overrightarrow{h^e_{T_x}}, \overleftarrow{h^e_{T_x}}])) \tag{25}$$

$$c^e_{T_x} = \tanh(f_c([\overrightarrow{c^e_{T_x}}, \overleftarrow{c^e_{T_x}}])) \tag{26}$$

where $f_h, f_c \in \mathbb{R}^{n\times 2n}$ are also the trainable parameters. Therefore the total number of trainable parameters in the encoder is $2 \times 4(n^2 + nm + 2n) + 2 \times 2n^2 = 12n^2 + 8nm + 16n$.

**Attention.** The encoder output obtained above has twice the hidden dimension from the previous section. I would also include the cell state $c^d_{T_x}$ into the calculation of the context vector, by $h^d_t = [h^d_t, c^d_t]$. The number of trainable parameters here is then: $2(2n)^2 + 2n$. The hidden dimension of the context vector obtained here has also doubled up.

**Decoder.** The RNN in the decoder is unidirectional. However I have a larger context vector and decided to include the cell state into the hidden state after the RNN decoding, the total number of trainable parameters in the decoder has also increased: $2(2n)^2 + 2n + 4(n^2 + n(m + 2n) + 2n)) + (4n + m)|V|$

The change in the model architecture here does resolve any particular readability issue. Nothing significant has improved, except there is a drop in the lowest validation loss, meaning that the model's convergence has improved. The ROUGE scores has also slightly increased.

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 19,354,200 |
| Encoder | $12n^2 + 8nm + 16n$ | 1,804,800 |
| Decoder | $2(2n)^2 + 2n + 4(n^2 + n(m + 2n) + 2n)) + (4n + m)\|V\|$ | 98,934,000 |

Table 11: The number of trainable parameters in each part of the model. The total number is 120,093,000.

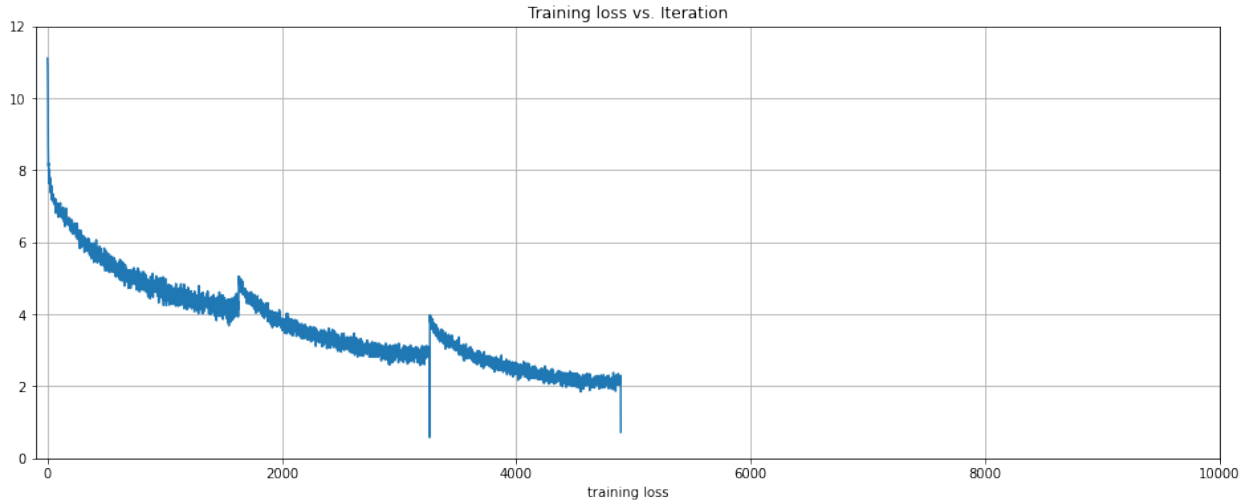


Figure 6: Training loss of each batch (the order is not shuffled)

### 4.4 + GloVe word embedding

In the past sections, the models' word embedding are being trained along with the encoder and decoder. With a pre-trained word embedding trained from a larger corpus, a better word representation vector can be used. It is believed that such word embedding can boost the model performance, without modifying the model's architecture. The pre-trained word embedding used here is GloVe (Global Vectors for Word Representation), collected from the corpus: Wikipedia 2014 + Gigaword 5, with 6B tokens and 400K vocabularies, in vectors with dimension = 300. The convergence is improved and higher ROUGE scores are achieved.

### 4.5 Reducing vocab size

Starting from this section, the CNN dataset will be used with the Bi-LSTM + Attention model. Reducing the vocabulary size (80k) will speed up the training time. Also, due to the long lengths of the sequences, I have reduced the batch size to 16 and switch to iteration training instead. This means, the validation loss will be calculated for every certain number of iterations (500 in this case), instead of every epoch. It is believed that it can help getting a more accurate model. However, reducing the vocabulary size causes problems.

Generally, the result is not good at all. The convergence is slow. The lowest validation loss is at iteration 15500, 4.7216, which is even higher than the baseline model on the Gigaword dataset. The ROUGE scores obtained is low too.

So what is the reason behind the low ROUGE scores? Many nonsensical summaries are being generated by the poor model convergence. Apparently this is something to do with the <UNK> tokens. Several problems have arisen.

**OOV Problem.** In the following example, there are lots of numerical data, which will all be identified as OOV and mapped into <UNK> tokens. With lots of <UNK> token, the model becomes harder to train. Although some keywords from the document can still be picked up, the sentence is still hard to interpret with the <UNK> tokens.

**document**
cnn rape has turned into a weapon of war in the democratic republic of congo , with the number of attacks on women having grown threefold over the past few years , human rights activists said in october . anneke van woudenberg ...

| Epoch | Validation loss |
|---|---|
| 1 | 4.4025 |
| 2 | **4.2564** |
| 3 | 4.3994 |

Table 12: The validation losses

| | |
|---|---|
| ROUGE-1 | 24.1081 |
| ROUGE-2 | 8.5416 |
| ROUGE-3 | 3.6197 |
| ROUGE-L | 22.4925 |

Table 13: ROUGE scores

**reference summary**
human rights watch : 200,000 women , girls raped in eastern congo since 1998 . condition of women more dire as the army fights armed groups in the country .

**generated summary**
congo has been fighting since the war in congo has been persecuted since $< UNK >$ . congo has been given a weapon of war in congo since $< UNK >$ .

**Word Repetition.** The <UNK> tokens are not only affecting the interpretability of the summaries. Some serious cases show that the summaries can be completely unreadable. Here is an extreme example, where there are only 3 meaningful words in the generated summary. This may due to the <UNK> tokens have seriously deteriorate the training, making the conditional probability of generating an <UNK> after an <UNK> is high. Therefore once an <UNK> token appears, an <UNK> token is very likely comes next, and so on.

**document**
cnn a pair of giant trimarans are battling for line honors in the rolex fastnet race , one of yachting 's classic offshore races . loick peyron 's maxi banque populaire was leading fellow french entry , the seb josse-skippered gitana 11 , by eight nautical miles as they rounded fastnet rock off south-west ireland and headed for the finish in plymouth in south-west england . maxi banque populaire is considered the world 's fastest offshore boat with massive 140-foot long hulls , much bigger than the 77-feet of gitana 11 , so the relatively small gap between the two was a ...

**reference summary**
leading trimarans round fastnet rock in yachting 's classic fastnet race . loick peyron 's maxi banque populaire was leading battle for line honors .

**generated summary**
$< UNK >< UNK >< UNK >< UNK >< UNK >< UNK >< UNK >< UNK >< UNK >$
$< UNK >. < UNK >< UNK >< UNK >< UNK >< UNK >< UNK >$ km race in $< UNK >$ .

**Redundant Information.** Sometimes a word isn't repeating itself consecutively, but appears in different places in the summary text. I would consider the word "congo" here as a redundant word rather than simply repeating itself.

**document**
cnn rape has turned into a weapon of war in the democratic republic of congo , with the number of attacks on women having grown threefold over the past few years , human rights activists said in october . anneke van woudenberg , senior researcher with ...

**reference summary**
human rights watch : 200,000 women , girls raped in eastern congo since 1998 . condition of women more dire as the army fights armed groups in the country .

**generated summary**
congo has been fighting since the war in congo has been persecuted since $< UNK >$ . congo has been given a weapon of war in congo since $< UNK >$ .

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | 0 | 0 |
| Encoder | $12n^2 + 8nm + 16n$ | 1,804,800 |
| Decoder | $2(2n)^2 + 2n + 4(n^2 + n(m + 2n) + 2n)) + (4n + m)\|V\|$ | 98,934,000 |

Table 14: The number of trainable parameters in each part of the model. The total number is 100,738,800.

| Epoch | Validation loss |
|---|---|
| 1 | 4.1048 |
| 2 | **3.9115** |
| 3 | 3.9613 |

Table 15: The validation losses

| | |
|---|---|
| ROUGE-1 | 25.6854 |
| ROUGE-2 | 9.7915 |
| ROUGE-3 | 4.4563 |
| ROUGE-L | 24.8279 |

Table 16: ROUGE scores

## 5 Pointer-Generator

When words are converted into indices, if the word is not found in the vocabulary list, it is OOV (out-of-vocabulary words), it will get mapped to the `unk` token and its corresponding index. This happens when the word we encountered is a rare word [7], or it is a new word during test time. The hybrid pointer-generator network proposed by See, et al. (2017) is aimed to solve handle the unknown words.

**Extended Vocabulary.** So far, each example pair consists of 2 fields: source and target. The very first thing to do right before the training is to convert the words from those fields into integers. Each integers represents a one-hot vector for look-up purpose in the embedding layer. Without the considering in OOV words, all unknown words will be mapped to the integer that represents `<UNK>` (e.g. 3 in this case). So now there is a `src` integer sequence and a `trg` integer sequence for each example pair. In this section, we want to handle the unknown words. First of all, we create an OOV set for each example pair, all unknown words in an example pair will be converted to numbers $\in \mathbb{Z}_{\geq |V|}$. Please note that an extended vocabulary set for each example is the union of the "global" vocabulary set and the example's OOV set. The words will then be converted to indices `src\_extended\_vocab` and `trg\_extended\_vocab` according to the extended vocabulary set. Since the model is trained in batches, we have to define a maximum OOV length `oov` here for each batch, to extend dimension of the final word distribution in the output layer. Now each example consists of 4 integer sequences, `src` and `trg` are the input to the encoder and the decoder respectively. `src\_extended\_vocab` is responsible for calculating the word distribution in the output layer and finally `trg\_extended\_vocab` handles the calculation of the loss.

**Architecture.** The vocabulary distribution produced inside the decoder becomes a 2-layered fully-connected feed-forward network rather than just one. For each example in each batch at each decoding step:

$$P_{vocab} = softmax(V'(V([h_t^d, y_t, c_t^*] + k) + k') \in [0, 1]^{|V|} \tag{27}$$

where $V' \in \mathbb{R}^{|V| \times n}$ and $V \in \mathbb{R}^{n \times 4n+m}$ are 2 linear layers with biases $k'$ and $k$ respectively.

Next, a probabilistic value is needed to be used as a weight between generating the word from a vocabulary distribution and copy the word from the source.

$$p_{gen} = \sigma(w_{ptr}^T [c_t^*, h_t^d, y_t] + b_{ptr}) \in [0, 1] \tag{28}$$

[7] If it is a rare word, it has low frequency in the training data, and will be trimmed off from the vocabulary list before training.

| Iteration | Validation loss |
|---|---|
| 00000 | 13.4847 |
| 01000 | 5.9410 |
| ... | ... |
| 15500 | **4.7216** |
| 16000 | 4.7342 |
| 16500 | 4.7723 |
| 17000 | 4.8530 |

Table 17: The validation losses

| | |
|---|---|
| ROUGE-1 | 17.9348 |
| ROUGE-2 | 3.8409 |
| ROUGE-3 | 1.0783 |
| ROUGE-L | 18.8614 |

Table 18: ROUGE scores

where $w_{ptr}^T \in \mathbb{R}^{4n+m}$ with bias $b_{ptr}$. To copy a word from the source, the word is generated from the attention distribution.

$$P_{attn} \in [0,1]^{|V|+oov}, P_{attn_i} = 0 + \sum_{j:i=w_j} \alpha_{tj} \tag{29}$$

where $oov$ is the maximum number of the OOV words among the sources in that batch, and $w_j$ indicates the word at position $j$ in the source, $\alpha$ is the attention weight explained earlier.

Before calculation the word distribution, the vocabulary distribution has to be padded to fit the size of the attention distribution (extended vocabulary distribution). The word distribution is then the weighted average of the extended vocabulary distribution and the attention distribution. The loss calculation will be based on this as well.

$$P_{extended\_vocab} = [P_{vocab}, \mathbf{0}] \in [0,1]^{|V|+oov} \tag{30}$$

$$P_{word} = p_{gen} P_{extended\_vocab} + (1 - p_{gen}) P_{attn} \in \mathbb{R}^{|V|+oov} \tag{31}$$

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 24,000,000 |
| Encoder | $12n^2 + 8nm + 16n$ | 1,804,800 |
| Decoder | $2(2n)^2 + 2n + 4(n^2 + n(m+2n) + 2n)) + n(4n + m + \|V\|) + \|V\| + n$ | 26,693,300 |

Table 19: The number of trainable parameters in each part of the model (Reduced Vocabulary). The total number is 52,498,100.

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 24,000,000 |
| Encoder | $12n^2 + 8nm + 16n$ | 1,804,800 |
| Decoder | $2(2n)^2 + 2n + 4(n^2 + n(m+2n) + 2n))$ $+n(4n + m + \|V\|) + \|V\| + n + (4n + m + 1)$ | 26,694,801 |

Table 20: The number of trainable parameters in each part of the model (Pointer Generator). The total number is 52,499,601. Only 1,501 extra trainable parameters is needed for the generation probability.

By using the pointer-generator here, the results has drastically improved. The model converged at iteration 10500 with validation loss 3.9048 (See Table 21). The ROUGE scores have also increased (See Table 22). In See et al.'s work, they found that the pointer-generator is quicker to train. In this experiment, the model also converged earlier then without the pointer-generator (iteration 15500 and iteration 10500 respectively).

Two main issues have solved by the pointer-generator model.

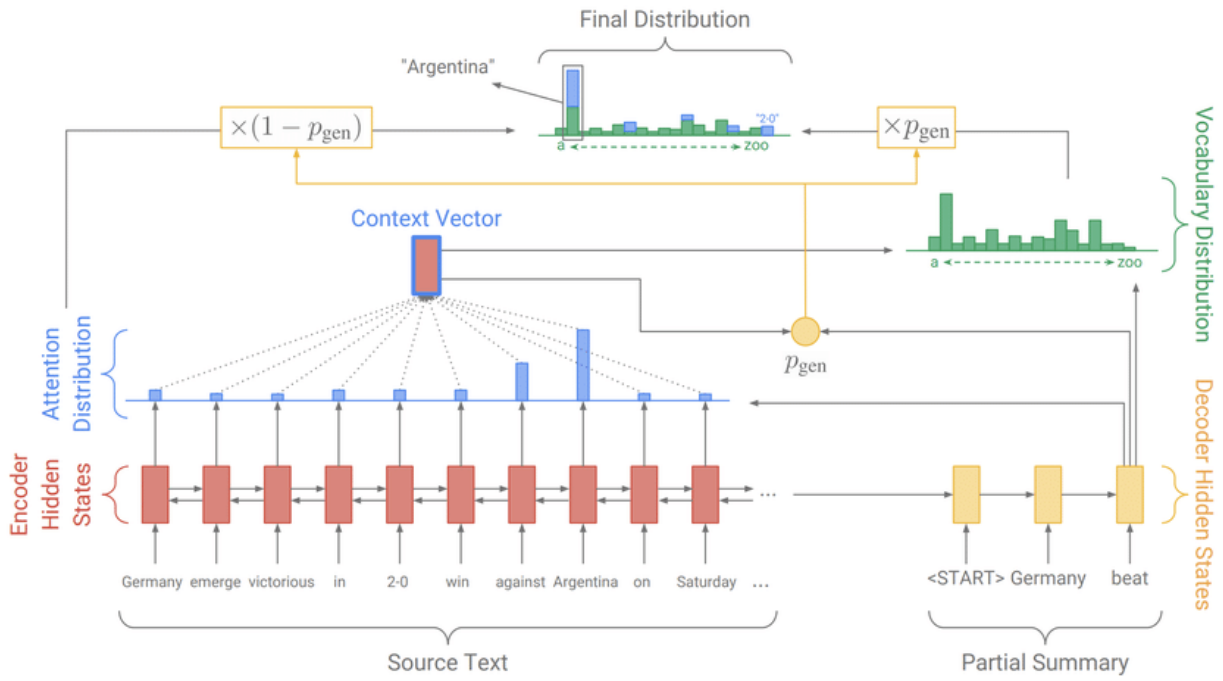


Figure 7: The pointer generator model

| Iteration | Valid loss |
|---|---|
| 00000 | 9.3708 |
| 01000 | 4.6228 |
| 02000 | 4.3556 |
| ... | ... |
| 10000 | 3.9053 |
| 10500 | **3.9048** |
| 11000 | 3.9424 |
| 11500 | 3.9394 |
| 12000 | 3.9249 |

Table 21: The validation losses

**Reduced UNK.** Originally, the average percentage of <UNK> tokens in the generated summaries is about 9.8%. The pointer-generator has successfully reduced that amount to less than 1%.

**Copy from source.** Let's return to the first example from the previous section. Key numerical details (200,000 and 1998) which were OOV words and supposed to be mapped into UNK> tokens now appears in the generated summary. However, there is a repeat and incorrect information in the middle. The problem of showing redundant and repetition information still exist.

**reference**
human rights watch : 200,000 women , girls raped in eastern congo since 1998 . condition of women more dire as the army fights armed groups in the country .

**w/o pointer-generator**
congo has been fighting since the war in congo has been persecuted since $< UNK >$ . congo has been given a weapon of war in congo since $< UNK >$ .

**w/ pointer-generator**
human rights watch says 200,000 women and girls have been raped in eastern congo . 1998 people were girls in eastern congo since 1998

| | |
|---|---|
| ROUGE-1 | 24.3825 |
| ROUGE-2 | 7.1653 |
| ROUGE-3 | 3.0409 |
| ROUGE-L | 23.0794 |

Table 22: ROUGE scores

| w/o pointer-generator | w/ pointer-generator |
|---|---|
| 0.0983 | 0.0091 |

Table 23: Ratios of $< UNK >$ in the test generated summaries by a Bi-LSTM + Attention model

### 5.1 + Coverage

Also mentioned in same paper, the coverage mechanism is designed to reduce repetition on top of a pointer-generator model. In their experiment, they implemented coverage on a trained model. In here, I have trained the model with coverage at the beginning. To define coverage, it is a vector $c^t$ which is the sum of attention weights over all previous decoding steps.

$$c^t = \sum_{t'=0}^{t-1} \alpha_{t'} \tag{32}$$

And there is also a slight modification on calculating the energies.

$$e_{tj} = v^T \tanh(W_s h_{t-1}^d + W_h h_j^e + w_c c_j^t) \tag{33}$$

where $W_s, W_h \in \mathbb{R}^{2n \times 2n}$ and $w_c \in \mathbb{R}^{2n}$. Finally, the loss function at each step is also modified as follows:

$$loss_t = -\log(p(\hat{y}_t)) + covloss_t = -\log(p(\hat{y}_t)) + \sum_j \min(\alpha_{tj}, c_j^t) \leq -\log(p(\hat{y}_t)) + \sum_j \alpha_{tj} = 1 - \log(p(\hat{y}_t)) \tag{34}$$

where the coverage loss $covloss_t$ is bounded by 1 as shown.

| Component | Number of trainable parameters | Value |
|---|---|---|
| Embedding | $m\|V\|$ | 24,000,000 |
| Encoder | $12n^2 + 8nm + 16n$ | 1,804,800 |
| Decoder | $2(2n)^2 + 2n + 4(n^2 + n(m + 2n) + 2n))$ $+n(4n + m + \|V\|) + \|V\| + n + (4n + m + 1) + 2n$ | 26,695,401 |

Table 24: The number of trainable parameters in each part of the model. The total number is 52,500,201.

There is a slight improvement. I have compared the number of repetitions between with and without coverage on the same Bi-LSTM + Attention + Pointer-Generator model. The indicator I used is simply the number of repetitions in the sentence over the length of the sentence (See Table 22). With coverage, the proportion of repetitions in sentences in the test set has reduced by at least 3%.

| w/o coverage | w/ coverage |
|---|---|
| 0.4154 | 0.3790 |

Table 25: Ratios of repetitions in the test generated summaries by a Bi-LSTM + Attention model + Pointer-Generator (stop-words included).

There are also improvements in the ROUGE scores

| Iteration | Valid loss |
|---|---|
| 00000 | 11.1837 |
| 01000 | 6.1547 |
| 02000 | 5.8925 |
| 03000 | 5.7405 |
| 04000 | 5.6454 |
| 05000 | 5.5729 |
| ... | ... |
| 10000 | 5.4377 |
| 10500 | **5.4328** |
| 11000 | 5.4766 |
| 11500 | 5.4784 |
| 12000 | 5.4508 |

Table 26: The validation losses

| | |
|---|---|
| ROUGE-1 | 25.6496 |
| ROUGE-2 | 6.7768 |
| ROUGE-3 | 2.6697 |
| ROUGE-L | 19.8242 |

Table 27: ROUGE scores

# 6 Extension

In this section, some models that beyond basic Seq2seq models will be discussed. They will not necessarily improve the summarisation performance or solving any particular issue, but independent investigations on them are also interesting to look at.

## 6.1 + Self-Critic Policy Gradient

### 6.1.1 Problems with normal Seq2seq

The cross-entropy loss has been used all the time to train the models. Two main problems have been raised in the work of Keneshloo, et al. (2019). First of all, as mentioned, the exposure bias arises when model is not exposed to its previous predictions when predicting the next word in the training time. During test time, the model's prediction may suffer from the accumulated error. Secondly, there is a mismatch between the evaluation metric (ROUGE) and the training objective (cross-entropy). Minimising the cross-entropy loss doesn't necessarily maximise ROUGE. The work of Paulus, et. al.(2017) has also mentioned that potentially valid summaries may be penalised by the cross-entropy loss due to different orders of tokens while ROUGE doesn't.

### 6.1.2 Reinforcement Learning

The goal of an agent in reinforcement learning is to maximise the expected reward obtained in the environment state by its action and the agent chooses its actions by its policy. In a seq2seq summarisation task, The reward $r \in \mathbb{R}_+$ we

try to maximise is the ROUGE score, which ROUGE-1 will be used here for simplicity. Next the state $s_t \in S$ can be considered as a tuple of the previous decoder hidden state and the encoder output. The action $a_t \in A$ is actually the predicted target $\hat{y}_t \in \{0, ..., |V| + oov - 1\}$. Last but not least, the policy $\pi_\theta$ is the Seq2seq model, which is a policy network that maps states to actions $\pi_\theta : S \to A$.

As we have an enormous space of the Seq2seq model, it is a heavy burden for the value-base RL approaches, while with the use of policy-based RL, parameter $\theta$ can be learned instead of memorising a large amount of state and state-action values, especially for the stochastic policy. Therefore, the policy gradient will be used to test whether it can improve the result. The following modification of the model is based on a part of the work of Paulus, et. al.

### 6.1.3 Mixed Loss

**Cross-Entropy Loss.** In my experiment, a Bi-LSTM + Attention + Pointer-Generator model will be trained by the cross-entropy loss as in section 8.

$$loss_{CE} = -\frac{1}{T_y}\sum_{t=1}^{T_y}\log(p(\hat{y}_t)) = -\frac{1}{T_y}\sum_{t=1}^{T_y}\log(p(\hat{y}_t|y_1, ..., y_{t-1}, \mathbf{x})) \tag{35}$$

**Policy Gradient.** As ROUGE is a discrete metrics, it is impossible for us to obtain its gradient, however it doesn't mean that we can't train according to it. The training follows the REINFORCE Algorithm mentioned in Keneshloo et al.'s work, by the following loss function. It suggests to used a pre-trained policy before using policy gradient.

$$loss_{RL} = \frac{1}{T_y}(r_b - r(\mathbf{y}^s))\sum_{t=1}^{T_y}\log(p(y_t^s|y_1^s, ..., y_{t-1}^s, \mathbf{x})) \tag{36}$$

where $r_b$ is the baseline reward, the average reward of the sampled action sequences in the batch. Here the self-critic model suggests to use the action sequence from greedy search $r_b = r(\hat{\mathbf{y}})$, rather than using sampled actions. No ground truth labels $y_t$ is used during training, hence exposure bias can be reduced. Finally, the objective function will be used here is a hybrid version of $loss_{CE}$ and $loss_{RL}$.

$$loss_{mixed} = \gamma loss_{RL} + (1 - \gamma)loss_{CE} \tag{37}$$

where $\gamma \in [0, 1]$ is a scaling factor.

In this experiment, a pre-trained model will be used, and trained with $\gamma \in [0.9, 1]$ increasingly.

To verify is model able to train with the policy gradient, we can see that the from iteration 0 to iteration 800 there is an upward trend and achieved the highest validation ROUGE-1. The downward trend starting from iteration 800 indicates overfitting. The model at iteration 800 is chosen, and used for testing.

| Iteration | $\gamma$ | Valid ROUGE-1 |
|---|---|---|
| 0000 | 0.9 | 0.2158 |
| 0100 | 0.9 | 0.2342 |
| 0200 | 0.9 | 0.2263 |
| 0300 | 0.9 | 0.2214 |
| 0400 | 0.9 | 0.2289 |
| 0500 | 1.0 | 0.2298 |
| 0600 | 1.0 | 0.2298 |
| 0700 | 1.0 | 0.2532 |
| 0800 | 1.0 | **0.3105** |
| 0900 | 1.0 | 0.2580 |
| 1000 | 1.0 | 0.2110 |

Table 28: The validation ROUGE-1

A similar ROUGE-1 score is achieved on the test set. The overall ROUGE scores have also improved except for ROUGE-L (See Table 29).

The policy gradient method has drastically boosted ROUGE-1. However, the rise in ROUGE-2 and ROUGE-3 are not significant. The method is not aimed to improve any explicit readability-related issues. An example below:

| ROUGE scores | Pre-trained | RL trained |
|---|---|---|
| ROUGE-1 | 21.3473 | 31.1960 |
| ROUGE-2 | 3.0413 | 6.5898 |
| ROUGE-3 | 0.6403 | 2.0573 |
| ROUGE-L | 16.6071 | 11.3979 |

Table 29: ROUGE scores

**document**
new york cnn prince harry , the younger son of britain 's prince charles and princess diana , offered his condolences to september 11 victims friday in his first official trip overseas . prince harry , 24 , the third in line to britain 's throne , is making his first official trip overseas ...

**reference summary**
prince harry meets with relatives of september 11 victims . third in line to british throne making first official visit overseas .

**generated summary**
prince harry , harry harry , first his overseas official overseas overseas overseas . the harry harry britain to the 11 children 's his official official to the 11 children 's 2001 , a officials in the rehab.

**Serious repetition.** In the above generated summary, the word "harry" has appeared 5 times, "official" 3 times, "overseas" 4 times and "11" twice. We can see that the model tries to boost ROUGE-1 by "cheating", repeating words. After trained by policy gradient, the repetition is worsen. For future work, as suggested by Paulus et al.'s paper, intra-temporal attention on input sequences and intra-decoder attention can be imposed to reduce repetition. Coverage mechanism can also be used instead, as See et al's work mentioned that the temporal attention may distort the signal from the attention mechanism and reducing performance.

| Pre-trained | RL-trained |
|---|---|
| 0.4191 | 0.6533 |

Table 30: Ratios of repetitions in the test generated summaries by a Bi-LSTM + Attention model + Pointer-Generator (stop-words included).

**Readability.** There is also no readability improvement found during training with policy gradient, despite of the boost in ROUGE-1.

### 6.2 Transformer

The model developed here is a modification of the Ben Trevett's simplified transformer. In short, the transformer model is formed by an encoder and a decoder as well, except they are not RNNs anymore. They are stacked encoder layers and decoder layers respectively, and each layer consists of several sublayers. Without the use of RNNs, the model gets the information of the words position by a positional encoding layer after word embedding.

| Parameters | Symbol | Value |
|---|---|---|
| Vocabulary dimension | $\|V\|$ | 64514 |
| Model dimension | $d_m$ | 300 |
| Feed-forward dimension | $d_{ff}$ | 600 |
| Number of heads | $h$ | 10 |
| Number of layers | $l$ | 3 |
| Maximum sequence length | $T_{max}$ | 50 |
| Batch size | $b$ | 128 |

Table 31: Transformer's hyper-parameter

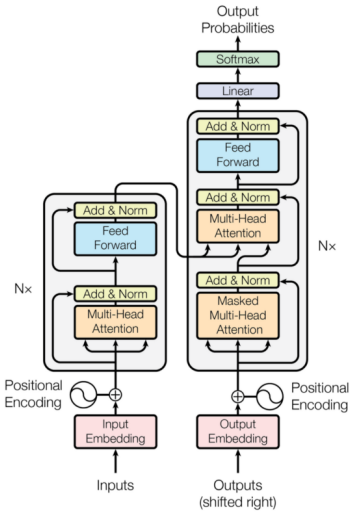


Figure 8: A Transformer model architecture

### 6.2.1 Inside a layer

There are 3 types of sublayer in a layer. 1. Multi-head attention (MHA), 2. Position-wise feed-forward networks (FFN) 3. Layer normalisation (LayerNorm). For an encoder layer, it consists of 1 MHA, 1 FFN and each of them comes before a LayerNorm. For a decoder layer, it has 2 MHA, 1 FFN and similarly, each of them attached with a LayerNorm behind.

**Multi-Head Attention.** In the paper, the embedding dimension $m$ and hidden dimension $n$ are considered the same: model dimension $d_m$. The inputs of a MHA are query, key and value. If it is in an encoder layer, then query, key and value are all embedded source input. For a decoder layer, as there are 2 MHAs in it, the inputs for the first MHA are embedded target input. The inputs for the second MHA are embedded target input, and 2 encoder outputs respectively.

$$MultiHead(Q, K, V) = [head_1, ..., head_h]W^O \tag{38}$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{39}$$

where

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{40}$$

where $W^O, W^K, W^Q, W^V \in \mathbb{R}^{d_m \times d_m}$.
There are $4d_m^2$ trainable parameters.

**Position-wise Feed-Forward Networks.** Apart from the multi-head attention sublayer, the the position-wise feed-forward network is also a crucial part in an encoder layer/decoder layer.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{41}$$

where $W_1 \in \mathbb{R}^{d_{ff} \times d_m}$, $W_2 \in \mathbb{R}^{d_m \times d_{ff}}$, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^{d_m}$.
There are $d_m + d_{ff} + 2d_m d_{ff}$ trainable parameters in total.

**Layer Normalisation.** The LayerNorm defined as follows[8]: in It consists of $2d_m$ trainable parameters.

$$y = LayerNorm(x) = (\frac{x - \mathbb{E}(x)}{\sqrt{Var(x) + 10^{-5}}} \odot \gamma) + \beta \tag{42}$$

where $\gamma, \beta \in \mathbb{R}^{d_m}$ are affine transform parameters.

[8] `https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html`

| Model | Trainable parameters | Value |
|---|---|---|
| Embedding | $d_m\|V\|$ | 19,354,200 |
| Encoder | $[4d_m^2 + d_m + d_{ff} + 2d_m d_{ff} + 2(2d_m)] \times l$ | 2,166,300 |
| Decoder | $[2(4d_m^2) + d_m + d_{ff} + 2d_m d_{ff} + 3(2d_m)] \times l + d_m\|V\|$ | 22,602,300 |

Table 32: Trainable parameters in the transformer. The total number is 44,122,800.

### 6.2.2 Investigation in Positional Encoding (PE)

In the original Ben Trevett's work, a learning PE (trainable) is used instead of a fixed PE (not trainable) described in the paper. Here I will compare the performance of using a learning PE, fixed PE and a fixed PE variant.

**Learning PE.** This a basic trainable embedding layer $\in \mathbb{R}^{T_{max} \times d_m}$. If it is used, there will be $d_m T_{max} = 300 \times 50 = 15,000$ extra trainable parameters.

**Fixed PE.** This is the fixed PE matrix $\in \mathbb{R}^{T_{max} \times d_m}$ defined in the paper:

$$PE_{pos,2i} = sin(\frac{pos}{10000^{\frac{2i}{d_m}}}) \tag{43}$$

$$PE_{pos,2i+1} = cos(\frac{pos}{10000^{\frac{2i}{d_m}}}) \tag{44}$$

where $pos$ and $i$ are the position and its dimension respectively. It is believed that the model can learn the relative position by itself.

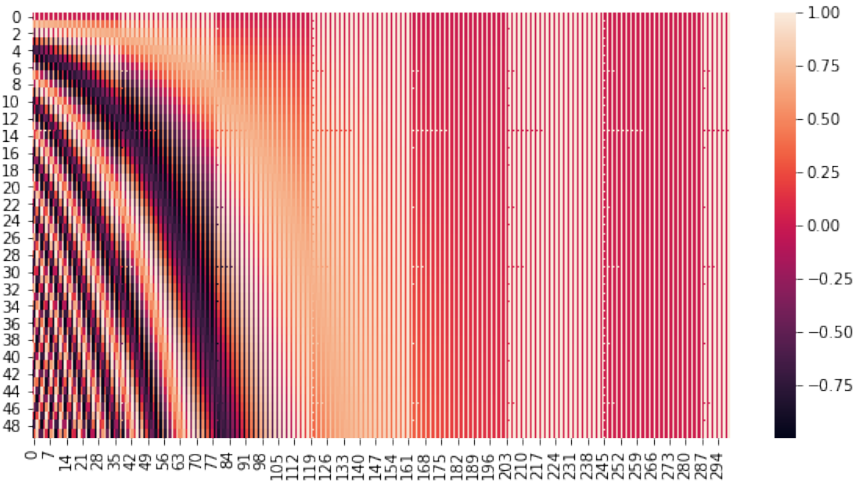


Figure 9: Fixed positional encoding heat-map

**PE variants.** The transformer model is first introduced by Vaswani et al. (2017). The definition of the above PE may differ in different papers. In H. Li et al. (2019)'s work, the PE is defined as follows:

$$PE_{pos,2i} = sin(\frac{pos}{T_{max}^{\frac{2i}{d_m}}}) \tag{45}$$

$$PE_{pos,2i+1} = cos(\frac{pos}{T_{max}^{\frac{2i}{d_m}}}) \tag{46}$$

It has simply replaced 10000 by the max length $T_x$ (The Maximum Variances Positional Encoding Algorithm will not be covered here).

Looking at the validation losses, the learning PE has outperformed the original fixed PE, and the fixed PE variant has achieved the lowest validation loss among all. It may due to the max length $T_{max}$ is included in the PE calculation and becomes a better fit to the training data. A similar conclusion can be drawn from the ROUGE scores.

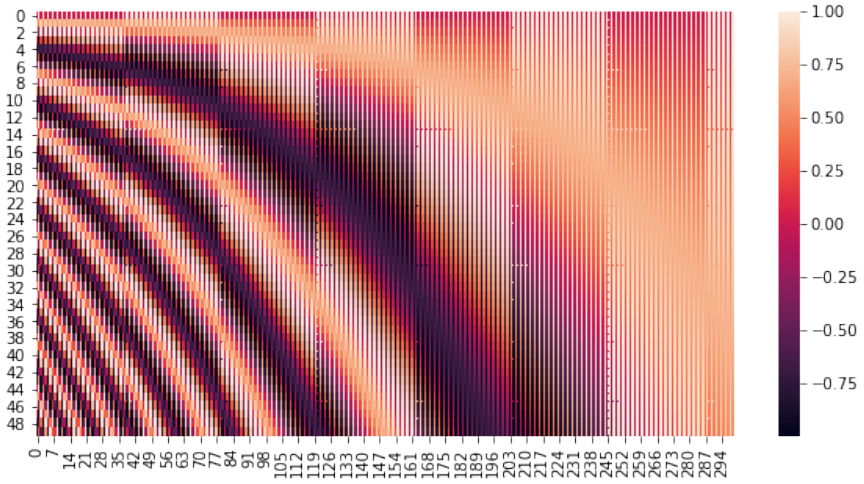

Figure 10: Fixed positional encoding variant heat-map

| Epoch | Learning PE | Fixed PE | PE variant |
|---|---|---|---|
| 1 | 5.5674 | 5.7733 | 5.4695 |
| 2 | 4.9768 | 5.2565 | 4.8453 |
| 3 | 4.7691 | 5.0405 | 4.6139 |
| 4 | 4.6939 | 4.9349 | 4.5374 |
| 5 | 4.6271 | 4.8799 | 4.5069 |
| 6 | 4.5906 | 4.8642 | 4.4846 |
| 7 | **4.5754** | **4.8415** | 4.4846 |
| 8 | 4.5868 | 4.8608 | **4.4792** |
| 9 | – | – | 4.4896 |

Table 33: The validation losses

### 6.2.3 Investigation in Optimiser

So far I have been using the Adam Optimiser with fixed learning rate. A modified version of the Adam Optimiser with varying learning rate is introduced in the paper. The change in learning rate is defined below:

$$lr = d_m^{-0.5} \min(step^{-0.5}, step \times warmup^{1.5}) \tag{47}$$

where $warmup = 4000$, as suggested. Here I have determined the change in performance between using the Adam Optimiser with $lr = 0.001$ and the Modified Optimiser with $lr = 0$ initially. The Modified Optimiser has improved the result.

| ROUGE scores | Learning PE | Fixed PE | PE variant |
|---|---|---|---|
| ROUGE-1 | 21.9457 | 17.8085 | 22.7125 |
| ROUGE-2 | 6.5989 | 4.4055 | 7.1044 |
| ROUGE-3 | 2.5290 | 1.4983 | 2.4005 |
| ROUGE-L | 20.0342 | 16.3494 | 20.9179 |

Table 34: ROUGE scores

| Parameters | Value |
|---|---|
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.98 |
| $\epsilon$ | $10^{-9}$ |
| $warmup$ | 4000 |

Table 35: Optimiser's hyper-parameter

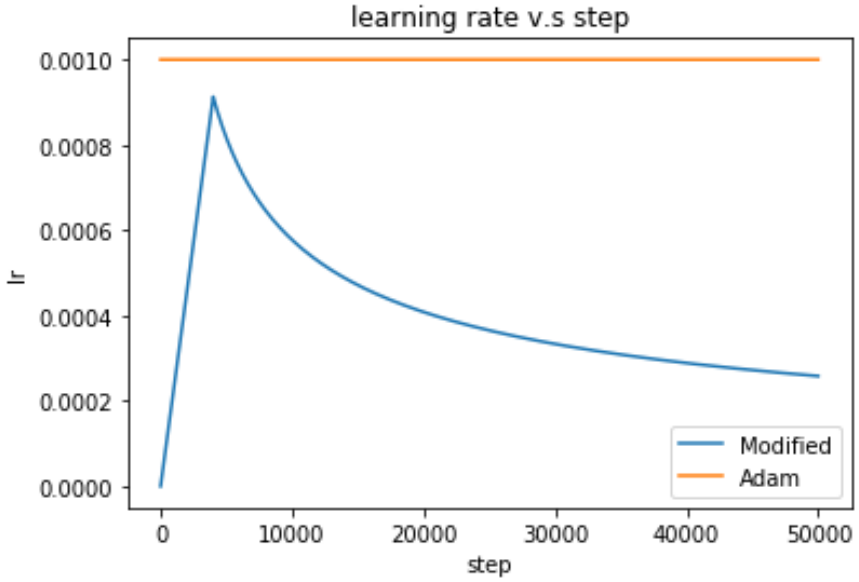


Figure 11: Fixed positional encoding heat-map

| Epoch | Adam Optimiser | Modified Optimiser |
|---|---|---|
| 1 | 5.4695 | 5.6137 |
| 2 | 4.8453 | 4.8559 |
| 3 | 4.6139 | 4.5389 |
| 4 | 4.5374 | 4.4338 |
| 5 | 4.5069 | **4.3966** |
| 6 | 4.4846 | 4.4573 |
| 7 | 4.4846 | – |
| 8 | **4.4792** | – |
| 9 | 4.4896 | – |

Table 36: The validation losses

#### 6.2.4 Investigation in Regularisation

The purpose of regularisation is to prevent or reduce the effect of overfitting. There are 2 types of regularisation introduced: 1. Residual dropout and 2. Label Smoothing. Regularisation was never mentioned in all previous sections. These techniques were explicitly covered in the paper, investigations on them are worth to look at.

| ROUGE scores | Adam Optimiser | Modified Optimiser |
|---|---|---|
| ROUGE-1 | 22.7125 | 25.0492 |
| ROUGE-2 | 7.1044 | 8.3097 |
| ROUGE-3 | 2.4005 | 3.2506 |
| ROUGE-L | 20.9179 | 22.7452 |

Table 37: ROUGE scores

**Residual Dropout.** A dropout rate of 0.1 is used here as suggested. Applying a dropout layer to a neural network produces a "thinned" network by randomly dropping units with a given probability (dropout rate) during training. It was first introduced in Srivastava et al. (2014)'s paper, where a more detailed explanation can be found. A dropout layer is placed at the end of every sub-layer and after summing the positional encoding with the word embedding.

**Label Smoothing.** The concept of label smoothing is not mentioned in the paper, but Müller et al. (2020)'s work explains it all. With a smoothing parameter[9] $\alpha \in [0, 1]$ and a cross-entropy loss. This $loss_{CE}$ may looks different from the previous version but actually they have the same meaning.

$$loss_{CE} = -\frac{1}{T_y} \sum_{k=1}^{|V|} u_k \log(p(y_k)) \tag{48}$$

where $u_t \in \{0, 1\}$ is an indicator. $u_t = 1$ for the correct class and 0 otherwise. The modified indicator of label smoothing is that:

$$u_k^{LS} = (1 - \alpha)u_k + \frac{\alpha}{|V|} \tag{49}$$

The cross-entropy loss with label smoothing is then:

$$loss_{CE}^{LS} = -\frac{1}{T_y} \sum_{k=1}^{|V|} u_k^{LS} \log(p(y_k)) = -\frac{1}{T_y} \left( (1 - \alpha) \sum_{k=1}^{|V|} u_k \log(p(y_k)) + \frac{\alpha}{|V|} \sum_{k=1}^{|V|} \log(p(y_k)) \right) \tag{50}$$

A slight improvement can been seen in the validation losses and ROUGE scores.

| Epoch | w/o regularisation | w/ regularisation |
|---|---|---|
| 1 | 5.6137 | 5.8266 |
| 2 | 4.8559 | 5.0499 |
| 3 | 4.5389 | 4.7397 |
| 4 | 4.4338 | 4.5464 |
| 5 | **4.3966** | 4.4664 |
| 6 | 4.4573 | 4.4229 |
| ... | ... | ... |
| 12 | – | 4.3720 |
| 13 | – | **4.3601** |
| 14 | – | 4.3649 |
| 15 | – | 4.3710 |

Table 38: The validation losses

| ROUGE scores | w/o regularisation | w/ regularisation |
|---|---|---|
| ROUGE-1 | 25.0492 | 25.9478 |
| ROUGE-2 | 8.3097 | 8.6911 |
| ROUGE-3 | 3.2506 | 3.1974 |
| ROUGE-L | 22.7452 | 23.9057 |

Table 39: ROUGE scores

[9]Used $\alpha = 0.1$ here. Normal cross-entropy loss if $\alpha = 0$.

## 7 Conclusion

Here provided some key concepts of the models and their development background, limitations and tools used. Then explained how each major model component works mathematically, and described the models built with their generated result. The main purpose of this part is to evaluate how the change in model architecture resolves major machine summarisation problems, quantitatively and qualitatively. Models developed in the extension section are for experimental purposes, while they are not specifically built to tackle certain summarisation issues. However, they serve as the starting points of future development.

## References


[1] Ilya Sutskever, Oriol Vinyals and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *arXiv preprint arXiv:1409.3215*, 2014.

[2] Alexander M. Rush, Sumit Chopra and Jason Weston. A Neural Attention Model for Abstractive Sentence Summarization. In *arXiv preprint arXiv:1509.00685*, 2015.

[3] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau and Yoshua Bengio. On the properties of neural machine translation: EncoderDecoder approaches. In *arXiv preprint arXiv:1409.1259*, 2014b.

[4] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *arXiv preprint arXiv:1409.0473*, 2017.

[5] Abigail See, Peter J. Liu and Christopher D. Manning. Get To The Point: Summarization with Pointer-Generator Networks. In *arXiv preprint arXiv:1704.04368*, 2017.

[6] Yaser Keneshloo, Tian Shi, Naren Ramakrishnan and Chandan K. Reddy. Deep Reinforcement Learning For Sequence to Sequence Models. In *arXiv preprint arXiv:1805.09461*, 2019.

[7] Romain Paulus, Caiming Xiong and Richard Socher. A Deep Reinforced Model for Abstractive Summarization. In *aarXiv preprint arXiv:1705.04304*, 2017.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N.Gomez, Lukasz Kaiser and Illia Polosukhin. Attention Is All You Need. In *arXiv preprint arXiv:1706.03762*, 2017.

[9] Hailiang Li, Adele, Y.C. Wang, Yang Liu, Du Tang, Zhibin Lei and Wenye Li. An Augmented Transformer Architecture for Natural Language Generation Tasks. In *arXiv preprint arXiv:1910.13634*, 2019.

[10] Nitish Srivastava and Geoffrey Hinton and Alex Krizhevsky and Ilya Sutskever and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *Journal of Machine Learning Research*, 2014.

[11] Rafael Mller, Simon Kornblith and Geoffrey Hinton. When Does Label Smoothing Help? In *arXiv preprint arXiv:1906.02629*, 2019.