Extractive Document Summarization Using Convolutional Neural Networks - Reimplementation

Leo Laugier - 3033157038
Master's Student
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley
leo_laugier@berkeley.edu

Evan Thompson - 3033157454
Master's Student
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley
thompe5@berkeley.edu

Alexandros Vlissidis - 3033156349
Master's Student
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley
alex_vlissidis@berkeley.edu

I. Problem Definition and Motivation

Extractive Summarization is a method, which aims to automatically generate summaries of documents through the extraction of sentences in the text. This task is challenging because compared to key-phrase extraction, text summarization needs to generate a whole sentence that described the given document, instead of just single phrases. Another challenge for this task has been the manual generation of text summaries for supervised learning. Legacy algorithms use predefined handcrafted features of the text for representation. This makes it extremely painful to build an effective algorithm. We implemented an algorithm described by Y. Zhang *et al.*[1], where a Convolutional Neural Network (CNN) [2] approach is proposed for this task. This has the benefit that it can use word-embedding to represent text and the neural network can extract features automatically.

CNNs have traditionally been used in computer vision tasks, as this was their intended original use [3]. However, recent work has shown that they can be very effective in Natural Language Processing (NLP) [4] tasks as well. The specific model we implemented is a regression process for sentence ranking. The architecture of this method consists of a convolution layer followed by a max-pooling layer, on top of a pre-trained *word2vec* [5], [6] mapping. We implemented this new proposed method and perform experiments on single-document extractive summarization. The original paper supports that it can achieve superior performannce than state-of-the-art systems. Our goal was to verify the results and extend the analysis using different parameters. In addition, we open sourced our implementation. This is important because this method does not require any prior knowledge in the text summarization field, hence can be used by anyone and provide state-of-the-art performance.

II. Related Work and Comparisons

During the last few years there have been many publications focused on using deep learning methods for text summarization documents. There are two types of document summarization; extractive and abstractive. The former uses sentences from the given document to construct a summary, and the latter generates a novel sequence of words using likelihood maximization. As mentioned in the introduction we are focusing on related work in extractive text summarization. SummaRuNNer [7] achieves state-of-the-art performance in single document text summarization. It proposes an GRU-RNN network, which gives the advantage of having a model that is easily interpretable. However, this method only in some cases achieves state-of-the-art performance and does not propose a method to extend to multi-document summarization. RNNs have been used to rank sentences in [8], showing incredible performance in this task. This proposed method is limited by the fact that it uses hand-crafted features to represent the input to the RNN. Probably the closest paper to the one we are implementing is [9], which makes use of a hierarchical CNN to summarize documents. The advantage of the method we are implementing is that it has a much simpler architecture and is therefore more computationally inexpensive and is easier to tune. Most recently a Restricted Boltzmann Machine [10] (RBM), is proposed in [11]. However, it is only applied to extractive document summarization in factual reports.

To the best of our knowledge, we did not find any other implementation of text summarization using Convolutional Neural networks, this is why we chose to implement this method.

III. Background

ROUGE: Recall-**O**riented **U**nderstudy for **G**isting **E**valuation (ROUGE) [12] is a metric used for automatic text summarization developped at the University of Southern California.
There exists five variations of ROUGE:

- **ROUGE-N** where $N \in \{1, 2\}$ counting the overlapping *N-gram*[1] between a list of sentences and the ground truth summary.
- **ROUGE-L** evaluates statistics based on the *Longest Common Subsequence*[2] (LCS) [13].

[1] An N-gram is a sequence of N words

[2] The Longest Common Subsequence problem counts the longest sequence of N-grams appearing in two sentences

- **ROUGE-W** adding more weight to consecutive Longest Common Subsequences.
- **ROUGE-S** counting *Skip-bigrams*[3].
- **ROUGE-SU** that fixes a bias resulting from ROUGE-S. Indeed, ROUGE-S does not perform well on sentences without skip-bigram appearing in the ground-truth summary. This is why ROUGE-SU mixes ROUGE-S and ROUGE-1 to compare sentences to summaries.

Salience Score: Zhang *et al.* (authors of the original paper), define a salience score, in order to rank sentences according to their similarity with the ground truth summary. This makes sense for extractive summarization, since the more similar a sentence is with the summary, the more likely it is that the summary contains it. y_i is the salience score of sentence x_i, defined as:

$$y_i = \alpha \cdot R_1 + (1 - \alpha) \cdot R_2 \in \mathbb{R} \tag{1}$$

Where, R is the ROUGE-N score for the i^{th} sentence with its associated summary. α is a coefficient to perform the weighted average of R_1 and R_2, which was set to 0.5 in the original paper. We used that value, since we intend to directly compare our results to theirs.

IV. Dataset

The **D**ocument **U**nderstanding **C**onference (DUC) dataset has become a standard in text summarization and is the one used by Zhang *et al.* in the original paper to compare their proposed method with state-of-the-art. It is composed by news articles in English from the National Institute of Standards and Technology's Text Retrieval Conferences (NIST TREC) to train and test the model, see Table I. The original paper used the DUC2001 dataset for training and the DUC2002 dataset for testing. We followed their design choice in order to have a like-for-like model comparison.

Phase	**Dataset**	**Documents number**	**Sentences number**
Training	DUC2001	358	12,831
Testing	DUC2002	533	15,223

TABLE I: Datasets for training and testing the model.

We requested these datasets through an Individual Application on the DUC website.[1]

V. Approach

In our implementation we decided to use Python, due to its versatility and fast production capabilities, as well as the great support it has from deep learning frameworks. We also decided to implement this project in Keras, since it allows fast-prototyping and experimentation. In addition, we used ROUGE-1 and ROUGE-2 [4] as an evaluation metric in order to be able to compare our model's performance with the original paper and the state-of-the-art systems. After the project proposal phase, the code implementation was split into four main stages.

A. Data Pre-processing

The original paper published very few details about the pre-processing steps taken, so we had to make our own design choices, which may vary from the original authors'. See section VI for more detailed implementation details.

For our pre-processing stage we split the documents into sentences and obtained their word embedding using Google's pre-trained *word2vec*[4], which was trained on 100 billion words from the Google News dataset leading to an embedding of 3 million words at $k = 300$ latent dimensions. At the same time, the salience score for each sentence was computed. This process is described by Algorithm 1 below. The salience scores for each sentence, as well as its word embedding, are stored in pickle file formats for multiple reasons; it allows fast loading during training, but also it enables us to easily change our pre-processing stages without affecting the training stage. This process is described in Figure 1.

Algorithm 1 Preprocessing

```
procedure PREPROCESSING($D$, $summaries$)
  $y \leftarrow vector()$
  $X \leftarrow tensor()$
  for $doc_i \in D$ do
    for $s_j \in doc_i$ do
      $y_i \leftarrow salience(s_j, summaries_i)$
      for $word_k \in s_j$ do
        $X_{i,j} \leftarrow word2vec(word_k)$
return $X, y$
```

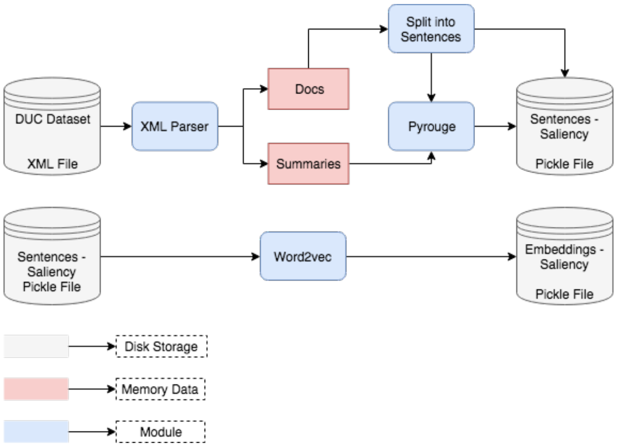

Fig. 1: Full pipeline used to produce the training set using Data Loader

[3] A skip-bigram is an ordered pair of words that are not necessarily consecutive

[1] http://www-nlpir.nist.gov/projects/duc/data.html

[4] https://code.google.com/p/word2vec

B. Model

The model we implemented is described in [14] and is composed of one convolutional layer, one max pooling layer, one dropout layer and one fully connected layer (see Fig. 2).

Input: We represent the input as a tensor $X \in \mathbb{R}^{s \times n \times k}$, where s is the number of sentences in the document, n is the length of the largest sentence and k is the word embedding latent dimension. Each sentence was zero-padded to length n. For the i^{th} sentence $x_i \in \mathbb{R}^{n \times k}$, each row $x_{i,j}^\top$ is the k-dimensional word embedding vector of each word in the i^{th} sentence.

$$x_i = \begin{bmatrix} x_{i,1}^\top \\ x_{i,2}^\top \\ \vdots \\ x_{i,n}^\top \end{bmatrix}$$

1D Convolutional layer: In order to encode memory in our network and mimic the operation of an RNN, the filters need to convolve over multiple words in the sentence and try to extract the score by looking at n-grams. Hence the first layer of our network is a one-dimensional convolution with l filters $(w_f)_{f \in [\![1;l]\!]} \in \mathbb{R}^{k \times m}$; $m \in \mathbb{N}$ being the window size of the filters, corresponding to the number of words the convolution takes into account for each sentence. Since it does not make sense to split words, our filter width (k) is the full word embedding dimension (300), thus combining full words to generate the output.

$\forall (f,j) \in [\![1;l]\!] \times [\![1;n-m+1]\!]$, we define a feature $c_{i,f,j} = \sigma_1(\langle w_f, \left[x_{i,j}, x_{i,j+1}, \ldots, x_{i,j+m-1}\right] \rangle_F + b_{f,j})$ where $b_{f,j} \in \mathbb{R}$ is the bias term and σ_1 is the first activation function, which is ReLU.

$$c_{i,f} = \begin{bmatrix} c_{i,f,1} \\ c_{i,f,2} \\ \vdots \\ c_{i,f,n-m+1} \end{bmatrix} \in \mathbb{R}^{n-m+1} \text{ is called a feature map.}$$

Therefore, for each feature map $f \in [\![1;l]\!]$ we build $n-m+1$ features capturing the meaning of m adjacent words and produce one feature per feature map.

Max pooling layer: Then, $\forall f \in [\![1;l]\!]$, we keep $h_{i,f} = max((c_{i,f,j})_{j \in [\![1;n-m+1]\!]})$ through a max pooling layer so that we keep the important feature.

Dropout layer: The fully connected layer is prone to overfitting so we regularize the model using a dropout layer.

$$\text{Let } r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_l \end{bmatrix} \in \mathbb{R}^f \text{ be the masking vector with probability } p.$$

The input to the fully connected layer is $h_i \odot r$. The original paper set $p = 0.5$.

Fully connected layer: We eventually apply a Fully Connected layer with a sigmoid activation and a single output. $\hat{y_i} = \sigma_2(w_r \cdot (h_i \odot r) + b_r) \in \mathbb{R}$ where $w_r \in \mathbb{R}^l$, $b_r \in \mathbb{R}$ and σ_2 is the sigmoid activation function.

Activation Function: The activation function used by Zhang et al. is the Sigmoid non-linearity.

Output: The output is a scalar $\hat{y_i} \in \mathbb{R}$ which we aim to approximate the salience score y_i defined by (1).

C. Training

For training, we load embeddings for words from pickle files and load the salience scores for all sentences.
Regularization: During training, we impose a l_2-norm constraint to the weights of the output FC layer, w_r, for generalization.

Loss function: The loss function we minimize in training is the binary cross-entropy:

$$\begin{aligned} \mathcal{L} \colon \mathbb{R}^2 &\longrightarrow \mathbb{R} \\ (y, \hat{y}) &\longmapsto \mathcal{L}(y,\hat{y}) = -y \cdot ln(\hat{y}) - (1-y) \cdot ln(1-\hat{y}) \end{aligned}$$

Optimizer: We trained our network with Adadelta update rule, which is the same as Zhang et al used. They didn't specify the learning rate and other hyperparameters used, so we had to experiment with those. We trained all of our networks for 10 epochs, except from very large models, which required up to 100 iterations to converge.

Network parameters: The network has a total of $l \cdot (m \cdot k + 1)$ weights: $l \cdot m \cdot k$ for $(w_f)_{f \in [\![1;l]\!]}$ and l for w_r.
It has $l \cdot (n-m+1) + 1$ biases: $l \cdot (n-m+1)$ for b and 1 for b_r.

D. Testing

For an unseen document from the testing corpus, a similar procedure with Algorithm 1 is followed. First the document is embedded as shown in Figure 1 and then the embeddings are fed forward to the network so as to compute the sentences' salience scores. We rank the sentences according to their salience scores. As we want to build a summary with non-redundant sentences, we select sentences based on [15], which proposes to only include sentences in the summary which are not too similar with what is already included in the summary. Where similarity is defined as $R_1(summary, sentence)$. First we add the sentence with the highest score to the empty summary, then we repeat until the length limit of the final summary is met: select the sentence with the next highest score, if the similarity of the sentence with the summary being built is less than a threshold t, then add the sentence to the summary. This procedure is shown in Algorithm 2 below.

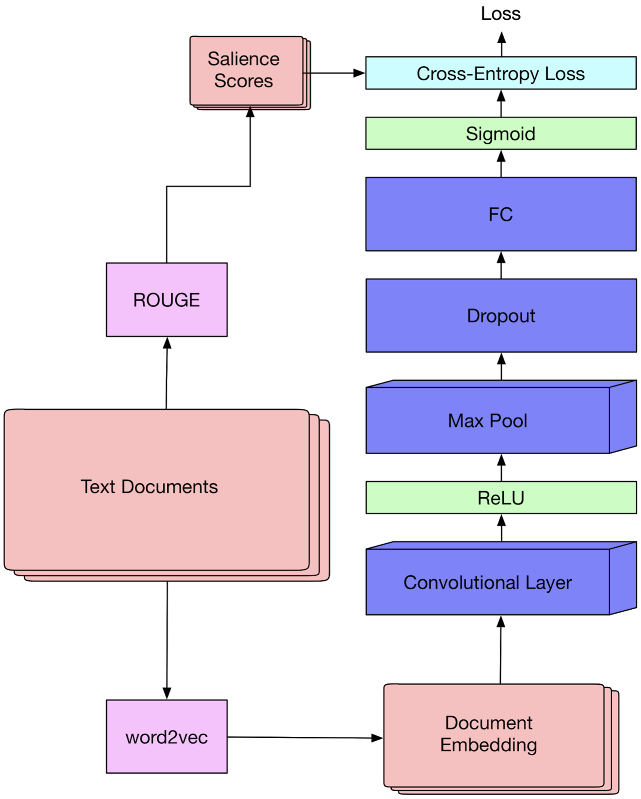

Fig. 2: Full training procedure

Algorithm 2 BuildSummary

```
procedure BuildSummary($t$)
    $summary \leftarrow \emptyset$
    $S \leftarrow \{s_1, \dots, s_N\}$
    for $s_i \in S$ do
        while $|summary| < 100$ do
            if $sim(summary, s_i) < t$ then
                $summary \leftarrow summary \cup s_i$
return $summary$
```

E. Evaluation metric

In this project, we used Rouge in two different ways. First, we produced salience scores y_i for each sentence (III. A.) with a weighted average of ROUGE-1 (counting the co-occurrences of words between the sentence and the associated ground-truth summary) and ROUGE-2 (for fluency purpose). Second, the evaluation metric to compare the summary generated from the model according to Algorithm 2, with the ground truth summary is ROUGE-N, $N \in \{1, 2\}$. This also allows us to compare our results with other state-of-the-art papers since ROUGE is a standard.

VI. Implementation Details

A. Data loader

To handle the loading of the training data, we focused on parsing the raw data from DUC and creating a dataset that would be easy to use in the training phase. This was not as straightforward of a task as we would have hoped. While the dataset from DUC appears to be properly working XML, we were unable to load it using several python modules due to discrepancies in the XML structure. When we first glanced at this project, we thought that because it appeared to be XML we would be able to easily extract the data. However, once it was obvious that this was not going to be possible through XML we decided to load the files and create a task specific parser. This parser is far from perfect, however.

We were then able to match each of the articles to their respective summaries. Once done, we then needed to break the large text dumps into sentences. Our original naive implementation was to just split the text on periods. We recognized the issues surrounding this and decided to use a third party library to handle this. We settled on the Natural Language Toolkit (NLTK) and changed our system to use it. After the text was split into sentences, ROUGE scores were calculated using the *Pyrouge*[5] Python module. This data was saved to disk to make the process of loading data much faster. This process, when started from scratch, can take well over an hour to complete but once done it no longer needs to be repeated.

B. Natural Language Toolkit (NLTK)

As previously mentioned, we used NLTK to accurately analyze the text and divide it into sentences. This is the most popular Python module for natural language processing. It makes use of a parser tree to analyze sentences and their structure and contains multiple libraries and word dictionaries, such as WordNet. NLTK provides a sentence tokenization algorithm for Sentence Boundary Disambiguation[6], which is approximately 95% accurate at splitting sentences. It was significantly more accurate at breaking sentences than our original naive solution. For instance, the example sentence, "Sen. Dianne Fienstein ran several meetings in Congress from 2:30 p.m. until 2:00 a.m. the following morning." would be broken up into 6 sentences: "Sen", " Dianne Fienstein ran several meetings in Congress from 2:30 p", "m", " until 2:00 a", "m", " the following morning". In this case the system would lose the meaning of the sentence as a whole, and split it into meaningless phrases, which are guaranteed to not be in the summary. NLTK, however, would return the sentence as a whole, thus maintaining the structure of the sentence and therefore improving the embedding of the sentence, aiding our network to make a more accurate prediction of its salience score.

C. Pyrouge

The original author of ROUGE published a Perl script which is unanimously used in text summarization as a golden standard. We used a Python wrapper around that script called Pyrouge, in order to have the same baseline evaluation metric as all the other systems in academia and the original author of the paper. Following Zhang et al. proposed settings we ran ROUGE-1.5.5 with options: -n 2 -m -u -c 95 -x -r 1000 -f A -p 0.5 -t -l 100.

[5] https://pypi.python.org/pypi/pyrouge/0.1.0

[6] https://en.wikipedia.org/wiki/Sentence_boundary_disambiguation

This runs ROUGE-1.5.5 for up to 2-grams, using the Porter Stemmer algorithm[7], for up to 100 words and include skip-grams in the calculation.

D. Padding issues

We ran into a mismatch in the padding size between the testing set and the training set. Because of the way we pad sentences to the longest sentence in the training dataset so that we are able to train the neural network, larger sentences in the testing set would be unable to fit into the smaller input of the neural network. We ultimately decided to just remove the very few sentences that were too long. Of the over 15,000 test sentences, only 111 of them were longer than the longest training sentence.

E. Building the model

For the model implementation we used Keras because of its ease of use and reliability. Our convolutional layer is essentially 1D, but we used a 2D convolutional Keras layer of 400 filters with a kernel window of (m, 300), where 300 is the embedding size and m is the number of words to convolve over, which was set to 3 by Zhang et al. This means that we are performing 1D convolution taking only 3 whole word embeddings. We did this in order to avoid reshaping problems with Keras, so we treated the embedded documents as 2D images with 1 channel. After that we used a MaxPooling2D layer, maintaining consistency with the previous Conv2D layer, pooling over all the filters keeping only 1 number from each convolution, as described in section V. B. Finally, we flatten the feature map from the max pooling layer and feed it into a single output FC neuron with dropout, which has an l2 regularization on its weights. Finally a sigmoid non-linearity is used as an activation function in the output neuron.

F. Training the model

As proposed by Zhang et al. we used the binary cross-entropy loss to minimize our problem, using the Adadelta update rule. They did not publish the hyper-parameters used in their paper, so we had to experiment. After trying multiple learning rates, as well as the Adam update rule, Adadelta with learning rate of 1.0 and a decay of 0.0 (recommended hyper-parameters from Keras) seemed to perform optimally. In order to evaluate our training performance we used a subset of DUC2001 for validation. We used a 80%-20% train-validation split. Since this is essentially a regression task, we used the Mean Absolute Error metric in order to evaluate our model on the validation set during training, after each epoch. This evaluation is does not fully reflect the performance of the model for our task, but gives an indication of the models ability to regress the salience scores. Ideally we would like to be able to do a test on a subset of DUC2002 using Algorithm 2 after every epoch, however we were limited by compute power and this would significantly slow down our productivity. Finally we trained most models for 10 epochs, as that seemed enough for them to converge, using a batch size of 256.

G. Hyper-parameters

The paper sets the hyperparameters as displayed on Table II.

Hyperparameter	**Value**
Number of filters l	400
Window size m	3
Dropout probability p	0.5
Regularization constant λ	Not mentionned
ROUGE weight α	0.5
First activation σ_1	ReLU
Second activation σ_2	Sigmoid

TABLE II: Hyperparameters proposed by Zhang *et al.*

Zhang *et al.* [1] did not specify the value they chose for the regularization constant, thus we tried different values (see VIII.)

H. Testing the model

When testing the model, we pre-process the test sentences, as described in section V. D, to fit them to the first layer. We embed the test sentences with word2vec, and then we expand the test matrices by adding 1 channel to each sentence matrix in order to match the required shape for convolution. Since the maximum size of the training sentences and test sentences was different, the test matrices were padded so that all the embedded matrices (see V.E.).

VII. Results - Replicated

A. ROUGE scores - Replicated

Table III shows the scores we obtained compared to those produced by the original paper [1] and the state of the art results.

System	**ROUGE-1**	**ROUGE-2**
Cssnsa.v2 [16]	48.05%	22.83%
Wpdv-xtr.v1 [**?**]	47.75%	22.27%
Zhang et al. [1]	**48.62**%	**21.99**%
Our implementation	42.48%	16.96%
Random summaries	32.14%	11.39%

TABLE III: Single Document Performance on DUC2002.

We observe a difference of 6.14% for ROUGE-1 and 5.03% for ROUGE-2 between our results and the original paper results. This discrepancy will be discussed in section IX. Random summaries is a naive text summarizer we built to get a baseline performance. It randomly selects sentences from the document until it hits the 100 word limit. Our implementation performs clearly better than the baseline, indicating that our method learns the features of a good sentence for a summary.

B. Example of predicted summary

In Table V, we provide an example of a predicted summary. We provide a comparison with the ground truth summary.

[7] https://tartarus.org/martin/PorterStemmer/

We let the reader find which one is artificial and which one is human (the answer is at *the bottom of this page*[8]).

Summary 1	**Summary 2**
"Rice was named the game's most valuable player. Joe Montana's 10-yard touchdown pass to John Taylor with 34 seconds left provided the winning margin. The victory was achieved through the brilliance of Montana and Jerry Rice, the wide receiver who caught 12 passes for 222 yards, both Super Bowl records. In a dramatic finish, the San Francisco 49ers football team won the Super Bowl on Sunday with a 20-16 victory over the Cincinnati Bengals. The winning score came at the end of a 92-yard drive, engineered by Montana, whose touchdown pass gave him a Super Bowl record with 357 yards passing."	"The San Francisco 49ers won the Super Bowl Sunday with a dramatic 20-16 victory over the Cincinnati Bengals. Joe Montana's 10-yard touchdown pass to John Taylor with 34 seconds remaining provided the win. The pass gave Montana a Super Bowl record with 357 yards passing. The victory was achieved through the brilliance of Montana and Jerry Rice, the wide receiver who caught 12 passes for 222 yards, both Super Bowl records. Rice was named the game's most valuable player. It was the fifth straight win for a National Football Conference team and the third Super Bowl win for Coach Bill."

TABLE IV: Guess which summary is the predicted summary of an article from *The Times Herald*, January 23, 1989

C. Example of summaries our model failed

We believe there is a slight issue with our algorithm when looking at documents with significant numbers of quotations. For example, we have provided a summary that sees this problem.

Predicted Summary	**True Summary**
"Superlatives? Another? See how big it is. Learn its age. Look at light created 14 billion years ago and arriving at your eye only this instant. Ride along to the beginning of everything, almost, to the Big Bang of 10 or 20 thousand million years ago. See stars born and see them die. Watch galaxies form. Follow the flow of gas into the Milky Way's halo. Peer deep into the universe, past so many stars in our own galaxy that it would take a person more than a lifetime to count, to a hundred billion other galaxies each with about 200 billion stars."	"The long delayed launching of the Hubble Space Telescope (HST) has evoked a burst of superlatives from the nation's scientists. From James Westphal at the California Institute of Technology we hear, "No one ever made anything that good." Edward Weiler, the chief NASA scientist on the HST project declares, "On a clear, dark night you can see a flashlight from two miles away. With a 28-power telescope you can see it on the moon 250,000 miles away. With the HST you could see a firefly in Australia from Washington." Others hail HST as the greatest thing since Galileo's telescope in 1609."

TABLE V: Poorly summarized article juxtaposed with the true summary (*Indiana Gazette*, August 8, 1989)

D. Training

Fig. 3 shows the mean average error and the loss for both training and validation.

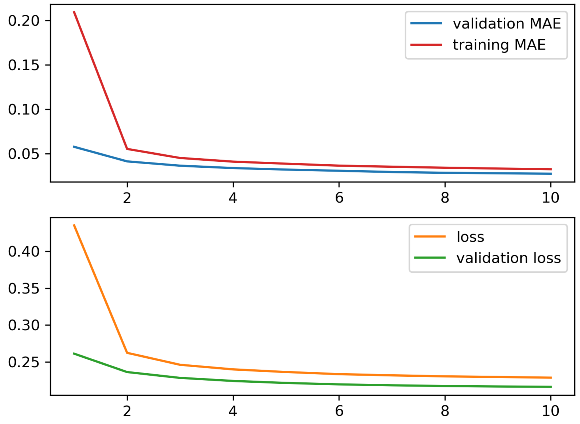

Fig. 3: Mean average error and loss value for training and validation - Replicated architecture

From the above training plots it can be deduced that the model converges quite quickly using the default parameters of the Adadelta update rule. The MAE curves do not add any more information than the loss curves as the cross-entropy loss is proportional to MAE if the output variable is Normally distributed. An interesting thing to note in these plots is that the validation loss is lower than the training loss. This can be attributed to the fact that Keras holds our the last 20% of the training set for validation. It is likely that the tail of our dataset is easier to fit than the rest. This, however, does not really affect our results as we are only interested in the trends of these curves.

VIII. Results - Extensions

We explored variations of the proposed model by modifying some parameters. This section presents our ROUGE-1 and ROUGE-2 results for different window sizes, dropout probabilities, numbers of filters and regularization constants.

A. Convolution window size investigation

In this section, we explore how the window size of the convolutional filters influence the ROUGE-1 and ROUGE-2 results (see Table VI). The default implementation (from Zhang *et al.*) sets the window size to 3×300 but increasing it up to 190×300 (the length of the longest sentence), we observed results very close to the original implementation and even surpassed their performance for ROUGE-2. This hyperparameter is extremely important since it defines how many words we "encapsulate" in the convolution phase.

[8] Summary 1 is the predicted summary!

Window size m	ROUGE-1	ROUGE-2
3 (Default)	42.48%	16.96%
10	41.99%	16.86%
50	42.19%	16.82%
100	42.21%	16.88%
190	**47.51**%	**22.41**%

TABLE VI: Extended results for different window sizes

B. Additional variations

We decided to vary the size of the dropout (**p**), number of filters (**l**), and regularization factor (**lambda**) to see how these hyperparamters impacted the scores for both ROUGE-1 and ROUGE-2. In our original training of our Neural Network, we noticed poor generalization and as such decided to try and adjust the dropout and regularization factor (it should be noted these numbers are with our nearly optimal CNN — not the original). We were also interested in how the added features of the different number of filters would increase or decrease the scores.

As can be seen from the chart below, changing these hyperparamemters led to very little change in both ROUGE-1 and ROUGE-2 scores. Due to this, we experimented more with changing the window size which led to significant changes in performance. However, given enough time, we would have liked to potentially perform a more exhausted search of hyperparameters in these domains.

p	l	lambda	ROUGE-1	ROUGE-2
0.5	400	0.01	42.48%	16.96%
0.1	400	0.01	42.60%	17.13%
0.5	200	0.01	42.45%	16.95%
0.5	400	0.1	42.56%	17.03%

TABLE VII: Extended results for different dropout probabilities p, numbers of filters l, regularization constant λ and σ_2

IX. Discussion

Our re-implementation of the original paper from Zhang *et al.*, provided reasonable scores, proving that the idea of using a CNN for document summarization is effective. However, the performance results we achieved did not match the ones stated in the original paper.

Since the original authors did not publish the steps they took in order to pre-process the DUC corpus, it is almost certain that we do not have identical training data for our models, even though we are using the same dataset. It is possible that the original authors had a custom parser for this dataset, whereas we had to build our own, using other generic NLP libraries such as NLTK, which are not perfect. In addition, training a CNN requires the tuning of multiple hyper-parameters, some of which are probably different from what the original authors had, such as, learning rate decay for Adadelta and regularization coefficient for the output FC layer. Another limitation is that the largest sentence in the testing corpus was larger than the largest sentence in the training corpus, therefore we discarded the 2 documents which included those sentences from our testing phase, and padded all the other test sentence embeddings to match the size of the training ones. This might be a very simplistic solution which might have affected our performance. All these factors were the only ones not specified in the original publication, hence they must account for the discrepancy in performance.

An interesting find of our project was to use filter sizes of 190×300, which outperformed the ROUGE-2 score of Zhang *et al.* on the DUC2002 dataset. It almost matched the state-of-the-art ROUGE-1 score. Our system ranks 3^{rd} and 2^{nd} in ROUGE-1 and ROUGE-2, respectively. These are reasonable results since instead of only looking at 3-4 words at each convolution, in order to extract the sentence's features, we look at the sentence as a whole using a spatial fully connected layer. So each of our 400 filters generates a scalar feature from the sentence embedding. This architecture has a significantly larger representation capacity than all the other models we tried. It totals to about 22.8 million parameters to optimize. This is very computationally expensive, therefore we used UC Berkeley's Savio super cluster, which trained it to 100 epochs in about 15 minutes. It is important to note that the testing phase was done on our laptops, which implies that the inferencing computational load does not render our model impractical, thus it can be used in the real world.

Another issue we came across during this project is that the DUC datasets are small datasets compared to other datasets used in computer vision with Convolutional Neural Networks: for instance the MNIST dataset has 60,000 training instances, so five more times training data than the DUC2001 dataset that we used for training. Furthermore, we do not have a great ratio between the 45% training data and the 55% testing data. This makes them hard to significantly improve the ROUGE scores.

In addition to that, we believe that the ROUGE metric does not compare the semantic meaning of sentences; ROUGE is mainly a N-gram counting method. For instance the ROUGE score of the sentences "Mr. President visited France." and "The Head of state traveled to Paris." will be 0. We thought that a new metric combining the benefits from both ROUGE (word count) and word embedding (the dot product between two latent vectors computes the semantic and syntactic similarity between the two embedded words) would capture more of the meaning of the sentences and therefore be more accurate at predicting, which sentences are representative of summaries. Furthermore, our testing method is flawed in the sense that it does not take into account the order of the sentences in the summary. Essentially it makes a blind ranking of the sentences independent of each other. This is one of the disadvantages of the CNN architecture vs as RNN language model.

In the future, we would like to build a system with which we can evaluate the system by building summaries on unseen documents after each epochs, and plotting the performance vs epochs. This would be the ideal evaluation metric for this task and would be a great tool for fine tuning the network and achieve even greater results. Another option is to explore training our own word embedding for this task. We did not prioritize this task because Google's pre-trained word2vec has a state-of-the-art word representation, which accurately captures the meaning of words. Hence, we considered this to be a detail improvement rather than an interesting aspect of this analysis. On the other hand, it would be of critical importance to have more time and computational capability to perform random search on all the hyper-parameters of the CNN, in order to extract as much performance as possible.

X. Conclusion

In this project we successfully built and trained our first deep learning architecture for NLP. This project was both challenging and thrilling since it explored neural networks, machine learning paradigms and NLP standard techniques. Conducting this implementation project from beginning to end taught us the challenges of pre-processing raw data, implementing a model using Keras, training and tuning this model and analyzing the importance of its hyper-parameters. Indeed, exploring the influence of parameters of any kind (probabilities, regularization, core hyperparameters) on the ROUGE-1 and ROUGE-2 results raised many interesting concerns and findings.

Looking at this project from a high-level point of view, we produced results obtained from building random summaries which, to the best of our knowledge has never been published and is a useful baseline for the interpretability of the ROUGE-1 and ROUGE-2 scores. Above all, we are proud to have reached state of the art results, ranking second right behind Cssnsa.v2 for ROUGE-2 [16].

The first step to any research project is to read about the state-of-the-art research from experts. The opportunity to open source the code we wrote will benefit not only data scientists working on automatic summarization (financial analysts, researchers, journalists, news reporters, politicians, academia, etc.) but also to the entire NLP community since we produced method to pre-process the DUC2001 and DUC2002 datasets (data loading and salience score computing).

For next steps, we propose to deepen the network by adding additional convolutional layers, applying to multi-documents and exploring how convolutional neural networks can apply to abstractive summarization.

XI. Acknowledgment

We would like to acknowledge the department of Computer Science of the University of California, Berkeley, the National Institute of Standards and Technology for the DUC datasets and the authors of the original paper.

XII. Link the the Github repository

https://github.com/alexvlis/extractive-document-summarization

References

[1] Y. Zhang et al, *Extractive Document Summarization Based on Convolutional Neural Networks*, IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, p. 918-922, 2016.

[2] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[5] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS 2013.*

[6] Chen, K., Corrado, G.S., Dean, J., & Mikolov, T. (2013). Efficient Estimation of Word Representations in Vector Space. *CoRR, abs1301.3781.*

[7] R. Nallapati, F. Zhai and B. Zhou, "SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents", *The Thirty-First AAAI Conference on Artificial Intelligence*, 2016.

[8] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou, "Ranking with recursive neural networks and its application to multi-document summarization", in *Proceedings of the 2015 AAAI Conference on Artificial Intelligence. AAAI*, 2015.

[9] M. Denil, A. Demiraj, N. Kalchbrenner, P. Blunsom, and N. de Freitas, "Modelling, visualising and summarising documents with a sin- gle convolutional neural network," *arXiv preprint arXiv:1406.3830*, 2014.

[10] Larochelle, H. Bengio, "Classification using discriminative Restricted Boltzmann Machines", *Proceedings of the 25th international conference on Machine learning - ICML*, p.536, 2008.

[11] S. Verma and V. Nidhi, "Extractive Summarization using Deep Learning", *arXiv:1708.04439v1*, 15 Aug. 2017.

[12] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, 2004, pp. 74–81.

[13] Lin, Chin-Yew and Franz Josef Och. 2004a. Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, Barcelona, Spain, July 21 - 26, 2004.

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

[15] Y. Li and S. Li, "Query-focused multi-document summarization: Combining a topic model with graph-based semi-supervised learning," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, p.11971207.

[16] J. M. Conroy, J. D. Schlesinger, D. P. O'Leary, M. E. Okurowski *Using HMM and Logistic Regression to Generate Extract Summaries for DUC* in Proceedings of the document understanding conference.

XIII. Appendix

In this appendix, we show the plots we get by training the convolutional neural network with different parameters. These plots were used to verify that the CNNs were training properly and see the overall trends through the epochs. this gave a good indication that further training past 10 epochs was not going to result in measurable gain.

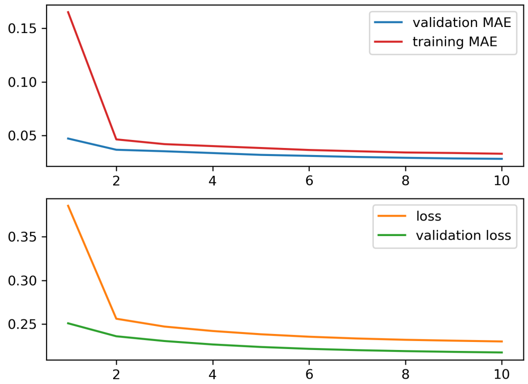

Fig. 4: Mean average error and loss value for training and validation - Window size $m = 10$

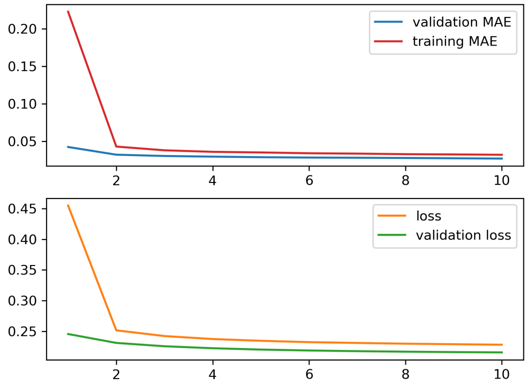

Fig. 5: Mean average error and loss value for training and validation - Window size $m = 50$

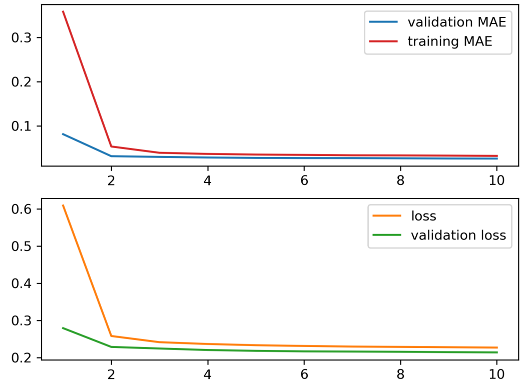

Fig. 6: Mean average error and loss value for training and validation - Window size $m = 100$

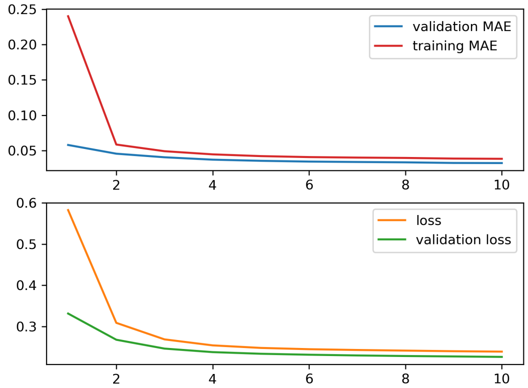

Fig. 7: Mean average error and loss value for training and validation - Regularization constant $\lambda = 0.1$

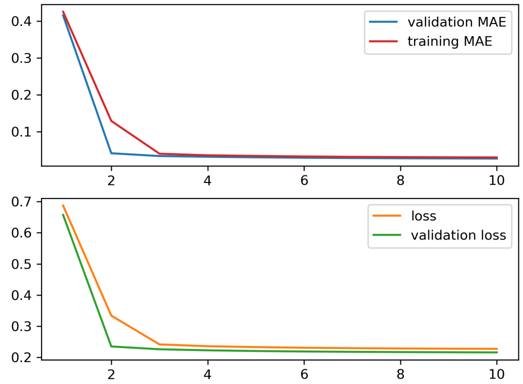

Fig. 8: Mean average error and loss value for training and validation - Dropout probability $p = 0.1$

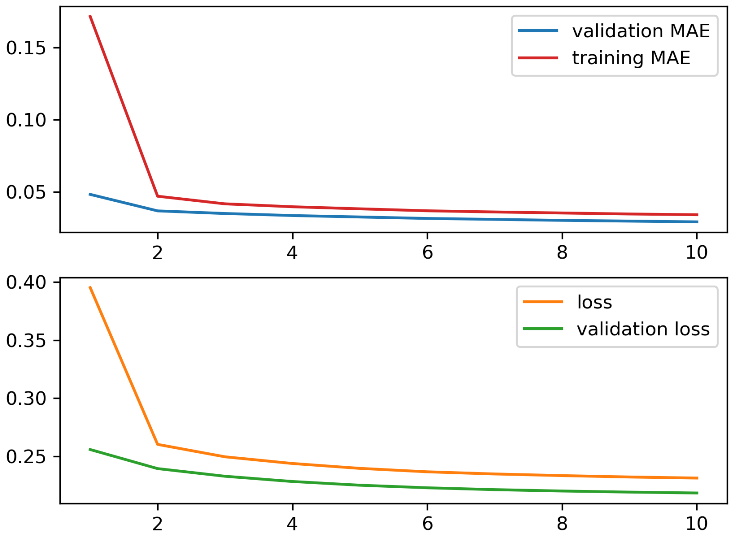

Fig. 9: Mean average error and loss value for training and validation - Number of filters $l = 200$

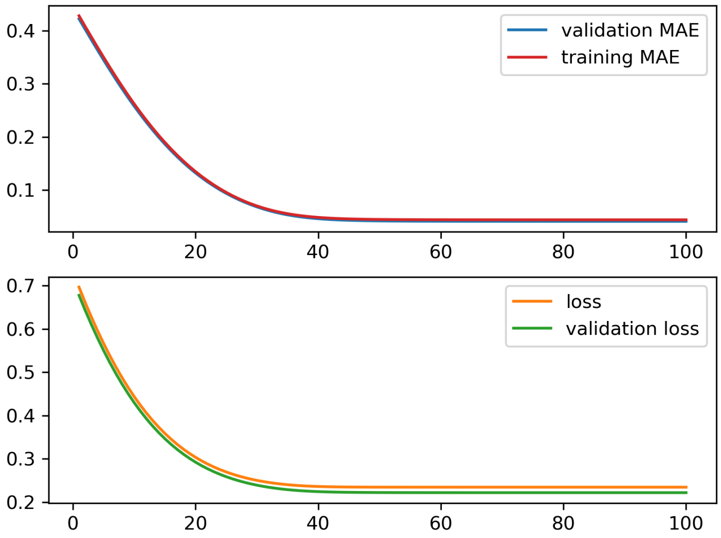

Fig. 10: Mean average error and loss value for training and validation - Window size $m = 190$