Evaluation of Neural and Non-Neural Techniques for Extractive Single Document Summarization

Aditya Saraf and Ron Fan
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195, USA
{sarafa,ronbo}@cs.washington.edu

Abstract

Current approaches to single document summarization generally belong to one of two categories of models. The first, neural models, typically involve encoding sentences and documents through recurrent or convolutional neural networks and then decoding a summary. The second, combinatorial or non-neural models, attempt to solve the problem of summarization algorithmically by posing the task as a variant of some common combinatoric problem, such as Maximum Coverage or Knapsack. We seek to compare these techniques across many axes, including runtime, performance, and the ability to generalize to new domains. We apply neural and non-neural extractive single document summarization models to two shared datasets and extract internal model scores to facilitate manual analysis of each model's tendencies in document summarization. We also design a visualizer that presents this information in an easily-navigable format.

1 Introduction

Single document summarization is a problem with many obstacles, and as a result, many distinct approaches. These approaches can generally be distinguished as either neural or non-neural models (Allahyari et al., 2017). Despite significant differences in model design and summary emission, many models are able to achieve results respectably comparable to state-of-the-art results (Hirao et al., 2013; Cheng and Lapata, 2016). This suggests that these models may have different strengths and weaknesses which can be augmented using knowledge of other successful approaches. Single document summarization techniques can be considered in three groups. For our work, we opted to restrict experimentation to extractive summarization techniques only. Abstractive summarization and extractive summarization with compression both require models to be capable of more than identifying key points in a document and summarizing those points, which is what we want to focus on.

1.1 Abstractive Summarization

Abstractive summarization refers to document summarization that generates novel sentences describing the document. This is almost always accomplished with neural models due to the lack of effective non-neural generative models. Example abstractive summarization models include a variety of RNN-based encoder-decoder models, sometimes with attention over inputs and outputs (Paulus et al., 2017). These models encode input documents as a single hidden layer and decode a sequence of words from the hidden state.

Abstractive summarization seems to be inappropriate for the purpose of comparing with non-neural models due to the on-going problem of coherence in longer generated sections (Paulus et al., 2017). We feel that the problem considered to be most challenging in abstractive summarization is not forming an accurate understanding of key points in a document, but rather, building readable sentences from a given set of talking points. Furthermore, combinatorial models are typically restricted to extractive summarization.

1.2 Extractive Summarization with Compression

Extractive summarization with compression refers to document summarization that generates novel sentences by applying grammatical constraints and rules to the original document. Some approaches to extractive summarization techniques convert textual units derived from the original document into well-formed summaries by using anaphoricity constraints to reduce the frequency of incomprehensible anaphoras in summaries and

enforcing grammaticality using rule-based grammaticality constraints (Durrett et al., 2016).

We chose to exclude compression techniques from our analysis due to what we perceived as an artificial, albeit very good, improvement of cohesion and ROUGE scores by enforcing a set of constraints which are difficult to generalize. We also felt that the implementation of such a system would introduce an unnecessary layer of complexity to our analysis.

1.3 Extractive Summarization

Extractive summarization is a summarization technique that forms summaries using chunks of texts taken directly from the input document. Extractive summarization techniques commonly extract at a sentential level, although they may also choose to extract individual words (Nallapati et al., 2017; Cheng and Lapata, 2016).

With extractive summarization, our neural model can make summaries that are identical to a concurring non-neural model. Although the resulting summaries may have lower levels of cohesion due to anaphoras and noise in documents, we believe they are the most fair for comparing to non-neural models which extract sentences from documents and make no modifications.

Our hypothesis was that combinatorial approaches would perform better at extractive summarization, since they more clearly exploited the structure of the problem. To this end, we built a simple combinatorial model and compared it against a state of the art neural model.

2 Related Work

Although much work has been done on neural and combinatorial models for extractive SDS[1], to the best of our knowledge, our paper is the first attempt to compare the different approaches on the same datasets. Our neural model is SummaRuNNer, which will be explained more in-depth in section 4 (Nallapati et al., 2017). Our combinatorial model was built from scratch, based on a reduction to the Maximum Coverage Problem, as explored in (Takamura and Okumura, 2009). Our MCP model is quite simple – we'll briefly survey some other state of the art combinatorial models.

Hirao et al. (2013) produce a model based on the Tree Knapsack Problem, which is similar to the traditional project selection problem taught as a common application of network flow. Their approach uses significant linguistic sophistication: they represent the document first as a Rhetorical Structure Theory Discourse Tree (REP-DT) and then convert it into a Dependency Discourse Tree (DEP-DT). They then trim the tree to preserve only the key semantic information, with the added benefit that anything that the summary semantically relies on (to be understood) will likely be included. Our goal was to implement a simpler combinatorial algorithm that could make use of our large training corpus, so we didn't use this approach.

Durrett et al. (2016) produce a custom ILP model based on information compression and anaphoracity constraints. These anaphoracity constraints deal with words (such as pronouns) that refer to information found earlier in the document, which might confuse readers if left out. Their model is extractive at the subsentential level - they compress sentences and rewrite pronouns in order to include more content and improve the summarys readability. Their model performed very well and could easily accommodate word weights, as they also had access to a large corpus (the paid New York Times corpus). But since our goal was to build a simple combinatorial system, we choose to instead reduce text summarization to MCP.

3 Neural Model

For the neural model we use for our later comparisons, we chose to use three models derived from the SummaRuNNer model introduced in Nallapati et al. (2017) and partially implemented by Zhao (2018). The SummaRuNNer model is designed for extractive single document summarization and generates binary classification labels indicating whether a sentence is included in the final summary or not.

3.1 RNN-RNN

This model is most similar to the one described in Nallapati et al. (2017). In the first layer, each sentence in the document is converted to a word embedding and fed into an RNN layer with bidirectional GRUs which encodes the words in the sentence. A 1D adaptive max pooling is applied and the result for each sentence is fed into another RNN layer which forms a document representation through a fully-connected linear layer. Finally, a feed-forward classifier determines emis-

[1] Single Document Summarization

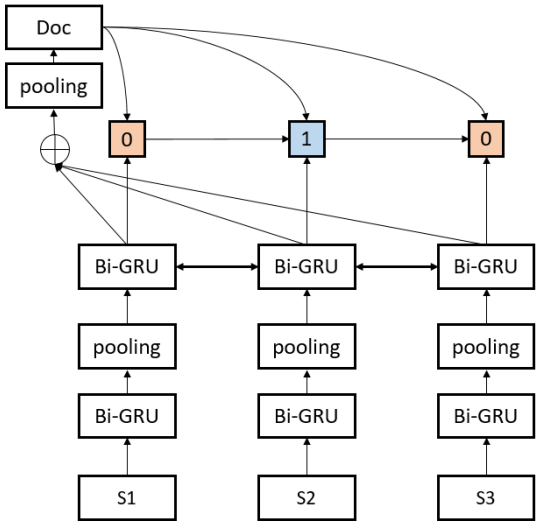

Figure 1: RNN-RNN: A two-layer RNN model using bidirectional GRUs. Figure from Zhao (2018).

sions by calculating a probability from a set of component scores made through linear and bilinear transformations.

3.2 CNN-RNN

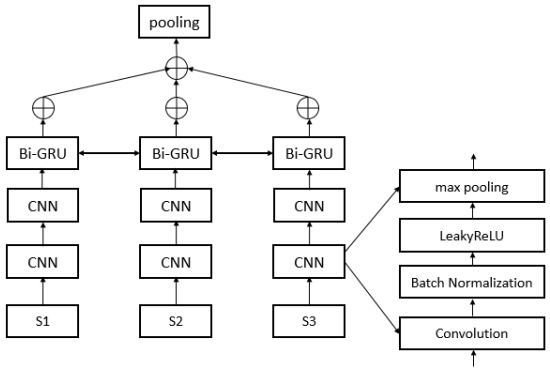

Figure 2: CNN-RNN: Two convolutional layers feeding into a recurrent GRU layer. Figure from Zhao (2018).

This model uses convolutions to encode sentences. The model takes in sets of words and performs two one-dimensional convolution operations on them. The first convolution reduces the number of channels and the second one simply convolves on the same set of channels again. Each convolution is followed by a 1D batch normalization and a LeakyReLU activation. The result of the convolution operations on each sentence is then fed into an RNN layer with bidirectional GRUs. The document encoding is formed through a fully-connected layer which applies another 1D batch normalization and a tanh activation. As before, a feed-forward classifier determines emissions.

3.3 RNN-RNN with Attention

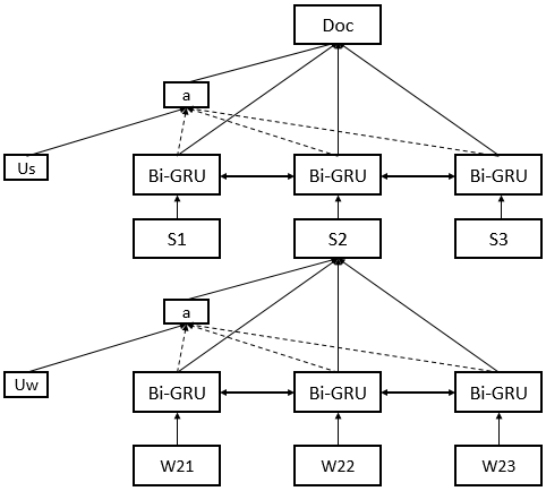

Figure 3: RNN-RNN with Attention: A two-layer RNN with attention. Figure from Zhao (2018).

This model is similar to the RNN-RNN model. The model has been augmented with attention, which is implemented using the words and sentences in the document as queries for an attention layer.

3.4 The SummaRuNNer Classifier

All three models we examined share the same SummaRuNNer-inspired classifier. The classifier is used to generate the emissions for each label.

$$
\begin{aligned}
P(y_j = 1 \mid \boldsymbol{h}_j, \boldsymbol{s}_j, \boldsymbol{d}) = \sigma(&W_c \boldsymbol{h}_j && \text{(content)} \\
&+ \boldsymbol{h}_j^T W_s \boldsymbol{d} && \text{(salience)} \\
&- \boldsymbol{h}_j^T W_r \tanh(\boldsymbol{s}_j) && \text{(novelty)} \\
&+ W_{ap} \boldsymbol{p}_j^a && \text{(abs. pos)} \\
&+ W_{rp} \boldsymbol{p}_j^r && \text{(rel. pos)} \\
&+ b) && \text{(bias)}
\end{aligned}
\tag{1}
$$

The classifier generates a probability through the formulation shown in Equation 1. Each of these components is assigned an interpretable definition by Nallapati et al. (2017). Each component is scored independently and the resulting scores are combined into a probability through a sigmoid function. We modified our models to track

and store these component scores which form the probabilities that the model uses to make its top-k decisions.

Content: The content weights measure the information content of a sentence. The layer is a linear transformation.

Salience: The salience weights measure the prominence of a sentence with respect to the whole document. The layer is a bilinear transformation combining the sentence encoding with the document encoding.

Novelty: The novelty weights measure the novelty of a sentence given the current summary representation. The current summary is squashed using a tanh activation and combined with the sentence encoding in a bilinear transformation.

Absolute and Relative Positional Importance: The absolute and relative positional importance scores are derived from 50-dimensional positional embeddings. The positional embeddings are learned through linear transformations. Absolute positional embeddings are based on numerical positions in the document. Relative positional embeddings divide the document into 10 segments and score sentences in the same segment identically.

3.5 Training

We trained our models with a hidden unit size of 200, batch size of 32, learning rate of 1e-3, and 5 epochs. Training for each model took approximately 48 hours.

4 Combinatorial Model

Our combinatorial model was based on a reduction from text summarization to the Weighted Maximum Coverage Problem (MCP). We first provide a formal definition of (weighted) MCP:

> Input: A collection of sets $S = \{S_1, S_2, \ldots, S_n\}$, an integer k, and a weight function, w, that assigns a weight to each $e_j \in S_1 \ldots S_n$.
>
> Output: A subset $S^{'} \subseteq S$ such that $\left|S^{'}\right| \leq k$.
>
> Objective: The weighted sum of all unique elements in all $S_i \in S^{'}$ is maximized.

Text summarization can be naturally reduced to MCP. Each sentence in the document is treated as a set of words, and the length of the summary is set to k. Then the goal is to find k of the sentences that maximizes the weighted sum of all unique words contained in the summary.

The MCP relies on the **Coverage Heuristic**. The Coverage Heuristic breaks the document into elementary units (such as words) and attempts to choose as many important unique units as possible, while retaining some of the original structure to ensure grammaticality. In our formulation, the elementary units of the document are exactly the words in the document, and we constrain the model to pick entire sentences to ensure grammaticality. The two assumptions behind this heuristic are:

1. Having a high number of important unique words likely means you have a high amount of semantic content.
2. Including a word multiple times in a summary does not increase the semantic content of the summary.

These assumptions are obviously not exactly true for real documents – semantic information may come from key phrases or sentences (the sum of the parts may be less than the whole) and using the same word in different contexts may increase the semantic content of the summary. But this is still a useful heuristic in practice.

Since the MCP is NP-complete, there exists no efficient algorithm to compute an optimal solution. For this reason, we frame the problem as an Integer Linear Program and use an ILP solver. Our ILP formulation is as follows:

$$\text{Maximize: } \sum_{e_j} w\left(e_j\right) * y_j$$

$$\text{Subject to: } \sum x_i \leq k$$

$$\sum_{e_j \in S_i} x_i \geq y_j$$

4.1 Learning Word Weights

We learned weights for the words[2] with logistic regression on the CNN/DailyMail training set (~310,000 documents). We used an L2 loss function, and converged in 730 iterations. Training took around 11 hours.

5 Data

In this section, we describe (1) our primary dataset, (2) major limitations of our dataset, and

[2]We used UNK tokens for any word that appeared less than 5 times in the training corpus.

(3) an alternate dataset included to test the models' ability to generalize to a different domain.

5.1 CNN/DailyMail Dataset

Our primary dataset is a collection of 300,000 short news articles compiled from CNN and DailyMail. This dataset is a very popular recent dataset (Cheng and Lapata, 2016; Nallapati et al., 2017; See et al., 2017) for text summarization that was originally adapted from the reading comprehension domain. We follow the approach of See et al. (2017) in using the non-anonymized version instead of the anonymized version, as the anonymized version has to be preprocessed using a Named Entity Recognition system. The dataset contains very short articles (around 70 sentences on average) with short, abstractive summaries (around 3.75 sentences on average). We partitioned the dataset into a training set of around 310k articles, and a test set of exactly 10k articles.

The biggest advantage of this dataset is its size. With over 300,000 articles that take up more than 1.5 GB when uncompressed, this data allows for large scale learning that traditional, ~3,000 document sets cannot accommodate. We wanted to compare both neural and combinatorial models in their best settings, so this dataset was a natural choice. The dataset is also freely available (unlike the large NYT corpus) and widely used, especially in the past few years.

5.2 CNN/DailyMail Limitations

However, the dataset has several limiting issues. One major issue is that because the file sizes and summary lengths were so short, any reasonable extractive model would often generate largely overlapping summaries. This didn't give the models much room to differentiate – most documents had some sentences that were obviously the most content-laden. One possible extension of our work could be using both models to filter this dataset for "diverse" documents (those documents with significantly different summaries for the neural and combinatorial models), which would be a useful comparison metric for future research in extractive SDS.

Besides the short articles, the dataset also contains a lot of noise. Sentence boundaries are fairly imprecise, and a lot of metadata is present at the beginning of some articles, which causes SummaRuNNer to erroneously choose semantically empty pseudo-sentences. Of course, this might simply be exposing a legitimate issue with SummaRuNNer's over-reliance on the "First-K" heuristic. In addition to the metadata noise, the abstractive "summaries" aren't really summaries. The bullet points were meant to be read alongside the document – readers either read only the bullet points or the bullet points and the full article. Thus, authors often included novel information in the bullet points, which no extracted summary could replicated. To illustrate an accurate performance ceiling, we evaluate "Oracle" scores alongside our other models, where the Oracle summary is derived from the generated extractive labels.

5.3 Australian Law Database

We also make use of a database containing around 4,000 legal cases from the Australasian Legal Information Institute (ALII). This database was processed and hosted on the UC Irvine Machine Learning Repository[3]. Similar to Galgani et al. (2012), we treat the "catchphrases" as abstractive summaries. These catchphrases are designed similarly to tags – they allow for more powerful search tools. In contrast to the CNN/DailyMail dataset, the catchphrases are very short (often just a few words), but there are many more of them (8 per document). This smaller dataset is less noisy, as the catchphrases never contain novel information. The documents themselves are much longer, with a few outlier documents containing more than 10,000 sentences. This dataset allows us to test two aspects of the models:

1. The ability to generalize to a different target domain – there aren't enough legal cases to retrain the models.

2. How drastically the runtime changes with larger documents.

5.4 Pre-processing

Since both datasets only contain abstractive summaries, we designed a system to assign each sentence an extractive label. Our method was similar to Nallapati et al. (2017) in that we tried to greedily maximize the ROUGE score. Our algorithm to generate extractions was as follows:

1. Parse a single document and split into abstractive summaries (highlights) and actual content.

[3] https://archive.ics.uci.edu/ml/datasets/Legal+Case+Reports

2. Post-process sentences by doing an additional regex split on whitespace with punctuation lookback. Numerous sentences in the original dataset were not delimited by new line characters, so this is a necessary step to ensure our model deals with individual sentences.

3. Tokenize each sentence and highlight using the NLTK MosesTokenizer.

4. Build unigram, bigram, and trigram sets from the sentences and highlights.

5. Count matches between sentences and highlights, giving more weight to bigram and trigram matches.

6. Sort sentences by score.

7. Let S be the number of sentences and H be the number of highlights. The algorithm picks up to $\min(1, \text{floor}(S/2))$ sentences as part of the summary. It picks at least the H sentences with the highest scores. After sentence H is picked, subsequent sentences will only be picked if their score was at least $(80 + 3 * X)\%$ of the previously-picked sentences score, where X is the number of sentences currently picked.

This is a metric designed to pick the most likely sentences roughly in accordance with the size of the abstractive summary, while also allowing for particularly similar sentences to both be chosen for completeness. Due to the noise in the dataset described previously, the extracted summaries are not always ideal. However, we find that they are generally reasonably close to what human-selected summaries might be, and are suitable for our purposes.[4]

6 Results

This section will first explain our methodology and experimental setup, then present the results of our experiments.

6.1 Methodology

One of the challenges we faced was determining the length of the summaries. We tested the obvious approach of matching the Oracle's summary lengths, but such an approach wouldn't generalize to unlabelled text, so we also tried various arbitrary summary lengths (3, 6, or 9 sentences). We evaluated our models with ROUGE-1 Recall, F1, and ROUGE-2 Recall using the JRouge library[5]. Our primary metric was ROUGE-1 Recall. We also timed our models – since our MCP implementation was in Java, we ran 3 dummy runs to warmup the JVM, then timed and averaged the results of the next 10 runs.

We tested 5 models:

1. First-K: This model simply chose the first K sentences, where K is the length of the summary. This provided an extremely simple baseline approach.

2. Unweighted Greedy MCP: This model reduces summarization to MCP, but doesn't use weights and solves using a greedy algorithm instead of an ILP solver.

3. MCP: This model uses the learned weights discussed in section 3 and uses GLPK 4.5 to solve the MCP optimally.

4. Neural: This is the ATTN-RNN version of SummaRuNNer discussed in section 4 (the various neural architectures explored had almost identical performance).

5. Oracle: This model uses the extractive labels obtained from the system discussed in section 5. This serves as a performance ceiling to compare with the other models.

Table 1: Full results on the CNN/DailyMail test set with summary lengths set to three sentences. Runtimes reported in milliseconds.

Model	ROUGE-1	ROUGE-1 F1	ROUGE-2	Runtime
First-K	0.31104	0.24458	0.12518	35241
U.G. MCP[6]	0.42982	0.25597	0.13335	37093
MCP	0.44425	0.28429	0.15821	61398
Neural	0.45483	0.31671	0.18239	1048249
Oracle	0.68059	0.37807	0.38756	N/A[7]

6.2 CNN/DailyMail Test Set

We compared our models on the 10,000 document CNN/DailyMail test set. All runtimes are reported in milliseconds. As Tables 1/2 show,

[4] We also used a stoplist and tokenizer to preprocess the text. We tried lemmatization, but it had a negative impact on performance.

[5] https://bitbucket.org/nocgod/jrouge/wiki/Home

[6] Unweighted Greedy MCP

[7] Oracle summaries are generated in pre-processing

Table 2: Full results on the CNN/DailyMail test set with summary length matching the Oracle summary length. Runtimes reported in milliseconds.

Model	ROUGE-1	ROUGE-1 F1	ROUGE-2	Runtime
First-K	0.410247	0.252497	0.174275	35048
U.G. MCP	0.533134	0.237928	0.183332	38967
MCP	0.547118	0.262058	0.207809	71355
Neural	0.5397474	0.292089	0.229425	1145616
Oracle	0.68059	0.37807	0.38756	N/A

Table 3: ROUGE-1 Recall scores as summary length increases. Summary length indicated in number of sentences.

	Summary Length		
Model	3	6	9
First-K	0.31104	0.45118	0.58371
U.G. MCP	0.42982	0.58056	0.66126
MCP	0.44425	0.59074	0.66616
Neural	0.45483	0.58693	0.66516

the combinatorial model was much faster than the neural model, processing roughly 163 articles per second vs the neural model's 9 articles per second. Both models performed competitively, with the neural model having a slight edge on shorter summaries and the combinatorial model having a slight edge on size-matched summaries. Note that the neural model had a higher F1 score in both experiments, demonstrating that the combinatorial model prefers to choose longer sentences.

We also tested how the models' ROUGE scores increased as we increased the summary lengths. Table 3 summarizes our results. The Oracle summaries were on average 4.8 sentences, but neither the MCP nor Neural models could match the Oracle's ROUGE-1 even with twice as many sentences.

6.3 Australian Legal Cases

Lastly, we tested our non-baseline models on the Australian Legal Case database, to test asymptotic runtimes as well as the ability to generalize. The neural model was not retrained and the MCP word weights were not re-learned. Table 4 summarizes our results. The Oracle F1 score is much lower in this dataset than the previous one, because the abstractive summaries ("catchphrases") were much shorter, so the precision was much lower. As expected, the MCP model generalizes better, as only one part of it is learned (word weights). The neural model relies heavily on domain specific training data.

7 Visualizer

In order to more easily analyze the results we collected, we built a summary visualizer. We combined the relevant scores and metadata into a set of JSON files and put together a web-based visualizer[8].

The visualizer shows scores for each sentence when hovered. Content, salience, novelty, absolute positional importance, relative positional importance, overall probability, and MCP model score are all included. The correct sentences for the summaries are highlighted in yellow, and the current selection is highlighted in green. The left column shows the sentences chosen by the RNN-RNN with Attention model, the center column shows the entire document, and the right column shows the sentences chosen by the MCP model. The visualizer also counts the number of matches made by each model.

8 Error Analysis

In order to fulfill our goals of understanding the strengths and weaknesses of neural and non-neural single document summarization models, we ran each model on the dataset we created. Besides the output labels that each model generated, we also collected information on the model's internal calculations. For the neural model we extracted each of the component scores used to calculate a probability for each sentence. For the maximum coverage model, we extracted the final scores associated with each sentence[9].

8.1 Neural Model

In order to better understand the weaknesses of the neural RNN-RNN with Attention model, we computed the content, salience, novelty, and positional scores for each sentence in the dataset. We took

[8] Available at https://rococode.github.io/primeapeNLP

[9] These scores represent the combined weight of all unique words in the sentence, but if another sentence is already chosen, the weights of this sentence would change. This isn't represented in the visualizer, but explains why the top K highest sentences aren't chosen.

Table 4: Non-baseline model performance on the legal case dataset. Runtimes listed in milliseconds.

Model	ROUGE-1	ROUGE-1 F1	ROUGE-2	Runtime
MCP	0.56808	0.11932	0.17202	176708
Neural	0.54434	0.15179	0.18272	1015356
Oracle	0.66658	0.17366	0.32877	N/A

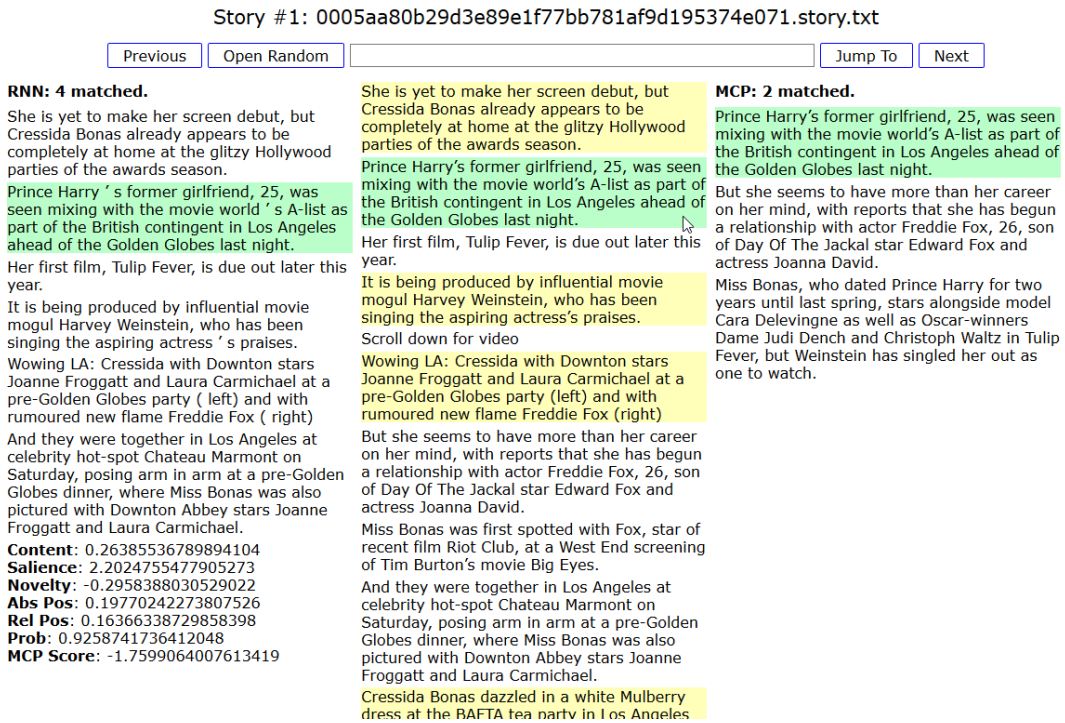

Figure 4: Our visualizer showing locations where the second sentence in the document is chosen and the scores associated with that sentence.

these scores and were able to make some interesting observations about common failures in the neural model.

Preference for early sentences: The biggest flaw in the neural model is the significant tendency to pick sentences near the beginning of a document. Many of the mistaken choices made by the neural model tend to be when it chooses metadata at the beginning of a document, such as the author name or timestamps. We believe that this inaccuracy stems from the formulation of the final classifier layer, which adds up the component scores to calculate a probability. The position of a sentence is influential several times in this sum, as part of the absolute positional importance, relative positional importance, and novelty. Absolute positional importance is an embedding-based score that constantly decreases as the document goes on. Relative positional importance is effectively the same type of score, except with segmentation into groups of sentences. Novelty, although not directly related to position, is calculated as a measure of redundancy of a sentence given the summary up to that point. Since novelty is measured according to the current state, earlier sentences tend to have better novelty scores since they are more likely to contain novel information. This is most clearly indicated by the fact that the first sentence of any document has a novelty score of 0 (the other sentences all have negative novelty).

Inconsistency in cross-document scoring: We noticed that our neural model is fairly inconsistent across different documents. In particular, metadata sentences like "By" and timestamps are often scored very differently despite being in the same position. While the model sometimes chooses sentences like "By" in the summary after allocating very high probabilities to them (sometimes upwards of 90%), it does not make this mistake consistently. The fact that two documents both starting with a single word sentence "By" can assign different scores to the sentence suggests that the salience component of our classifier may negatively affect the model's comprehension of the document.

8.2 MCP Model

The Maximum Coverage model has several weaknesses.

Preference for longer sentences: Since we budget by sentences instead of words, our MCP model always prefers longer sentences. This manifests in the significantly lower F1 scores when compared to the neural model. This weakness is

fairly easy to fix: we can modify our version of MCP to be Budgeted Weighted MCP, which is also easy to formulate as an ILP.

Weight Learning: Since we UNK words that appear infrequently, unseen NEs[10] are assigned a low weight, instead of the high weight that previously seen NEs have. The same logic applies to other features, such as Part-of-Speech (maybe verbs are content-heavy), word position, etc. We can fix this by adding more pre-processing (POS tagging, NER, etc.) and learning feature weights in our model.

Preference for Named Entities: One interesting example we found through the visualizer was an article on a baseball match. One sentence in the article just listed out the names of the players on the winning team, but had no important content. This sentence had an extremely large weighted sum, since some the Named Entities in the list were fairly popular (high weight) and there was a lot of them. One potential solution to this problem would be to compute the average weight of a sentence, and pick the k best such sentences – but this might make the model less robust to noise (an erroroneous sentence like "Donald Trump" would have a much higher score than "Donald Trump signed one of the most important Executive Orders").

Overlapping Sentence Structure: Sentences with a lot of important content, but only slightly different structure will not all be chosen – after the best such sentence is chosen, the others will all have much lower weights if they used a lot of the same words. This is an inherent weakness present in any model that reduces to MCP.

9 Conclusions

In this work, we used two very different datasets to compare neural models and a combinatorial model. Finally, we built a visualizer tool which simplifies the process of comparing the and non-neural decisions on the same document. From our work, we observed weaknesses in both models. The most prominent weaknesses were the excessive importance of position in the neural model, and the weaknesses surrounding Named Entities in the MCP model.

[10]Named Entities

10 Future Directions

There are several aspects of this paper that could be expanded upon. The most obvious improvement would be to replace the MCP model with the Compression/Anaphoricity model from Durrett et al. (2016). This would be a more even test – a SOTA[11] neural model against a SOTA combinatorial model. Another point of improvement would be using our error analysis to find adversarial inputs. Since the two models have distinct weaknesses, it should be possible to craft a set of documents such that the neural model heavily outperforms the MCP model on some instances and vice versa. For instance, a document with content-heavy sentences in the beginning, and a large list of Named Entities at the end would be adversarial to the MCP model and easy for the neural model. Such a document set could be an interesting "challenge" set for future researchers that would essentially test the edge cases of single document summarization.

We also think there is room for improvement on the neural model's final classification layer. Currently, our model decodes the document representation using the feed-forward classification layer from Nallapati et al. (2017). We believe that the formulation of probability is flawed due to the simple adding of each component score (content, salience, etc.). Each of these components is roughly on the same scale, as evident from our collected data, and they are added together without further weighting. This seems to be inaccurate, since it seems unlikely that all of these components should actually have the same degree of influence on the final model. It may be interesting and productive to add in an additional layer in the classifier that performs an extra linear or non-linear transformation on the component scores. This would allow the model to learn precisely how influential values like salience are supposed to be, and may result in improved performance.

[11]State of the Art

References

Mehdi Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. 2017. Text summarization techniques: A brief survey. *CoRR*, abs/1707.02268.

Jianpeng Cheng and Mirella Lapata. 2016. Neural

summarization by extracting sentences and words. *CoRR*, abs/1603.07252.

Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. 2016. Learning-based single-document summarization with compression and anaphoricity constraints. *CoRR*, abs/1603.08887.

Filippo Galgani, Paul Compton, and Achim G. Hoffmann. 2012. Combining different summarization techniques for legal text.

Tsutomu Hirao, Yasuhisa Yoshida, Masaaki Nishino, Norihito Yasuda, and Masaaki Nagata. 2013. Single-document summarization as a tree knapsack problem. In *EMNLP*.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *AAAI*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*.

Hiroya Takamura and Manabu Okumura. 2009. Text summarization model based on maximum coverage problem and its variant. In *EACL*.

Huaipeng Zhao. 2018. Summarunner. https://github.com/hpzhao/SummaRuNNer.