Policy Gradient Reinforcement Learning for Extractive Summarization

Kian Kenyon-Dean

April 24, 2018

COMP 767 Final Project

Profs. Doina Precup & Pierre-Luc Bacon

McGill University

Abstract

We explore the natural language processing task of *extractive summarization* (ES) in a reinforcement learning (RL) setting. ES has a natural extension to RL due to the fact that the primary evaluation metric, ROUGE, is a non-differentiable function that cannot be optimized directly with typical supervised learning methods. However, ROUGE can be used as a reward signal in an RL framework where the objective is for an agent to extract a summary by learning to maximize this reward on a training set. In this work, we experiment on a subset of the *CNN / Daily Mail* ES dataset and examine the application of linear policy-gradient reinforcement learning using the REINFORCE algorithm with function approximation. Using simple TF-IDF feature extraction methods, we find that the agent generalizes its experience from the training set onto the validation and test sets, performing at or above the level of the *Lead-3* baseline ES algorithm. Our examination of the learned weights and hyperparameters offers insights into future model design and feature extraction decisions with a lucidity that is too often lost with typical deep learning methods.

1 Extractive Summarization

The task of *extractive summarization* (ES) has the objective of selecting the sentences in an article that most adequately summarize the article. A technical analogy would pose that ES seeks to compress an article into a small subset of its sentences such that the amount of information lost from the compression is minimized. Formally, the task assumes a model is given an article $\mathbf{A} = \{x_1, \dots x_n\}$ consisting of textual units x_i (which can be words, phrases, sentences, etc.); here, we assume the x_i are sentences. In the popular *CNN / Daily Mail* dataset ([1]), the gold standard summary is an *abstractive*[1] summary of the document S^*, which consists of either three or four sentences.

Our formulation of this task poses the objective as producing a summary $\mathbf{S} = \{x_a, x_b, x_c\}$ of the document by selecting exactly three sentences x_j from $\mathbf{A}$. The goal is for this summary $\mathbf{S}$ to have the maximum possible ROUGE-1, ROUGE-2, and ROUGE-L scores[2] with the gold summary S^*; that is, to maximize the scoring function below:

$$\textit{score}(S, S^*) = \frac{1}{3}\Big(\text{ROUGE-1}(S, S^*) + \text{ROUGE-2}(S, S^*) + \text{ROUGE-L}(S, S^*)\Big) \tag{1}$$

Given the scoring function above, we can formally pose the objective as the goal of finding the optimal summary $\mathbf{S}$ defined by:

$$\begin{aligned} \mathbf{S} &= \underset{S}{\operatorname{argmax}}\, \textit{score}(S, S^*) \\ &\text{s.t.} \quad S = \{x_a, x_b, x_c\} \quad \text{where } a, b, c \in \{1, \dots, n\}, a \neq b \neq c \end{aligned} \tag{2}$$

Therefore, in an article with n sentences, there are precisely $\binom{n}{3}$ possible sentence combinations to select from. Since the optimal score is defined with respect to an abstractive summary, there is an upper bound on performance that is less than 1. This upper bound is defined by an *Oracle* algorithm that greedily extracts the three sentences that obtain the optimal mean-ROUGE score (see Algorithm 1, Appendix), making for an $O(n + (n-1) + (n-2)) = O(3n) = O(n)$ algorithm. This greedy expression is necessary because an algorithm that explores all combinations would be $O(\binom{n}{3}) \in O(n^3)$, which would be extremely inefficient. It would therefore be desirable to build a model that can also extract a summary at an $O(n)$ speed; at test-time, our proposed reinforcement learning model achieves this goal.

1.1 Related Work

In recent years, RL methods have become more and more popular for designing ES systems. [10] seems to have been the first attempt to systematically apply RL to this problem. While the authors did

[1] That is, the true summary is abstracted from the document, it is not made by extracting specific sentences or phrases. Thus, an extractive summarization system is limited by not being able to generate novel sentences, but confronts a simplified problem as we do not have to be concerned with grammaticality or logical consistency of generated text.

[2] See [4] for an explanation of each of these variants of the ROUGE metric.

not use the ROUGE score as the reward function in their work, they found that decent results could be obtained; however, significant feature engineering and parameter tuning had to be performed, and the model tended to converge to sub-optimal local solutions. In [9] the field observed more linear RL methods applied to the ES task, this time using the ROUGE score as the reward function. The authors found that the $TD(\lambda)$ learning algorithm attained notable performance when compared to $SARSA$, but did not experiment with policy gradient methods.

After the *CNN / Daily Mail* dataset was released [1], deep learning methods seemed to be viable techniques that could harness the dataset's abundance of data. In [11, 5] we observe supervised deep learning techniques using state-of-the-art recurrent neural network and long-short-term-memory models. Soon after, deep RL methods were proposed and experimented with by a variety of authors. Some used Q-learning RL methods with Deep Q-Networks (DQNs) [3], and others used policy gradient methods with the REINFORCE algorithm for learning [6, 15], or other policy gradient variants [7].

It should be noted, however, that all of the models presented above offer improvements upon the basic *Lead-3* baselines that do not exceed 3 points in any ROUGE metric score. This is a somewhat troubling fact: extremely large and advanced deep learning models trained for days or even weeks on end minimally improve upon the most basic deterministic solution that simply selects the first three sentences of the article to be the summary. It is with this fact in mind that we seek to go back to "square one" in the present work by looking at the problem in an isolated setting with an interpretable, linear model. In this work, we gain insights into the problem by examining ES in a controlled setting, where we can clearly understand what our model learns by avoiding (for now) the so-called "alchemy" of deep learning methods [8], until we can better understand the problem we are faced with.

2 Reinforcement Learning for Extractive Summarization

RL is well-suited for the ES task because the primary evaluation metric ROUGE [4] is a non-differentiable function. Performance can be optimized in an RL setting by using the ROUGE score directly as the reward function, as opposed to substituting a differentiable approximate loss function for the purpose of using standard supervised learning approaches to solve ES [5, 11].

In Figure 1 we present a general overview of our process from beginning to end. The intial pre-processing step consists of removing stopwords and punctuation from all articles. Doing so not only corresponds to having a stronger correlation to human judgments of summarization quality[3], but also improves the running-time of our models by removing a significant number of tokens from each article.

Following preprocessing, we use the linear policy-gradient Monte Carlo learning algorithm, REINFORCE [14, 12], to train an agent to build extractive summaries of news articles. This therefore necessitates useful feature extraction, hyperparameter tuning, and elucidation of our learning algorithm, as described below. One important feature of our learning process is the introduction of *batch-mean* weight updates (Section 2.2.1), which we found significantly improved generalization ability.

2.1 Feature Extraction

The use of function approximation based RL methods is crucial for the application of RL to real-world problems where tabular formulations are either computationally impractical or completely infeasible [12]. Here, we pose the following simple characteristics of our RL formulation of the problem:

- An episode of this task begins with a single article $\mathbf{A}$ for which we are expected to extract a three-sentence summary e; during training obtain reward with a gold-standard reference summary S^*.
- The *state* s, or S_t at time t, of the MDP is the set of sentences currently in the article $\mathbf{A}$, minus the set of sentences currently selected to be in the extractive summary e.

[3] In [4], the designers of ROUGE note that the correlation of ROUGE score and human-evaluations of summary quality improves when stopwords are removed for the purposes of computing ROUGE .

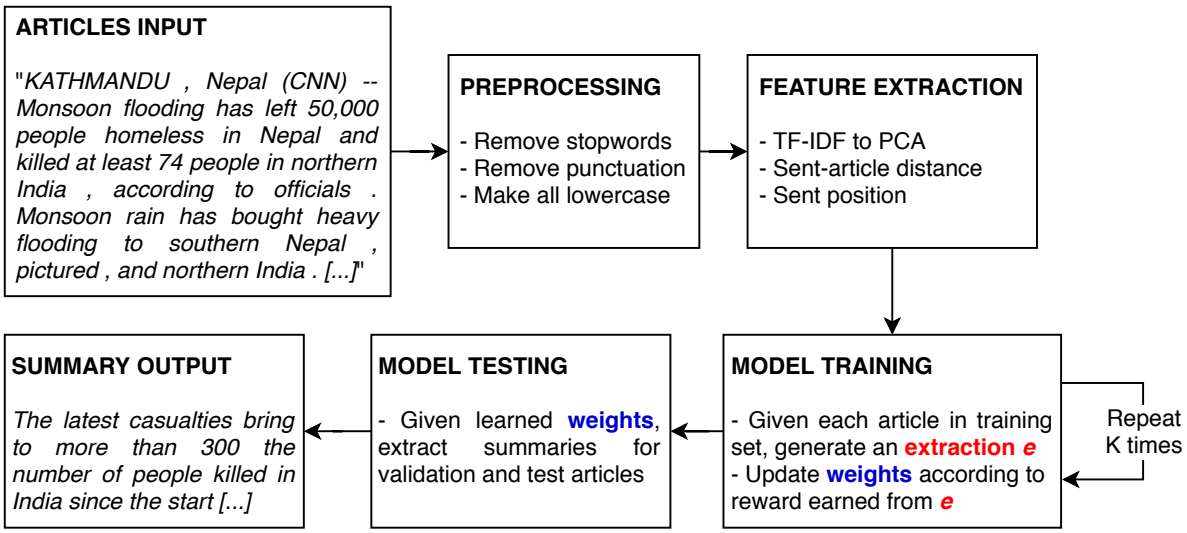

Figure 1: General depiction our RL-based pipeline for performing ES.

- An *action* a, or A_t at time t, in the MDP is the choice of selecting the a^{th} sentence in the article to be in the extraction e.
- There are exactly 3 time steps, and then the episode ends; at each time step we select a single sentence to be in the summary.
- A complete episode is: $S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2$
- Rewards $R_0 = R_1 = 0$ and $R_2 = score(e, S^*)$ (see Equation 1), meaning that the reward is earned only at the end of the episode.
- Updates U_t are defined according to the total return $G = \sum_t R_t$ and a baseline function $b(S_t)$ (if no baseline is used, $b(S_t) = 0$); generally, $U_t = G - b(S_t)$. We do not use discounting.
- During training, actions are selected stochastically according to the weighted probability distribution defined by the policy; during testing, actions are selected greedily according to the policy.

Given the above formulation of the problem, we thus propose the features described below, where we describe how we build state features $\boldsymbol{\phi}(s)$ and state-action features $\boldsymbol{\phi}(s, a)$. In this setting we sought to use simple methods with simple features to observe if we could obtain performance comparable with the dominant baseline of the field (*Lead-3*), and thus opt for simple TF-IDF feature and simple linear PCA vector compression. Future work may involve incorporating word embeddings and non-linear PCA methods to improve the quality of the extracted features. The rest of the section uses significant notation; refer to Table 4 in the Appendix for further clarity.

2.1.1 State Features

We use the common term-frequency inverse-document-frequency NLP feature extraction technique to build vector representations of sentences with respect to their document. For the purpose of hyper-parameter tuning and using a gridsearch to find the optimal models, we fit the TF-IDF feature extractor[4] and the PCA feature extractors on the training and validation sets combined; this is reasonable since these feature extraction techniques are unsupervised. Then, for the final testing, we use the previously

[4]See `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`.

determined optimal models and re-fit the TF-IDF and PCA extractors onto the full combined training, validation, and test sets. This is a realistic approach to the problem as it would be unscientific to perform a hyperparameter gridsearch by fitting the extractors onto the test set; however, fitting the unsupervised feature extractors to the test set at test time with pre-determined models is reasonable as that is not computationally expensive (unlike a gridsearch for optimal hyperparameters).

TF-IDF embeds the sentences into an m-dimensional space, which is then compressed to d dimensions using PCA (principal component analysis). We call this PCA-compressed vector of a sentence j the *sentence representation* $\mathbf{v}_j$. The full feature representation of an article is the mean of its sentence representations, $\mathbf{a}$. At time step $t = 0$, the state representation is the mean sentence representation over the sentences in the K-sentence article, equivalent to the article representation; that is,

$$\boldsymbol{\phi}(S_0) = \mathbf{a} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{v}_k$$

At each subsequent time step ($t = 1$ and $t = 2$), we update the state vector according to the action taken. Suppose action A_{t-1} has selected sentence k' to in the summary; then we set the features for state S_t as follows:

$$\boldsymbol{\phi}(S_t) = \frac{1}{K-t}\big[(K-t+1)\boldsymbol{\phi}(S_{t-1}) - \mathbf{v}_{k'}\big]$$

this removes the sentence representation $\mathbf{v}_{k'}$ from the state representation by taking it out of the average and rescaling accordingly.

2.1.2 State-Action Features

Policy gradient methods require features for representing state-action pairs. Thus, given an action a associated with sentence representation $\mathbf{v}_a$, we encode:

- *TF-IDF features*: the sentence representation corresponding to action a, $\mathbf{v}_a$;
- *Distance features*: the L1, L2, and cosine distance between $\mathbf{v}_a$ and $\mathbf{a}$, defining the distance vector $\mathbf{d}_a = \langle L_1(\mathbf{v}_a, \mathbf{a}), L_2(\mathbf{v}_a, \mathbf{a}), cos(\mathbf{v}_a, \mathbf{a})\rangle$;
- *Position features*: given the maximum number of sentences in an article in our batch, l, use a one-hot-encoding of a's position in the document via an l-dimensional position vector $\mathbf{p}_a$ such that the a^{th} component of $\mathbf{p}_a$ equals 1 and each other component equals 0.

Thus defining the following feature vector representation for state-action pairs s, a, where $\mathbf{x} \,\|\, \mathbf{y}$ indicates the concatenation of vectors:

$$\boldsymbol{\phi}(s, a) = \langle \boldsymbol{\phi}(s) \,\|\, \mathbf{v}_a \,\|\, \mathbf{d}_a \,\|\, \mathbf{p}_a \rangle$$

2.2 Policy Gradient Learning with REINFORCE

Our learning algorithm uses the standard formulation of REINFORCE with function approximation, as found in [12]. The specific form of the policy is defined by the following linear-softmax action preference function:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp\left(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)\right)}{\sum_{b \in \mathcal{A}(s)} \exp\left(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s, b)\right)} \tag{3}$$

where (for the purpose of computational efficiency) the generalized matrix form is defined as:

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(s) = \eta \exp\left[\boldsymbol{\Phi}(s)\,\boldsymbol{\theta}\right] \tag{4}$$

where exp[ ] denotes the component-wise application of the exponential function ($f(x) = e^x$) onto a vector, and η is the normalizing constant (1 over the sum of the components of vector exp $\left[\boldsymbol{\Phi}(s)\boldsymbol{\theta}\right]$).

We formulate the update rule for policy gradient learning with the REINFORCE algorithm by introducing an update vector $\mathbf{u}_{\boldsymbol{\theta}}(t)$. Our objective is to learn the weight vector $\boldsymbol{\theta}$ that can select the best actions in the environment, where updates are determined according to a learning rate α_π:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\pi \mathbf{u}_{\boldsymbol{\theta}}(t) \tag{5}$$

$$\begin{aligned} \mathbf{u}_{\boldsymbol{\theta}}(t) &= U_t \nabla \ln \pi_{\boldsymbol{\theta}}(A_t|S_t) \\ &= U_t \Bigg[\boldsymbol{\phi}(S_t, A_t) - \sum_{a \in \mathcal{A}(S_t)} \pi_{\boldsymbol{\theta}}(a|S_t) \boldsymbol{\phi}(a, S_t) \Bigg] \\ &= U_t \Bigg[\boldsymbol{\phi}(S_t, A_t) - \sum_{rows} \boldsymbol{\pi}_{\boldsymbol{\theta}}(S_t) \odot \boldsymbol{\Phi}(S_t) \Bigg] \end{aligned} \tag{6}$$

where $\mathbf{v} \odot \mathbf{M}$ indicates that the components of vector $\mathbf{v}$ scale each row of $\mathbf{M}$ separately (equivalent to creating a diagonal matrix $\mathbf{V}$ such that $\mathbf{V}_{ii} = \mathbf{v}_i$ and performing the matrix multiplication $\mathbf{VM}$; i.e., $\mathbf{v} \odot \mathbf{M} = \mathbf{VM}$); and, $\sum_{\text{rows}}$ indicates the summation of all matrix's rows into a single vector.

Policy gradient methods often can suffer from high variance in updates [12]. We therefore also experiment with the introduction of a state-value function $\hat{v}_{\mathbf{w}}$ as the baseline (i.e., $b(S_t) = \hat{v}_{\mathbf{w}}(S_t)$), with a learning rate $\alpha_{\hat{v}}$. The update rule for the state-value function (the baseline) is defined below:

$$\hat{v}_{\mathbf{w}}(s) = \mathbf{w}^\top \boldsymbol{\phi}(s) \tag{7}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\hat{v}} \mathbf{u}_{\mathbf{w}}(t) \tag{8}$$

$$\mathbf{u}_{\mathbf{w}}(t) = \Big[G - \hat{v}_{\mathbf{w}}(S_t) \Big] \boldsymbol{\phi}(S_t) \tag{9}$$

In our experiments, we examine the impact of performing weight updates at each time step versus just performing weight updates at the last time step. We coin this as the UpdateLast boolean hyperparameter, and experiment with it in the gridsearch (Section 3.2).

2.2.1 Batch-Mean Updates

The above methods incrementally learn from each article. The model would therefore be biased by the last article it learned from during training; while this may improve training accuracy, we doubted whether or not such a method would offer high-quality generalization. We therefore propose *batch-mean updates*. This is, in fact, more theoretically well-behaved than doing single-episodic updates; averaging over a batch of trajectories offers a better and lower variance approximation of the true integral which we are attempting to approximate with a sample-based expectation updates, see [12] for further explanation of these theoretical justifications.

Recall that an episode of is defined by exactly three time steps $E^{(i)} = \{t_0^{(i)}, t_1^{(i)}, t_2^{(i)}\}$. We now redefine the update rules according to a batch $\mathcal{B}$ of n unique episodes, where the batch set contains each time step in each episode $\mathcal{B} = \{t_j^{(i)}\}\ \forall i = 1 \dots n, j = 0, 1, 2$. In the Update-Last setting, we only perform updates on the last time step of the episode, setting $\mathcal{B}$ to only consist of those $t_j^{(i)}$ such that $j = 2$. Recalling that each episode occurs on a different article, we define the generalized updates below in Equation 10; as described above, note that these updates move more in the general direction of the "true" distribution of the MDP than single-step updates:

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha_\pi \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \mathbf{u}_{\boldsymbol{\theta}}(t) \\ \mathbf{w} &\leftarrow \mathbf{w} + \alpha_{\hat{v}} \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \mathbf{u}_{\mathbf{w}}(t) \end{aligned} \tag{10}$$

	$\alpha_{\hat{v}}$	α_{π}	TF-IDF Dim.	PCA Dim.	UpdateLast	BatchMean
PG-T	0	0.275	5000	1000	False	False
PG-V	0.05	0.15	5000	1000	True	True

Table 1: Policy Gradient hyperparameters ("Dim." means dimensionality) that attained the optimal mean-Rouge scores on the training (PG-T) and validation (PG-V) sets.

3 Experimental Design

The present work expresses a preliminary investigation into the application of policy-gradient based RL for the task of extractive summarization. Here, we experiment with linear methods, but it should be noted that when applied on a large scale problem they would likely insufficiently take advantage of a large dataset when compared to the application of deep learning methods in the same setting. Nonetheless, a useful consequence of using linear methods is that we can directly examine the impact of our different manually extracted features by analyzing the learned weights associated with those features. We therefore restrict the problem setting in order to provide lucidity and isolate the primary factors of variation concerning algorithmic performance in this setting.

We use the *CNN / Daily Mail* summarization dataset created by [1], but restrict the problem setting to the *CNN* subset, and further only use 200 training articles, 50 validation articles, and 50 final testing articles, all of which were randomly sampled from the complete dataset of $92,580$ articles. Given this small amount of training and test data, the results presented in this work should should be understood as preliminary to a complete study, but nonetheless offer insights into future model design.

It was further necessary to restrict the problem setting in this way because the model takes a relatively long time to train. Depending on the feature extraction settings, it could take anywhere between three and seven minutes to perform 500 training steps over each of the 200 articles; i.e., an average of five minutes to perform $100,000$ extractive summarizations (with weight updates), or about 333 summarizations per second. Restricting the training set allowed us to perform an extensive gridsearch over $4,600$ different hyperparameter configurations in a reasonable amount of time by parallelizing over 36 CPU cores, taking roughly 11 hours. Our exploration into hyperparameter settings in this restricted setting offers a preliminary intuition for which subset of hyperparameters to test when moving to larger-scale developments of the methods presented in this work.

3.1 Baselines

We compare with the following standard baseline algorithms:

- *Random*: uniformly at random selects sentences to be a part of the summary, a useful comparison in order to assess whether our model is actually learning.
- *Lead-3*: selects the first three sentences of the article to be the summary, a surprisingly difficult baseline to beat, even with deep methods [5, 11, 15].
- *Oracle*: an oracle that selects the three sentences which obtain the optimal mean-ROUGE score (Equation 1); this is an upper bound on model performance.

3.2 Hyperparameters

We performed a non-exhaustive gridsearch over several thousand different combinations over these parameter settings, each of which were given 500 training steps. In Table 1 above, we present the optimal models determined by the gridsearch over the following hyperparameters:

- Dimensionality of the TF-IDF vector representation;

Training	ROUGE-1	ROUGE-2	ROUGE-L	Mean
Oracle	0.595	0.287	0.441	0.441
Random	0.215	0.071	0.158	0.148
Lead-3	0.352	0.139	0.247	0.246
PG-V	0.360	0.142	0.254	0.252
PG-T	**0.380**	**0.156**	**0.278**	**0.271**
Validation				
Oracle	0.617	0.328	0.476	0.474
Random	0.184	0.051	0.132	0.122
Lead-3	0.383	0.166	0.283	0.278
PG-V	**0.393**	**0.171**	**0.290**	**0.285**
PG-T	0.323	0.132	0.245	0.235
Test				
Oracle	0.626	0.321	0.480	0.475
Random	0.230	0.088	0.172	0.164
Lead-3	**0.376**	**0.167**	**0.277**	**0.273**
PG-V	**0.375**	**0.167**	**0.278**	**0.273**
PG-T	0.344	0.145	0.257	0.249

Table 2: Train, validation, and test set performance. PG-V is the model optimized for maximal Validation set performance; PG-T is the model optimized for maximal Training set performance (see Table 1 for hyperparameter configuration of these models). **Boldface** indicates the best model performance for that subset (excluding the Oracle), or tie when the difference is negligible.

- Dimensionality of the PCA compression of the TF-IDF vectors;
- Value function learning rate, $\alpha_{\hat{v}} \in [0, 1]$ (when $\alpha_{\hat{v}} = 0$ this corresponds to not using a baseline);
- Policy gradient learning rate, $\alpha_{\pi} \in (0, 1]$;
- Whether or not to use batch-mean updates (see Section 2.2.1) – BATCHMEAN;
- Whether or not to perform weight updates at each time step or only at time step $t = 2$ (loosely equivalent to whether or not we should set $\gamma = 0$ or $\gamma = 1$) – UPDATELAST;

4 Results

In Table 2 we present the final results obtained by our models across the training, validation, and test sets. We note that the model that performed optimally on the *training* set (PG-T) *did not* use batch-mean updates, while the model the generalized best to the *validation* set (PG-V) *did* use batch-mean updates. These results confirm our hypothesis that batch-mean updates offer improved generalization, and is further confirmed by the similar performance of PG-V on the test set. Additionally, we note that the PG-T model did not use a baseline, while the best model for generalization did, and examination of gridsearch results showed that the top 20 validation-set models needed a baseline. In Figure 3 (Appendix), we show performance of the greedy policy on our learned $\pi_{\boldsymbol{\theta}}$ at each time step.

The results attained by our PG-V model obtained test set performance almost directly at the same level as *Lead-3* across all evaluation metrics. This suggests that linear RL methods can perform well in this setting, and that perhaps the heavy-weight deep learning methods [5, 11] may be overkill, or at least need to be revisited. Our results are even more encouraging due to the fact that several improvements to our model can clearly and easily be made, as explained in the conclusion (Section 5).

Most Positive		**Most Negative**	
Feature name	$\boldsymbol{\theta}_i$	*Feature name*	$\boldsymbol{\theta}_i$
3^{rd} PC of $\mathbf{v}_a$	0.168	2^{nd} PC of $\mathbf{v}_a$	-0.151
4^{th} PC of $\mathbf{v}_a$	0.103	1^{st} PC of $\mathbf{v}_a$	-0.030
5^{th} PC of $\mathbf{v}_a$	0.050	17^{th} PC of $\mathbf{v}_a$	-0.026
7^{th} PC of $\mathbf{v}_a$	0.018	9^{th} PC of $\mathbf{v}_a$	-0.022
8^{th} PC of $\boldsymbol{\phi}(s)$	0.014	16^{th} PC of $\mathbf{v}_a$	-0.022

Table 3: The largest weights of $\boldsymbol{\theta}$ and their associated features (PC stands for Principal Component); recall that $\mathbf{v}_a$ is the PCA-compressed TF-IDF representation of the sentence at position a in the article (for the action a), and that $\boldsymbol{\phi}(s)$ is the state representation, which is the mean of the PCA-TF-IDF sentence representations of the document (minus the previously selected sentences for the summary).

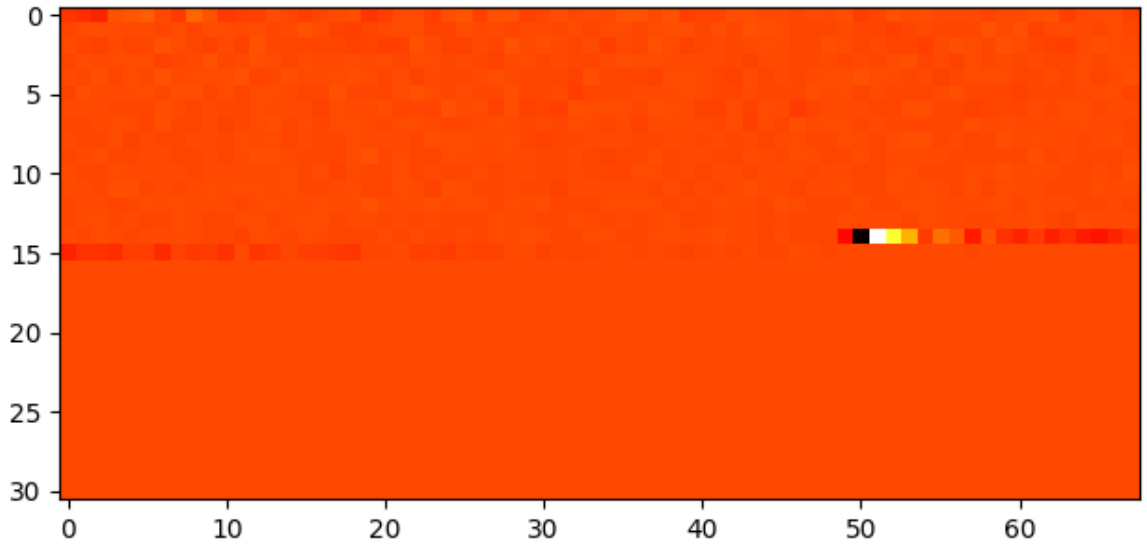

Figure 2: Learned weight vector $\boldsymbol{\theta}$ (reshaped into a matrix for visualization) for the PG-V model. Darker colors indicate the most negative values, and lighter more positive, with red being roughly zero.

4.1 Analysis

Linear RL methods are limited by the need for more explicit feature engineering, and suffer from having lower hypothesis-space expressivity than deep models [12]. Despite these shortcomings, they offer a significant advantage over deep RL methods by allowing us to directly interpret the learned weights as they modify the features. Additionally, linear RL offers convergence guarantees and is typically more stable than deep RL [12]. Indeed, while linear RL requires intensive feature engineering, the work of deep RL largely consists of intensive model engineering, which can take significantly more time to perform exhaustively.

In Figure 2 and Table 3 we thus take advantage of the interpretability benefits of linear RL by examining the learned weights of our PG-V model (see Section 3.2 and Table 1) because our policy gradient action preferences are linear in the features (e.g., $\pi_{\boldsymbol{\theta}}(a|s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a)$)). In Figure 2 we observe that the vast majority of features are near zero; indeed, almost half of the features have weights $|\theta_i| < 10^{-5}$, and over 70% of the features have weights $|\theta_i| < 10^{-3}$. This indicates that the model is learning precise weights and that it does not rely on a highly distributed weight representation, which implies that the model disentangles the factors of variation within the data, at least to a certain degree.

In Table 3, we see that the PCA-compressed TF-IDF representation of the sentence $\mathbf{v}_a$ offers crucial features for the model. Indeed, the first elements of $\mathbf{v}_a$, which correspond to the principal components computed from the original TF-IDF vector, offer strong signal for action selection. Interestingly, we found that the weights associated with the distance and position features $(\mathbf{d}_a, \mathbf{p}_a)$ are all essentially zero, with absolute values less than or equal to 10^{-19}. This is surprising, as one would expect that a mathematical similarity comparison between a sentence and its document would be informative for determining how much information or redundancy a certain sentence contributes to the summary. However, it is possible that the model simply learned the weights that offer the most useful combination of the sentence and article-level features, therefore not necessitating naive distance metrics.

Additionally, one would anticipate that the position of a sentence in the document would be an informative feature for summarization (e.g., we may expect that sentences in the beginning of an article would be more likely to be included in the summary), yet the weights learned suggest that they offered no signal whatsoever. This consequence may actually be desirable as it shows that the model has not overfit to the training set sentences by essentially memorizing the positions of sentences. Instead, our model seems to have learned general relationships between sentences and their article.

5 Conclusion

In this work we present a linear policy-gradient reinforcement learning method for performing extractive summarization of CNN news articles. We isolate the problem to a randomly sampled subset of the complete dataset, allowing us to directly examine the impact of features and model hyperparameters. Our results show that linear methods can obtain performance at or above the level of the basic *Lead-3* baseline algorithm for ES, just as deep learning methods do [5, 11]; this suggests that future work should be pursued in the area of linear RL, and that perhaps the contemporary deep learning methods should be rethought. Additionally, we found that using *batch-mean* learning updates significantly improved the ability of our model to generalize to unseen data, offering insight into learning algorithm design for future work in RL for ES.

While the high-quality results of our model are encouraging, what is perhaps even more encouraging is how simple our proposed model is, a testament to the powers of reinforcement learning. We use basic NLP features – TF-IDF with linear PCA – and even the slightly more engineered distance and position features turned out to be unnecessary. Future work would involve incorporating word embeddings, non-linear PCA, and stronger comparative features between sentences and their documents. Additionally, the baseline method we chose (learning a state-value function) is just one of many possible; future work may find that simply using *Lead-3* performance as a baseline would result in better learning. In the MDP, future experiments should be performed on the impact of the reward function, and whether it should be an average of the ROUGE scores, or some weighted combination of the three. From an RL perspective, we used a standard REINFORCE-with-baseline learning algorithm; however, there exist many variants of policy gradient learning that can offer significant improvements in learning speed and generalization abilities [13, 2]. Future work would involve experimenting with such algorithmic learning variants. Overall, while the possibilities for improvement are immense, this work reveals the importance of examining problems in simplified settings with simple model designs, where this approach offer insights into future model design in such a way that extremely complex models might neglect.

References

[1] Karl Moritz Hermann et al. "Teaching machines to read and comprehend". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1693–1701.

[2] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[3] Gyoung Ho Lee and Kong Joo Lee. "Automatic Text Summarization Using Reinforcement Learning with Embedding Features". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Vol. 2. 2017, pp. 193–197.

[4] Chin-Yew Lin. "ROUGE: A package for automatic evaluation of summaries". In: *Proceedings of the ACL 2004 Workshop on Text Summarization Branches Out*. Barcelona, Spain, 2004, pp. 74–81.

[5] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. "SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents." In: *AAAI*. 2017, pp. 3075–3081.

[6] Shashi Narayan, Shay B Cohen, and Mirella Lapata. "Ranking Sentences for Extractive Summarization with Reinforcement Learning". In: *arXiv preprint arXiv:1802.08636* (2018).

[7] Romain Paulus, Caiming Xiong, and Richard Socher. "A deep reinforced model for abstractive summarization". In: *arXiv preprint arXiv:1705.04304* (2017).

[8] Ali Rahmi and Ben Recht. *Reflections on random kitchen sinks*. 2017. URL: `http://www.argmin.net/2017/12/05/kitchen-sinks/`.

[9] Cody Rioux, Sadid A Hasan, and Yllias Chali. "Fear the reaper: A system for automatic multi-document summarization with reinforcement learning". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 681–690.

[10] Seonggi Ryang and Takeshi Abekawa. "Framework of automatic text summarization using reinforcement learning". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics. 2012, pp. 256–265.

[11] 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).

[12] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction – 2nd Edition*. Vol. 1. 1. MIT press Cambridge, 2018. URL: `https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view`.

[13] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[14] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[15] Yuxiang Wu and Baotian Hu. "Learning to Extract Coherent Summary via Deep Reinforcement Learning". In: *arXiv preprint arXiv:1804.07036* (2018).

Appendix

Symbol	Domain	Description
MDP parameters		
s	-	a state; i.e., the current article minus any sentences previously selected from it
a	-	an action; i.e., the choice of selecting a sentence from an article to be part of the extracted summary (the sentence's index)
$\mathcal{A}(s)$	-	the set of possible actions given a state s; i.e., the sentences in the article minus any previously selected
γ	$[0, 1]$	discount factor
t	$\{0, 1, 2\}$	time step in a summarization episode
S_t	-	the state at time step t
A_t	$\mathcal{A}(S_t)$	the action taken at time step t
$r(S_t, A_t)$	$[0, 1]$	the reward observed at time step t from taking action A_t in state S_t
G_t	$[0, 1]$	the return up to time step t of an episode
U_t	$\mathbb{R}$	the update target at time t, taking into account γ and $r(S_t, A_t)$
E	-	a summarization episode on an article, three time steps $E = \{t_0, t_1, t_2\}$
$\mathcal{B}$	-	the set of all time steps along all episodes in the training batch of articles; e.g., $\mathcal{B} = \{t_0^{(1)} \dots t_j^{(i)} \dots t_2^{(n)}\}$
Feature parameters		
m_s	$\mathbb{R}$	number of features for state representation
m_{sa}	$\mathbb{R}$	number of features for state-action representation
$\boldsymbol{\phi}(s)$	$\mathbb{R}^{m_s}$	feature vector for a state s
$\boldsymbol{\phi}(s, a)$	$\mathbb{R}^{m_{sa}}$	feature vector for a state-action combination s, a
$\boldsymbol{\Phi}(s)$	$\mathbb{R}^{\lvert\mathcal{A}(s)\rvert \times m_{sa}}$	feature matrix with rows $\boldsymbol{\phi}(s, a)\, \forall a \in \mathcal{A}(s)$
$\mathbf{v}_a, \mathbf{d}_a, \mathbf{p}_a$	-	specific features corresponding to action a: sentence PCA-compressed TF-IDF features, distance features, and position features (respectively)
Learning parameters		
$\mathbf{w}$	$\mathbb{R}^{m_s}$	learned weight vector for approximated the state-value function
$\hat{v}_{\mathbf{w}}(s)$	$\mathbb{R}$	the approximated state-value function (used as baseline)
$\alpha_{\hat{v}}$	$\mathbb{R}$	fixed learning rate for state-value function updates
$\mathbf{u}_{\mathbf{w}}(t)$	$\mathbb{R}^{m_s}$	the learning update vector for the state-value function for time t
$\boldsymbol{\theta}$	$\mathbb{R}^{m_{sa}}$	learned weight vector that defines the parameterized softmax policy
$\pi_{\boldsymbol{\theta}}(a\lvert s)$	$\mathbb{R}$	the parameterized softmax policy for a given state and action s, a
$\boldsymbol{\pi}_{\boldsymbol{\theta}}(s)$	$\mathbb{R}^{\lvert\mathcal{A}(s)\rvert}$	policy vector with components $\pi_{\boldsymbol{\theta}}(a\lvert s)\, \forall a \in \mathcal{A}(s)$
α_{π}	$\mathbb{R}$	fixed learning rate for policy gradient updates
$\mathbf{u}_{\boldsymbol{\theta}}(t)$	$\mathbb{R}^{m_{sa}}$	the policy gradient update vector for time t

Table 4: Notation used in the present work; those necessitating equations are defined above.

Algorithm 1 Oracle ES Algorithm. Obtains the near-optimal extractive summary of the document (may be slightly less than optimal since it doesn't examine all $\binom{n}{3}$ combinations), offering a soft upper-bound on any ES model's performance.

procedure ORACLE($\mathbf{A} = \{x_1, x_2, \ldots x_n\}$)
    $e \leftarrow \{\}$ ▷ Stores the extraction's sentences.
    while $e.size < 3$ **do**
        $x_{best} \leftarrow \emptyset$
        for $x_i \in \mathbf{A}$ **do** ▷ Greedily select the sentences one at a time.
            if $score(e \cup \{x_i\}) > score(e \cup \{x_{best}\})$ **then**
                $x_{best} \leftarrow x_i$
        $e \leftarrow e \cup \{x_{best}\}$
        $\mathbf{A} \leftarrow \mathbf{A} \setminus \{x_{best}\}$
    return e

Figure 3: Full average training set ROUGE scores obtained by our PG-V model when greedily selecting actions at each step according to the learned softmax policy; i.e., we pass the entire training set through the model at each training step by applying the greedy version of the model's softmax policy learned up to that step. This clearly shows that the model is learning over time.