Deep Q-Learning for Event Summarization

Francisco Javier Arceo
fja2114@columbia.edu

Chris Kedzie
kedzie@cs.columbia.edu

November 9, 2016

Abstract

We present a streaming extraction policy for multi-document summarization learned using a deep reinforcement learning recurrent neural network architecture. The rewards of our network are evaluated through manually reviewed summaries, which allow us to specify an end-to-end framework to optimize our extraction policy. By using raw text from our articles and mapping words into a higher embedding dimension through our recurrent neural network, we are able to learn a more robust representation than traditional n-gram models. We benchmark our model against random decisions, bag-of-words, and bag-of-bigrams models.[1]

1 Introduction

Recent research has focused on extractive (e.g., [8], [2], and [6]) multi-document summarization. The approach by [8] has shown that it is possible to select relevant sentences from a massive number of documents on the web to create summaries with meaningful content by adapting classifiers to maximize search policies. These systems operate in a streaming fashion and are capable of evaluating each sentence within each article to decide whether or not to include or ignore the sentence.

Unfortunately, these systems have still been shown to fall short of algorithms that employ simple heuristics [2], which may be due to inadequate capturing of the rich structure and often idiosyncratic information by traditional n-gram language models. In recent years, [1] have shown that natural language processing can be mapped to a higher order dimension to represent more powerful features for various language

[1] `https://github.com/franciscojavierarceo/DQN-Event-Summarization`

modeling tasks. These embeddings have proven incredibly powerful on a variety of different natural language processing tasks [citations needed].

In this paper we show that a deep recurrent neural network with a long short term memory (LSTM) is able to successfully learn an extractive summary policy by encoding the action, state, and reward into a Q-Learning model, similar to that of [4]. We show that on a variety of different metrics, our specification is able to reach state-of-the-art performance.

2 Extractive Streaming Summarization

In the extractive streaming summarization task, we are given as input a query (i.e., a short text description of a topic or an event), a document stream, and a time ordered set of sentences relevant to the query. Starting with an initially empty summary, an extractive, streaming summarization algorithm is intended to examine each sentence in order and when new and important (relative to the query) information is identified, add that sentence to the summary.

Implicit to this problem is the notion of system time – the summarization algorithm can only examine sentences that occur in the stream before the current system time. Advancing the system time gives the algorithm access to more sentences, although in practice the stream is sufficiently large enough that choices have to be made about how much history can be kept in memory. For many domains, e.g. crisis informatics, it is preferable for a summarization algorithm to identify important information as early as possible, and so the objective function should penalize a large disparity between the time a piece of information is first available to the algorithm and the system time at which that information is actually added to the summary.

Previous work in this area has either incremented the system time in fixed increments (e.g. an hour) [9, 8] or operated in a fully online setting [3, 7]. In both cases explicitly modeling the current state of the summary, the stream, and their relationship with the query is quite complicated and exhibits non-linear dynamics that are difficult characterize in traditional feature based models.

Additionally, the structured nature of the sentence selection task (sentence selection is highly dependent on the current summary state) suggests that imitation or reinforcement learning are necessary to obtain parity between training and testing feature distributions.

Early successes of deep reinforcement learning have used convolutional neural networks for video games (e.g., [10] and [11]) but recent work has leveraged recurrent neural networks for both video-based and text-based gaming environments (e.g. [4] and [12]).

This leads us to explore Deep Q-Networks (DQN) for three reasons: (1) both the representation and mode of interaction between the stream, summary, and query can be learned; (2) the embeddings can learn a more robust semantic representations than classic n-gram models; and (3) by randomly exploring the state space, the ϵ-greedy strategy used in DQN learns a policy that yields more consistency between train and test distributions.

3 Background

In the subsequent sections, we outline the architecture of our DQN and define the parameterization of our Q-function for the multi-document summarization task. Our architecture consists of three components: the (1) input, (2) action, and (3) value representations. By specifying these components into our neural network, we are able to develop an end-to-end extraction policy that fully propagates information both forward and backward.

3.1 States

In our architecture a state $s(x_t, \tilde{y}_t, q)$ is a function of the stream X observed up to the current system time t, the state of the current summary $\tilde{Y}$ at system time t, and the query q. For brevity we will use $s(t, q)$ where the dependence on $x_t, \tilde{Y}_t$ is assumed. $s(t, q)$ is itself three recurrent neural networks, one for encoding the summary, the stream, and the query respectively.

3.2 Actions

The set of possible actions a_t at each time step is $\mathcal{A} = \{select, skip\}$ where *select* corresponds to adding the current sentence x_t to the summary and incrementing the current system time, or *skip* where only t is incremented without changing the current summary. Thus, our current predicted summary at time t is defined as

$$\tilde{y}_{t+1} = \begin{cases} \tilde{y}_t \cup \{x_t\}, & \text{if } a_t = select \\ \tilde{y}_t, & \text{if } a_t = skip. \end{cases} \tag{1}$$

Since $\tilde{y}_t$ grows linearly with the number of selected sentences, we mitigate potential capacity constraints by using a queue to hold the last K elements of the current predicted summary.

3.3 Reward

The reward for a given action is measured by relative gain in ROUGE-2 F1 score of the predicted summary $\tilde{y}_t$ measured against a gold standard summary Y. When only one gold summary reference is used, ROUGE-N Recall is calculated as

$$\text{ROUGE-NR}(\tilde{y}, Y) = \frac{\sum_{g \in \text{ngrams}(Y,N)} \min\left(\text{count}(g, \tilde{y}), \text{count}(g, Y)\right)}{\sum_{g \in \text{ngrams}(Y,N)} \text{count}(g, Y)}$$

where $\text{ngrams}(Y, N)$ returns the set of ngrams of order N in the summary Y and $\text{count}(g, Y)$ is the count of occurrences of ngram g in Y.

Similarly, ROUGE-N Precision is calculated as

$$\text{ROUGE-NP}(\tilde{Y}, Y) = \frac{\sum_{g \in \text{ngrams}(Y,N)} \min\Big(\text{count}(g, \tilde{Y}), \text{count}(g, Y)\Big)}{\sum_{g \in \text{ngrams}(\tilde{Y},N)} \text{count}(g, \tilde{Y})}$$

and the F_1 is simply the harmonic mean of the two:

$$\text{ROUGE-NF1}(\tilde{Y}, Y) = \frac{2 \times \text{ROUGE-NP}(\tilde{Y}, Y) \times \text{ROUGE-NR}(\tilde{Y}, Y)}{\text{ROUGE-NP}(\tilde{Y}, Y) + \text{ROUGE-NR}(\tilde{Y}, Y)}$$

The reward r at time t is then:

$$r_t = \text{ROUGE-NF1}(\tilde{y}_t, Y) - \text{ROUGE-NF1}(\tilde{y}_{t-1}, Y)$$

3.4 Deep Q-Learning

We define an architecture similar to that of [12] and map our three inputs (query, sentence, and current predicted summary) into LSTM embeddings according to **Figure 1**. By formulating our extraction task as a Markov Decision Process (MDP) we can express the state-action tuples as transitions states.

Our extraction policy then takes as input an action a_t in state s_t at time t and returns the expected reward. We initialize our actions and Q-function at random and by taking the optimal action given our expected reward, we are able to iteratively update our Q-function using the Bellman equation [13] satisfying

$$\hat{Q}_{t+1}(s, a) = \text{E}[r + \gamma \text{max}_{a'} \hat{Q}_t(s', a') | s, a] \tag{2}$$

where $\gamma \in \mathbb{R}$ is the discount rate applied to future rewards and the expectation is taken over transitions in state s and action a. By iterating over the Q-function, it

converges asymptotically to the optimal policy, i.e., $\hat{Q}_t \rightarrow \hat{Q}^*$ as $t \rightarrow \infty$, but in finite iterations we can use a neural network as our function approximator, where our weights our θ can be optimized using the following loss function

$$\mathcal{L}_i(\theta_i) = \mathrm{E}_{\hat{s},\hat{a}}[\big(y_i - Q(\hat{s},\hat{a};\theta_i)\big)^2] \tag{3}$$

where

$$y_i = \mathrm{E}_{\hat{s},\hat{a}}[r_i + \gamma \mathrm{max}_{a'} Q(s',a';\theta_{i-1})|s',a']. \tag{4}$$

Differentiation of the loss function yields the update equation,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathrm{E}_{\hat{s},\hat{a}}[2\big(y_i - Q(\hat{s},\hat{a};\theta_i)\big)\nabla_{\theta_i} Q(\hat{s},\hat{a};\theta_i)] \tag{5}$$

that learns the parameters using stochastic gradient descent with RMSprop [5] on sampled batches of stored transitions.

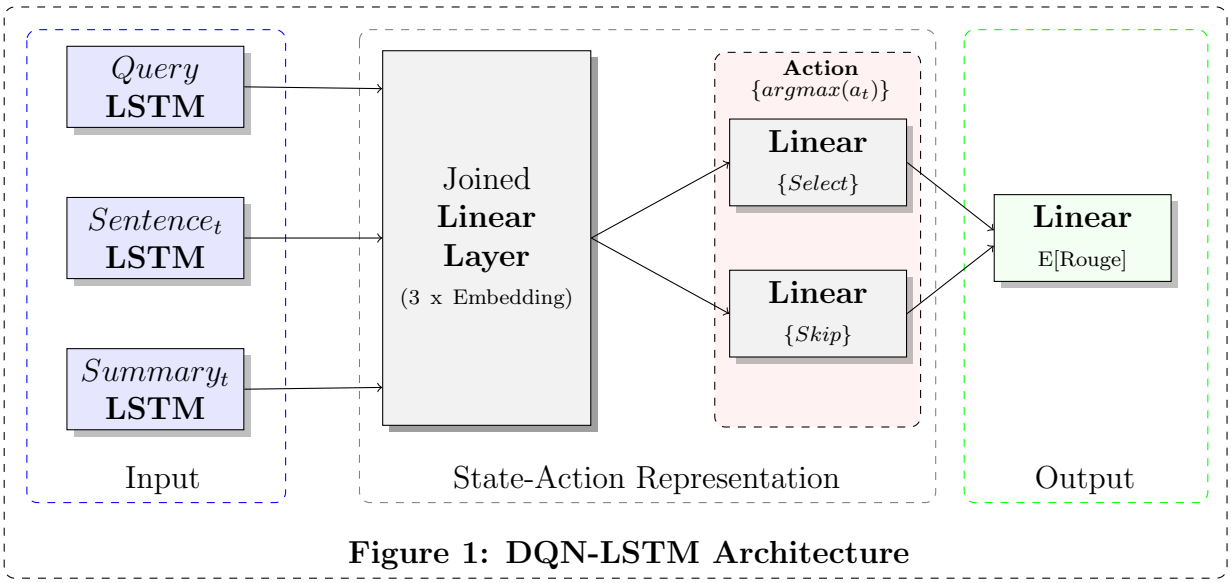

Figure 1: DQN-LSTM Architecture

4 Learning an Extraction Policy

4.1 Settings

Since the state of our current summary at time t depends on the state at time $t-1$, we execute our Q-function in the feedforward stage using single batches to allow for proper updating of the predicted summary, yet during the backpropagation phase, we sample random mini-batches from the stored transitions. We set our embedding dimension to 100, the future discount rate γ to 0.50, and the maximum summary length to 300 tokens.

The weights of the Q-function and the action sequence are initialized at random and we randomly explore the state-space for 200 of the 1000 epochs by employing an ϵ-greedy search strategy during our action selection that linearly decays to a base exploration rate of 0.01. We describe in full detail the training of our DQN-LSTM in **Algorithm 1**.

4.2 Experiments

Different experiments were done to study the effect of each hyperparameter on the model. Specifically, we evalute (1) the settings of K to assess the impact on the final predicted summary, (2) values of gamma to understand the effects on the parameter updates, (3) the memory size on the speed of convergence, and (4) the base exploration rate on the states explored. We provide visualization of the impact using 5-fold cross-validation on the different hyperparameters of our model in **Figure 2**.

5 Results

We evaluate our learned extracted summaries on ROUGE-N F1, Recall, and Precision on the full set of nuggets averaged across all of the queries on the full predicted summary $\tilde{Y}_d$, $\forall d \in \mathcal{D}$. We compare our DQN-LSTM to bag-of-words and bag-of-bigrams neural networks, our results are presented in **Table 1**.

Model	ROUGE-1F1	ROUGE-1Recall	ROUGE-1Precision
DQN-LSTM	TBD	TBD	TBD
DQN-BOW	TBD	TBD	TBD
DQN-BiBOW	TBD	TBD	TBD
Random	TBD	TBD	TBD

Table 1. Extracted summary performance on manually reviewed nuggets

Algorithm 1 DQN-LSTM for Event Summarization Training Procedure

Input: $\{\mathcal{D}$: Event queries, X_d: Input sentences, N: Number of epochs$\}$
Output : $\{\hat{Q}$: extraction policy, $\tilde{Y}$: event summary for query $d\}$

Initialize memory $\Gamma = \{\emptyset\}_{d=1}^{|\mathcal{D}|}$
Initialize summaries $\tilde{Y} = \{\emptyset\}_{d=1}^{|\mathcal{D}|}$
Initialize action-value function $\hat{Q}$ with random weights
for *epoch* $= 1, .., N$ **do**
  for $d \in \mathcal{D}$ **do**
    X_d= {Extract $t = 1, ..., T_d$ sentences for query d}
    $\tilde{Y}_d$= {Extract $t = 1, ..., T_d$ summaries for query d}
    for $(x_t, \tilde{y}_t) \in (X_d, \tilde{Y}_d)$, **do**
      Set $s_t = s(x_t, \tilde{y}_t, d)$
      $\forall a_t \in \mathcal{A}(s_t) = \{select, skip\}$ compute $\hat{Q}(s_t, a_t)$
      Select $a_t^* = \text{argmax}_{a_t} \hat{Q}(s_t, a_t)$
      if $random() < \epsilon$ **then**
        Set a_t^* to random action with $\Pr(a_t) = \frac{1}{|\mathcal{A}|}$
      end if
      if $a_t^* = \{select\}$ **then**
        Update $\tilde{y}_{t+1} = \tilde{y}_t \cup \{x_t\}$
      end if
      Execute action a_t^* and observe reward r_t and new state s_{t+1}
      Update $\Gamma_d = \Gamma_d \cup \{[s_t, a_t^*, r_t, s_{t+1}]\}$
    end for
  end for
  for $j = 1, ..., J$ transitions sampled from Γ **do**

$$\text{Set } y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \text{max}_{a'} \hat{Q}(s_{j+1}, a'; \theta) & \text{if } s_{j+1} \text{ is non-terminal} \end{cases}$$

    Perform gradient step on $\mathcal{L}(\theta) = (y_j - \hat{Q}(s_j, a_j; \theta))^2$
  end for
end for

6 Conclusion

We have presented an end-to-end deep reinforcement learning architecture to learn an extraction policy for multi-document summarization. By treating the incoming sentence streams from different documents as inputs into a recurrent neural network, we are able to learn both a richer embedding representation of the three input components and a robust extraction policy.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.

[2] Cristina Gârbacea and Evangelos Kanoulas. The university of amsterdam (ilps. uva) at trec 2015 temporal summarization track.

[3] Qi Guo, Fernando Diaz, and Elad Yom-Tov. Updating users about time critical events. In *European Conference on Information Retrieval*, pages 483–494. Springer, 2013.

[4] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[5] Geoffrey Hinton, N Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent.

[6] Chris Kedzie. Extractive and abstractive event summarization over streaming web text.

[7] Chris Kedzie, Fernando Diaz, and Kathleen McKeown. Real-time web scale event summarization using sequential decision making. *Proceedings of the Joint Conference on Artificial Intelligence*, 2016.

[8] Chris Kedzie, Kathleen McKeown, and Fernando Diaz. Predicting salient updates for disaster summarization. In *Proceedings of the 53rd annual meeting of the ACL and the 7th International Conference on Natural Language Processing*, pages 1608–1617, 2015.

[9] Richard McCreadie, Craig Macdonald, and Iadh Ounis. Incremental update summarization: Adaptive sentence selection based on prevalence and novelty. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 301–310. ACM, 2014.

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[12] Karthik Narasimhan, Tejas D Kulkarni, and Regina Barzilay. Language understanding for textbased games using deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Citeseer, 2015.

[13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1.