# DQN-LSTM for Event Summarization

Chris Kedzie      Francisco Javier Arceo

kedzie@cs.columbia.edu      fja2114@columbia.edu

September 18, 2016

# 1 Notes/Questions

- Word-level RNN

- for first part we will ignore time

- Okay, so we have 3 separate LSTMs

  1. Query level (we have 44 queries)
  2. Input Stream/Sentence (each query has  200k)
  3. Current Summary (initially empty)

- From there it gets interesting...because they're different sizes

- For all of the DQN games we have a terminal iteration and observed outcome, what about for us?

  1. We will process all queries

- We process all of the embeddings

## 1.1 Thoughts

Here's a stupidly worded brain dump of how I think this should go

- Randomly initialize the Query and Stream word vectors

- Initialize the updated summary with 0s

- Set estimated current summary to empty

- Sample $T$ streams from data (in mini-batches) by randomly choosing to select or skip streams (for a single query)

  1. Probably makes sense to sample from streams by weight exponentially as a function of time (older streams will be penalized directly)

  2. Use epsilon-greedy strategy that decays to a small percent for random sampling actions

  3. If improvement in $ROUGE > 0$ then 1, if not 0

  4. Sum $ROUGE$ for all sentences, make that label

- Employ pooling Feedforward ReLU network to learn $E[TotalROUGE]$

- Now given current summary, evaluate summary on current metrics

- Back prop

# 2 Extractive Streaming Summarization

In the extractive streaming summarization task, we are given as input a query, e.g. a short text description of a topic or an event, and document stream, 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) [5, 4] or operated in a fully online setting [2, 3]. 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.

Additionaly, 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.

This leads us to explore deep Q networks (DQN) for two reasons. First, both the representation and mode of interaction between the stream, summary, and query can be learned. Second, the learned Q function (plus random noise) controls the state space that is explored, ensuring more consistancy between train and test distribtuions than for example naive imitation learning (possibly – I'm not totally happy with this sentence).

## 2.1 Problem Definition - FJA

We specify our DQN architecture by estimating the Q function, $Q^*$, with 3 separate RNN-LSTMs representing the query, $t^{th}$ stream, and current predicted summary, which is then fed into a feedforward Multilayer Perceptron. After evaluation of our current state in our DQN, the model is backpropagated in the standard way, we refer to our architecture as a DQN-LSTM. Similar to [8] we use LSTM-DQN.

## 2.2 Algorithm

## 2.3 Problem Definition

DQN learns by using an a $\epsilon$-greedy search policy to generate a sequence of state, action, next state, reward 4-tuples using the current learned Q-network to evaluate candidate actions. These tuples are sampled from and used to estimate the true Q function.

**States** In our setup a state $s(X_{\leq t}, \tilde{Y}_{\leq t}, q)$ 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 $X_{\leq t}, \tilde{Y}_{\leq t}$ is assumed. $s(t, q)$ is itself three recurrent neural networks, one for encoding the summary, the stream, and the query respectively.

**Actions** The set of possible actions at each time step is $\mathcal{A} = \{select, skip\}$ where $select$ corresponds to adding the current sentence $x_t$ to the summary and

**Algorithm 1 Streaming DQN-LSTM**

---

**Input:** $\{x_q, \pi_q^*\}_{\forall q \in \mathcal{Q}}$, number of iterations $N$, mixture parameter $\epsilon$

**Output:** $\hat{\pi}$

---

 1: **The notation is not very clear – WORK IN PROGRESS**
 2: Initialize experience memory $\Gamma$
 3: Initialize action-value function $\mathcal{Q}$ with random weights
 4: $i \leftarrow \emptyset$
 5: **for** $n \in \{0, 1, .., N-1\}$ **do**
 6:    **for** $q \in \mathcal{Q}$ **do**
 7:       **for** $t \in \{0, 1, ..., T-1\}$ **do**
 8:          Execute $\pi_i$ $t$ times and reach $s_t$
 9:          **for** $a \in \mathcal{A}(s_t)$ **do**
10:             With probability $\epsilon$ set $\pi^0 = \pi_q^*$, else $\pi_i$
11:             Compute $c_t(a)$ by executing $\pi^0$
12:             $\Gamma \leftarrow \Gamma \cup \{[\phi(s_t), a, c_t(a)]\}$
13:          **end for**
14:       **end for**
15:       Sample random minibatches of transitions $\gamma_j$ from $\Gamma$
16:       Set $y_j = r_j$ for terminal $\phi_{j+1}$
17:       Perform gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$
18:       $i \leftarrow i + 1$
19:    **end for**
20: **end for**

---

**Algorithm 2** Streaming DQN-LSTM

---

**Input:** $\{x_q, \pi_q^*\}_{\forall q \in \mathcal{Q}}$, number of iterations $N$, mixture parameter $\epsilon$

**Output:** $\hat{\pi}$

---

  1: **The notation is not very clear – WORK IN PROGRESS**
  2: Initialize $\pi_0$, $\Gamma$
  3: Initialize action-value function $\mathcal{Q}$ with random weights
  4: $i \leftarrow \emptyset$
  5: **for** $n \in \{0, 1, .., N-1\}$ **do**
  6:     **for** $q \in \mathcal{Q}$ **do**
  7:         **for** $t \in \{0, 1, ..., T-1\}$ **do**
  8:             Execute $\pi_i$ $t$ times and reach $s_t$
  9:             **for** $a \in \mathcal{A}(s_t)$ **do**
10:                 With probability $\epsilon$ set $\pi^0 = \pi_q^*$, else $\pi_i$
11:                 Compute $c_t(a)$ by executing $\pi^0$
12:                 $\Gamma \leftarrow \Gamma \cup \{[\phi(s_t), a, c_t(a)]\}$
13:             **end for**
14:         **end for**
15:         Sample random minibatches of transitions $\gamma_j$ from $\Gamma$
16:         Set $y_j = r_j$ for terminal $\phi_{j+1}$
17:         Perform gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$
18:         $i \leftarrow i + 1$
19:     **end for**
20: **end for**

---

incrementing the current system time,

$$\tilde{Y}_{\leq t+1} = \tilde{Y}_{\leq t} \cup \{x_t\}$$
$$t = t + 1$$

or *skip* where only $t$ is incremented without changing the current summary

$$\tilde{Y}_{\leq t+1} = \tilde{Y}_{\leq t}$$
$$t = t + 1.$$

**Reward** The reward for a given action will be measured by relative gain in ROUGE-2 F1 score of the predicted summary $\tilde{Y}_{\leq t}$ 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 ngrams$(Y, N)$ returns the set of ngrams of order $N$ in the summary $Y$ and count$(g, Y)$ is the count of occurences of ngram $g$ in $Y$.

Similarly, ROUGE-N Precision is calculated as

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

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:

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

TODO: Think about how to incorporate a time penalty into the reward.

# 3  Useful Links

Just adding some links to useful things
```
http://trec.nist.gov/
http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf
```

# 4  To-Do's

| Task | Status | Person |
|---|---|---|
| Read DQN Paper [7] | Complete | Francisco |
| Read Asynchronous RL Paper [6] | Briefed | Francisco |
| Read Regularization Paper [1] | Complete | Francisco |
| Read Disaster Summarization Paper [4] | Complete | Francisco |
| Read Sequential Decision Making Paper [3] | Complete | Francisco |
| Install Torch | Complete | Francisco |
| Small simulation of MLP in Torch | Complete | Francisco |
| Small simulation of DQN in Torch | Incomplete | Francisco |
| Download data | Incomplete | Francisco |
| Explore data | In-Progress | Francisco |

# References

[1] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[2] 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.

[3] Chris Kedzie, Fernando Diaz, and Kathleen McKeown. Real-time web scale event summarization using sequential decision making. *arXiv preprint arXiv:1605.03664*, 2016.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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