Sample-efficient Deep Reinforcement Learning for Dialog Control

Kavosh Asadi
Brown University
kavosh@brown.edu

Jason D. Williams
Microsoft Research
jason.williams@microsoft.com

Abstract

Representing a dialog policy as a recurrent neural network (RNN) is attractive because it handles partial observability, infers a latent representation of state, and can be optimized with supervised learning (SL) or reinforcement learning (RL). For RL, a policy gradient approach is natural, but is sample inefficient. In this paper, we present 3 methods for reducing the number of dialogs required to optimize an RNN-based dialog policy with RL. The key idea is to maintain a second RNN which predicts the value of the current policy, and to apply experience replay to both networks. On two tasks, these methods reduce the number of dialogs/episodes required by about a third, vs. standard policy gradient methods.

1 Introduction

We study the problem of using reinforcement learning (RL) to optimize a controller represented as a recurrent neural network (RNN). RNNs are attractive because they accumulate sequential observations into a latent representation of state, and thus naturally handle partially observable environments, such as dialog, and also robot navigation, autonomous vehicle control, and others.

Among the many methods for RL optimization [Sutton and Barto, 1998], we adopt the *policy gradient* approach [Williams, 1992]. Policy gradient approaches are a natural fit for recurrent neural networks because both make updates via stochastic gradient descent. They also have strong convergence characteristics compared to value-function methods such as Q-learning, which can diverge when using function approximation [Precup et al., 2001]. Finally, the form of the policy makes it straightforward to also train the model from expert trajectories (ie, training dialogs), which are often available in real-world settings.

Despite these advantages, in practice policy gradient methods are often sample inefficient, which is limiting in real-world settings where explorational interactions – ie, conducting dialogs – can be expensive.

The contribution of this paper is to present a family of new methods for increasing the sample efficiency of policy gradient methods, where the policy is represented as a recurrent neural network (RNN). Specifically, we make two changes to the standard policy gradient approach. First, we estimate a *second* RNN which predicts the expected future reward the policy will attain in the current state; during updates, the value network reduces the error (variance) in the gradient step, at the expense of additional computation for maintaining the value network. Second, we add experience replay to both networks, allowing more gradient steps to be taken per dialog.

This paper is organized as follows. The next section reviews the policy gradient approach, Section 3 presents our methods, Sections 4 and 5 present results on two tasks, Section 6 covers related work, and Section 7 briefly concludes.

2 Preliminaries

In a reinforcement learning problem, an agent interacts with a stateful environment to maximize a numeric reward signal. Concretely, at a timestep t, the agent takes an action a_t, is awarded a real-valued reward r_{t+1}, and receives an observation vector $\mathbf{o}_{t+1}$. The goal of the agent is to choose actions to maximize the discounted sum of future rewards, called the return, G_t. In an episodic problem, the return at a timestep t is:

$$G_t = \sum_{i=1}^{T-t} \gamma^{i-1} r_{t+i} \ , \qquad (1)$$

where T is the terminal timestep, and γ is a discount factor $0 \leq \gamma \leq 1$.

In this paper, we consider policies represented as a recurrent neural network (RNN). Internally the RNN maintains a vector representing a latent state $\mathbf{s}$, and the latent state begins in a fixed state $\mathbf{s}_0$. At each timestep $t = 1, 2, \ldots$, an RNN takes as input an observation vector $\mathbf{o}_t$, updates its internal state according to a differentiable function $F(\mathbf{s}_{t-1}, \mathbf{o}_t | \boldsymbol{\theta}_f) = \mathbf{s}_t$, and outputs a distribution over actions $\mathbf{a}_t$ according to a differentiable function $G(\mathbf{s}_t | \boldsymbol{\theta}_g) = \mathbf{a}_t$, where $\boldsymbol{\theta} = (\boldsymbol{\theta}_f, \boldsymbol{\theta}_g)$ parameterize the functions. F and G can be chosen to implement long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997], gated recurrent unit [Cho et al., 2014], or other recurrent (or non-recurrent) models. $\pi^{\theta}(a_t | \mathbf{h}_t)$ denotes the output of the RNN at timestep t.

Past work has established a principled method for updating the parameters $\boldsymbol{\theta}$ of the policy $\pi^{\boldsymbol{\theta}}$ via RL [Williams, 1992, Sutton et al., 2000, Peters and Schaal, 2006] via stochastic gradient descent:

$$\triangle \boldsymbol{\theta}_d = \sum_{t=0}^{T-1} \triangledown_{\boldsymbol{\theta}} \log \pi^{\boldsymbol{\theta}}(a_t | \mathbf{h}_t) G_t \ . \qquad (2)$$

While this update is unbiased, in practice it has high variance and is slow to converge. Williams [1992] and Sutton et al. [2000] showed that this update can be re-written as

$$\triangle \boldsymbol{\theta}_d = \sum_{t=0}^{T-1} \triangledown_{\boldsymbol{\theta}} \log \pi^{\theta}(a_t | \mathbf{h}_t)(G_t - b), \qquad (3)$$

where b is a baseline, which can be an arbitrary function of states visited in dialog d. Note that this update assumes that actions are drawn from the same policy parameterized by θ – ie, this is an *on-policy* update.

Throughout the paper, we also use importance sampling ratios that enable us to perform off-policy updates. Assume that some behavior policy μ is used to generate dialogs, and may in general be different from the target policy π we wish to optimize. At timestep t, we define the importance sampling ratio as

$$\rho_t = \frac{\pi(a_t | \mathbf{h}_t)}{\mu(a_t | \mathbf{h}_t)}. \qquad (4)$$

3 Methods

3.1 Benchmarks

Before introducing our methods, we first describe our two benchmarks. The first uses (2) directly. The second uses (3), with b computed as an estimate of the average return of π:

$$b = \frac{\sum_{d \in \mathcal{D}} w(d) G_0^d}{\sum_{d \in \mathcal{D}} w(d)} \qquad (5)$$

where $\mathcal{D}$ is a window of most recent episodes (dialogs), and the weight of each dialog $w(d)$ is $w(d) = \Pi_{t=0}^{T(d)-1} \rho_t$. To compute ratios using (4), the policy that generated the data is μ, and the current (ie, target) policy is π.

3.2 Method 1: State value function as baseline

Our first method modifies parameter update (3) by using a per-timestep baseline:

$$\triangle \boldsymbol{\theta}_d^{\text{on}} = \sum_{t=0}^{T-1} \triangledown_{\boldsymbol{\theta}} \log \pi^{\boldsymbol{\theta}}(a_t | \mathbf{h}_t) \big(G_t - \hat{V}(\mathbf{h}_t, \mathbf{w}) \big), \qquad (6)$$

where $\hat{V}(\mathbf{h}_t, \mathbf{w})$ is an estimate of the value of $\pi^{\boldsymbol{\theta}}$ starting from $\mathbf{h}_t$, and $\mathbf{w}$ parameterizes $\hat{V}$. This method allows a gradient step to be taken in light of the value of the current state. We implement $\hat{V}(\mathbf{h}_t, \mathbf{w})$ as a *second* RNN, and update its parameters at the end of each dialog using supervised learning, as

$$\triangle \mathbf{w}_d^{\text{on}} = \sum_{t=0}^{T-1} \big(G_t - \hat{V}(\mathbf{h_t}, \mathbf{w}) \big) \triangledown_{\mathbf{w}} \hat{V}(\mathbf{h_t}, \mathbf{w}) \qquad (7)$$

Note that this update is also on-policy since the policy generating the episode is the same as the policy for which we want to estimate the value.

3.3 Method 2: Experience replay for value network

Method 2 increases learning speed further by reusing past dialogs to better estimate $\hat{V}(\mathbf{h_t}, \mathbf{w})$. Since the policy changes after each dialog, past dialogs are *off-policy* with respect to $\hat{V}(\mathbf{h_t}, \mathbf{w})$, so a correction to (7) is needed. Precup et al. [2001] showed that the following off-policy update is equal to the on-policy update, in expectation:

$$\triangle \mathbf{w}_d^{\text{off}} = \sum_{t=0}^{T-1} \prod_{i=0}^{t-1} \rho_i \Big(\prod_{j=0}^{T-1} \rho_j G_t - \hat{V}(\mathbf{h_t}, \mathbf{w}) \Big) \triangledown_{\mathbf{w}} \hat{V}(\mathbf{h_t}, \mathbf{w}).$$

Our second method takes a step with $\triangle\mathbf{w}_d^{\text{on}}$ with d as the last dialog, and one or more steps with $\triangle\mathbf{w}_d^{\text{off}}$ where d is sampled from recent dialogs.

3.4 Method 3: Experience replay for policy network

Our third method improves over the second method by applying experience replay to the policy network. Specifically, Degris et al. [2012] shows that samples of the following expectation, which is under behavior policy μ, can be used to estimate the gradient of the policy network representing the policy π^θ:

$$\mathbf{E}[\rho_t \nabla_{\boldsymbol{\theta}} \log \pi^\theta(a_t|\mathbf{h}_t) Q^\pi(\mathbf{h}_t, a_t)|\mu] \qquad (8)$$

We do not have access to Q^π, but since $Q^\pi(\mathbf{h}_t, a_t) = r_t + \gamma V^\pi(\mathbf{h}_{t+1})$, we can state the following off-policy update for the policy network:

$$\triangle\boldsymbol{\theta}_d^{\text{off}} = \sum_{t=0}^{T-1} \rho_t \nabla_{\boldsymbol{\theta}} \log \pi^\theta(a_t|\mathbf{h}_t) \left(r_t + \gamma\hat{V}(\mathbf{h_{t+1}}, \mathbf{w}) - \hat{V}(\mathbf{h_t}, \mathbf{w})\right)$$

Method 3 first applies Method 2, then updates the policy network by taking one step with $\triangle\boldsymbol{\theta}_d^{\text{on}}$, followed by one or more steps with $\triangle\boldsymbol{\theta}_d^{\text{off}}$ using samples from recent dialogs.

4 Problem 1: dialog system

To test our approach, we created a dialog system for initiating phone calls to a contact in an address book, taken from the Microsoft internal employee directory. Full details are given in Williams and Zweig [2016]; briefly, in the dialog system, there are three entity types – `name`, `phonetype`, and `yesno`. A contact's name may have synonyms ("Michael"/"'Mike") and a contact may have more than one phone types (eg "work", "mobile") which in turn have synonyms (eg "cell" for "mobile").

To run large numbers of RL experiments, we then created a stateful goal-directed simulated user, where the goal was sometimes not covered by the dialog system, and where the behavior was stochastic – for example, the simulated user usually answers questions, but can instead ignore the system, provide additional information, or give up. The user simulation was parameterized with around 10 probabilities.

We defined the reward as $+1$ for successfully completing the task, and 0 otherwise. $\gamma = 0.95$ was used to incentivize the system to complete dialogs faster rather than slower. For the policy network, we defined F and G to implement an LSTM with 32 hidden units, with a dense output layer with a softmax activation. The value network was identical in structure except it had a single output with a linear activation. We used a batch size of 1, so we update both networks after completion of a single dialog. We used Adadelta with stepsize $\alpha = 1.0$, $\rho = 0.95$, and $\epsilon = 10^{-6}$. Dialogs took between 3 and 10 timesteps. Every 10 dialogs, the policy was frozen and run for 1000 dialogs to measure average task completion.

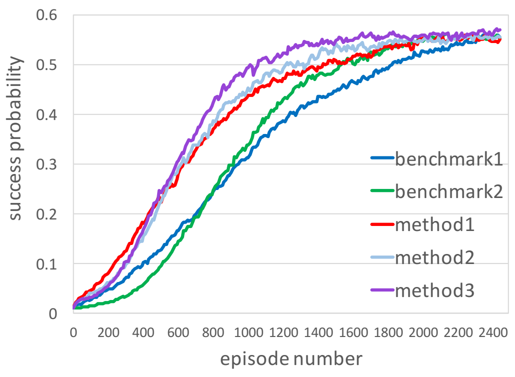

Figure 1: Number of dialogs vs. average task success over 200 runs for the dialog task.

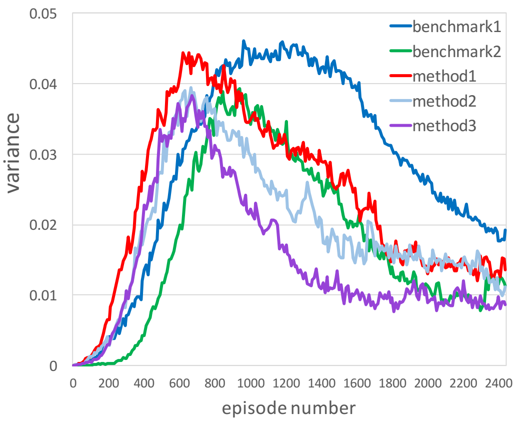

Figure 2: Number of dialogs vs. variance in task success among 200 runs.

Figures 1 and 2 show mean and variance for task completion over 200 independent runs. Compared to the benchmarks, our methods require about one third fewer dialogs to attain asymptotic performance, and have lower variance.

5 Problem 2: lunar lander

To test generality and provide for reproducibility, we sought to evaluate on a publicly available

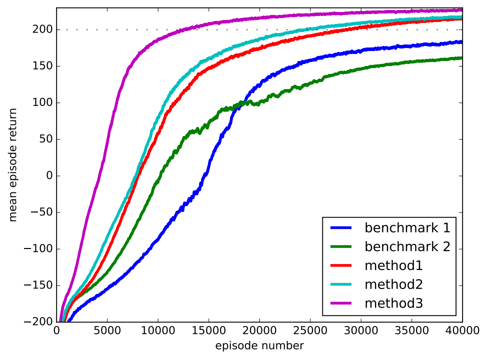

Figure 3: Number of epsidoes vs. average return over 200 runs for the lunar lander task.

dialog task. However, to our knowledge none exists, and so we instead applied our method to a public but non-dialog RL task, called "Lunar Lander", from the OpenAI gym.[1] This domain has a continuous (fully-observable) state space in 8 dimensions (eg, x-y coordinates, velocities, leg-touchdown sensors, etc.), and 4 discrete actions (controlling 3 engines). The reward is +100 for a safe landing in the designated area, and -100 for a crash. Using the engines will also result in a negative cost as explained in the link below. Episodes finish when the spacecraft crashes or lands. We used $\gamma = 0.99$.

Since this domain is fully observable, we chose definitions of F and G in the policy network corresponding to a fully connected neural networks with 2 hidden layers, followed by a softmax normalization. We further chose RELU activations and 16 hidden units, based on limited initial experimentation. The value network has the same architecture except for the output layer that has a single node with linear activation. We used a batch episode size of 10, as we found that with a batch of 1 divergence appears frequently. We used a stepsize of 0.005 and used Adam algorithm [Kingma and Ba, 2014] with its default parameters. Methods 2 and 3 performs 5 off-policy updates per each on-policy update for the value network. Method 3 performs 3 off-policy updates per each on-policy update for the policy network.

Results are in Figure 3, and show a similar increase in sample efficiency as in the dialog task.

[1] `https://gym.openai.com/envs/LunarLander-v2`

6 Related work

Since neural networks naturally lend themselves to policy gradient-style updates, much past work has adopted this broad approach. However, most work has studied the fully observable case, whereas we study the partially observable case. For example, AlphaGo [Silver et al., 2016] applies policy gradients (among other methods), but Go is fully observable via the state of the board.

Several papers that study fully-observable RL are related to our work in other ways. Degris et al. [2012] investigates off-policy policy gradient updates, but is limited to linear models. Our use of experience replay is also off-policy optimization, but we apply (recurrent) neural networks. Like our work, Fatemi et al. [2016] also estimates a value network, uses experience replay to optimize that value network, and evaluates on a conversational system task. However, unlike our work, they do not use experience replay in the policy network, their networks rely on an external state tracking process to render the state fully-observable, and they learn feed-forward networks rather than recurrent networks.

Hausknecht and Stone [2015] applies RNNs to partially observable RL problems, but adopts a Q-learning approach rather than a policy gradient approach. Whereas policy gradient methods have strong convergence properties, Q-learning can diverge, and we observed this when we attempted to optimize Q represented as an LSTM on our dialog problem. Also, a policy network can be pre-trained directly from (near-) expert trajectories using classical supervised learning, and in real-world applications these trajectories are often available [Williams and Zweig, 2016].

7 Conclusions

We have introduced 3 methods for increasing sample efficiency in policy-gradient RL. In a dialog task with partially observable state, our best method improved sample efficiency by about a third. On a second fully-observable task, we observed a similar gain in sample efficiency, despite using a different network architecture, activation function, and optimizer. This result shows that the method is robust to variation in task and network design, and thus it seems promising that it will generalize to other dialog domains as well. In future work we will apply the method to a dialog system with real human users

References

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Franois Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. Policy networks with two-stage training for dialogue systems. In *SIGDIAL2016*, 2016.

Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015. URL `http://arxiv.org/abs/1507.06527`.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2219–2225. IEEE, 2006.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Sander Dieleman Marc Lanctot, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489, 2016.

R Sutton and A Barto. *Reinforcement Learning: an Introduction.* MIT Press, 1998.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS) 12, Denver, USA*, pages 1057–1063, 2000.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

Jason D. Williams and Geoffrey Zweig. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *CoRR*, abs/1606.01269, 2016. URL `http://arxiv.org/abs/1606.01269`.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8 (23), 1992.