BATCH POLICY GRADIENT METHODS FOR IMPROVING NEURAL CONVERSATION MODELS

Kirthevasan Kandasamy *
Carnegie Mellon University, Pittsburgh, PA, USA
kandasamy@cs.cmu.edu

Yoram Bachrach ♭
Digital Genius, London, UK
yorambac@gmail.com

Ryota Tomioka, Daniel Tarlow, David Carter
Microsoft Research, Cambridge, UK
{ryoto,dtarlow,dacart}@microsoft.com

* ♭ This work was done when KK/YB was an intern/employee at Microsoft Research, Cambridge, UK.

ABSTRACT

We study reinforcement learning of chat-bots with recurrent neural network architectures when the rewards are noisy and expensive to obtain. For instance, a chat-bot used in automated customer service support can be scored by quality assurance agents, but this process can be expensive, time consuming and noisy. Previous reinforcement learning work for natural language uses on-policy updates and/or is designed for on-line learning settings. We demonstrate empirically that such strategies are not appropriate for this setting and develop an off-policy batch policy gradient method (BPG). We demonstrate the efficacy of our method via a series of synthetic experiments and an Amazon Mechanical Turk experiment on a restaurant recommendations dataset.

1 INTRODUCTION

Chat-bots are one of the classical applications of artificial intelligence and are now ubiquitous in technology, business and everyday life. Many corporate entities are now increasingly using chat-bots to either replace or assist humans in customer service contexts. For example, Microsoft is currently actively building a chat bot to optimise and streamline its technical support service.

In these scenarios, there is usually an abundance of historical data since past conversations between customers and human customer service agents are usually recorded by organisations. An apparently straightforward solution would be to train chat-bots to reproduce the responses by human agents using standard techniques such as maximum likelihood. While this seems natural, it is far from desirable for several reasons. It has been observed that such procedures have a tendency to produce very generic responses (Sordoni et al., 2015). For instance, when we trained chat-bots via maximum likelihood on a restaurant recommendations dataset, they repeatedly output responses to the effect of `How large is your group?`, `What is your budget?` etc. Further, they also produce responses such as `Let me look that up.` or `Give me a second.` which, although permissible for a human agent to say, are not appropriate for a chat-bot. Although there are ways to increase the diversity of responses (Li et al., 2015), our focus is on encouraging the bot to meaningfully advance the conversation. One way to address this problem is to provide some form of weak supervision for responses generated by a chat-bot. For example, a human labeller, such as a quality assurance agent, could score each response generated by a chat-bot in a conversation with a customer. This brings us to the reinforcement learning (RL) paradigm where these rewards (scores) are to be used to train a good chat-bot. In this paper we will use the terms score, label and reward interchangeably. Labelled data will mean conversations which have been assigned a reward of some form as explained above.

Nonetheless, there are some important differences in the above scenario when compared to the more popular approaches for RL.

- **Noisy and expensive rewards**: Obtaining labels for each conversation can be time consuming and economically expensive. As a result, there is a limited amount of labelled data available. Moreover, labels produced by humans are invariably noisy due to human error and subjectivity.

- **Off-line evaluations:** Unlike conventional RL settings, such as games, where we try to find the optimal policy while interacting with the system, the rewards here are not immediately available. Previous conversations are collected, labelled by human experts and then given to an algorithm which has to make do with the data it has.
- **Unlabelled Data:** While labelled data is limited, a large amount of unlabelled data is available.

If labelled data is in short supply, reinforcement learning could be hopeless. However, if unlabelled data can be used to train a decent initial bot, say via maximum likelihood, we can use *policy iteration* techniques to refine this bot by making local improvements using the labelled data. Besides chat-bots, this framework also finds applications in tasks such as question answering (Ferrucci et al., 2010; Hermann et al., 2015; Sachan et al., 2016), generating image descriptions (Karpathy & Fei-Fei, 2015) and machine translations (Bahdanau et al., 2014) where a human labeller can provide weak supervision in the form of a score to a sentence generated by a bot.

To contextualise the work in this paper, we make two important distinctions in policy iteration methods in reinforcement learning. The first is on-policy vs off-policy. In on-policy settings, the goal is to improve the current policy on the fly while exploring the space. On-policy methods are used in applications where it is necessary to be competitive (achieve high rewards) while simultaneously exploring the environment. In off-policy, the environment is explored using a behaviour policy, but the goal is to improve a different target policy. The second distinction is on-line vs batch (off-line). In on-line settings one can interact with the environment. In batch methods, which is the setting for this work, one is given past exploration data from possibly several behaviour policies and the goal is to improve a target policy using this data. On-line methods can be either on-policy or off-policy whereas batch methods are necessarily off-policy.

In this paper, we study reinforcement learning in batch settings, for improving chat bots with Seq2Seq recurrent neural network (RNN) architectures. One of the challenges when compared to on-line learning is that we do not have interactive control over the environment. We can only hope to do as well as our data permits us to. On the other hand, the batch setting affords us some luxuries. We can reuse existing data and use standard techniques for hyper-parameter tuning based on cross validation. Further, in on-line policy updates, we have to be able to "guess" how an episode will play out, i.e. actions the behaviour/target policies would take in the future and corresponding rewards. However, in batch learning, the future actions and rewards are directly available in the data. This enables us to make more informed choices when updating our policy.

Related Work

Recently there has been a surge of interest in deep learning approaches to reinforcement learning, many of them adopting Q-learning, e.g. (He et al., 2015; Mnih et al., 2013; Narasimhan et al., 2015). In Q-learning, the goal is to estimate the *optimal action value function* Q^*. Then, when an agent is at a given state, it chooses the best greedy action according to Q^*. While Q-learning has been successful in several applications, it is challenging in the settings we consider since estimating Q^* over large action and state spaces will require a vast number of samples. In this context, policy iteration methods are more promising since we can start with an initial policy and make incremental local improvements using the data we have. This is especially true given that we can use maximum likelihood techniques to estimate a good initial bot using unlabelled data.

Policy gradient methods, which fall within the paradigm of policy iteration, make changes to the parameters of a policy along the gradient of a desired objective (Sutton et al., 1999). Recently, the natural language processing (NLP) literature has turned its attention to policy gradient methods for improving language models. Ranzato et al. (2015) present a method based on the classical REINFORCE algorithm (Williams, 1992) for improving machine translation after preliminary training with maximum likelihood objectives. Bahdanau et al. (2016) present an actor-critic method also for machine translation. In both cases, as the reward, the authors use the BLEU score of the output and the translation in the training dataset. Here, the rewards are deterministic and cheaply computable, and hence does not reflect difficulties inherent to training chat-bots where labels are noisy and expensive. Li et al. (2016) develop a policy gradient method bot for chat-bots. However, they use user defined rewards (based on some simple rules) which, once again, are cheaply obtained and deterministic. Perhaps the closest to our work is that of Williams & Zweig (2016) who use a REINFORCE based method for chat bots. We discuss the differences of this and other methods in greater detail in

Section 3. The crucial difference between all of the above efforts and ours is that they use on-policy and/or on-line updates in their methods.

The remainder of this manuscript is organised as follows. In Section 2 we review Seq2Seq models and Markov decision processes (MDP) and describe our framework for batch reinforcement learning. Section 3 presents our method BPG and compares it with prior work in the RL and NLP literature. Section 4 presents experiments on a synthetic task and a customer service dataset for restaurant recommendations.

2 PRELIMINARIES

2.1 A REVIEW OF SEQ2SEQ MODELS

The goal of a Seq2Seq model (Cho et al., 2014; Kalchbrenner & Blunsom, 2013; Sutskever et al., 2014) in natural language processing is to produce an output sequence $y = [a_1, a_2, \ldots, a_T]$ given an input sequence x. Here $a_i \in \mathcal{A}$ where $\mathcal{A}$ is a vocabulary of words. For example, in machine translation from French to English, x is the input sequence in French, and y is its translation in English. In customer service chat-bots, x is the conversation history until the customer's last query and y is the response by an agent/chat-bot. In a Seq2Seq model, we use an encoder network to represent the input sequence as a euclidean vector and then a decoder network to convert this vector to an output sequence. Typically, both the encoder and decoder networks are recurrent neural networks (RNN) (Mikolov et al., 2010) where the recurrent unit processes each word in the input/output sequences one at a time. In this work, we will use the LSTM (long short term memory) (Hochreiter & Schmidhuber, 1997) as our recurrent unit.

In its most basic form, the decoder RNN can be interpreted as assigning a probability distribution over $\mathcal{A}$ given the current "state". At time t, the state s_t is the input sequence x and the words $y_{t-1} = [a_1, \ldots, a_{t-1}]$ produced by the decoder thus far, i.e. $s_t = (x, y_{t-1})$. We sample the next word a_t from this probability distribution $\pi(\cdot|s_t)$, then update our state $s_{t+1} = (x, y_t)$ where $y_t = [y_{t-1}, a_t]$, and proceed in a similar fashion. The vocabulary $\mathcal{A}$ contains an end-of-statement token `<EOS>`. If we sample `<EOS>` at time $T + 1$, we terminate the sequence and output y_T.

2.2 A REVIEW OF MARKOV DECISION PROCESSES (MDP)

We present a formalism for MDPs simplified to our setting. In an MDP, an agent takes an action a in a state s and transitions to a state s'. An *episode* refers to a sequence of transitions $s_1 \to a_1 \to s_2 \to a_2 \to \cdots \to a_T \to s_{T+1}$ until the agent reaches a terminal state s_{T+1}. At a terminal state, the agent receives a reward. Formally, an MDP is the triplet $(\mathcal{S}, \mathcal{A}, R)$. Here, $\mathcal{S}$ is a set of states and $\mathcal{A}$ is a set of actions. When we take an action a at state s we transition to a new state $s' = s'(s, a)$ which, in this work, will be deterministic. $\mathcal{A}$ will be a finite but large discrete set and $\mathcal{S}$ will be discrete but potentially infinite. $R : \mathcal{S} \to \mathbb{R}$ is the expected reward function such that when we receive a reward r at state $s \in \mathcal{S}$, $\mathbb{E}[r] = R(s)$. Let $\mathcal{S}_0 \subset \mathcal{S}$ be a set of terminal states. When we transition to any $s \in \mathcal{S}_0$, the episode ends. In this work, we will assume that rewards are received only at a terminal state, i.e $R(s)$ is nonzero only on $\mathcal{S}_0$.

A policy π is a rule to select an action at a given state. We will be focusing on stochastic policies $\pi : \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ where $\pi(a|s)$ denotes the probability an agent will execute action a at state s. We define the *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ of policy π, where $V(s)$ is the expected reward at the end of the episode when we follow policy π from state s. For any terminal state $s \in \mathcal{S}_0$, $V^\pi(s) = R(s)$ regardless of π. We will also find it useful to define the *action-value function* $Q^\pi : \mathcal{S} \times \mathcal{A} :\to \mathbb{R}$, where $Q^\pi(s, a)$ is the expected reward of taking action a at state s and then following policy π. With deterministic state transitions this is simply $Q^\pi(s, a) = V^\pi(s'(s, a))$. It can be verified that $V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s, a)\right]$ (Sutton & Barto, 1998).

2.3 SET UP

We now frame our learning from labels scenario for RNN chat-bots as an MDP. The treatment has similarities to some recent RL work in the NLP literature discussed above.

Let x be the input and $y_{t-1} = [a_1, \ldots, a_{t-1}]$ be the words output by the decoder until time t. The state of our MDP at time t of the current episode will be $s_t = (x, y_{t-1})$. Therefore, the set of states $\mathcal{S}$ will be all possible pairs of inputs and partial output sequences. The actions $\mathcal{A}$ will be the vocabulary. The terminal states $\mathcal{S}_0$ will be (x, y) such that the last literal of y is `<EOS>`. The stochastic policy π will be a Seq2Seq RNN which produces a distribution over $\mathcal{A}$ given state s_t. When we wish to make the dependence of the policy on the RNN parameters θ explicit, we will write π_θ. When we sample an action $a_t \sim \pi(\cdot|s_t)$, we deterministically transition to state $(x, [y_{t-1}, a_t])$. If we sample $a_{T+1} =$ `<EOS>` at time $T+1$, the episode terminates and we observe a stochastic reward.

We are given a dataset of input-output-reward triples $\{(x^{(i)}, y^{(i)}, r^{(i)})\}_{i=1}^n$ where $y^{(i)} = (a_1^{(i)}, \ldots, a_{T_i}^{(i)}, \texttt{<EOS>})$ is the sequence of output words. This data was collected from possibly multiple *behaviour policies* which output $y^{(i)}$ for the given input $x^{(i)}$. In the above customer service example, the behaviour policies could be chat-bots, or even humans, which were used for conversations with a customer. The rewards r_i are scores assigned by a human quality assurance agent to each response of the chat-bot. Our goal is to use this data to improve a given target policy π_θ. We will use q to denote the distribution of the data. $q(s)$ is the distribution of the states in the dataset, $q(a|s)$ is the conditional distribution of an action given a state, and $q(s,a) = q(s)q(a|s)$ is the joint distribution over states and actions. q will be determined by the initial distribution of the inputs $x^{(i)}$ and the behaviour policies used to collect the training data. Our aim is to find a policy that does well with respect to q. Specifically, we wish to maximise the following objective.

$$J(\theta) = \sum_{s \in \mathcal{S}} q(s) V^{\pi_\theta}(s) \tag{1}$$

This is similar to objectives used in on-line off-policy policy gradient literature where q is replaced by the limiting distribution of the behaviour policy (Degris et al., 2012). In the derivation of our algorithm, we will need to know $q(a|s)$ to compute the gradient of our objective. In off-policy reinforcement learning settings this is given by the behaviour policy which is readily available. If the behaviour policy if available to us, then we can use this too. Otherwise, a simple alternative is to "learn" a behaviour policy. For example, in our experiments we used an RNN trained using the unlabelled data to obtain values for $q(a|s)$. As long as this learned policy can capture the semantics of natural language (for example, the word `apple` is more likely than `car` when the current state is $(x, \texttt{I ate an})$), then it can be expected to do reasonably well. In the following section, we will derive a stochastic gradient descent (SGD) procedure that will approximately minimise (1).

Before we proceed, we note that it is customary in the RL literature to assume stochastic transitions between states and use rewards at all time steps instead of the terminal step. Further, the future rewards are usually discounted by a discount factor $\gamma < 1$. While we use the above formalism to simplify the exposition, the ideas presented here extend naturally to more conventional settings.

3 Batch Policy Gradient

Our derivation follows the blueprint in Degris et al. (2012) who derive an off-policy on-line actor critic algorithm. Following standard policy gradient methods, we will aim to update the policy by taking steps along the gradient of the objective $\nabla J(\theta)$.

$$\nabla J(\theta) = \nabla \mathbb{E}_{s\sim q}\left[\sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right] = \mathbb{E}_{s\sim q}\left[\sum_{a\in\mathcal{A}} \nabla\pi_\theta(a|s) Q^{\pi_\theta}(s,a) + \pi_\theta(a|s)\nabla Q^{\pi_\theta}(s,a)\right].$$

The latter term inside the above summation is difficult to work with, so the first step is to ignore it and work with the approximate gradient $g(\theta) = \mathbb{E}_{s\sim q}[\sum_{a\in\mathcal{A}} \nabla\pi_\theta(a|s) Q^{\pi_\theta}(s,a)] \approx \nabla J(\theta)$. Degris et al. (2012) provide theoretical justification for this approximation in off policy settings by establishing that $J(\theta) \leq J(\theta + \alpha g(\theta))$ for all small enough α. Expanding on $g(\theta)$, we obtain:

$$\begin{aligned} g(\theta) &= \mathbb{E}_{s\sim q}\left[\sum_{a\in\mathcal{A}} \pi_\theta(a|s) \frac{\nabla\pi_\theta(a|s)}{\pi_\theta(a|s)} Q^{\pi_\theta}(s,a)\right] = \mathbb{E}_{\substack{s\sim q \\ a\sim q(\cdot|s)}}\left[\rho(s,a)\psi(a,s)Q^{\pi_\theta}(s,a)\right] \\ &= \mathbb{E}_{(s_t,a_t)\sim q(\cdot,\cdot)}\left[\rho(s_t,a_t)\psi(a_t,s_t)(Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t))\right]. \end{aligned} \tag{2}$$

Here $\psi(a,s) = \frac{\nabla\pi_\theta(a|s)}{\pi_\theta(a|s)} = \nabla \log \pi_\theta(a|s)$ is the *score function* of the policy and $\rho(s,a) = \pi_\theta(a|s)/q(a|s)$ is the *importance sampling* coefficient. In the last step, we have used the fact that

$\mathbb{E}[\pi(a|s)\psi(a|s)h(s)] = 0$ for any function $h : \mathcal{S} \to \mathbb{R}$ of the current state (Szepesvári, 2010). The purpose of introducing the value function V^{π_θ} is to reduce the variance of the SGD updates – we want to assess how good/bad action a_t is relative to how well π_θ will do at state s_t in expectation. If a_t is a good action ($Q^{\pi_\theta}(s_t, a_t)$ is large relative to $V^{\pi_\theta}(s_t)$), the coefficient of the score function is positive and it will change θ so as to assign a higher probability to action a_t at state s_t.

The $Q^{\pi_\theta}, V^{\pi_\theta}$ functions are not available to us so we will replace them with estimates. For $V^{\pi_\theta}(s_t)$ we will use an estimate $\widehat{V}(s_t)$ – we will discuss choices for this shortly. However, the action value function is usually not estimated in RL policy gradient settings to avoid the high sample complexity. A sensible stochastic approximation for $Q^{\pi_\theta}(s_t, a_t)$ is to use the sum of future rewards from the current state (Sutton & Barto, 1998)[1]. If we receive reward r at the end of the episode, we can then use $Q^{\pi_\theta}(s_t, a_t) \approx r$ for all time steps t in the episode. However, since $q(a_t|s_t)$ is different from $\pi_\theta(a_t|s_t)$ we will need to re-weight future rewards via importance sampling $r \prod_{i=t}^{T} \rho(s_i, a_i)$. This is to account for the fact that an action a given s may have been more likely under the policy $\pi_\theta(\cdot|s)$ than it was under $q(\cdot|s)$ or vice versa. Instead of directly using the re-weighted rewards, we will use the so called λ–return which is a convex combination of the re-weighted rewards and the value function (Sutton, 1988; 1984). In our setting, they are defined recursively from the end of the episode $t = T + 1$ to $t = 1$ as follows. For $\lambda \in (0, 1]$,

$$\tilde{r}^{\lambda}_{T+1} = r, \qquad \tilde{r}^{\lambda}_{t} = (1 - \lambda)V^{\pi_\theta}(s_{t+1}) + \lambda\rho(s_t, a_t)\tilde{r}^{\lambda}_{t+1} \quad \text{for } t = T, \ldots, 1. \tag{3}$$

The purpose of introducing λ is to reduce the variance of using the future rewards alone as an estimate for $Q^{\pi_\theta}(s_t, a_t)$. This is primarily useful when rewards are noisy. If the rewards are deterministic, $\lambda = 1$ which ignores the value function is the best choice. In noisy settings, it is recommended to use $\lambda < 1$ (see Sec 3.1 of (Szepesvári, 2010)). In our algorithm, we will replace $\tilde{r}^{\lambda}_{t}$ with r^{λ}_{t} where V^{π_θ} is replaced with the estimate $\widehat{V}$. Putting it all together, and letting α denote the step size, we have the following update rule for the parameters θ of our policy:

$$\theta \;\leftarrow\; \theta + \alpha\rho(s_t, a_t)\psi(s_t, a_t)(r^{\lambda}_{t} - \widehat{V}(s_t)).$$

In Algorithm 1, we have summarised the procedure where the updates are performed after an entire pass through the dataset. In practice, we perform the updates in mini-batches.

An Estimator for the Value Function: All that is left to do is to specify an estimator $\widehat{V}$ for the value function. We first need to acknowledge that this is a difficult problem: $\mathcal{S}$ is quite large and for typical applications for this work there might not be enough data since labels are expensive. That said, the purpose of $\widehat{V}$ in (2), (3) is to reduce the variance of our SGD updates and speed up convergence so it is not critical that this be precise – even a bad estimator will converge eventually. Secondly, standard methods for estimating the value function based on minimising the projected Bellman error require the second derivatives, which might be intractable for highly nonlinear parametrisations of $\widehat{V}$ (Maei, 2011). For these two statistical and computational reasons, we resort to simple estimators for V^{π_θ}. We will study two options. The first is a simple heuristic used previously in the RL literature, namely a constant estimator for $\widehat{V}$ which is equal to the mean of all rewards in the dataset (Williams, 1992). The second uses the parametrisation $\widehat{V}(s) = \sigma(\xi^\top \phi(s))$ where σ is the logistic function and $\phi(s) \in \mathbb{R}^d$ is a Euclidean representation of the state. For $\widehat{V}(s)$ of the above form, the Hessian $\nabla^2_\xi \widehat{V}(s)$ can be computed in $\mathcal{O}(d)$ time. To estimate this value function, we use the $\mathsf{GTD}(\lambda)$ estimator from Maei (2011). As $\phi(s)$ we will be using the hidden state of the LSTM. The rationale for this is as follows. In an LSTM trained using maximum likelihood, the hidden state contains useful information about the objective. If there is overlap between the maximum likelihood and reinforcement learning objectives, we can expect the hidden state to also carry useful information about the RL objective. Therefore, we can use the hidden state to estimate the value function whose expectation is the RL objective. We have described our implementation of $\mathsf{GTD}(\lambda)$ in Appendix A and specified some implementation details in Section 4.

[1] Note $Q^{\pi_\theta}(s_t, a_t) = V^{\pi_\theta}(s_{t+1})$ for deterministic transitions. However, it is important not to interpret the term in (2) as the difference in the value function between successive states. Conditioned on the current time step, $V^{\pi_\theta}(s_t)$ is deterministic, while $V^{\pi_\theta}(s_{t+1})$ is stochastic. In particular, while a crude estimate suffices for the former, the latter is critical and should reflect the rewards received during the remainder of the episode.

Algorithm 1 Batch Policy Gradient (BPG)

Given: Data $\{(x_i, y_i, r_i)\}_{i=1}^n$, step size α, return coefficient λ, initial θ_0.

- Set $\theta \leftarrow \theta_0$.
- For each epoch $k = 1, 2, \ldots$
  - ▶ Set $\Delta\theta \leftarrow \mathbf{0}$
  - ▶ For each episode $i = 1, \ldots, n$
    - $r^\lambda_{T+1} \leftarrow r_i$
    - $\rho_t \leftarrow \pi_\theta(a^{(i)}_t|s^{(i)}_t)/q(a^{(i)}_t|s^{(i)}_t)$ for $t = 1, \ldots, T^{(i)}$.
    - For each time step in *reverse* $t = T^{(i)}, \ldots, 1$
      - (i) $r^\lambda_t \leftarrow (1-\lambda)\widehat{V}(s^{(i)}_{t+1}) + \lambda\rho_t r^\lambda_{t+1}$
      - (ii) $\Delta\theta \leftarrow \Delta\theta + \frac{1}{T^{(i)}}\rho_t\psi(s^{(i)}_t, a^{(i)}_t)(r^\lambda_t - \widehat{V}(s^{(i)}_t))$
      - (iii) Compute updates for the value function estimate $\widehat{V}$.
  - ▶ Update the policy $\theta \leftarrow \theta + \alpha\Delta\theta$
  - ▶ Update the value function estimate $\widehat{V}$.

Comparison with Other RL Approaches in NLP

Policy gradient methods have been studied extensively in *on policy* settings where the goal is to improve the current policy on the fly (Amari, 1998; Williams, 1992). To our knowledge, all RL approaches in Seq2Seq models have also adopted on-policy policy gradient updates (Bahdanau et al., 2016; Li et al., 2016; Ranzato et al., 2015; Williams & Zweig, 2016). However, on policy methods break down in off-policy settings, because any update must account for the probability of the action under the target policy. For example, suppose the behaviour policy took action a at state s and received a low reward. Then we should modify the target policy θ so as to reduce $\pi_\theta(a|s)$. However, if the target policy is already assigning low probability to $a|s$ then we should not be as aggressive when making the updates. The re-weighting $\rho(s, a)$ via importance sampling does precisely this.

A second difference is that we study the *batch* setting. Standard on-line methods are designed for settings where we have to continually improve the target while exploring using the behaviour policy. A crucial component in these methods is a need to estimate the future rewards at the current state and the future actions that will be taken by both the behaviour and target policies. In order to tackle this, previous research either ignore future rewards altogether (Williams, 1992), resort to heuristics to distribute a delayed reward to previous time steps (Bahdanau et al., 2016; Williams & Zweig, 2016), or make additional assumptions about the distribution of the states (Degris et al., 2012; Maei, 2011). However, when data is available in batch settings, the λ-return from a given time step can be computed directly (3) since the future action and rewards are known. Access to this information provides a crucial advantage over techniques designed for on-line settings.

4 Experiments

Implementation Details: We implement our methods using Chainer (Tokui et al., 2015), and group sentences of the same length together in the same batch to make use of GPU parallelisation. Since different batches could be of different length we do not normalise the gradients by the batch size as we should take larger steps after seeing more data. However, we normalise by the length of the output sequence to allocate equal weight to all sentences. We truncate all output sequences to length 64 and use a maximum batch size of 32. We found it necessary to use a very small step size (10^{-5}), otherwise the algorithm has a tendency to get stuck at bad parameter values. While importance re-weighting is necessary in off-policy settings, it can increase the variance of the updates, especially when $q(a_t|s_t)$ is very small. A common technique to alleviate this problem is to clip the $\rho(s_t, a_t)$ value (Swaminathan & Joachims, 2015). In addition to single $\rho(s_t, a_t)$ values, our procedure has a product of $\rho(s_t, a_t)$ values when computing the future rewards (3). The effect of large ρ values is a large weight $\rho_t(r^\lambda_t - \widehat{V}(s_t))$ for the score function in step (ii) of Algorithm 1. In our implementation,

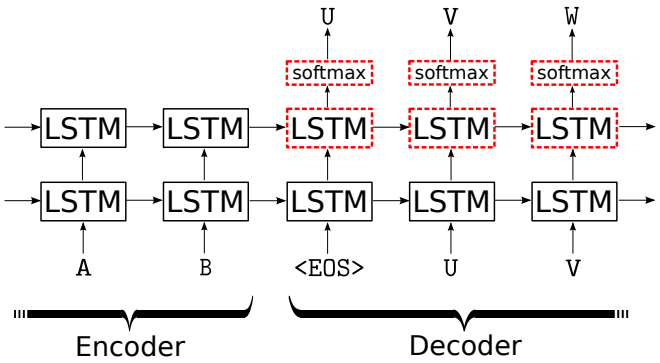


Figure 1: Illustration of the encoder and decoder RNNs used in our experiments. In this example, the input to the encoder is $x = (\ldots, \texttt{A}, \texttt{B}, \texttt{<EOS>})$ and the output of the decoder is $y = (\texttt{U}, \texttt{V}, \texttt{W}, \ldots)$. We use four different LSTMs for the bottom and top layers of the encoder and decoder networks. In our RL algorithms, we only change the top LSTM and the softmax layer of the decoder RNN as shown in red dashed lines.

we clip this weight at 5 which controls the variance of the updates and ensures that a single example does not disproportionately affect the gradient.

RNN Design: In both experiments we use deep LSTMs with two layers for the encoder and decoder RNNs. The output of the bottom layer is fed to the top layer and in the decoder RNN, the output of the top layer is fed to a softmax layer of size $|\mathcal{A}|$. When we implement GTD(λ) to estimate V^{π_θ} we use the hidden state of the bottom LSTM as $\phi(s)$. When performing our policy updates, we only change the parameters of the top LSTM and the softmax layer in our decoder RNN. If we were to change the bottom LSTM too, then the state representation $\phi(s)$ would also change as the policy changes. This violates the MDP framework. In other words, we treat the bottom layer as part of the environment in our MDP. To facilitate a fair comparison, we only modify the top LSTM and softmax layers in all methods. We have illustrated this set up in Fig. 1. We note that if one is content with using the constant estimator, then one can change all parameters of the RNN.

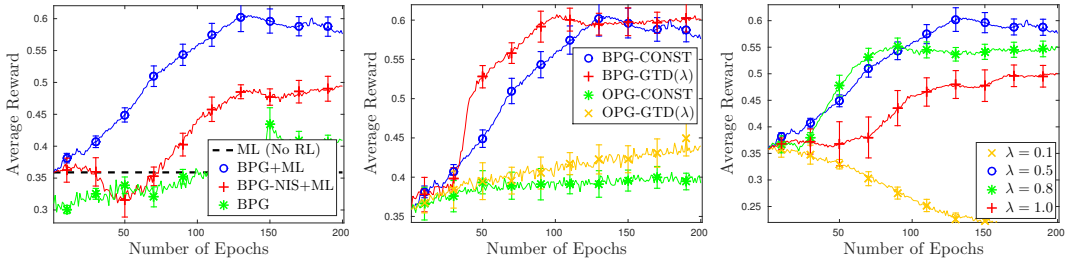
4.1 Some Synthetic Experiments on the Europarl Dataset

To convey the main intuitions of our method we compare our methods against other baselines on a synthetic task on the European parliament proceedings corpus (Koehn, 2005). We describe the experimental set up briefly, deferring details to Appendix B.1. The input sequence to the RNN was each sentence in the dataset. Given an input, the goal was to reproduce the words in the input without repeating words in a list of forbidden words. The RL algorithm does not explicitly know either goal of the objective but has to infer it from the *stochastic* rewards assigned to input output sequences in the dataset. We used a training set of 500 input-output-reward triplets for the RL methods.

We initialised all methods by maximum likelihood training on 6000 input output sequences where the output sequence was the reverse of the input sequence. The maximum likelihood objective captures part of the RL objective. This set up reflects naturally occurring practical scenarios for the algorithm where a large amount unlabelled data can be used to bootstrap a policy if the maximum likelihood and reinforcement learning objectives are at least partially aligned. We trained the RL algorithms for 200 epochs on the training set. At the end of each epoch, we generated outputs from the policy on test set of 500 inputs and scored them according to our criterion. We plot the test set error against the number of epochs for various methods in Fig. 2.

Fig. 2(a) compares 3 methods: BPG with and without maximum likelihood initialisation and a version of BPG which does not use importance sampling. Clearly, bootstrapping an RL algorithm with ML can be advantageous especially if data is abundantly available for ML training. Further, without importance sampling, the algorithm is not as competitive for reasons described in Section 3. In all 3 cases, we used a constant estimator for $\widehat{V}$ and $\lambda = 0.5$. The dashed line indicates the performance of ML training alone. BPG-NIS is similar to the algorithms of Ranzato et al. (2015); Williams & Zweig (2016) except that there, their methods implicitly use $\lambda = 1$.

Fig. 2(b) compares 4 methods: BPG and its on-line version OPG with constant (CONST) and GTD(λ) estimators for $\widehat{V}$. The on-line versions of the algorithms are a direct implementation of the method in Degris et al. (2012) which do not use the future rewards as we do. The first observation is that while GTD(λ) is slightly better in the early iterations, it performs roughly the same as using a constant estimator in the long run. Second, BPG performs significantly better than OPG. This should not be surprising – a close inspection of Degris et al. (2012) reveals that the algorithm sees the reward only at the end of the episode. This is typical of many RL algorithms where the reward



(a) (b) (c)

Figure 2: Results for synthetic experiments. (a): Comparison of BPG with and without maximum likelihood (ML) initialisation and BPG without importance sampling (BPG-NIS). The dotted line indicates performance of ML alone. (b): Comparison of BPG with its online counterparts OPG. We compare both methods using a constant estimator (CONST) for the value function and GTD(λ). (c): Comparison of BPG with different values of λ. All curves were averaged over 10 experiments where the training set was picked randomly from a pool. The test set was the same in all 10 experiments. The error bars indicate one standard error.

appears in the update only at the time step it is realised. For the reward to get factored into policy updates in earlier time steps, it has to percolate through updates for the estimate of the value function $\widehat{V}$. While this is reasonable in on-line settings where we can explore indefinitely to collect a large number of samples and estimate $\widehat{V}$, it is difficult when one only has a limited number of labelled samples. Finally, we compare BPG with different choices for λ in Fig. 2(c). As noted previously, $\lambda < 1$ is useful with stochastic rewards, but choosing too small a value is detrimental. The optimal λ value may depend on the problem.

4.2 Restaurant Recommendations

We use data from an on-line restaurant recommendation service. Customers log into the service and chat with a human agent asking recommendations for restaurants. The agents ask a series of questions such as food preferences, group size etc. before recommending a restaurant. The goal is to train a chat-bot (policy) which can replace or assist the agent. For reasons explained in Section 1, maximum likelihood training alone will not be adequate. By obtaining reward labels for responses produced by various other bots, we hope to improve on a bot initialised using maximum likelihood.

Data Collection: We collected data for RL as follows. We trained five different RNN chat-bots with different LSTM parameters via maximum likelihood on a dataset of 6000 conversations from this dataset. The bots were trained to reproduce what the human agent said (output y) given the past conversation history (input x). Then, we generated responses from these bots on 1216 separate conversations and had them scored by workers on Amazon Mechanical Turk (AMT). For each response by the bots in each conversation, the workers were shown the history before the particular response and asked to score (label) each response on a scale of $0 - 1 - 2$. We collected scores from three different workers for each response and used the mean as the reward in our RL framework.

Policies and RL Application: Next, we initialised 2 bots via maximum likelihood and then used BPG to improve them using the labels collected from AMT. For the 2 bots we used the following LSTM hidden state size H, word embedding size E and BPG parameters. These parameters were chosen arbitrarily and are different from those of the bots used in data collection described above.

- Bot-1: $H = 512$, $E = 256$. BPG: $\lambda = 0.5$, GTD(λ) estimator for $\widehat{V}$.
- Bot-2: $H = 400$, $E = 400$. BPG: $\lambda = 0.5$, constant estimator for $\widehat{V}$.

Testing: We used a separate test set of 500 conversations which had a total of more than 3500 input-output (conversation history - response) pairs. For each Bot-1 and Bot-2 we generated responses before and after applying BPG, totalling 4 responses per input. We then had them scored by workers on AMT using the same set up described above. The same worker labels the before-BPG and after-BPG responses from the same bot. This controls spurious noise effects and allows us to conduct a paired test. We collected about 7500 before and after label pairs each for Bot-1 and Bot-2 and compare them using a paired t-test and a Wilcoxon signed rank test.

	Mean (ML)	Mean (BPG+ML)	Paired t-test	Wilcoxon signed rank
Bot-1	0.8941 ± 0.0103	0.9066 ± 0.0104	0.15022	0.23463
Bot-2	0.7052 ± 0.0098	0.7216 ± 0.0099	0.08735	0.06286

Table 1: The results on the Mechanical Turk experiments using the restaurant dataset. The first two columns are the mean labels of all responses before and after applying BPG on the bots initialised via maximum likelihood. The last two columns are the p-values using a paired t-test and a paired Wilcoxon signed rank test. For both Bot-1 and Bot-2 the before and after responses were scored by the same worker. The results above are on 7439 such scores for each bot. Bot-2 is statistically significant at the 10% level on both tests.

Results: The results are shown in Table 1. The improvements on Bot-2 are statistically significant at the 10% level on both tests. While Bot-1 is not significant we can still see some improvement. The large p-values for Bot-1 are due to the noisy nature of AMT experiments and we believe that we can attain significance if we collect more labels which will reduce the standard error in both tests. In Appendix B.2 we present some examples of conversation histories and the responses generated by the bots before and after applying BPG. We qualititavely discuss specific kinds of issues that we were able to overcome via reinforcement learning.

5 Conclusion

We presented a policy gradient method for batch reinforcement learning to train chat-bots. The data to this algorithm are input-output sequences generated using other chat-bots/humans and stochastic rewards for each output in the dataset. This setting arises in many applications, such as customer service systems, where there is usually an abundance of unlabelled data, but labels (rewards) are expensive to obtain and can be noisy. Our algorithm is able to efficiently use minimal labelled data to improve chat-bots previously trained through maximum likelihood on unlabelled data. While our method draws its ideas from previous policy gradient work in the RL and NLP literature, there are some important distinctions that contribute to its success in the settings of interest for this work. Via importance sampling we ensure that the probability of an action is properly accounted for in off-policy updates. By explicitly working in the batch setting, we are able to use knowledge of future actions and rewards to converge faster to the optimum. Further, we use the unlabelled data to initialise our method and also learn a reasonable behaviour policy. Our method outperforms baselines on a series of synthetic and real experiments.

The ideas presented in this work extend beyond chat-bots. They can be used in applications such as question answering, generating image descriptions and machine translation where an output sentence generated by a policy is scored by a human labeller to provide a weak supervision signal.

Acknowledgements

We would like to thank Christoph Dann for the helpful conversations and Michael Armstrong for helping us with the Amazon Mechanical Turk experiments.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Vivek S Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5):291–294, 1997.

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.

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

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pp. 1693–1701, 2015.

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

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, pp. 413, 2013.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3128–3137, 2015.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pp. 79–86, 2005.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

Jiwei Li, Will Monroe, Alan Ritter, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Hamid Reza Maei. *Gradient temporal-difference learning algorithms*. University of Alberta, 2011.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pp. 3, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Mrinmaya Sachan, Avinava Dubey, and Eric P Xing. Science question answering using instructional materials. *arXiv preprint arXiv:1602.04375*, 2016.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pp. 1057–1063, 1999.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts, Amherst, 1984.

Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 814–823, 2015.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

Jason D Williams and Geoffrey Zweig. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

APPENDIX

A IMPLEMENTATION OF GTD(λ)

We present the details of the GTD(λ) algorithm (Maei, 2011) to estimate a value function in Algorithm 2. However, while Maei (2011) give an on-line version we present the batch version here where the future rewards of an episode are known. We use a parametrisation of the form $\widehat{V}(s) = \widehat{V}_\xi(s) = \sigma(\xi^\top \phi(s))$ where $\xi \in \mathbb{R}^d$ is the parameter to be estimated. $\sigma(z) = 1/(1+e^{-z})$ is the logistic function.

The algorithm requires two step sizes α', α'' below for the updates to ξ and the ancillary parameter w. Following the recommendations in Borkar (1997), we use $\alpha'' \ll \alpha$. In our implementations, we used $\alpha' = 10^{-5}$ and $\alpha'' = 10^{-6}$. When we run BPG, we perform steps (a)-(f) of Algorithm 2 in step (iii) of Algorithm 1 and the last two update steps of Algorithm 2 in the last update step of Algorithm 1.

The gradient and Hessian of $\widehat{V}_\xi$ have the following forms,

$$\nabla_\xi \widehat{V}_\xi(s) = \widehat{V}_\xi(s)(1-\widehat{V}_\xi(s))\phi(s), \qquad \nabla^2_\xi \widehat{V}_\xi(s) = \widehat{V}_\xi(s)(1-\widehat{V}_\xi(s))(1-2\widehat{V}_\xi(s))\phi(s)\phi(s)^\top.$$

The Hessian product in step (d) of Algorithm 2 can be computed in $O(d)$ time via,

$$\nabla^2_\xi \widehat{V}_\xi(s) \cdot w = \left[\widehat{V}_\xi(s)(1-\widehat{V}_\xi(s))(1-2\widehat{V}_\xi(s))(\phi(s)^\top w)\right]\phi(s).$$

Algorithm 2 GTD(λ)

Given: Data $\{(x_i, y_i, r_i)\}_{i=1}^n$, step sizes α', α'', return coefficient λ, initial ξ_0.

- Set $\xi \leftarrow \xi_0$, $w \leftarrow \mathbf{0}$.
- For each epoch $k = 1, 2, \ldots$
  - Set $\Delta\xi \leftarrow \mathbf{0}$, $\Delta w \leftarrow \mathbf{0}$.
  - For each episode $i = 1, \ldots, n$
    - Set $r^\lambda_{T+1} \leftarrow r_i$, $g^\lambda_{T+1} \leftarrow 0$, $q^\lambda_{T+1} \leftarrow \mathbf{0}$
    - $\rho_t \leftarrow \pi_\theta(a^{(i)}_t | s^{(i)}_t)/q(a^{(i)}_t | s^{(i)}_t)$ for $t = 1, \ldots, T^{(i)}$.
    - For each time step in reverse $t = T^{(i)}, \ldots, 1$:
      (a) $g^\lambda_t \leftarrow \rho_t\Big((1-\lambda)\widehat{V}_\xi(s^{(i)}_{t+1}) + \lambda\rho_t r^\lambda_{t+1}\Big)$
      (b) $q^\lambda_t \leftarrow \rho_t\Big((1-\lambda)\nabla_\xi\widehat{V}_\xi(s^{(i)}_{t+1}) + \lambda q^\lambda_{t+1}\Big)$
      (c) $\delta_t \leftarrow g^\lambda_t - \widehat{V}_\xi(s^{(i)}_t)$
      (d) $h_t \leftarrow \big(\delta_t - w^\top\nabla_\xi\widehat{V}_\xi(s^{(i)}_t)\big)\nabla^2_\xi\widehat{V}_\xi(s^{(i)}_t)\cdot w$
      (e) $\Delta w \leftarrow \Delta w + \frac{1}{T^{(i)}}\big(\delta_t - w^\top\nabla_\xi\widehat{V}_\xi(s^{(i)}_t)\big)\nabla_\xi\widehat{V}_\xi(s^{(i)}_t)$
      (f) $\Delta\xi \leftarrow \Delta\xi + \frac{1}{T^{(i)}}\Big(\delta_t\nabla_\xi\widehat{V}_\xi(s^{(i)}_t) - q^\lambda_t w^\top\nabla_\xi\widehat{V}_\xi(s^{(i)}_t) - h_t\Big)$
  - $w \leftarrow w + \alpha''\Delta w$.
  - $\xi \leftarrow \xi + \alpha'\Delta\xi$.

B ADDENDUM TO EXPERIMENTS

B.1 DETAILS OF THE SYNTHETIC EXPERIMENT SET UP

Given an input and output sequence, we used the average of five Bernoulli rewards Bern(r), where the parameter r was $r = 0.75 \times r_{\text{r}} + 0.25 \times r_{\text{f}}$. Here r_{r} was the fraction of common words in the input and output sequences while $r_{\text{f}} = 0.01^{p_{\text{f}}}$ where p_{f} is the fraction of forbidden words in the dataset. As the forbidden words, we used the 50 most common words in the dataset. So if an input

had 10 words of which 2 were forbidden, an output sequence repeating 7 of the allowed words and 1 forbidden word would receive an *expected* score of $0.75 \times (8/10) + 0.25 \times 0.01^{(1/8)} = 0.7406$.

The training and testing set for reinforcement learning were obtained as follows. We trained 4 bots using maximum likelihood on 6000 input output sequences as indicated in Section 4.1. The LSTM hidden state size H and word embedding size E for the 4 bots were, $(H, E) = (256, 128), (128, 64), (64, 32), (32, 16)$. The vocabulary size was $|\mathcal{A}| = 12000$. We used these bots to generate outputs for 500 different input sequences each. This collection of input and output pairs was scored stochastically as described above to produce a pool of 2000 input-output-score triplets. From this pool we use a fixed set of 500 triplets for testing across all our experiments. From the remaining 1500 data points, we randomly select 500 for training for each execution of an algorithm. For all RL algorithms, we used an LSTM with 16 layers and 16 dimensional word embeddings.

B.2 Addendum to the AMT Restaurant Recommendations Experiment

More Details on the Experimental Set up

We collected the initial batch of training data for RL as follows: We trained, via maximum likelihood on 6000 conversations, five RNN bots whose LSTM hidden size H and word embedding size E were $(H, E) = (512, 512), (256, 256), (128, 128), (512, 256), (256, 64)$. The inputs x were all words from the history of the conversation truncated at length 64, i.e. the most recent 64 words in the conversation history. The outputs were the actual responses of the agent which were truncated to length 64. As the vocabulary we use the $|\mathcal{A}| = 4000$ most commonly occurring words in the dataset and replace the rest with an `<UNK>` token.

Using the bots trained this way we generate responses on 1216 separate conversations. This data was sent to AMT workers who were asked to label the conversations on the following scale.

- **2**: The response is coherent and appropriate given the history and advances the conversation forward.
- **1**: The response has some minor flaws but is discernible and appropriate.
- **0**: The response is either completely incoherent or inappropriate and fails to advance the conversation forward.

Some Qualitative Results

In Tables 2 and 3 we have presented some examples. The text in black/grey shows the conversation history, the response in blue is by the bot trained via maximum likelihood (ML) alone and in red is the bot after improvement using our BPG reinforcement learning algorithm.

The first two examples of Table 2 present examples where the ML algorithm repeated generic questions (on budget, group size etc.) even though they had already been answered previously. After applying BPG, we are able to correct such issues, even though there are some grammatical errors. In the second, third and fourth example, we see that the ML+BPG bot is able to take context into consideration well when responding. For example, the customer asks for oriental/Mexican/Italian food. While the ML bot doesn't take this into consideration, the ML+BPG bot is able to provide relevant answers. However, in the third example, the name of the restaurant suggests that the food might be Indian and not Mexican. In the final example of Table 2 the customer asks a direct question about smoking. The ML bot provides an irrelevant answer where as the ML+BPG bot directly responds to the question.

In some examples, the ML bot had a tendency to produce sentences that were grammatically correct but nonsensical, sensible but grammatically incorrect, or just complete gibberish. The first three examples of Table 3 present such cases. Occasionally the opposite happened. The last example of Table 3 is one such instance.

Customer:	Hi there, I'm looking for a restaurant recommendation.	
Agent:	How many in your party?	
Customer:	5, including children.	
ML:	How many people will be joining you?	**[0]**
ML+BPG:	That is a fun size, and what is your budget joining you?	**[1.5]**

...		
Customer:	Hey... Sorry, it's the anniversary of my wife's death. I always do oriental tonight.	
Agent:	I'm sorry to hear that. I will do my best to find just what you need.	
Customer:	Thank you very much.	
Agent:	What kind of budget are you looking at?	
Customer:	Doesn't matter... $100 sounds like a good number.	
Agent:	Will it be just you, or are others going to join you?	
Customer:	It's just me tonight, but if you are off soon I wouldn't mind a friendly ear. No pressure.	
ML:	What type of budget are you looking to stay within?	**[0.33]**
ML+BPG:	How about: Café Hungry <UNK> ? They serve sea American to oriental on site such?	**[1.33]**

Customer:	I would love to find a great Mexican restaurant!	
Agent:	Great, and will anybody be joining you?	
Customer:	There will be a total of 3 of us.	
...		
Customer:	Could you find me a place with outdoor seating?	
Agent:	How about: The Painted Burrito. They serve Mexican food.	
Customer:	Eh, I've heard that one isn't great. Can you please suggest another?	
ML:	How about: The Hungry .w seafood restaurant? they serve sea and American food.	**[0.33]**
ML+BPG:	I am happy to, one moment. How about: Diva Indian Bistro Boston bar? They serve Mexican food.	**[1.67]**

Customer:	Hello, I'm looking for a restaurant.	
Agent:	I can help! Please tell me what kind of foods you most enjoy. I see you are in the Cambridge area.	
Customer:	I would like a restaurant that serves Italian food.	
Agent:	Are you looking for something a little less expensive or are you looking for a high end restaurant?	
Customer:	I am within a budget, 20 to 50 dollars.	
Agent:	I highly recommend The Redhouse Restaurant. It serves ...	
Customer:	Is there another restaurant? the restaurant would also need to host a party size of 5 people.	
ML:	How about: Amuse? They serve French and American food.	**[0]**
ML+BPG:	How about: Ocean prime - Boston Seaport? They serve sea, American and Italian food.	**[2]**

...		
Agent:	How about Cafe at Taj Boston?	
Customer:	Oh, I just ate there last night. Anything else?	
Agent:	Try Mela Restaurant at 578 Tremont St. Anything else I can help you with today?	
Customer:	That sounds nice. Do they allow smoking?	
ML:	This sure is within your budget for all.	**[0.5]**
ML+BPG:	No.	**[1.5]**

Table 2: Some qualitative results on the AMT experiment using restaurant data. In each cell, the text in black/grey show the history of a conversation between the customer and a human agent. The dots ... mean that the part of the conversation history is hidden. The blue response is by the bot trained using maximum likelihood (ML) alone. The red response is by the bot improved with BPG after ML. The numbers on the right of the blue/red responses are the average scores each response received from 2-3 AMT workers.

Agent:	How may I help you?	
Customer:	I'd like to find a French restaurant.	
Agent:	Do you have a budget in mind?	
Customer:	Between $50 and $75.	
Agent:	How many will be in your party?	
Customer:	Just me.	
ML:	And how will you be dining?	**[0]**
ML+BPG:	How will you be traveling to the restaurant?	**[2]**

Agent:	Hello, how may I help you?	
Customer:	We are kinda hungry, any suggestion in mind?	
Agent:	What type of cuisine do you like?	
Customer:	I want them Americano food.	
ML:	Very evening. Will you be dining alone?	**[1]**
ML+BPG:	Excellent choice. How many people will be in your party?	**[2]**

Customer:	I'd like to find a good local restaurant.	
Agent:	What type of cuisine would you like?	
Customer:	I have a hankering for Indian curry.	
...		
Agent:	Will you be needing parking?	
Customer:	We'll be driving altogether, so yes.	
ML:	Thank you for 1. How about: T. w your curry this place should be dining?	**[0]**
ML+BPG:	How about: The Maharaja? They serve Indian and American food.	**[2]**

Agent:	How can I help you today?	
Customer:	I would like a restaurant recommendation.	
Agent:	Okay what type of resturant?	
Customer:	Chinese would be great.	
Agent:	Who will be dining, how will you be travelling?	
Customer:	Just me and by bus.	
ML:	Okay, thanks. Do you have a budget in mind?	**[2]**
ML+BPG:	Do you want a wonderful place evening, or whats your party adults, and a place.	**[0]**

Table 3: Qualitative results on the AMT experiment using restaurant data, continued. See caption under Table 2 for more details.