Convergent Reinforcement Learning Control with Neural Networks and Continuous Action Search

Minwoo Lee[1] and Charles W. Anderson[2]

***Abstract*—We combine a convergent TD-learning method and direct continuous action search with neural networks for function approximation to obtain both stability and generalization over inexperienced state-action pairs. We extend linear Greedy-GQ to nonlinear neural networks for convergent learning. Direct continuous action search with back-propagation leads to efficient high-precision control. A high dimensional continuous state and action problem, octopus arm control, is examined to test the proposed algorithm. Comparing TD, linear Greedy-GQ, and nonlinear Greedy-GQ, we discuss how the correction term contributes to learning with nonlinear Greedy-GQ algorithm and how continuous action search contributes to learning speed and stability.**

I. Introduction

Temporal difference (TD) learning is a key method in reinforcement learning for solving Markov Decision Processes (MDPs). Without waiting for the final returns, TD methods learn from incomplete sequences of interactions with an environment. This makes TD applicable to various problems such as real-time adaptive control. A common problem in reinforcement learning on continuous actions and states is that there are an infinite number of state and action pairs; successful control often requires fine discretization, but this can result in the need for a prohibitive amount of experience [17].

To overcome the lack of experience in practical reinforcement learning tasks, an agent must be able to *generalize* based on limited experience. To do so, Q-values must be approximated using parameterized representation. Representing Q functions with function approximations, such as neural networks, has been successfully applied on various problems, especially in robotics such as navigation control [6], [10], robot walking [2], and robot soccer [18], [19]. The value function can be either linear or nonlinear, and it can be differentiable.

High-precision control with continuous actions that have the highest Q value can solve the real-world problems efficiently. Several studies have shown that continuous actions allow the solution of problems that are impossible to solve with coarse discretization of action space. To cover all important actions, expensive fine discretization is required. Without using fine discretization, previous studies proposed alternative ways. From a finite set of actions, some researchers obtain real-valued actions by interpolating discrete actions based on the value functions. Millan, et al., [15] sample real-valued actions from neighbors incrementally based on the approximated value function. Hasselt, et al., [24] select actions that have the highest Q value from the interpolator. Lazaric, et al., [3] use Sequential Monte Carlo (SMC) methods, which resample real continuous actions according to an importance sampling.

In addition to the use of real-valued actions, solutions to real-world problems also require stability in performance. Many popular TD algorithms, when combined with nontabular function approximation, diverge; Baird [1] and Tsitsiklis and Van Roy [23] show that learning parameters can go to infinity. Only linear function approximation with on-policy learning, for which samples are drawn from the current behavior policy, is known to be stable [20], [22], [23]. There is no guarantee of convergence when combined with nonlinear function approximation in on-policy learning.

Without sampling from the evaluated policy, in off-policy learning, an agent learns policies different from the one being executed. Off-policy learning can be applied on many applications since it can learn an optimal policy while executing an exploratory one, learn from demonstrations, and learn multiple tasks in parallel [5]. Instability of off-policy learning, however, has been a major issue. Sutton, et al., [21] first theoretically showed the convergence of off-policy temporal difference algorithm such as GTD, GTD2, and TDC, whose time and memory complexity are linear with the number of features in the function approximation. Maei, et al., [13] extend this approach to nonlinear function approximation and Maei, et al., [14] propose the first convergent TD learning algorithm for off-policy control with linear function approximation.

In this paper, we bring together key contributions to tackle the hard problem of learning to control a simulated octopus arm. In particular, we extend the theoretical work of linear Greedy-GQ to nonlinear neural networks and perform continuous action search by using Scaled Conjugate Gradient (SCG) [16].

This work is organized as follows. Section II presents a summary of MDPs, reinforcement learning, and recent work on convergent off-policy Q learning method, Greedy-GQ algorithm along with the notation. For clear notation, matrices are written in bold, and all the other variables are column vectors except scalars α, β, γ, δ, and ϵ. Section III derives the nonlinear Greedy-GQ algorithm for neural networks and derives the gradient update for continuous action search with back-propagation. In Section IV, after examining how the correction term in Greedy-GQ affects learning, we present the empirical results to demonstrate the stability of continuous nonlinear Greedy-GQ with an octopus arm experiment. Finally, in Section V, main conclusions about this research are presented, and some possible directions for future work are discussed.

[1]Minwoo Lee is a Ph.D student in the Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA `lemin@cs.colostateedu`

[2]Charles W. Anderson is a faculty of the Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA `anderson@cs.colostate.edu`

II. Temporal Difference Learning and Greedy-GQ

As developed in dynamic programming and Monte Carlo methods, Temporal difference (TD) learning uses bootstrapping to update the value estimates. A Markov decision process (MDP) is defined as a tuple $(S, A, P^a_{ss'}, R, \gamma)$, where for each time step $t = 0, 1, 2, \cdots$, with probability $P^a_{ss'}$, action $a_t \in A$ in state $s_t \in S$ transitions to state $s_{t+1} = s' \in S$, and the environment emits a reward $r_{t+1} \in R$. The value function V^π for a policy $\pi : S \rightarrow A$ is defined as:

$$V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_t = s, \pi],$$

where γ is a discounting factor. The value function satisfies the following Bellman equation:

$$V^\pi(s) = T^\pi V^\pi(s) = R^\pi(s) + \gamma P^\pi V^\pi(s),$$

where $R^\pi(s)$ is a reward function for the state s, P^π is the state transition probability under the policy π, and T^π is known as a Bellman operator. Similarly, for control problems, we can define the action value method, $Q^\pi(s, a)$:

$$Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_t = s, a_t = a, \pi].$$

When a value function approximation Q_θ is linear, it is defined as $Q_\theta(s, a) = \phi(s, a)^\top \theta$, where ϕ is a feature vector for state s and action a, and $\theta \in \mathbb{R}^d$ is a d dimensional weight vector. Action a can be selected either greedily or randomly. To balance the limited experience from greedy action selection and suboptimal random action selection, an action can be chosen greedily most of the time but randomly with small probability ϵ—this is called ϵ-greedy. Off-policy TD control known as Q-learning directly approximates $Q^* = \max_a Q(s, a)$, the optimal action-value function, which is independent of the policy being followed. Thus, the temporal difference error in Q-learning is defined as $\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$, and the algorithm has the update rule $\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi_t$, where α_t is a learning rate, and $\phi_t = \phi(s_t, a_t)$. Although TD is known to converge for on-policy control, there is no guarantee in off-policy problems [9].

Greedy-GQ [12], [14] is proved convergent in the mean square projected Bellman error (MSPBE) using stochastic gradient descent. To simplify the notation, removing π and omitting the arguments for Q, another objective function, MSPBE $J(\theta)$, can be written as:

$$\begin{aligned} \mathrm{J}(\theta) &= \|Q_\theta(s, a) - \Pi T Q_\theta(s, a)\|^2_\mu \\ &= \mathbb{E}[\delta(\theta) \nabla \mathrm{Q}^\top]^\top \mathbb{E}[\nabla \mathrm{Q} \nabla \mathrm{Q}^\top]^{-1} \mathbb{E}[\delta(\theta) \nabla \mathrm{Q}], \end{aligned}$$

where the projection operator $\Pi = \phi(\phi^\top D \phi)^{-1} \phi^\top D$, and D is a diagonal matrix whose diagonal values are the probability distribution $\mu(s, a)$ over the state-action pairs. The norm $\|Q\|^2_\mu = \int Q^2(s, a) \mu(ds, da)$. We denote the gradient of Q w.r.t. θ at s and a by ∇Q. Including the inverse matrix, g^* can be defined from $\mathrm{J}(\theta)$:

$$g^* = \mathbb{E}[\nabla \mathrm{Q} \nabla \mathrm{Q}^\top]^{-1} \mathbb{E}[\delta(\theta) \nabla \mathrm{Q}].$$

To avoid computation of the inverse matrix, Greedy-GQ estimates g^* by g_t along with the weight vector θ_t. Now, from MSPBE, the gradient descent learning Greedy-GQ updates weights as follows:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha_t \delta_t \phi_t - \alpha \gamma \phi'_t (\phi_t^\top g_t), \\ g_{t+1} &= g_t + \beta_t (\delta_t - \phi_t^\top g_t) \phi_t, \end{aligned}$$

where $\phi'_t = \phi(s', a')$ and $\beta_t > \alpha_t$. The term $-\alpha \gamma \phi'_t (\phi_t^\top g_t)$ is a correction term for the gradient descent direction.

III. Function Approximation with Neural Networks

To overcome the curse of dimensionality that arises with tabular function approximation, several approaches such as neural networks, radial basis functions, fuzzy sets, and support vector machines have been proposed. Neural networks is widely used as function approximators to represent nonlinear Q function. The following derivation leads to nonlinear Greedy-GQ algorithm with neural networks.

A. Nonlinear Greedy-GQ

Similar to the extension of linear TDC [21] to nonlinear TDC [13], we can extend linear Greedy-GQ [14] to nonlinear Greedy-GQ as follows:

$$\begin{aligned} \theta_{t+1} &= \Gamma\big(\theta_t + \alpha_t \delta_t \phi_t - \alpha \gamma \phi' (\phi_t^\top g_t) - \alpha_t h_t\big), \\ g_{t+1} &= g_t + \beta_t (\delta_t - \phi_t^\top g_t) \phi_t, \\ h_t &= (\delta_t - \phi_t^\top g_t) \nabla^2 Q_{\theta_t}(s_t) g_t. \end{aligned}$$

where $\phi_t = \nabla_\theta Q(s_t, a_t)$, and $\Gamma : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a mapping that projects the weights onto a smooth boundary parameter space. The projection mapping prevents the parameters from diverging in the early stage of the algorithm because of nonlinearities. A particular definition of Γ is used to prove convergence [12], but in practice, $\Gamma(x) = x$ is often used and is used here as well.

B. Greedy-GQ(0) with Neural Networks

Function approximation for $Q_{\mathbf{v},w}(s, a)$ is computed by two-layer neural networks where $\mathbf{v}$ is the weight matrix for the hidden layer and w is the weight vector for the output layer. The forward pass for Q can be defined as:

$$\begin{aligned} z &= h(x^\top \mathbf{v}) \\ z_m &= h(x^\top v^{(m)}) \\ y &= Q_{\mathbf{v},w}(s, a) = zw = h(x^\top \mathbf{v}) w \end{aligned}$$

When $h = \tanh$ is the activation function, $\nabla h = (1 - h^2)$, so we have the following gradient of Q w.r.t. the weight vector $v^{(m)}$ for hidden unit m:

$$\begin{aligned} \phi_{v^{(m)}} = \nabla_{v^{(m)}} Q(s, a) &= w_m \nabla_{v^{(m)}} (h(x^\top v^{(m)})) \\ &= w_m (1 - {z_m}^2) x^\top \end{aligned}$$

The $\phi_{\mathbf{v}}$ can be denoted as the following matrix, and from this, we can rewrite the vector $\phi_{v^{(m)}}$ as a m-th row of $\phi_{\mathbf{v}}$ matrix.

$$\begin{aligned} \phi_{\mathbf{v}} &= \nabla_v Q(s, a) = w \odot (1 - z^2)^\top x^\top \\ \phi_{v^{(m)}} &= [\phi_{\mathbf{v}}]_m, \end{aligned}$$

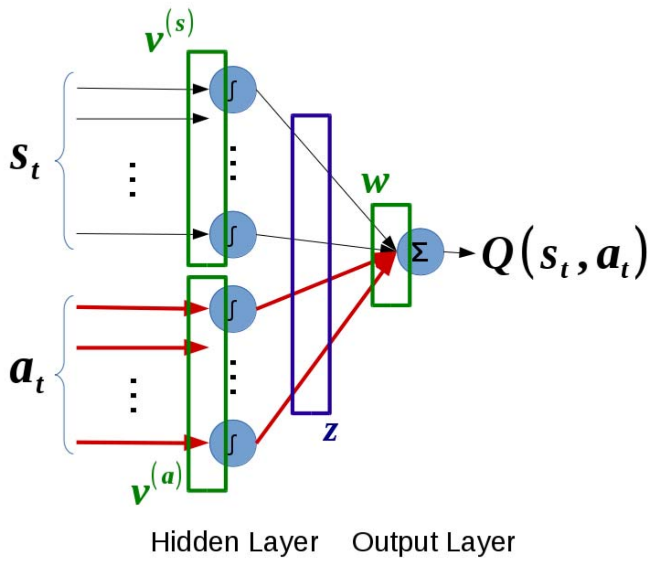


Fig. 1: The Two Layer Neural Networks for Q function approximation. $\mathbf{v} = [\mathbf{v^{(s)}}, \mathbf{v^{(a)}}]$ are the hidden unit weights, and w are the output unit weights for state and action input s_t and a_t at time t respectively. z is the output from the hidden layer.

where $\odot$ denotes component-wise multiplication.

$$\begin{aligned}
\nabla_{v^{(m)}} \phi_{v^{(m)}} = \nabla^2_{v^{(m)}} Q(s,a) &= \nabla_{v^{(m)}} (\nabla_{v^{(m)}} Q(s,a)^\top) \\
&= \nabla_{v^{(m)}} (w_m (1 - z_m^2) x^\top) \\
&= w_m (\nabla_{v^{(m)}} 1 - \nabla_{v^{(m)}} z_m^2) \odot x^\top \\
&= w_m (-2 z_m \nabla_{v^{(m)}} z_m)) \odot x^\top \\
&= w_m (-2 z_m ((1 - z_m^2) x^\top) \odot x^\top \\
&= -2 z_m (w_m (1 - z_m^2) x^\top) \odot x^\top \\
&= -2 z_m \phi_{v^{(m)}} \odot x^\top
\end{aligned}$$

For simplicity, we define $\overline{\phi_{v^{(m)}}}$ as:

$$\begin{aligned}
\overline{\phi_{v^{(m)}}} = (\nabla^2 Q(s,a)) g_{{v^{(m)}}_t} &= (-2 z_m \phi_{v^{(m)}} \odot x^\top) g_{{v^{(m)}}_t} \\
&= -2 z_m \phi_{v^{(m)}} (x^\top g_{{v^{(m)}}_t}) \\
&= -2 z_m (x^\top g_{{v^{(m)}}_t}) \phi_{v^{(m)}}
\end{aligned}$$

The update for the linear output layer follows the linear Greedy-GQ weight update since the second derivative is zero, which makes h_t zero. From the derivation above, the backward pass can be defined as:

$$\begin{aligned}
v_{t+1} &= \Gamma\Big(v_t + \alpha_v \Big[\delta_t \phi_{v_t} - \gamma \phi'_{v_t} (\phi_{v_t}^\top g_{v_t}) - h_t\Big]\Big) \\
g_{v_{t+1}^{(m)}} &= g_{v_t^{(m)}} + \beta_v (\delta_t - \phi_{v_t^{(m)}}^\top g_{v_t^{(m)}}) \phi_{v_t^{(m)}} \\
h_t^{(m)} &= (\delta_t - \phi_{v_t^{(m)}}^\top g_{v_t^{(m)}}) \overline{\phi_{v^{(m)}}}
\end{aligned}$$

$$\begin{aligned}
w_{t+1} &= w_t + \alpha_w \Big[\delta_t \phi_{w_t} - \gamma \phi'_{w_t} (\phi_{w_t}^\top g_{w_t})\Big] \\
g_{w_{t+1}} &= g_{w_t} + \beta_w (\delta_t - \phi_{w_t}^\top g_{w_t}) \phi_{w_t}
\end{aligned}$$

Summarizing the derivation, Algorithm 1 shows how to use nonlinear Greedy-GQ(0). ActionSelection() can be defined as any action selection scheme such as ϵ-greedy, softmax or noise-added greedy action. To learn from successive predictions, this model can be easily generalized with eligibility trace to Greedy-GQ(λ) as discussed in Maei, et al. [12].

Algorithm 1 Nonlinear Greedy-GQ(0) with 2-layer neural networks

```
**Initialization:** $g_0$ to 0, $\mathbf{v_0}$ and $w_0$ as random values.
**Choose** proper small positive learning rate $\alpha_\mathbf{v}$, $\alpha_w$, $\beta_\mathbf{v}$ and $\beta_w$, and set values for $\gamma \in (0, 1]$.
**Repeat** for each episode:
**Select** $a$ from $s$ by ActionSelection() to arrive at $s_{t+1}$.
**Observe** sample, $(s, a, r, s')$ at time step $t$, where $q'^* = Q_{\mathbf{v},w}(s', a'^*), a'^* = \text{argmax}_a Q_{\mathbf{v},w}(s', a)$.
**for** each observed sample and $Q_{\mathbf{v},w}(s,a) = w^\top z$ where $x = x(s,a)$, $x' = x(s',a')$, $z = \tanh(x^\top \mathbf{v})$, and $z' = \tanh(x'^\top \mathbf{v})$ **do**
  $q = Q_{\mathbf{v},w}(s,a)$
  $q' = Q_{\mathbf{v},w}(s',a')$
  $\phi_\mathbf{v_t} \leftarrow \nabla_v q = w^\top x(1 - z^2)$
  $\phi'_\mathbf{v_t} \leftarrow \nabla_v q' = w^\top x'(1 - z'^2)$
  $\phi_{w_t} \leftarrow \nabla_w q = z$
  $\phi'_{w_t} \leftarrow \nabla_w q = z'$
  $\delta_t \leftarrow r + \gamma q'^* - q$

  **for** each hidden unit $m$ **do**
    $\overline{\phi_{v_t^{(m)}}} \leftarrow (\nabla^2 q) g_{v_t} = -2z(x^\top g_{v_t}) \phi_\mathbf{v_t}$
    $h_t^{(m)} \leftarrow (\delta_t - \phi_{v_t^{(m)}}^\top g_{v_t^{(m)}}) \overline{\phi_{v_t^{(m)}}}$
  **end for**
  $\mathbf{v_{t+1}} \leftarrow \Gamma\Big(\mathbf{v_t} + \alpha_\mathbf{v} \big[\delta_t \phi_\mathbf{v_t} - \gamma \phi'_\mathbf{v_t} (\phi_\mathbf{v_t}{}^\top g_\mathbf{v_t}) - h_t\big]\Big)$
  $w_{t+1} \leftarrow w_t + \alpha_w \Big[\delta_t \phi_{w_t} - \gamma \phi'_{w_t} (\phi_{w_t}^\top g_{w_t})\Big]$
  **for** each hidden unit $m$ **do**
    $g_{v_{t+1}^{(m)}} \leftarrow g_{v_t^{(m)}} + \beta_\mathbf{v} (\delta_t - \phi_{v_t^{(m)}}^\top g_{v_t^{(m)}}) \phi_{v_t^{(m)}}$
  **end for**
  $g_{w_{t+1}} \leftarrow g_{w_t} + \beta_w (\delta_t - \phi_{w_t}^\top g_{w_t}) \phi_{w_t}$
**end for**
```

C. Continuous Action Search

The best action leads to the maximum estimated Q value on each step. At time t, with trained weights $\mathbf{v_t}$ and w_t, the estimated optimal action a_t^* is determined by this one step search [17]:

$$a_t^* = \underset{a}{\text{argmax}}\, Q_{\mathbf{v_t}, w_t}(s_t, a).$$

Here, we use back-propagation to calculate the derivative of Q with respect to the continuous action input. We can find the best action a^* that maximizes $Q(s_t, a)$ by using gradient ascent of $Q(s_t, a)$ with respect to a. Since the feed-forward output is Q, the gradient step is derived as below:

$$\frac{\partial Q(s_t, a)}{\partial a} = w_t^\top \odot (1 - z^2) \mathbf{v_t^{(a)}}^\top.$$

$\mathbf{v_t^{(a)}}$ are the weights applied to action input ($\mathbf{v} = [\mathbf{v^{(s)}}, \mathbf{v^{(a)}}]$). In the beginning, the network approximates the Q function poorly, and the found action is not likely to be the optimal. This will lead to further exploration in early stages. However, as the training goes on, or as ϵ decreases to exploit the learned policy,

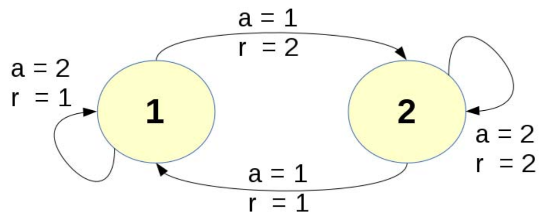


Fig. 2: 2-state model

the accuracy of approximation grows, and back-propagation search will be close to the optimal. For faster search, we use Moller's SCG [16] that uses gradients with approximate second order derivatives.

IV. Experiments

We examine the learning performance of nonlinear Greedy-GQ by comparing it with linear Greedy-GQ and nonlinear TD. Also, we demonstrate how the correction term affects the convergence or the performance of learning. For easier analysis on the effect of the correction term, first we test a simple toy problem, consisting of 2 discrete states and actions. Next, the combination of nonlinear Greedy-GQ and continuous action control is examined on an octopus arm problem. From our pilot tests, exponential decaying ϵ-greedy for exploration control outperformed softmax, greedy, or greedy with Gaussian noise, so the following experiments used ϵ-greedy with quickly decreasing ϵ as:

$$\epsilon_{t+1} = \epsilon_t \times e^{\frac{\log \epsilon_N}{N}},$$

where N is the number of episodes to train.

A. 2-state model

TD, linear Greedy-GQ(0) and nonlinear Greedy-GQ(0) are tested on the simple 2-state model shown in Fig. 2. The environment changes its state when an action is 1 and remains in the current state when the action is 2. The model can be summarized as follows:

$$s' = \begin{cases} s & \text{if } a = 2 \\ 1 & \text{if } s = 2 \text{ and } a = 1 \\ 2 & \text{if } s = 1 \text{ and } a = 1 \end{cases}$$

$$R(s,a) = \begin{cases} 2 & \text{if } s = a \\ 1 & \text{otherwise} \end{cases}$$

Although this 2-state model is simple, the shape of the optimal Q function is not linear. A linear function approximation cannot cover both the state-action $(1,1)$ and $(2,2)$, and it makes hard to reach an optimal policy.

Linear and nonlinear Greedy-GQ(0) were applied to the 2-state model for 1500 episodes (100 steps per each episode). This was repeated 100 times. Fig. 3 shows the mean of the 100 learning curves. From pilot tests, $\gamma = 0.9$, $\alpha = 0.0001$, and $\beta = 0.01$ are selected for linear Greedy-GQ(0), and $\gamma = 0.9$, $\alpha_v = 0.0001$, $\alpha_w = 0.0001$, $\beta_v = 0.01$, and $\beta_w = 0.001$ for nonlinear Greedy-GQ(0). Here, the subscripts v and w represent the parameters for hidden layer and output layer respectively. Neural networks in nonlinear Greedy-GQ(0) show the best performance with 20 hidden units. ϵ decays exponentially from 1 to 0.01. As shown in Fig. 3, nonlinear Greedy-GQ(0) solves what is optimized sum of rewards to the 2-state model problem, however, linear Greedy-GQ(0) is not able to learn. Linear function approximation cannot find good parameters for this 2-state problem.

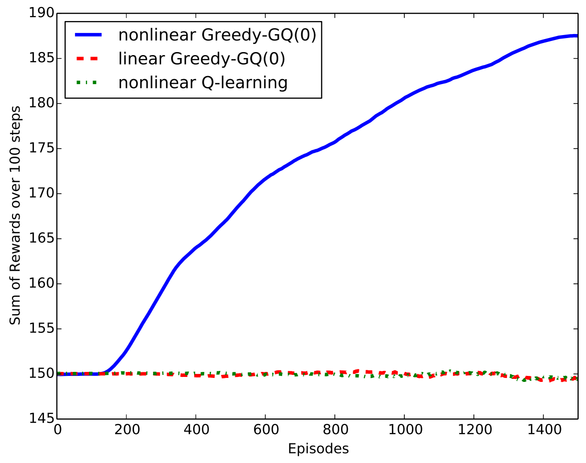


Fig. 3: Sum of rewards per episode on 2-state model. TD (with nonlinear neural networks function approximation) and linear Greedy-GQ(0) could not find an optimal policy. For smooth curve, moving average (over 100 episodes) is used.

In Fig. 3, we can also see that nonlinear Q-learning without Greedy-GQ correction term cannot solve the problem. This means there are some factors that disturb learning. To observe this, we examine the effects of the correction terms by changing β. All combinations of βs ($\beta_v, \beta_w \in \{0, 0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 0.99, 1.\}$) are tested and the cases with computation overflow are omitted. In Fig. 4, we observe that Q-learning without correction term ($\beta_v = \beta_w = 0$) cannot learn well. Interestingly, when there is small or no correction on the hidden layer ($\beta_v \leq 0.1$), it performs poorly. Correcting the gradient direction on the hidden layer is crucial for this 2-state model. In the two-layer neural networks, only the hidden layer contains a nonlinear operation, the activation function, and the output layer is linear sum of outputs from hidden units. When the target Q function is not linear, corrections in the nonlinear layer seem to be necessary for good performance.

Fig. 5a shows reward versus episodes for a single run with $\beta = [1, 0.01]$. There is a big jump in the curve around the 350th repetition. In Fig. 5b, we can see the large oscillation in TD error and its estimation, and then they drop around the 350th episode (35,000th step) as well (TD error is recorded for each state transition and 100 steps are recorded for each episode). We observe that the discrepancy from TD estimation grows in the beginning with random exploration, but as the exploration rate decreases, the TD estimation gets closer to the TD error. From this example run, Fig. 5 presents the TD gradient and Greedy-GQ(0) correction both in the hidden layer and the output layer. Unlike the output layer correction that has little effect on the gradient direction, the correction in

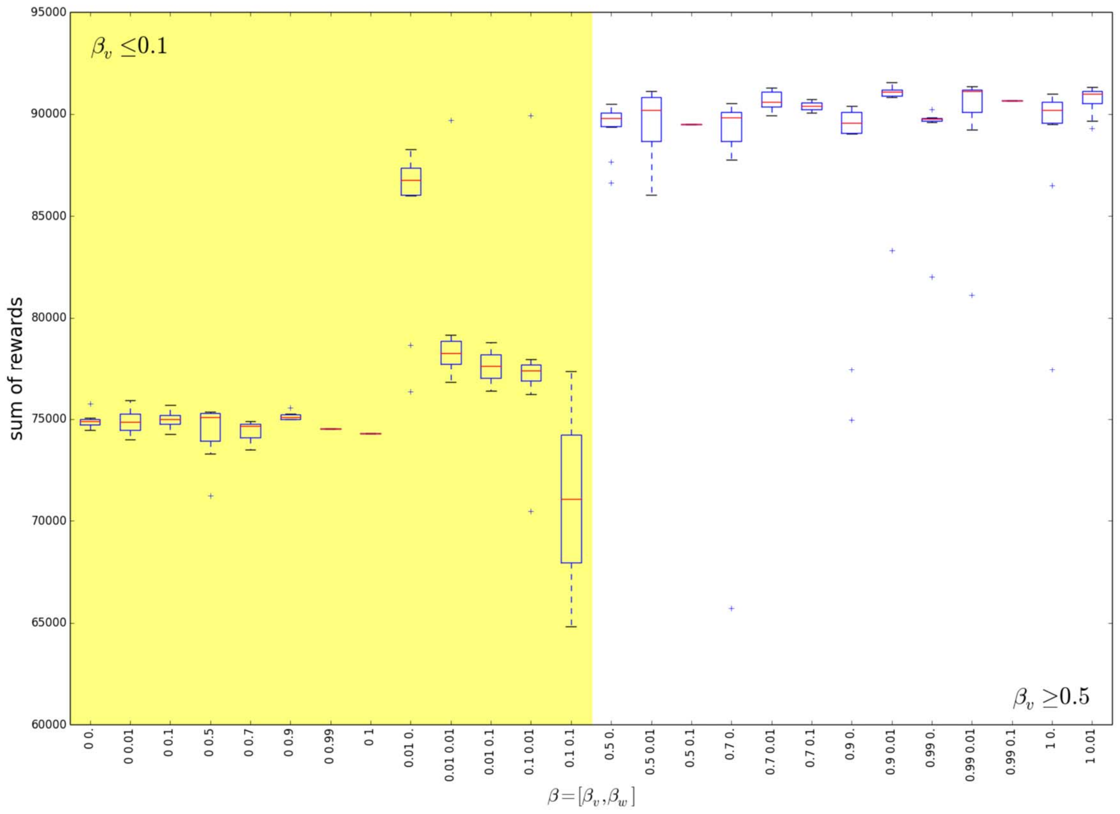


Fig. 4: The effects of the correction term in 2-state model. y-axis on the plot shows the sum of area under the reward curve. The x-axis is sorted by β_v and β_w. The yellow area groups the results with $\beta_v \leq 0.1$, and the rest shows the results with $\beta \geq 0.5$. For 10 runs, small learning rates β_v for the hidden layer result in lower performance.

the hidden layer has a large impact on the gradient. When the TD error hits the maximum, and at the same time when the learning curve jumps up, the correction term modifies the gradient direction to reduce the TD error and eventually leads to an optimal policy.

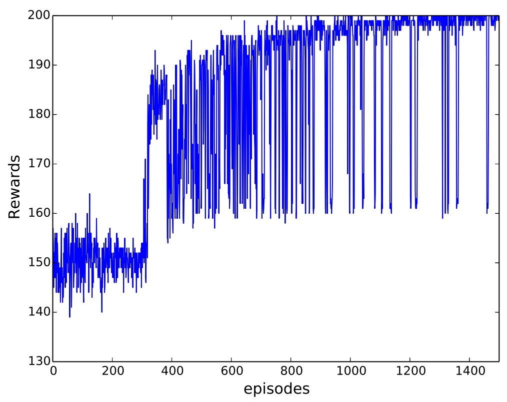
B. Octopus Arm

A real octopus arm is a complex organ with many degrees of freedom. Here, the Yekutieli's two-dimensional model [25] is used. The model is composed of spring muscles for each compartment. The basis of the model is that muscular hydrostats maintain a constant volume [8], so forces are transferred among the segments. Gravity, buoyancy, fluid friction, internal particle repulsion and pressure, and muscle contractions are computed each time step. For our experiments, we used the octopus arm model in the RL-competition [4] (Fig. 6). The problem contains 10 compartments, and the ventral, dorsal, and transverse muscles are controlled by independent activation variables. To make the problem simple, Engel, et al., [7] used 6 predefined discrete actions. In this paper, without reducing the action space, we train the arm to learn from the full action space defined by 33-dimensional real-valued actions. For the 10-compartment example, the state space is defined by 82 continuous values: base angle, angular velocity, x-y coordinate positions of each joints, and their velocities.

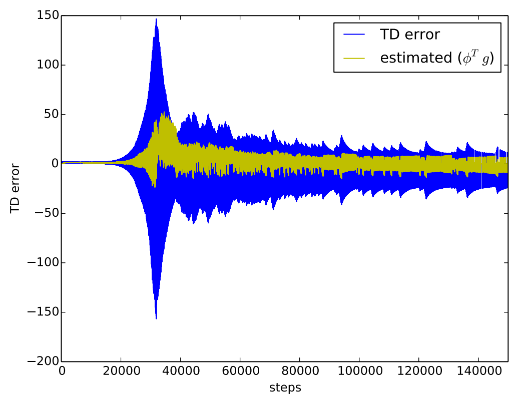
We place the goal at (4, 3) as in Fig. 6. Initially the base of the arm is placed close to (0, 0) and the arm is straight toward the right. The target task is touching the goal with any part of the arm. On each time step, the arm receives -0.01 as a penalty, and if it reaches the goal, it gets 10 as a reward. The maximum number of steps per each episode is limited to 1000 steps. Thus, if the arm touches the goal at the last moment, the total reward will be 0. If it fails to reach the goal, the total will be -10. Positive, larger rewards are obtained when the goal is reached sooner. The episodes are repeated 500 times with exponentially decreasing ϵ value from 1 to 0.01. From pilot tests, $\alpha = [0.0001, 0.0001]$ is selected. Each experiment was run 10 times.

Among all β combinations, $\beta = [0.1, 0.0001]$ shows the best performance and less variation than other β values. From this, as we examined in previous section, we can verify the stability contribution of the correction term empirically.

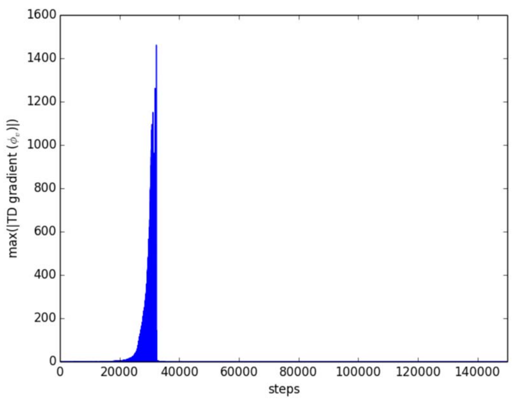
Fig. 7 compares the performance between continuous and discrete actions. After failing to touch the goal in the beginning, continuous control finds sequences of continuous actions that result in reaching the goal. With 6 discrete actions, it fails to reach the goal in 500 repetitions. It may need more



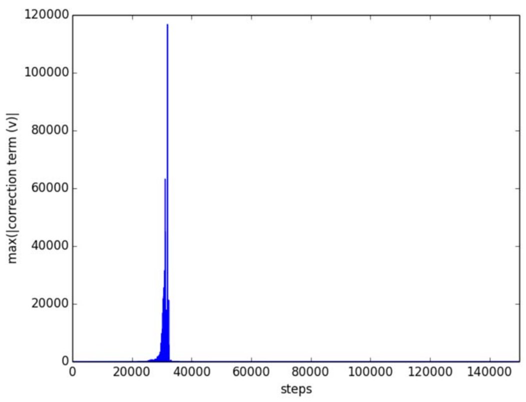
(a) Examplar learning curve



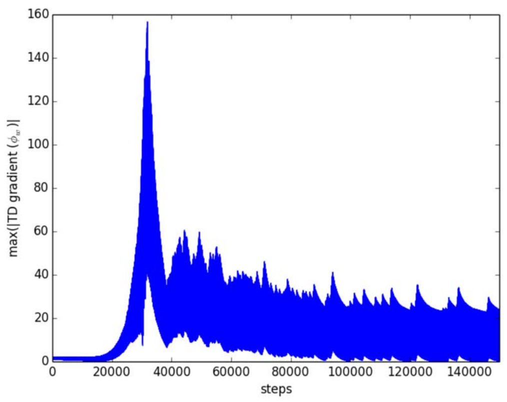
(b) TD error (δ) and approximation ($\delta_k g_{w_k}$) in Greedy-GQ(0)



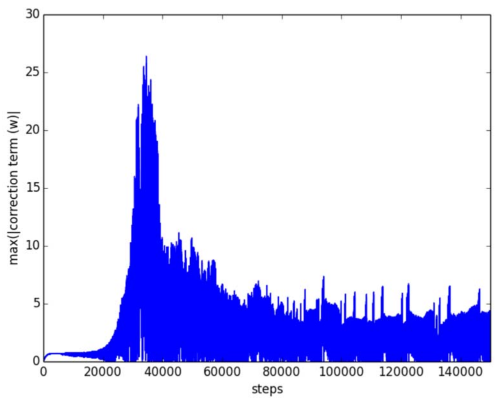
(c) TD gradient (hidden)



(d) Greedy-GQ(0) correction (hidden)

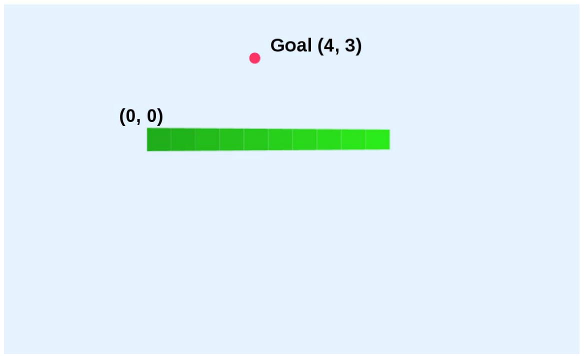


(e) TD gradient (output)

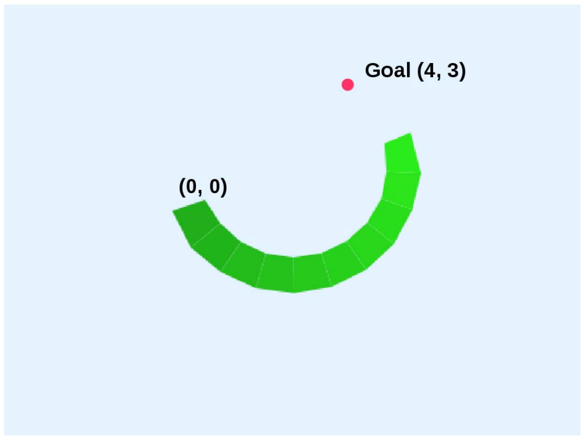


(f) Greedy-GQ(0) correction (output)

Fig. 5: TD gradient and Greedy-GQ(0) correction term in hidden and output layer

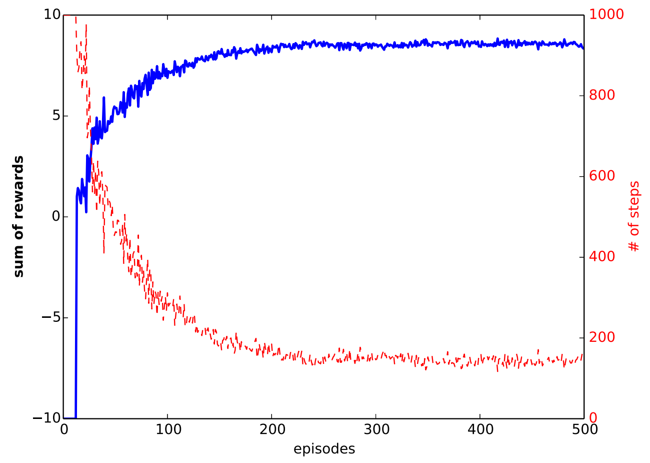


(a) Initial position

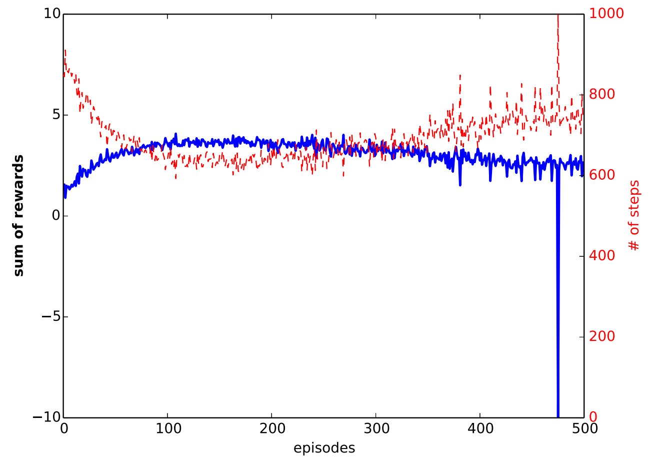


(b) Controlling the arm

Fig. 6: Benchmark octopus arm control problem



(a) Continuous actions



(b) 6 discrete actions

Fig. 7: Performance comparison between continuous action and discrete action control. Thick lines show the sum of rewards and dash lines show the number of steps per each episode. TD(0) fails to learn in 500 repetitions, so the learning curve is omitted.

experience to learn the policy to touch the goal. Although it is not presented, Q-learning(0) with or without continuous action search has been tested, but a positive learning curve was never observed, indicating further experience may be needed. Only nonlinear Greedy-GQ(0) with continuous actions was able to solve the task.

Reviewing the state trajectories, we also found that with discrete actions, the arm failed to find the short sequence of anti-clockwise movements towards to the goal. It only experienced clock-wise rotation to reach the goal, which costs more than 850 steps. Continuous action provides the additional flexibility during the early exploration stage to find the shorter paths of less than 200 steps. The number of steps in Fig. 7 shows this as well: continuous action (Fig. 7a) shows various experience that reaches to the goal in a wide range of steps, but discrete action (Fig. 7b) limits the experience to paths of more than 600 steps.

V. Conclusion

This paper extends linear Greedy-GQ to a nonlinear form with feed-forward neural networks for stable learning performance and then combines it with continuous action search. An analysis reveals that the nonlinear Greedy-GQ correction term is crucial for training the hidden layer when the target Q function is nonlinear. Testing on a complex octopus problem with a high dimensional continuous domain, we observe that the continuous action search improves the learning performance (both in speed and asymptotic level). Future research activity will follow in three main directions: examining more real-world problems, applying Regularized Off-Policy TD learning (RO-TD) [11] to be robust to noisy inputs, and development of batch-oriented version of nonlinear Greedy-GQ and comparative research against other state-of-the-art algorithms such as CACLA [24] and SMC-learning [3].

References

[1] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of 12th International Conference on*

Machine Learning, pages 30–37, 1995.

[2] H. Benbrahim and J. A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3):283–302, 1997.

[3] A. L. M. R. A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Proc. Adv. Neural Inf. Process. Syst*, 20:833–840, 2008.

[4] R. Community. Rl-competition. http://www.rl-competition.org/, 2009. [Online; accessed 03-July-2014].

[5] T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[6] M. Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994.

[7] Y. Engel, P. Szabo, and D. Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. *Advances in Neural Information Processing Systems*, 18:347, 2006.

[8] W. M. Kier and K. K. Smith. Tongues, tentacles and trunks: the biomechanics of movement in muscular-hydrostats. *Zoological Journal of the Linnean Society*, 83(4):307–324, 1985.

[9] J. Z. Kolter. The fixed points of off-policy td. pages 2169–2177, 2011.

[10] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[11] B. Liu, S. Mahadevan, and J. Liu. Regularized off-policy td-learning. pages 845–853, 2012.

[12] H. R. Maei. *Gradient temporal-difference learning algorithms*. PhD thesis, University of Alberta, 2011.

[13] H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. pages 1204–1212, 2009.

[14] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton. Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning*, pages 719–726, 2010.

[15] J. D. R. Millán, D. Posenato, and E. Dedieu. Continuous-action q-learning. *Machine Learning*, 49(2-3):247–265, 2002.

[16] M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[17] J. Santamaria, R. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163, 1997.

[18] P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *In International Conference on Machine Learning*, volume 1, pages 537–544, 2001.

[19] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[20] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT press, 1998.

[21] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, 2009.

[22] V. Tadić. On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, 42(3):241–267, 2001.

[23] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

[24] H. Van Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. IEEE International Symposium on*, pages 272–279, 2007.

[25] Y. Yekutieli, R. Sagiv-Zohar, R. Aharonov, Y. Engel, B. Hochner, and T. Flash. Dynamic model of the octopus arm. i. biomechanics of the octopus reaching movement. *Journal of neurophysiology*, 94(2):1443–1458, 2005.