Recurrent Network-based Deterministic Policy Gradient for Solving Bipedal Walking Challenge on Rugged Terrains

Doo Re Song
J.P. Morgan & Co
3 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Chuanyu Yang
The University of Edinburgh
15 PUBLICATIONS **37** CITATIONS
SEE PROFILE

Christopher Mcgreavy
The University of Edinburgh
6 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Zhibin Li
The University of Edinburgh
85 PUBLICATIONS **1,032** CITATIONS
SEE PROFILE

Recurrent Deterministic Policy Gradient Method for Bipedal Locomotion on Rough Terrain Challenge

Doo Re Song
University of Edinburgh, UK
sdr2002@gmail.com

Chuanyu Yang, Christopher McGreavy, Zhibin Li
School of Informatics, University of Edinburgh, UK
zhibin.li@inf.ed.ac.uk

***Abstract*— This paper presents a deep learning framework that is capable of solving partially observable locomotion tasks based on our novel interpretation of Recurrent Deterministic Policy Gradient (RDPG). We study on bias of sampled error measure and its variance induced by the partial observability of environment and subtrajectory sampling, respectively. Three major improvements are introduced in our RDPG based learning framework: tail-step bootstrap of interpolated temporal difference, initialisation of hidden state using past trajectory scanning, and injection of external experiences learned by other agents. The proposed learning framework was implemented to solve the Bipedal-Walker challenge in OpenAI's gym simulation environment where only partial state information is available. Our simulation study shows that the autonomous behaviors generated by the RDPG agent are highly adaptive to a variety of obstacles and enables the agent to effectively traverse rugged terrains for long distance with higher success rate than leading contenders.**

I. Introduction

The morphology of humanoid robots is similar to that of humans which provides the ability to traverse complex and dynamic terrains that are easily accessible to humans. Humanoid robots generally exhibit high manoeuvrability and flexibility, thus are capable of achieving locomotion while navigating through uneven terrain and stepping over obstacles. Considering the physical limitations of wheeled robots, there are many advantages to choosing bipedal locomotion over wheeled locomotion. Moreover, knowledge of bipedal locomotion can also help us design better exoskeletons that can benefit the lives of people with gait abnormalities. As a result, bipedal locomotion has attracted increasing attention in recent years.

Most bipedal walking controls are achieved using deterministic and analytic engineering approaches. Yet, there are also a large amount of works that attempt to use machine learning approaches, such as reinforcement learning (RL), to achieve bipedal walking. RL can be applied to model-free learning for bipedal walking. This has been attempted in various action policy learning algorithms based on Markov Decision Process (MDP) [1], [2].

Increasing amounts of researchers are interested in investigating the potential of implementing Deep Reinforcement Learning (DRL) methods to allow agents to perform dynamic motor tasks in complex environments. DRL is a RL framework that combines reinforcement learning with deep neural networks. Within DRL, deep neural networks are utilized to approximate value function $V(s)$ or $Q(s,a)$, action policy $\pi(a|s)$, and state transition and reward model [3]. DRL research has an growing interest in recent years due to its capability as a non-linear state abstraction tool which deals with action decision as well as evaluation. A great breakthrough in the development of DRL is its first human-level performance on simple Atari games by Mnih et al. [4]. As a proceeding, our motivation is at investigating the potential of implementing DRL methods to allow agents to perform dynamic motor tasks in complex environments, such as robot locomotion.

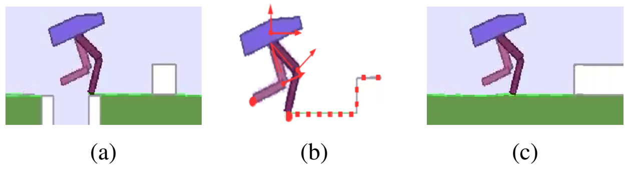


Fig. 1: Partially observable scenario: (a) Real world environment; (b) Incomplete observation of environment; (c) Incorrect perception built on the incomplete observation;.

A variety of work by computer science and robotics researchers has used DRL to solve bipedal locomotion. Peng et al. has successfully trained a bipedal humanoid character to traverse rugged terrain in a 2D simulation environment using Continuous Actor Critic Learning Automation (CACLA) [5]. They later extended their work to include a hierarchical DRL framework and trained a bipedal humanoid character to perform complex locomotion tasks in a 3D simulation environment [6] [7].

However, the MDP framework of RL lacks some necessary components for walking. During sensing, a walking agent is unable to fully observe its environment (Fig.1b), meaning inferences must be made about the environment using previous observations and actions made by the agent to fill in the gaps in observability; the estimated state is then used to infer actions (Fig.1a), otherwise the agent can only act the same as it does in less risky circumstance (Fig.1c). However, the MDP process has no functionality to distinguish them. State of the art RL algorithms with MDP is able to succeed in fully observable virtual environments [4] but not in the partially observable environment. In reality however, robots never gather sufficient information to generate optimal actions, and generally the obtained information is usually noisy as well. Therefore, partial observability is a critical issue in robot locomotion when RL is applied to real world robotics. Advancements in the improved decision framework of agents allow reasonable estimation of the true state of the environment when full observability is not possible.

There are related works that introduced Partially Observable Markov Decision Process (POMDP) [8] to RL to address the issue of incomplete observability in the environment. Wierstra et al. suggested an early form of Recurrent Neural Network (RNN) based RL, where the benefits of the POMDP framework provided by the standard RNN were reported in the partially observable environment though tasks were simple [9]. Mnih et al. reported that the RNN-based RL algorithms with a sensible observation-action evaluator can improve the learning performance of RL agents beyond human expert level if the observation is well provided [10]. There has been works that combined RNN with DRL to solve bipedal locomotion. Hess et al. proposed a RNN-based DRL called Recurrent Deterministic Policy Gradient (RDPG) [11] that is a combination of Long Short Term Memory network (LSTM) [12] and Deterministic Policy Gradient (DPG) [13], and the emergence of locomotion behaviour was obtained on flat terrain environment.

Finally, recent works applied RNN-based RL to locomotion learning tasks and observed emergence of walking behaviour from scratch by Heess et al. [14] and Duan et al. [15]. Nonetheless, the above works on RNN-based RL are hardly utilisable to solve locomotion POMDP in reality. This is because they do not offer the way to reuse the data collected from the far past or other agents, for training the current agent. In addition, constraints and issues on training the RNNs with small subtrajectories are barely discussed.

In this work, we examine the above two issues in a simple but reflective bipedal locomotion POMDP task on rugged terrains. Our methodology upon those challenges can successfully improve the learning performance of existing algorithms (see Table I in IV). Our contributions are below three aspects of training the RNN-based policy and value functions:

- Tail-step bootstrap for interpolated Temporal Difference on Value estimator;
- Configuration of initial hidden state via scanning past trajectory;
- Transfer of behavior via injecting external experience.

This paper is organized as follows. The principles of the RDPG algorithm is presented in section II. Our proposed improvement for training RDPG is elaborated in section III. The simulation setup and simulation results are presented in section IV, followed by final conclusion in section V.

II. Preliminaries

A. *Partially observable markov decision process and long short term memory*

A POMDP is an action decision process that uses knowledge of observations and actions from previous time-steps to augment current observations, since the true state information is not available. To use DRL in walking control, a method is clearly needed for representing the previous observations; these will be useful in inferring the true current state.

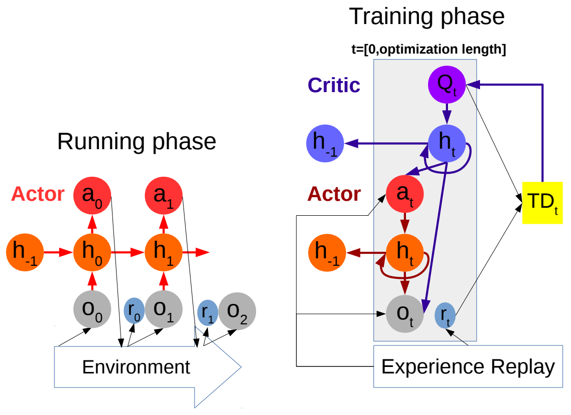


Fig. 2: Recurrent deterministic policy gradient (RDPG).

Since the true state can only be estimated via complete belief propagation from the terminal state, relaxed form of policy improvement methods via value approximation have been used, notably point-based value iteration methods [16]. Emergence of deep neural network models with temporal memory offered a new venue on the POMDP problem by point estimating the value that also references to the past observations. The Long Short Term Memory (LSTM) is variant of RNNs that allows the access to information further back in the history of the network via gate mechanism [12]. Therefore, the gated-RNNs are the widely used in DRL agents for POMDP tasks [10], [11].

B. *Recurrent deterministic policy gradient algorithm*

The RDPG is an application of Deterministic Policy Gradient algorithm [13] where the policy is a point-estimator built by RNNs [11]:

$$\begin{aligned}\pi(a_t|(o_i,a_i)_{i=:t-1}) &= p(a_t|b_t) \approx p(a_t|o_t,h_{t-1}) \\ &\equiv \delta\left(a_t - f^{Actor}(o_{0:t},h_{-1})\right)\end{aligned}$$

The policy gradient on the POMDP task is approximated as follows:

$$\begin{aligned}\nabla_\omega J(\pi^\omega) &= \int ds\rho(s)[\nabla_a Q^\theta(s,a)|_{a\sim\pi^\omega(s)} \cdot \nabla_\omega \pi^\omega(a|s)] \\ &\approx \mathbb{E}_{s\sim\rho^{\pi^\omega}}[\nabla_a Q^\theta(o,a|h)|_{a\sim\pi^\omega(o,h)} \cdot \nabla_\omega \pi^\omega(a|o,h)]. \quad (1)\end{aligned}$$

As a member of Actor-Critic algorithm [3] [17], the Critic network approximates the state-action value and send the score message to the Actor at policy improvement phase in the form of action gradient on the Critic's geometry ($\nabla_a Q^{Critic}(o,a)$).

In the value expectation step, the square of Temporal Difference (*TD*) is minimized. *TD* is an error measure between the current and target Q-values: $TD_t := Q_t^{target} - Q^{behavioral}(o_t,a_t|h_{t-1})$, where the behavioral Critic function Q^{beh} estimates the Q-value. The target Critic Q^{tar} is expanded to certain time-steps and then estimated by the target Critic function by Bellman equation. The original DDPG [18] and RDPG [11] uses target Q networks to provide the Q-values instead of using the learned Q^{beh} networks. The weights of the target Q network is updated by slowly tracking the learned networks via "soft updates": $\theta^{target} = \tau\theta + (1 -$

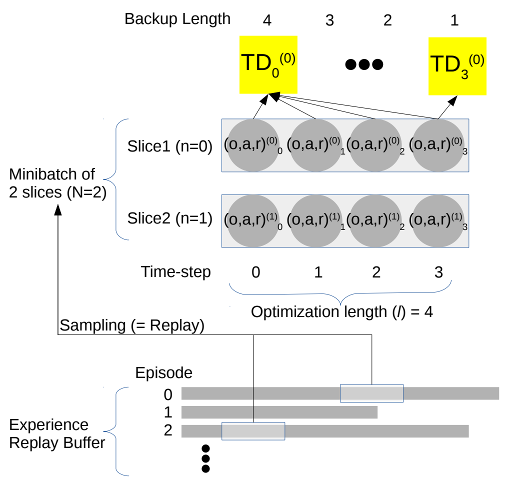

Backup Length
4
3
2
1
$TD_0^{(0)}$
$TD_3^{(0)}$
Slice1 (n=0)
Slice2 (n=1)
Minibatch of 2 slices (N=2)
$(o,a,r)^{(0)}_0$ $(o,a,r)^{(0)}_1$ $(o,a,r)^{(0)}_2$ $(o,a,r)^{(0)}_3$
$(o,a,r)^{(1)}_0$ $(o,a,r)^{(1)}_1$ $(o,a,r)^{(1)}_2$ $(o,a,r)^{(1)}_3$
Time-step
0
1
2
3
Optimization length (l) = 4
Sampling (= Replay)
Episode
0
1
2
Experience Replay Buffer

Fig. 3: Episodic *Experience replay* buffer for training.

$\tau)\theta^{target}$. The slow change of target Q networks provides a more consistent Q-value, therefore improving the stability of the value expectation step.

The graph representation of RDPG on the POMDP framework is shown in Fig.2. The agent's POMDP decides the action based on its *belief* (b_t) of the true state estimated by the *history* (h_{t-1}) representation [11] of the past observation-action trajectory and the current *observation* of the state (o_t).

In order to facilitate exploration of new behavioursr, DPG algorithm noises its action via first order stochastic process such as Ornstein-Uhlenbeck process. In addition, we peroidically noise to the parameter space of the Actor [19].

C. Minibatch sampling from experience replay buffer

All the pairs of state transition, action, and reward experience are saved in an *Experience replay* buffer [20] for the minibatch gradient update during the running phase. The *Experience replay* buffer also provides training data in the form of minibatch. Since this technique decorrelates the training data from the data collected on the current episode, it has been preferred over on-line [10] or episodic-wise learning methods [21] by recent works on DRL [4] [11] [18].

In our model, the buffer stores the experience pair at each time-step $(o,o',a,r)_t$ and forms a bundle of episodic trajectories (Fig.3) so that the RDPG model will not be trained with slices having terminal state in the middle. The episodic buffer provides the minibatch of experience for training by following steps. First, the agent picks episodes to sample. Second, the agent samples one slice of certain length l each from the chosen episode, and stacks the slices as a minibatch. Lastly, the RNN-based Actor and Critic networks read the entire slices, measure their objectives, and generate the update gradient of parameters for the Policy Iteration.

A major challenge in applying the minibatch-based training method to the RNN-based RL agents is that the training data is fed with only a fraction of each episode while the RNN memorizes each of the entire episode during the run.

III. Learning methods

Following the discussion in Sec. II-B and II-C, our work focus on issues of Policy Iteration process of RDPG while the

Algorithm 1: Recurrent Deterministic Policy Gradient

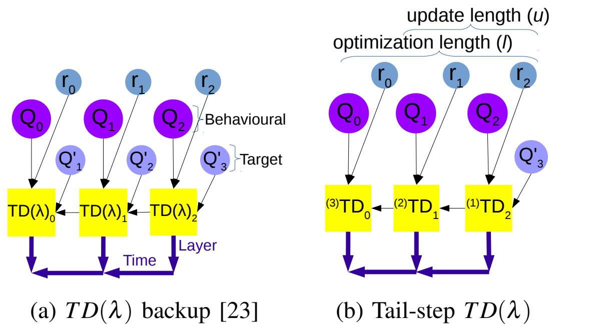
```
def RDPG:
Initialise parameters of Actor and Critic $\omega, \theta$
Initialise Experience replay buffer $R$
Set optimization/update/scan length as $l/u/s$   # $l \geq u$
for episode=1 to $M$ do
    Initialise OU-noise $\mathcal{N}_{epi}$ and Noise() parameters
    Initialise observation $o_0$, and hidden states $h_0$
    while not terminated do
        Generate action: $a_t = \pi^{\omega}(o_t, h^{Actor}_{t-1}) + \mathcal{N}_{epi,t}$
        Receive $o_{t+1}, r_t$ from the environment$(o_t, a_t)$
        Store transition$(o_t, a_t, r_t, o_{t+1})$ to $R$
    end
    Denoise() parameters and then Update()
end

def Update:
Sample $N$ sliced trajectories of length $s+l$ from $R$
Get $h_{-1}$ from $h_{init}$ by scanning $s$ time-steps (Eqn.4)
Get TD gradient of the minibatch (Eqn.3):
  $\bigtriangledown_{\theta_{old}} L = -\frac{1}{N}\sum_{n=1}^{N}\Big((\lambda\frac{1-\lambda^u}{1-\lambda})^{-1}\sum_{i=0}^{u-1}\lambda^{i}\cdot{}^{(l-i)}TD_i^{(n)}\cdot$
  $\bigtriangledown_{\theta_{old}} Q^{\theta}(o_i^{(n)}, a_i^{(n)}|o_{0:i}^{(n)}, h_{-1}^{(n),Critic})\Big)$
Update Critic by minimizing the TD loss with ADAM
  optimizer [22]: $\theta_{new} \leftarrow ADAM(\theta_{old}, \bigtriangledown_{\theta_{old}} L)$
Get Actor gradient of the mini-batch (Eqn.1): $\bigtriangledown_{\omega_{old}} J =$
  $\frac{1}{N}\sum_{n}^{N}\Big(l^{-1}\sum_{i=0}^{l-1}\bigtriangledown_a Q^{\theta}(o_i^{(n)}, a|h_{-1+i}^{(n),Critic})|_{a\sim\pi^{\omega}(o_i^{(n)}, h_{-1+i}^{(n),Actor})}\cdot$
  $\bigtriangledown_{\omega_{old}}\pi^{\omega}(a|o_i^{(n)}, h_{-1+i}^{(n),Actor})\Big)$
Update Actor by maximizing the Q-value with ADAM
  optimizer: $\omega_{new} \leftarrow ADAM(\omega_{old}, \bigtriangledown_{\omega_{old}} J)$
```

update gradients are calculated from sampled minibatches. A pseudocode of our model RDPG algorithm is detailed in Algorithm 1.

A. Modifications on the optimization of RDPG

1) Tail-step bootstrap of interpolated TD: The original RDPG algorithm proposed by Heess et al. [11] measured one-step *TD* for every experience pair within the minibatch. The problem with the one-step TD is that the obtained reward can only be used to directly update the value of the specific state action pair since the values of other state action pairs can only be updated indirectly through the Q-value function Q(s, a). If the observation of environment (o) does not truly represent the true state (s) such as POMDP tasks, the short backup length induces the bias of the estimated Q-value. This justifies the necessity of the Critic referring to farther time-steps.

Giving the longest but equal backup length for every *TD* measures of sampled experience pairs is crucial for theoretically justifying the Q-value's convergence bound. However, minibatch sampling shortens down the backup length for later experience pairs within the sampled subtrajectories (See Fig.3). Unless the full Monte-Carlo sums of future rewards are utilised to update Critic, which is not the case for the



(a) $TD(\lambda)$ backup [23] (b) Tail-step $TD(\lambda)$

Fig. 4: TD measurement in a subtrajectory. Thick purple line: backpropagation of Critic update gradient.

DPG algorithm, the problem sustains. A common practice is to ignore the equality on length but to backup with truncated multi-step TD's:

$$^{(i)}TD_t = \sum_{t'=t}^{t+i-1} \gamma^{t'-t} r_{t'} + \gamma^i Q^{tar}_{t+i} - Q^{beh}(o_t, a_t | h_{t-1})$$

is the multi-step *TD* [24] with backup length i. The length i varies by the position of experience pairs within the sampled subtrajectories. This raises another issue on the variance control of sampled rewards sum for each experience pair due to uneven backup length, resulting in lack of principled appreciation on recency of TD's [25].

In terms of bias control, recent advances either refer to the whole trajectories [25] or boostrap impractically long temporal horizon [26], or train the agent episodically online from backward direction [10]. They are not suitable for subtrajectory sampled training. In addition, the control on variance was barely discussed.

Our method is motivated from the work of Harb et al. for use of $TD(\lambda)$ in the context of RNN-based Q-learning [23] (Fig.4a). Although the $TD(\lambda)$ measurement improved the learning performance, their work 1) did not resolve the problem of shortening backup length of the TD's in the later time-steps of subtrajectories, and 2) discrete action control tasks can explore much diverse action values for similar states since action representation is categorical.

In order to ensure the maximum backup length for every experience pair in the minibatch, we propose a new method of updating the Critic. First, we generalize the *TD* measurement as an interpolation of multi-step *TD*'s:

$$Z^{-1}\sum_{i=1}^{l} w(i)\cdot {}^{(i)}TD_t, \text{ where } Z^{-1}\sum_{i=1}^{l} w(i) = 1 \tag{2}$$

On the other hand, our method enables a principled backup scheme while still measuring only one *TD* with the longest backup length each experience pair (Fig.4b).

In our method, the target Q-value is only being calculated at the end of each slice of sampled trajectories, which in turn measures $TD(\lambda)_{\lambda=1}$ for every point. Hence each slice with length l measures $\{^{(l-x)}TD_{t+x}\}_{x=0\cdots l-1}$ as that of $TD(\lambda)_{\lambda=1}$. However, an interpolated-TD_t with backup length l requires $\{^{(i)}TD_t\}_{i=1\cdots l}$. They correspond to each other as follows:

$$\{^{(l-x)}TD_{t+x} \text{ is measured for } interpolated\text{-}TD_{t+x}\}_{x\in[0,l-1]}$$

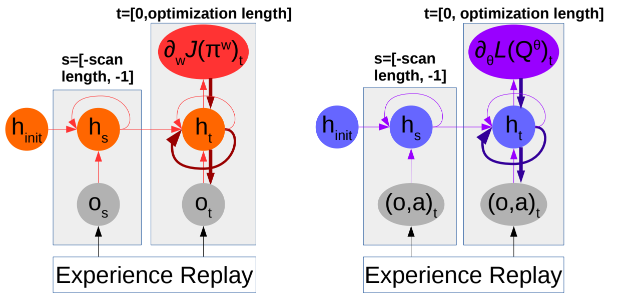


Fig. 5: Scanning strategy that looks back from constant h_{init} to h_{-1} on Actor (left) and Critic (right) during training. Without the scanning interval, h_{-1} is equal to h_{init}.

Without waiting for the other elements of the *interpolated-TD*'s [1], the agent updates the Critic with the part of them we measured on the minibatch, which are $\{^{(l-x)}TD_{t+x}\}_{x=0\cdots l-1}$. However, all the other constituents will eventually be visited by sufficient rounds of replay with slices of experience from different location[2]. Therefore, every experience points gather all the multi-step *TD*'s as constituents for the l-length interpolated-TD in the end.

To minimize the bias of our interpolation in fixed l, we apply decaying weight factor $w(i) = (\lambda\frac{1-\lambda^l}{1-\lambda})^{-1}\lambda^i$ (Eqn.2) to $^{(i)}TD$'s , resulting in the interpolated-TD_t as $(\lambda\frac{1-\lambda^l}{1-\lambda})^{-1}\sum_{i=1}^{l}\lambda^i\cdot{}^{(i)}TD_t$. This is an adoptation of $TD(\lambda)$ [27] in a fixed sample length while both action and state visitation conditions are ignored, since our agent learns from contiuous state-action space. Eventually, the tail-step bootstrap of the interpolated-TD to the Critic becomes:

$$(\lambda\frac{1-\lambda^l}{1-\lambda})^{-1}\sum_{i=0}^{l-1}\lambda^i\cdot{}^{(l-i)}TD_{t+i} \tag{3}$$

per sliced trajectory (subtrajectory) sampled on a minibatch. When $\lambda = 1$, our method becomes equivalent to the truncated multi-step TD bootstrapping where $w(i) = 1\ \forall i$, which shows that this bootstrapping method actually trains the Critic in interpolated manner upto fixed optimization length.

Throughout experiments, we search for the optimum backup length (update length) for the bootstrapping conserved for every experience pairs. Lastly, we also investigate on the optimal length to backpropagate the update gradient of LSTM-Critic through time (optimization length: see Fig.4b).

2) Hidden state initialization via trajectory scanning: One limitation of the minibatch gradient update for RNN-based DRL is: the historic representations of the networks during the training phase no longer accurately reflect the truth history obtained from the real trial. This is because the sliced trajectories of minibatches decorrelate from the past state-action pairs within the episodes, hence the initial hidden states h_{-1} of the slices become completely *unaware*

[1] For instance, we still need $\{^{(i)}TD_{t+i}\}_{i=1\cdots,i-1,i+1,\cdots l}$ to estimate the $interpolated\text{-}TD_{t+i}$ since only one $^{(l-i)}TD_{t+i}$ out of constituents is measured in the sampled subtrajectory.

[2] We ensure the eveness of visitation by uniformly sampling subtrajectories and episodically saving and removing trajectories.

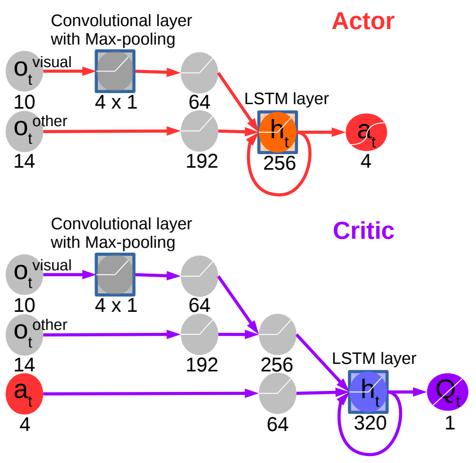


Fig. 6: Network design of Actor and Critic.

of the episodic history. On the other hand, the episodic-wise learning method is utilized since each training data is read from the first or last time-step of its episode [10] [21], and the training data is highly correlated to each other.

However, the importance of having meaningful initial hidden states has been actively searched since the LSTM is capable of memorizing useful long-term history [28]. We aim to develop a method to enforce the h_{-1} to represent meaningful history while the minibatch cuts off the past trajectories of the slices (Fig.3).

We introduce a recent technique mitigating this issue in discrete control domain: scanning the past trajectory before the optimization interval (Fig.5) [23] [29]. By scanning, h_{-1} represents the history ahead of the interval by:

$$\begin{aligned} h_{-1}^{Actor} &= f^{Actor}\left((o_t)_{t=-scanning\ length:-1} \middle| h_{init}\right) \\ h_{-1}^{Critic} &= f^{Critic}\left((o_t, a_t)_{t=-scanning\ length:-1} \middle| h_{init}\right) \end{aligned} \tag{4}$$

where f is the recurrent part of Actor or Critic representing the past trajectory as a history vector, and h_{init} is a zero vector. The scanning intervals solely purpose is to generate h_{-1} and does not generate any update gradient. In this paper, we explicitly validate the scanning strategy in terms of its compatibility on the Actor-Critic framework for continuous action decision task by measuring learning performance as well as analysing locomotion behaviour.

3) Experience injection for behaviour transfer: From prior simulations, we have found that the RDPG agents tend to learn monotonic behaviors. *Monotonic* behavior means that the agent sticks to one pattern of action behavior (or motion) to solve every category of landscape. Monotonic behavior is bad for traversing complex terrains as it is necessary to have different behaviors to negotiate different terrain features.

We have observed that even though RDPG agents can be easily trapped in a monotonic behavior, their monotonic behaviors differ from each other. It is possible to transfer the different monotonic behaviors of different agents into a new agent, guiding the new agent to learn combination of the provided behaviors.

We propose a method called *experience injection* to transfer knowledge between agents to resolve the issue of monotonic behavior. In our proposed experience injection, knowledge is transferred from source agents (teachers) to a recipient agent (student) in the form of trajectories of state-action-reward pairs. Concretely, the experience injection provides the interpolated-TD_t measure (Eqn.2) to the student's Critic as:

$$\sum_{i=1}^{l} w(i) \cdot {}^{(i)}TD_t^{\theta_{student}} \Big|_{a\sim \bigcup_{j\in teachers,student} \pi^j(o),\ (o',r)\sim env(o,a)}$$

By injecting trajectories of well-trained agents into the experience replay buffer of a new agent, we can introduce good learned behavior into the new agent so that it can learn more optimal behavior by deciding non-monotonic action chain. Teachers can be trained from various policy learning algorithms, network designs, and even reward designs before injection. Normalization issue of multiple teacher policies can be ignored since DPG is an off-policy learning algorithm. However, we do not dominate the experience replay buffer with external trajectories since the trajectory distribution distorts [30] and anneal down the contribution of injected trajectories over episodes.

The idea of injecting external experience into an agent is well examined in context of inverse reinforcement learning [31]. However, our injection method does not limit experiences to be bounded near to the current policy of student. Policy distillation [32] property of experience injection method is justified since the Critic is shaped with diverse behaviours of multiple teachers, and then the Actor maximises Q-value solely guided by its Critic.

B. Network structure

We design[3] the Actor (Fig.6) as a four-layer network. The first layer transforms the visual observation with 64 channels of convolutional filters followed by max-pooling. The second layer converts the first layer and the other non-visual observation with 64 and 192 feedforward neurons, respectively and then concatenates them. The second layer is the LSTM of 256 neurons with Relu. The top layer takes feedforward neurons with tanh for 4-dimensional(D) action output. The output of the Actor network is a 4-D torque reference for knee-hip joints of both legs (Fig.7). The tanh activation function is to limit the output range within [-1,1].

The Critic (Fig.6) shares the structure of the Actor but with one more feedforward layer between the second and the third layer. The additional layer augments the 4-D action output to 64 neurons and then concatenates with the second layer. Moreover, we increase the size of LSTM layer of the critic network to 320 neurons. The top layer of the Critic uses linear identity activation instead of tanh activation.

During the policy improvement step, we completely truncate out the temporal Backpropagation path (BPTT) of the

[3]Other network structures were also tested with less depth, pyramidal shape, or without convolutional layer. However, the performance nor behaviour of RDPG does not vary much on our experiments due to the structure. In contrast, the PPO algorithm for our comparison study in table I, 3-layer pyramidal feedforward networks are critically favoured for both Actor and Critic.

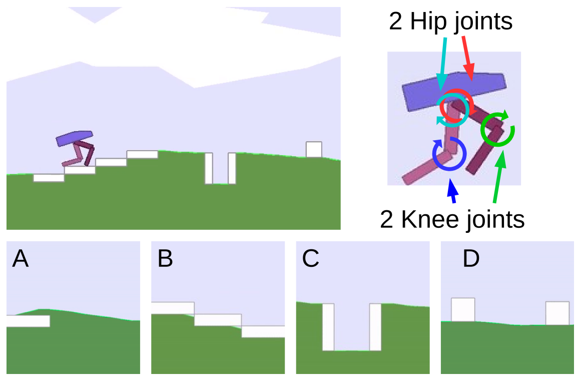


Fig. 7: Terrain feature. A. Slope. B. Stair. C. Gap. D. Hurdles.

LSTMs. Without BPTT, the estimated action at each time-step only refers to the current Q-values for applying the policy gradient $\triangle\omega_t = \nabla_{a_t} Q_t(b_t, a_t) \cdot \nabla_\omega \pi(h, o_t)|_{h=h_t}$.

IV. Simulation Study

A. Simulation environment setup

1) OpenAI Gym BipedalWalker environment: The bipedal walking task that we aim to solve is the OpenAI's BipedalWalker challenge[4]. The 2D simulation environment is partially observable to the bipedal walking agent.

The bipedal character has 4 degrees of freedom, 2 hip and knee joints. The simulation environment provides 24-dimensional sensory feedback. This information consists of 10-D LIDAR (visual) with limited range, 4-D translational/rotational displacement and velocity of hull, 8-D rotational displacement and velocity of the joints, and 2-D binary contact of the feet terrain. The control loop runs the same frequency as the physics simulation at 50Hz.

The goal of the challenge for the bipedal agent is to traverse a variety of rugged terrains (Fig.7) for 360 points without falling. The environment runs episodically, ie an episode terminates if: the body of robot touches the environment, or the agent reaches the goal, or the maximum runtime (40s) is out.

2) Reward design: Our reward design solely focuses on facilitating the agent to move as quick as possible ($\triangle x_t / \triangle t$) with mild penalty against collision to the ground ($r_{c,t} = -20$):

$$r_t = \frac{\triangle x_t}{\triangle t} + \delta(\text{collision}_t == \text{True}) \cdot r_{c,t}$$

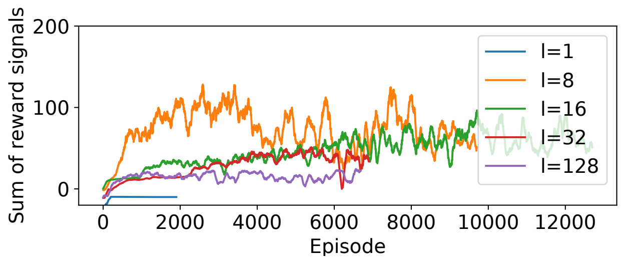
Although the resultant behaviour is less realistic due to high torque of actuators, we found that posture and stability penalties significantly deter the emergence of gait behaviours due to the partial observability, frequency and difficulty of the obstacles (See Table I).

B. Simulation results

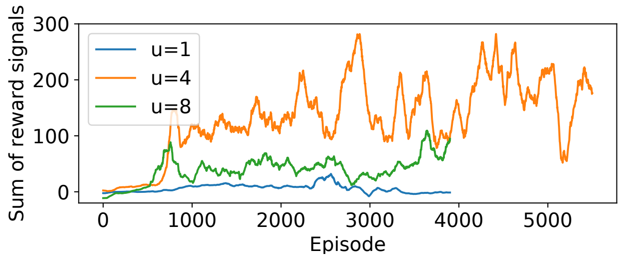
Our DRL framework has successfully trained policies for a stable and dynamic locomotion[5]. It is capable of negotiating a diversity of terrain features including slopes, stairs, gaps and hurdles in a very agile manner. We have also benchmarked our modified RDPG with the existing Feedforward network-based Policy gradient algorithms including DPG and KL-divergence controlled Proximal Policy Optimization algorithm (FFN-PPO$_{KL}$)[6] [26].

[4]https://gym.openai.com/envs/BipedalWalkerHardcore-v2

[5]Video is available at https://youtu.be/ijU4MfdaF8k



(a) Learning curves of agents with various optimization length (l). Update length (u) is set as half of the optimization length.



(b) Learning curves of agents with various update length. Optimization length is fixed as 8 time-steps. The agents are trained on the terrain without obstacles otherwise the agents with u=1 or 8 do not learn to walk.

Fig. 8: Learning performance of agents with various optimization and update length for our interpolated TD method for Critic.

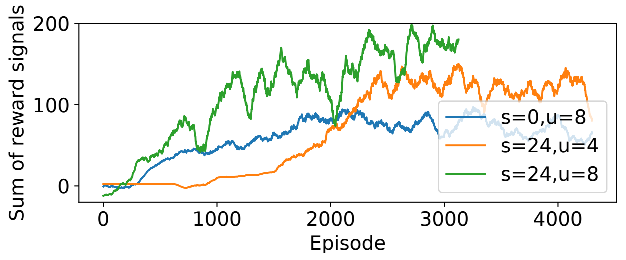
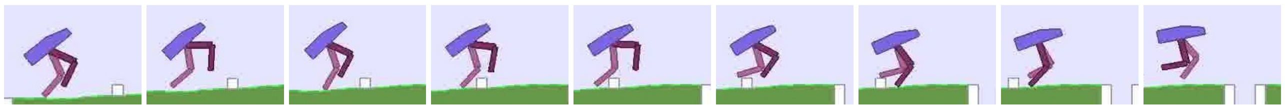


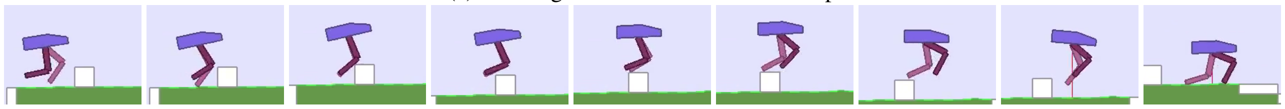
Fig. 9: Learning curves of agents with various scanning interval lengths (s). Optimization length is set as 8.

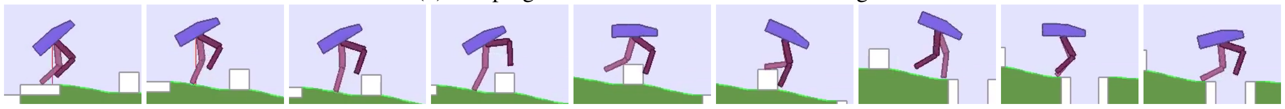
Learning performance is measured with R(100MA) - episodic sum of rewards averaged over the last 100 episodes - and Success ratio - percentage of episodes out of 200 the best agent reaches the goal without fall. The best learning performance of the models are summarized in Table I. Snapshots of locomotion of our RDPG agent are featured in Fig.10.

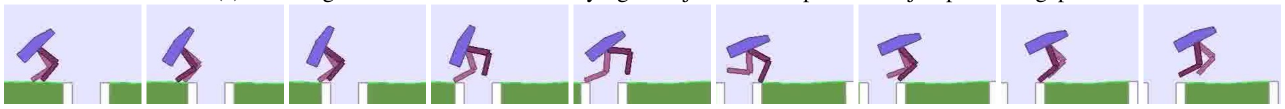
1) Optimal TD backup interval search in interpolation method: We grid-search the optimal optimization length and the ratio of update length. Fig.8 shows that the optimality of policy pursues balance on the variance of Q^{π}_{Critic} over the range of interval lengths.

[6]We adopt active divergence control with Advantage normalization [14]. Target D_{KL} is search over trials for stable policy improvement. The agent is trained over 15000 episodes where the policy does not improve over 3000 episodes.


(a) Walking on flat terrain with a bump


(b) Jumping over a hurdle with stable landing


(c) Crossing over a hurdle followed by agile adjustment of posture to jump over a gap


(d) Leaping over a gap

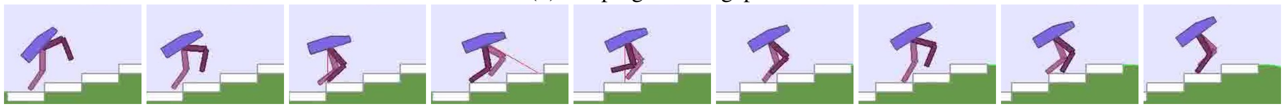

(e) Walking over stairs

Fig. 10: Terrain specific agile behaviors generated by our RDPG agent trained with experience injection.

TABLE I: Summary of performance of neural DPG.

Model	Top R_{100MA}	Success(%)
Our RDPG [a]	227.5	23.5
Our RDPG with injection	**238.5**	**32.5**
Baseline RDPG [11][b]	N/A	0.0
DDPG [18][c]	194.6	28.0
DDPG with injection	187.0	11.5
FFN-PPO_{KL} [26]	72.7	0.5

[a]Optimization/Update/Scanning length set as 8/8/120
[b]TD(0) backup without scanning: N/A means the agent does not walk
[c]Network spec is same as Fig.6 except the LSTM layer being feedforward

As mentioned in Section III-A.1, longer optimization length results in the Critic network estimating discounted rewards in farther time-steps rather than nearer ones; hence the Critic produces a state-action value that is less correlated to the current time-step. Shortening the update length to one time-step hampers learning performance, as can be seen in the figure. When the update length is 1 so that the interpolated-TD is equal to the 1-step TD backup, the agent is not able to learn a successful walking policy as in Fig 8b that the performance converges within a few hundreds of episodes and no longer improves.

2) Initializing hidden state via trajectory scanning improves motion behavior: Fig.9 shows that using the scanning strategy facilitates better learning performance as well as training speed of the agent. Scanning the past trajectory may provide a more reasonable initial hidden LSTM state to both the Actor and Critic network. This results in improved learning performance due to the reduced bias of the Q-value estimated by the RNN-based Critic. Faster improvements on learning curve at optimal optimization length (o= 8 versus 4 with s=24) means that the sampled TD offers better variance on the reduced bias by the scanned initial hidden state. In terms of behavior, the scanning strategy enables the agent to appropriately learn from experiences. This was not the case for the agent with the hidden state h_{-1} initialized as zero (blue line in Fig.9).

Another evidence of the bias reduction on TD by scanning is that the optimal update length changes from 4 to 8 when the optimization length is 8 (Fig.8b). Since the scanning method enables h_{-1} to represent meaningful history of the optimization interval, TD's in early time-steps (Fig.4b) in the interval properly contributes to training the Critic, hence the extension of u upto l improves the performance.

3) Experience injection facilitates diverse behaviors: Interestingly, this study reveals that the RDPG agent can learn *non-monotonic* motion behavior with the help of the experience injection. It is particularly useful in adapting to irregular terrains. Fig.10 shows that the external experience injection takes advantage of the behaviors acquired by previous trained agents, as the new agent is capable of generating diverse set of agile behaviors to stably traverse over different terrain types. Our proposed use of experience injection also facilitates faster learning with the highest reward (Table I). This is not the case for the DDPG agent. We suspect that the lack of memorization capability of past with Feedforward-network rather deters the correct policy gradient of the DDPG agent since the state and observation distributions are changed by our injection method.

As shown in Fig.10b, the agent trained with experience injection can learn how to jump over a hurdle and cross over by putting one leg ahead first (Fig.10c). Previously, those two distinct behaviors were only achieved separately by different trained policies. These results show the agent trained with proposed experience injection is capable of combining the knowledge and behaviors from multiple policies, and hence can generate a better policy. Moreover, as shown in Fig.10b and 10e, some novel responses with better optimality that naturally leads to agile motions emerge, such as stable landing as well as step-by-step climbing over stairs.

V. Conclusion and future work

In this paper, we introduced several methods to improve induced bias and variance of error measure in principled way for RDPG algorithm in partially observable environment. Our work improved previous optimization methods in terms of error measurement, initial hidden state initialization and their correlation. Furthermore, we suggested an effective method of knowledge transfer for episodic experience replay given that the agent estimated better belief-state from observation via memory. The robot controlled by our proposed RDPG with this improved optimization process was trained in the OpenAI's simulation where the terrain and obstacles are partially observable and monotonic locomotion fails. Overall, our study provided evidence that the RDPG agent with proposed optimization methods improves policy via better belief-state representation. As a future work, one can measure the effect of memory by auxillary-task learning on the future observation [33].

References

[1] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2849–2854.

[2] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 3030–3035.

[3] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[5] X. B. Peng, G. Berseth, and M. Van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 80, 2015.

[6] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[7] C. Yang, T. Komura, and Z. Li, "Emergence of human-comparable balancing behaviours by deep reinforcement learning," in *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*. IEEE, 2017, pp. 372–377.

[8] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[9] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of IGPL*, vol. 18, no. 5, pp. 620–634, 2009.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[11] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 387–395.

[14] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[15] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[16] T. Smith and R. G. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," in *UAI*, 2005.

[17] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning Representations*, 2018.

[20] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[21] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR, abs/1507.06527*, 2015.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] J. Harb and D. Precup, "Investigating recurrence and eligibility traces in deep q-networks," *arXiv preprint arXiv:1704.05495*, 2017.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*. MIT Press, 1998.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[28] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, 2015, pp. 2048–2057.

[29] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning." in *AAAI*, 2017, pp. 2140–2146.

[30] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.

[31] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, 2016, pp. 49–58.

[32] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[33] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.