Improving sample efficiency and exploration in upside-down reinforcement learning

Mohammadreza Nakhaei, Reza Askari Moghadam

Journal of Information and Intelligence xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of Information and Intelligence

journal homepage: www.journals.elsevier.com/journal-of-information-and-intelligence

Improving sample efficiency and exploration in upside-down reinforcement learning

Mohammadreza Nakhaei [a], Reza Askari Moghadam [b,*]

[a] Robot Learning Lab, Department of Electrical Engineering and Automation, Aalto University, Espoo 02150, Finland
[b] Laboratoire d'Imagerie Biomédicale, Sorbonne Université, CNRS, INSERM, Paris F-75006, France

ARTICLE INFO

Keywords:
Reinforcement learning
Supervised learning
Intrinsic reward
Recurrent neural networks

ABSTRACT

Supervised learning has been demonstrated to be a stable approach for training deep neural networks. Upside down reinforcement learning solves reinforcement learning problems by using supervised learning, but this method suffers from weak sample efficiency in comparison to state-of-art reinforcement learning algorithms, mostly due to poor exploration. In this paper, we propose modifications to address this issue. To encourage better exploration, entropy maximization, noisy layer, and artificial curiosity are used in training upside-down reinforcement learning agents. Furthermore, to model sequences in reinforcement learning, recurrent neural networks are used in behavior function. We particularly propose deep clockwork RNN for this purpose. To prevent overfitting and underfitting due to a large or small number of updates respectively, we propose proportional number of updates according to the amount of new collected data instead of a fixed number in each iteration. This algorithm outperformed the original upside-down reinforcement learning and the results for several standard environments are presented.

1. Introduction

Traditional deep reinforcement learning (DRL) algorithms learn to predict reward, either by approximating value function (or Q-function) and determining the action with the higher value function in a given state, or by searching for policies that maximize expected returns, or a combination of both [1,2]. DRL algorithms have been successfully applied to many fields including video games, robotics, finance, crop management, and healthcare [3–8]. However, these algorithms suffer from several drawbacks including instability and sensitivity to hyper-parameters [9,10] limiting application in real-world.

The upside down reinforcement learning (UDRL) algorithm solves reinforcement learning (RL) problems using supervised techniques by considering environment feedbacks (such as reward and time horizon) as input [11,12]. More precisely, UDRL agents do not predict rewards, but observe commands in form of desired return and desired horizon meaning "get a specified reward within a specified time". By interacting with the environment and selecting samples with better performance, agents learn to map state and generate commands to action probabilities through backpropagation and gradient descent. It has been shown that UDRL can outperform traditional DRL algorithms in some problems, especially when the reward is sparse [12]. But in general, UDRL algorithms suffer from sample efficiency, mainly because of poor exploration strategy. To address these issues, some techniques are presented in this paper.

* Corresponding author.
E-mail addresses: mohammadreza.nakhaei@aalto.fi (M. Nakhaei), reza.askari_moghadam@sorbonne-universite.fr, reza.askari_moghadam@upmc.fr (R. Askari Moghadam).

https://doi.org/10.1016/j.jiixd.2025.04.004
Received 2 March 2024; Received in revised form 16 April 2025; Accepted 24 April 2025
Available online xxxx

Please cite this article as: M. Nakhaei, R. Askari Moghadam, Improving sample efficiency and exploration in upside-down reinforcement learning, Journal of Information and Intelligence, https://doi.org/10.1016/j.jiixd.2025.04.004

NoisyNet was proposed to improve the exploration of RL agents by adding parametric noise to the parameters of the network [13]. Unlike other strategies, mechanisms of generalization and exploration are dependent and noise parameters are learned along with the network parameters. The combination of NoisyNet and RL agents such as deep Q-learning (DQN) and asynchronous advantage actor critic (A3C) improved the performance in many different environments. Entropy maximization encourages a balance between exploration and exploitation and therefore, results in better performance [14]. Entropy can be considered as a measure of uncertainty in a system. Adding entropy to the loss function encourages the agents to be less certain of certain actions, leading to more stochasticity and avoiding suboptimal policies. Utilization of this technique with traditional policy-based RL algorithms improved efficiency [15–18]. Artificial curiosity (AC) and intrinsic reward are efficient and structured approaches for improving exploration by encouraging agents to search for novel regions, autonomous knowledge acquisition, learning skills, and integrating model-based and model-free RL algorithms [16,19–29]. In artificial curiosity and intrinsic motivation applications, reward signal is augmented with intrinsic reward in different methods including learning process, compression process, information-theoretic measures, prediction error, surprise-based, and visitation count [30–42]. Pixels, random features, variational auto-encoders, and inverse dynamic models were used in the prediction error method for intrinsic reward without considering extrinsic reward in Ref. [32]. A dynamic model for intrinsic reward, despite being simple, and performed well in many environments. Furthermore, combining this method with the UDRL algorithm requires little change and additional computation.

Recurrent neural networks (RNNs) are suitable for dealing with sequential problems such as speech recognition, time series prediction, natural language processing, and video processing [43]. The UDRL algorithm turns RL problems into supervised learning problems and episodes can be viewed as sequential data. Therefore, utilization of RNN is easily possible with little modification. RNNs can recognize possible sequential patterns in environments and improve the sample efficiency of the UDRL algorithm. A simple RNN is unable to learn long-time dependencies due to gradient vanishing [43]. To address this issue, different architectures such as long short-term memory (LSTM), gated recurrent unit (GRU), and clockwork RNN were proposed [44–46]. In clockwork RNN, the hidden layer is partitioned into separate modules with specified clock rates, which enables the network to recognize different patterns with different rates. In this paper, we propose deep clockwork RNN, which can consist of several hidden layers with different neuron numbers and time clocks. Using this architecture enables the UDRL agent to discover more complex sequential patterns.

In the primary version, [12], in each iteration, the number of updates is constant. In some environments like CartPole and Pong, episode length has a correlation with the agent's performance and at the beginning of the training, episodes have fewer steps, therefore, there are fewer samples in the replay buffer. Due to many updates and a limited number of samples, overfitting might occur. Furthermore, these samples contain less valuable information compared to samples obtained later in the training. On the other hand, as training progress and episode lengths become longer, the same number of updates is applied and some valuable samples might not be used in training. In addition, in different environments episode length varies and the number of updates should be tuned respectively. To deal with these issues, we propose considering the number of updates propositional to the number of total new samples added to the replay buffer in each iteration.

This paper is organized into six sections. Section 1 is the introduction and describes the basic ideas. In Section 2, background information about UDRL and clockwork RNN are explained. Section 3 explains the proposed modifications and techniques in more detail. In Section 4, the results of experiments are demonstrated and compared to each other. In Section 5, the results of proposed UDRL are compared with the results of RL method for some standard evaluation environments. Finally, Section 6 summarizes this paper and provides the conclusion.

2. Preliminaries

2.1. Primary upside-down reinforcement learning algorithm

In traditional reinforcement learning, a policy is a function that maps states to actions. It can be deterministic (outputting single action given states), or stochastic (outputting probability distribution given states). In UDRL, behavior function is defined, which takes states and commands as input and maps them to actions. Policy, in traditional RL, is trained to maximize cumulative expected return, behavior function. On the other hand, it is trained to follow the input commands. In theory, behavior function can be trained by any policy that generates all possible trajectories in the environment given sufficient time, but this procedure is not practical and efficient. Furthermore, the main goal is to achieve higher returns in shorter horizons. Therefore, a subset of trajectories with the highest returns is considered for the training behavior function. Therefore, a replay buffer is used to store a fixed maximum number of trajectories with the highest returns. Replay buffer is also used for sampling exploratory commands for generating new experiences and trajectories.

A high-level description of this algorithm is described in Algorithm 1. First, the replay buffer is initialized with a few trajectories by the random policy. In each iteration, behavior policy is improved by supervised training on previous experiences collected from replay buffer. For a fixed number of times, new exploratory commands are determined from the replay buffer, and new trajectories are generated by the interaction of behavior function and environment. Training finishes when stopping criterion, such as reaching maximum steps, is met.

Algorithm 1: High-level description of primary upside-down reinforcement learning

Initialize replay buffer with few specified trajectories generated from random policy.
Until the stop criterion is not met **do**
  Train behavior function on collected samples for n-update times (Algorithm 2).
  Sample exploratory commands (Algorithm 3) and generate episodes for n-episode times.
  Evaluate if necessary.

The behavior function at any time takes states and desired commands as input and returns action probability. For training the behavior function, samples from the replay buffer with desired inputs are selected. Action probabilities for these samples are determined and compared to real taken actions. For discrete action space, the cross-entropy function is used as the loss function. In continuous action space, the behavior function predicts the mean and the standard deviation for each action and the logarithm of the normal distribution probability is used for the loss function. Another variation is to use a fixed number for the standard deviation and the behavior function only predicts the mean for each action. In this case, mean squared error is used as the loss function. Algorithm 2 illustrates sample collection and training in each iteration.

Algorithm 2: Training of behavior function

Input: batch size, behavior function B.
Initialize samples $E \leftarrow \phi$.
For $i = 1$ to batch size:
  Select a trajectory from the replay buffer.
  Select time step indices randomly: $t_1 < T$, where T is length of the selected episode.
  Compute desired return: $d^{\mathrm{r}} = \sum_{t=t_1}^{T} r_t$.
  Compute desired horizon: $d^{\mathrm{h}} = T - t_1$.
  Append $(s_{t_1}, d^{\mathrm{r}}, d^{\mathrm{h}}, a_{t_1})$ to E.
Compute action probabilities for behavior function: $P\left(a \middle| s, d^{\mathrm{r}}, d^{\mathrm{h}}\right) = B(s, d^{\mathrm{r}}, d^{\mathrm{h}}; \theta)$.
Compute loss function.
Compute gradients and update behavior function parameters.

For generating new trajectories, potentially with higher returns, and also evaluation, initial commands must be computed and used in the behavior function. For this purpose, a number of trajectories with the highest returns are selected from the replay buffer. This number is a hyper-parameter and remains fixed in training. Sampling exploratory commands for generating new episodes and evaluation is described in Algorithm 3.

Algorithm 3: Sampling of exploratory commands

Input: number of episodes with highest returns (N).
Output: initial desired return and horizon for generating new episodes and evaluation.
Select N top episodes with the highest returns.
Calculate the average return (M) and standard deviation (S) of these episodes.
Calculate the average time horizon of selected episodes and set it to the initial desired horizon d_0^{h}.
For generating new episode, sample the initial desired return from uniform distribution $\mu[M, M+S]$.
For evaluation, set the initial desired return to M.

For generating new episodes, the initial state is observed and initial commands are sampled. The behavior function selects an action and interacts with the environment. A tuple of state, reward, and next state are stored and inputs, state, and command, are updated; reward is subtracted from the desired reward and desired horizon decreases by one step. The desired horizon cannot be below one, and in evaluation desired returns cannot exceed the maximum possible return. This process is repeated until the episode terminates. The generated trajectory is saved in the replay buffer. For evaluation, greedy action selection is used and action with the highest probability is always selected.

2.2. Clockwork recurrent neural network

Similar to a simple RNN, clockwork RNN consist of input, hidden, and output layers. The difference between a simple RNN and a clockwork RNN is that neurons in the hidden layer are partitioned into g modules with clock period $T_n \in \{T_1, \cdots, T_g\}$. These modules are internally fully connected, but recurrent connections only exist from slower modules to faster ones. One of the best choices for the clock period of modules is to use exponential series: Module j has clock period of 2^{j-1}. The output of the network is calculated using the following equations:

$$H^t = f_{\mathrm{h}}\left(W^{\mathrm{h}}H^{t-1} + W^{\mathrm{i}}X^t\right), \tag{1}$$

$$O^t = f_{\mathrm{o}}(W^{\mathrm{o}}H^t), \tag{2}$$

where H^t is hidden state at time step t, X^t is input of the network, O^t is the output of the network, W^{h}, W^{i}, and W^{o} are hidden, input, and output weight matrices, respectively. f_{h} and f_{o} are non-linear activation functions. In this architecture, hidden weight matric is divided into g block rows:

$$W^{\mathrm{h}} = \begin{pmatrix} w_1^{\mathrm{h}} \\ \vdots \\ w_g^{\mathrm{h}} \end{pmatrix}. \tag{3}$$

At each time step, if the module i is activated, the value of w_i^{h} is used in the forward pass, otherwise, the value for the row w_i^{h} is considered to be zero. Since recurrent connections only exist from slower modules to faster ones and modules' time cycles are sorted, the W^{h} is a block-upper triangular matrix.

At each time step, only the modules that the remainder of dividing time step by module's time period become zero are active ($t \bmod T_i = 0$). Therefore, low-clock-rate can recognize long-term patterns in sequential data and keep information in memory. Fast-clock-rate modules are able to focus on high frequent information. The number of modules, hidden neurons, and clock periods are hyper-parameters of this architecture.

3. Modifications

In this section, the modifications to the original UDRL algorithm are explained. They boost the performance in the term of sample efficiency. First, the strategies to improve data collection procedure are described, then we propose utilizing sequence models in the context of upside-down RL. UDRL apply pure supervised learning to solve RL tasks without bootstrapping and temporal-difference learning, therefore, sequence models can be applied straightforwardly to improve credit assignment and long-term memory.

3.1. Entropy maximization

Entropy maximization generalizes the standard objective in reinforcement learning with an entropy term aiming to partially maximize entropy for each visited state. The parameter α determines the relative importance of entropy term; higher α leads to more stochasticity and vice versa:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E\left[\sum_{t=t_0}^{T} r(s_t, a_t) + \alpha H(.|s_t)\right]. \tag{4}$$

During the training in UDRL, the standard goal is to minimize the loss function. To maximize entropy, the entropy term should be subtracted from the standard loss function in gradient descent. Furthermore, in contrast to RL algorithms, the entropy term is conditioned on state and command. The modified loss function is defined as:

$$L_{\mathrm{total}} = L_{\mathrm{std}} - \alpha H = L_{\mathrm{std}} + \alpha \sum P(a|s,c)\log P(a|s,c). \tag{5}$$

3.2. Noisy layer

In noisy networks, some weights and biases are augmented with parametric noise. These parameters are adapted during training with gradient descent. Consider a simple linear layer of a neural network, represented by

$$y = \omega x + b, \tag{6}$$

where x is the input of the layer, ω is the weight matrix, and b is the bias. Noisy layer consists of added noise parameters and noise random variables:

$$y = (\mu^{\omega} + \sigma^{\omega} \odot \epsilon^{\omega})x + \mu^b + \sigma^b \odot \epsilon^b. \tag{7}$$

The parameters $\mu^{\omega}, \mu^b, \sigma^{\omega}, \sigma^b$ are learnable and $\epsilon^{\omega}, \epsilon^b$ are noise random variables. In the main paper [13], two options for random variables are proposed: independent Gaussian noise and factorized Gaussian noise. In independent Gaussian noise, noise applied to each parameter is independent and sample from the normal distribution. Factorized Gaussian noise uses an independent noise for each output and another for each input. The main aim is to reduce computation time for generating random numbers. In this paper, independent Gaussian noise is used. Note that in evaluation, the random noise is ignored and the noisy layer works like a simple fully connected layer.

3.3. Artificial curiosity and intrinsic motivation

As described earlier, several methods have been proposed for intrinsic reward. We select the dynamic-based intrinsic reward because this method is scalable, computationally efficient, stable, and requires little modification to the basic algorithm. In this method, a separate network is used that takes the current state and action as input and predicts the next state. The predicted next state is compared to the real one, and the loss function (mean squared error) of this network is used as the intrinsic reward with a coefficient, α_{ac}. This coefficient is a hyper-parameter that determines the relative weight of the intrinsic reward to the desired return, and a higher value leads to more curiosity. Higher intrinsic reward indicates regions where the agent has little knowledge and might contain valuable information. The intrinsic reward is added to the desired returns of the samples only in training stage. This encourages the agent to take actions leading to uncertain regions according to dynamic predictions, exploring novel regions of environment consequently. As training progresses, the dynamic model improves, and the next state is predicted more accurately. Therefore, the role of curiosity decreases. The process of computing and using intrinsic reward is illustrated in Algorithm 4.

Algorithm 4: Intrinsic reward calculation

Input: samples for training behavior function, artificial curiosity network.
Output: updated desired return of samples.
Select samples from replay buffer.
Predict next state given current state and action: $S'_{\text{pred}} = AC(s, a; \theta_{\text{ac}})$.
Compute loss function: $L = \left| S'_{\text{pred}} - S' \right|^2$.
Train artificial curiosity network by gradient descent.
Determine intrinsic reward: $R_{\text{ac}} = \alpha_{\text{ac}} L$.
Add intrinsic reward to desired return: $d^{\text{r}} := d^{\text{r}} + R_{\text{ac}}$.
Use $\left(s, d^{\text{r}}, d^{\text{h}}, a\right)$ to train behavior policy in Algorithm 2.

3.4. Recurrent neural networks in behavior function

In most environments, Markov decision process (MDP) conditions exist and fully connected networks are sufficient for behavior function, but using RNNs can improve sample efficiency by considering the sequential nature of reinforcement learning problems. RNNs take the history of the agent as well as the current condition for making decisions. In partially observable environments where the MDP condition is not satisfied, recurrent networks in behavior function are able to learn the task specified in the environment. In this paper, LSTM, GRU, and deep clockwork RNN are used in the behavior function. For the initial state, previous hidden states are considered to be zero. One issue with recurrent networks is that the samples in mini-batch must have the same sequential length. To resolve this, two options were investigated. First, we set the batch size to one and collect a sample from the replay buffer where the sequence starts at a random time step t_1 and ends when the episode terminates. Second, we collected multiple samples from the replay buffer and set the sequential length to a pre-defined length by adding zeros to shorter samples and masking them during training. The second solution in practice ends up in better convergence and faster training.

3.4.1. Deep clockwork RNN

As explained before, clockwork RNN partitions the hidden layer into modules, considering different clock period for each one. At each time step, modules that satisfy the condition are active. In deep clockwork RNN, the output layer is omitted, and the hidden layer becomes the input for the next layer. Therefore, many layers with different numbers of modules, number of neurons, and time periods can be stacked together. In addition, connections between slower modules to faster modules are also omitted so that the network has fewer parameters and is trained faster through backpropagation. Fig. 1 compares the difference between original clockwork weight matrix for hidden state and proposed version where $w_{i,j} \in R^{n_i \times n_j}$ is the corresponding weight matrix of W^{h} that connects module j with n_j neurons to module i with n_i neurons. Fig. 2 illustrates deep clockwork architecture.

The hidden state of each layer is stored in a list for the next prediction. The following equation illustrates the output of each layer. For convenience, the input layer is considered to be the zeros hidden layer in the network architecture:

$$\begin{cases} H_0^t = X^t, \\ H_j^t = f_{\text{h}}\left(W_j^{\text{h}} H_j^{t-1} + W_j^{\text{i}} H_{j-1}^t\right), \; j = 1 : J, \\ O^t = f_{\text{o}}\left(W^{\text{o}} H_J^t\right) \end{cases} \tag{8}$$

where the H_j^t is the value of the hidden state of the jth layer at time step t, W_j^{h} is the weight matrix for hidden state for the jth layer which contains different modules with different clock rates, W_j^{i} is the weight matrix for the input which the hidden state from the previous layer, and W^{o} is the weight matrix for computing the ouput from the hidden state of the last layer, acting as a linear layer. The hidden state at the first timestep for each layer is considered to be zero. Each layer contains modules with different clock rate to capture

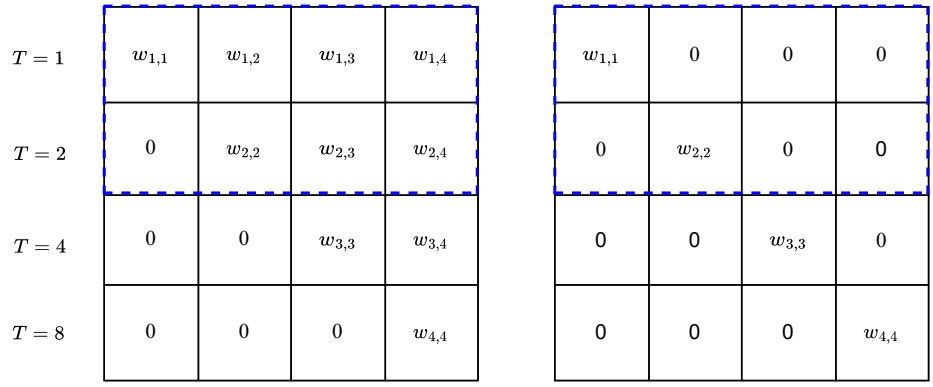

Fig. 1. Difference between original clockwork weight matrix for hidden state (W^{h}) and the modification. As an example, assume four modules $t = 6$, so only the first two modules are activated (blue dashed lines). The modified weight matrix has fewer parameters and is more suitable for stacking layers on top of each other for deep networks.

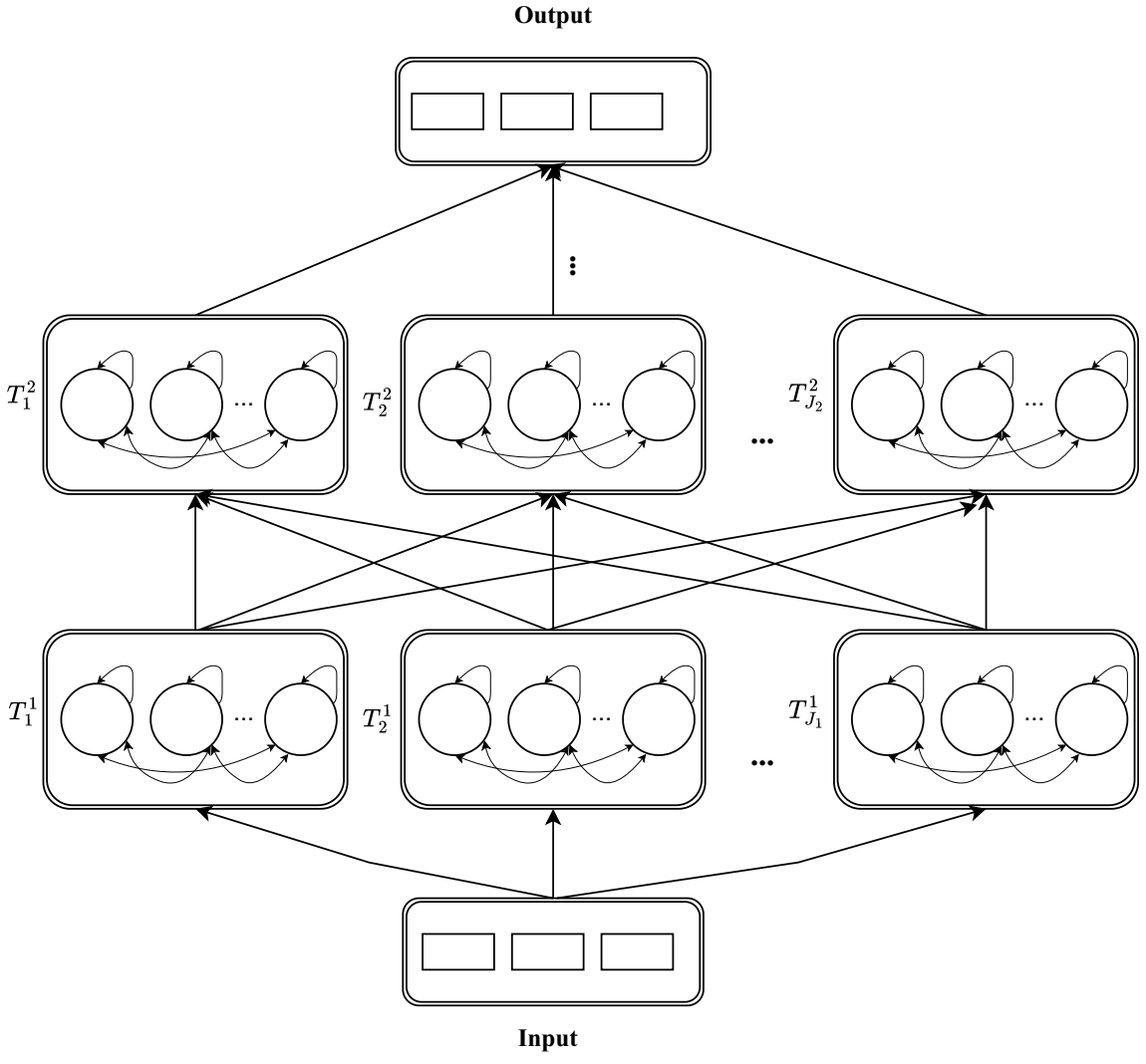

Fig. 2. Deep clockwork RNN architecture.

temporal patterns with different frequencies. During generating new episodes and evaluation, the time step of the episode should be taken into account for selecting actions and calculating the hidden state for the next time step.

4. Experiments

The goal of experiments is to compare the performance of primary UDRL agent with proposed methods. Six environments from Gym RL library are considered for this purpose: CartPole, Acrobot, and LunarLander, InvertedPendulum, InvertedDoublePendulum, and Hopper [47]. In all the simulations, five random seeds (0∼4) were used for simulation and the average and the standard deviation is obtained.

Similar to the primary paper, in the behavior function, input commands are scaled by fixed coefficients and transformed by a sigmoidal fully-connected layer. States are transformed to an embedding by a linear layer and tanh as activation function. Transformed states and commands are multiplied element-wise together. The output is used as input to the next hidden layer similar to a standard fully-connected network. The hyper-parameters are provided in the Appendix.

4.1. Entropy maximization

To encourage exploration, the entropy term is added to the loss function with a scaling coefficient β. For different values of β experiments were run, and results are illustrated in Fig. 3.

As Fig. 3 demonstrates, choosing the right value of the coefficient factor β can improve the performance in some environments. Low values do not encourage enough exploration and higher values make agents unable to decide with enough certainty. Adding entropy term deteriorated the performance in InvertedPendulum environment, especially with higher coefficient value, and did not lead to improvement in the performance of the agent in InvertedDoublePendulum environment. For these environments, taking greedy actions according to the data leads to the optimal policy and adding entropy term can reduce performance. One common approach is to adapt the coefficient to maintain certain amount of entropy during training, but this approach requires considering another optimizer to change the coefficient.

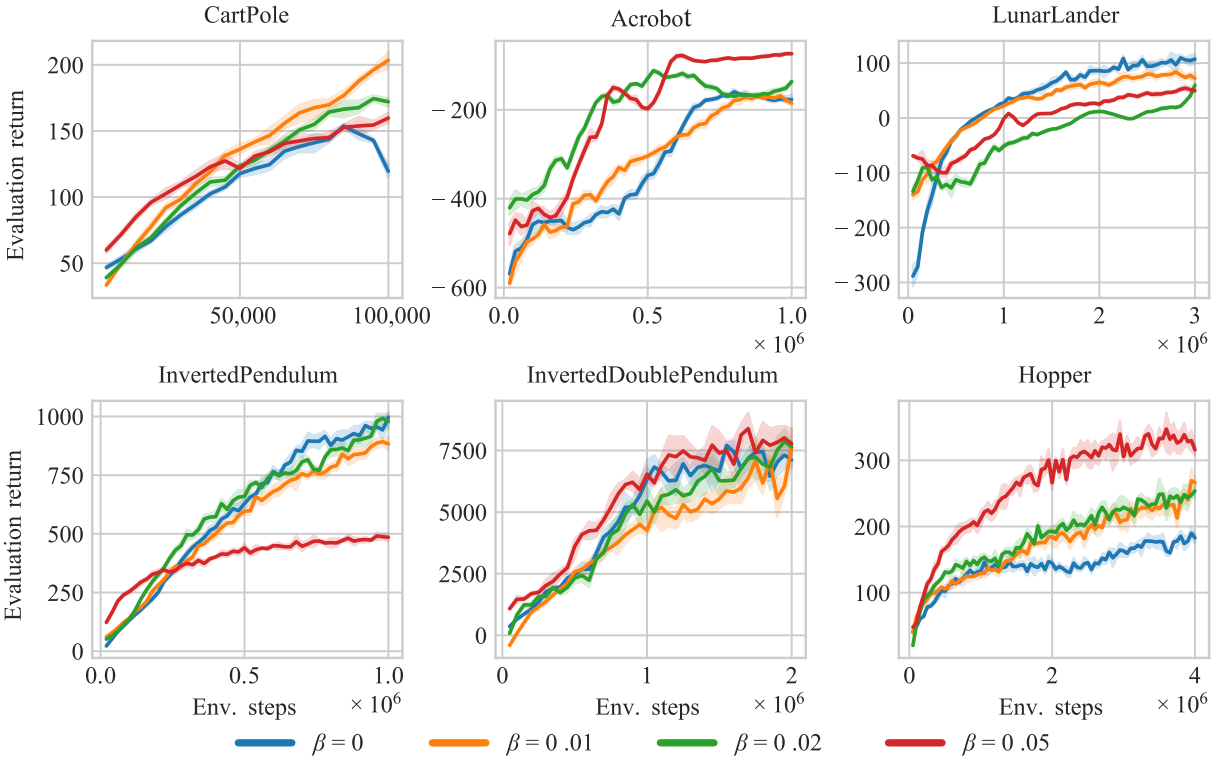

Fig. 3. Entropy bonus results, mean of evolution reward for 5 random seeds with different coefficients.

4.2. Noisy layer and artificial curiosity

In the behavior function, noisy layers are used instead of simple linear layers for some of the hidden layers. Using too many noisy layers in behavior function might lead to weaker performance and even instability due to too much randomness. As explained earlier, a separate network is used for the calculation of intrinsic reward, which can have a different architecture than behavior function. In a more complex network, it may take longer for the network to learn the dynamics of the environment, therefore, curiosity participates for a longer period in training. On the other hand, too simple networks cannot learn due to an insufficient number of parameters, and curiosity is overestimated in training. For simplicity, the same hidden layers are used in the artificial curiosity network. Experiments were carried out with considering noisy layer, artificial curiosity, and combination of both. Fig. 4 demonstrates the results of the experiments.

In environments where exploration is challenging, noisy layers and intrinsic reward lead to better results, and in other environments, the performance of agent did not change considerably. Unlike adding entropy bonus, these methods did not lead to weaker performance. Our intuition is that entropy term changes the output probability distribution of the behavior function and makes the agent less certain in evaluation, but this technique does not change the output probability distribution in evaluation, but provide more diversity in the training data.

4.3. Recurrent neural networks

In the behavior function, different recurrent architectures including LSTM, GRU, and clockwork RNN are used. In clockwork RNN architecture, exponential series of periods is used for each layer, module i has clock period of 2^{i-1}. Fig. 5 demonstrates the effect of using these architectures on sample efficiency. Except for LunarLander, using recurrent networks improves the sample-efficiency by considering the sequence patterns in the environment. Deep clockwork RNN performs better LSTM and GRU since it can capture both high-frequency and low-frequency patterns by having different time clock for different modules at different levels.

4.4. Proportional number of updates

Instead of using a fixed number of updates per iteration, the number of updates in each iteration is determined proportionally to the number of interaction steps in that iteration. At each iteration, the number of new transitions is computed and the number of updates for training the behavior policy is determined proportionally. When there are more new transitions in the buffer, there are more new data,

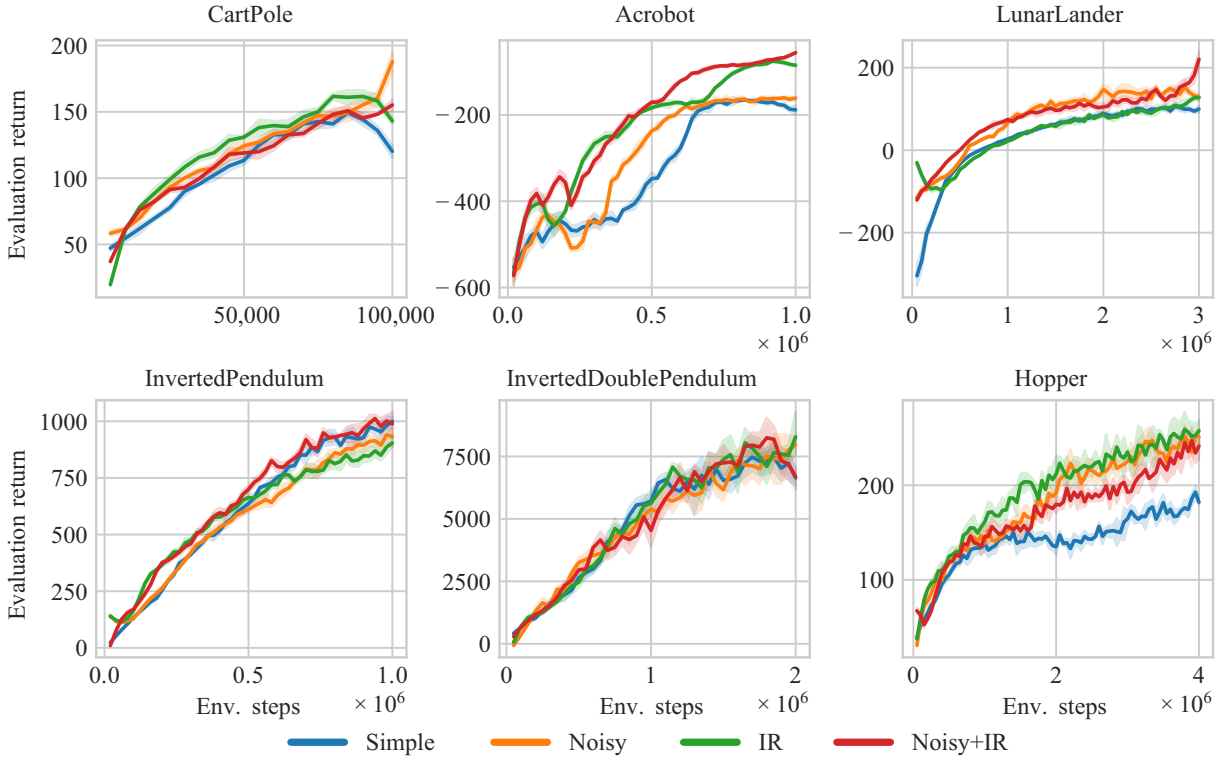

Fig. 4. The performance of UDRL, noisy UDRL, AC UDRL, and noisy AC UDRL agents.

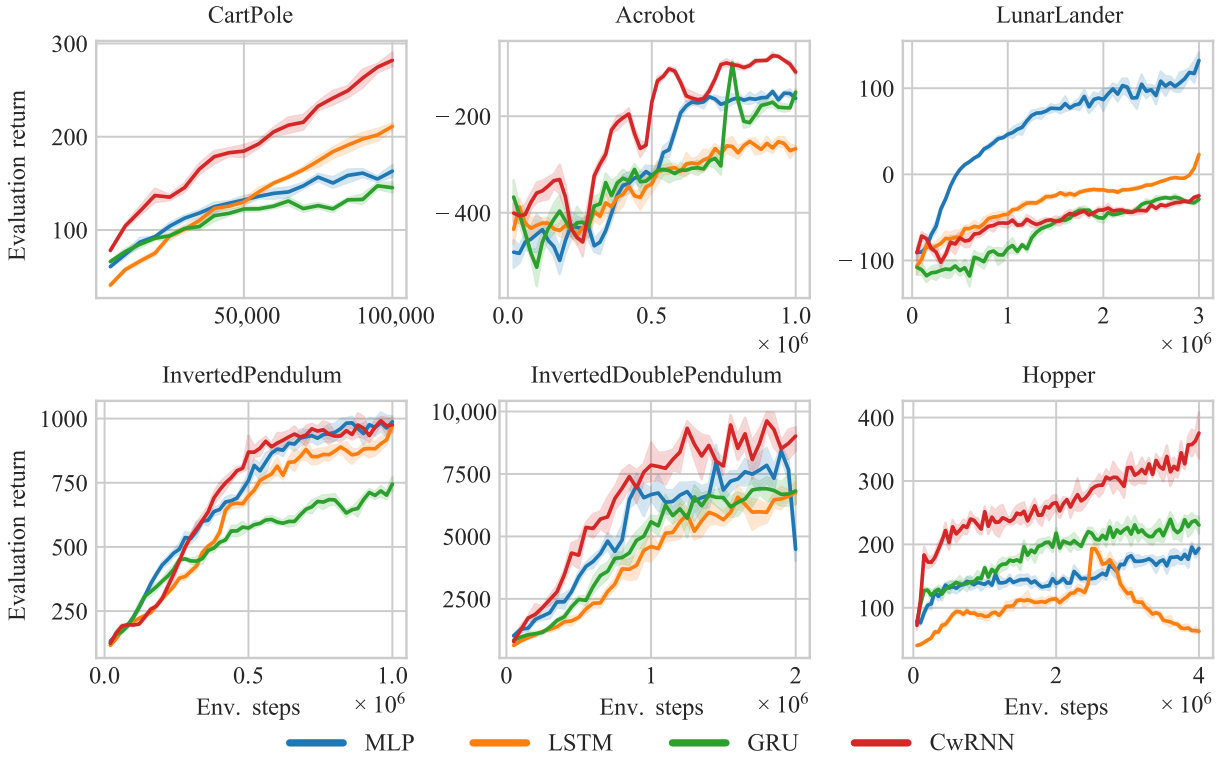

Fig. 5. The effect of using recurrent architectures in behavior function.

therefore larger number of updates. Fig. 6 compares the methods. Shaded areas represent mean ± standard deviation of evaluation reward.

Except for Acrobot environment, using proportional number of updates leaded to better results or make the training more stable by reducing the variance of training for different random seeds.

4.5. Combining proposed method

In this section, the proposed methods including, entropy maximization, intrinsic motivation, noisy layer, utilization of recurrent architecture in the behavior function in case of better results, and the proportional number of updates are implemented together. It is expected that the combination of these methods outperforms each of them. As Fig. 7 illustrates, proposed methods significantly improved sample-efficiency and performance in different environments.

Fig. 8 demonstrate the final test performance of the agent for different desired returns. We exclude Hopper and LunarLander since for different desired returns, we cannot compute the desired horizon easily. Except for Acrobot, the agent performs according to the desired return and can follow the order. Our intuition for Acrobot environment is that there is a discontinuity in the returns, if the agent can swing up the pole, it suddenly gets a large reward.

5. Comparison with reinforcement learning

In this section, the results of upside-down RL and the results of on-policy and off-policy RL are compared. DQN [48] and deep deterministic policy gradient (DDPG) [49] are considered as off-policy RL baselines and proximal policy optimization (PPO) [18] is assumed as on-policy RL baseline. Fig. 9 show the results and comparisons. While upside-down RL requires more training steps, it is more stable, especially in environments with continuous actions.

We also consider the case where the agent gets the total reward at the last time step. In this setting RL algorithm fails while upside-down RL can learn to accomplish the task and improves the performance, as illustrated in Fig. 10.

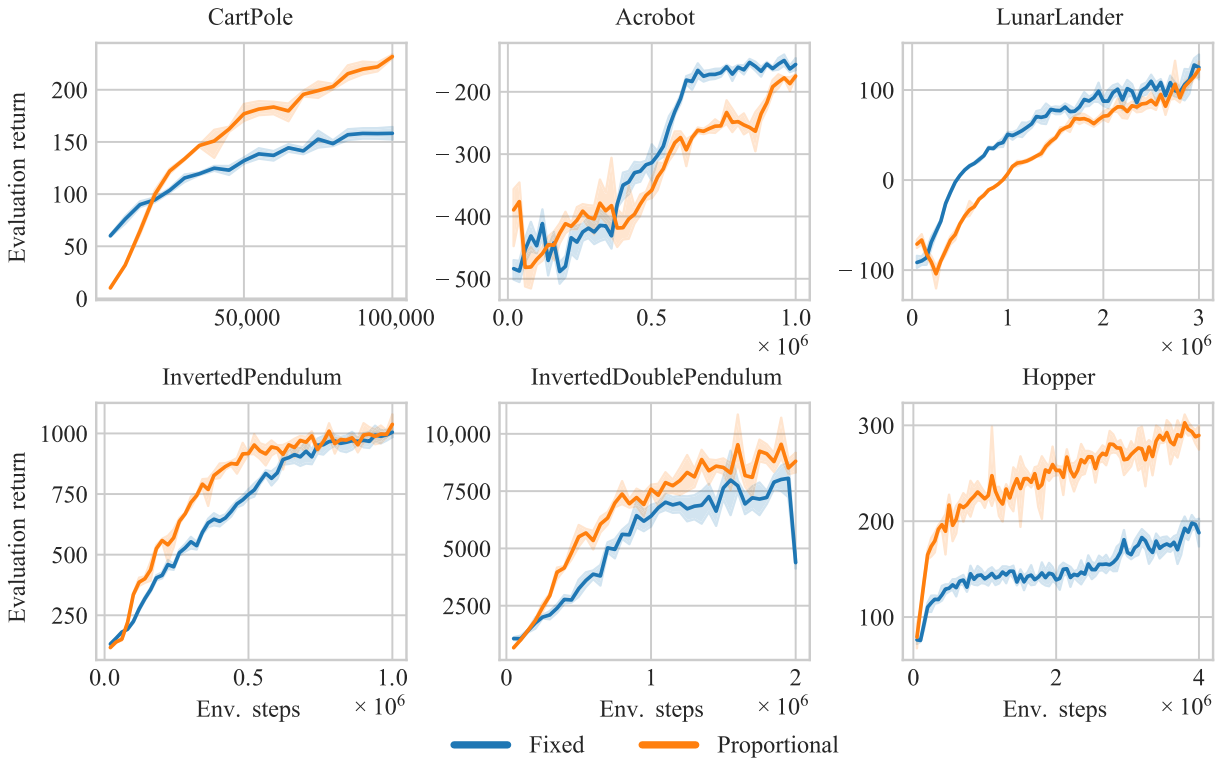

Fig. 6. Proportional versus fixed number of updates.

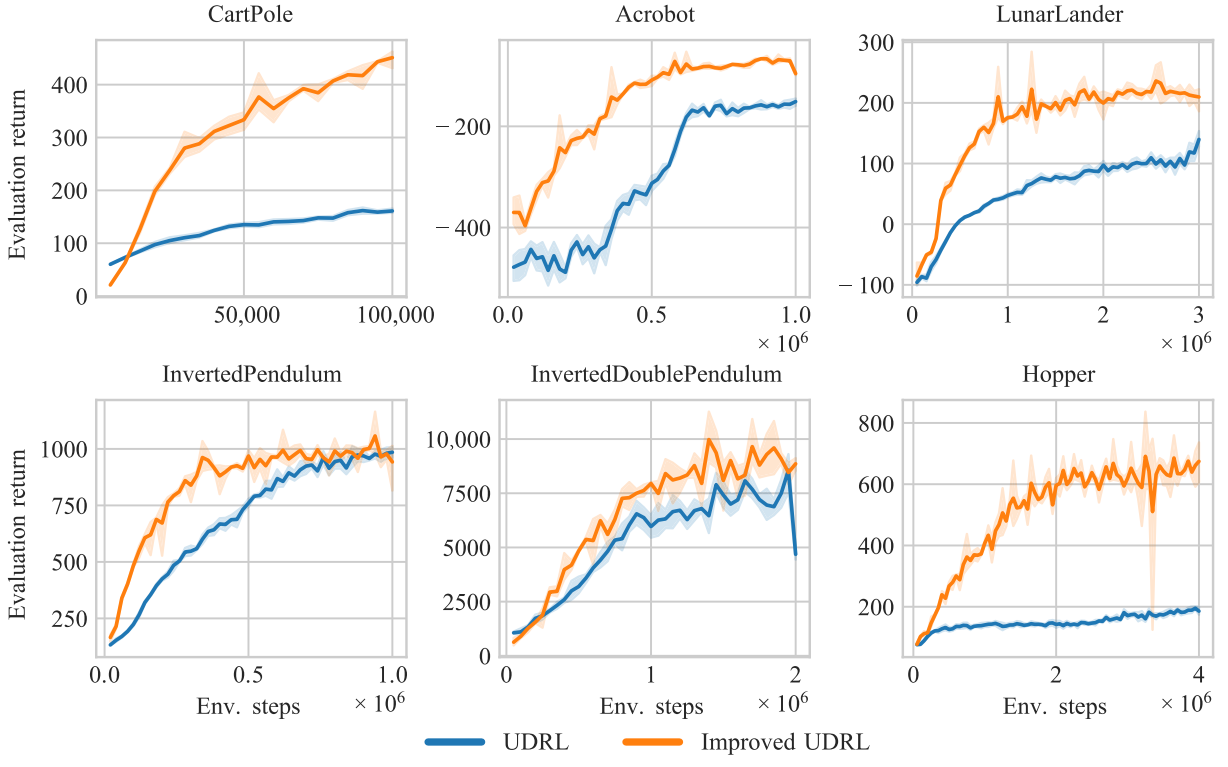

Fig. 7. Primary versus improved UDRL

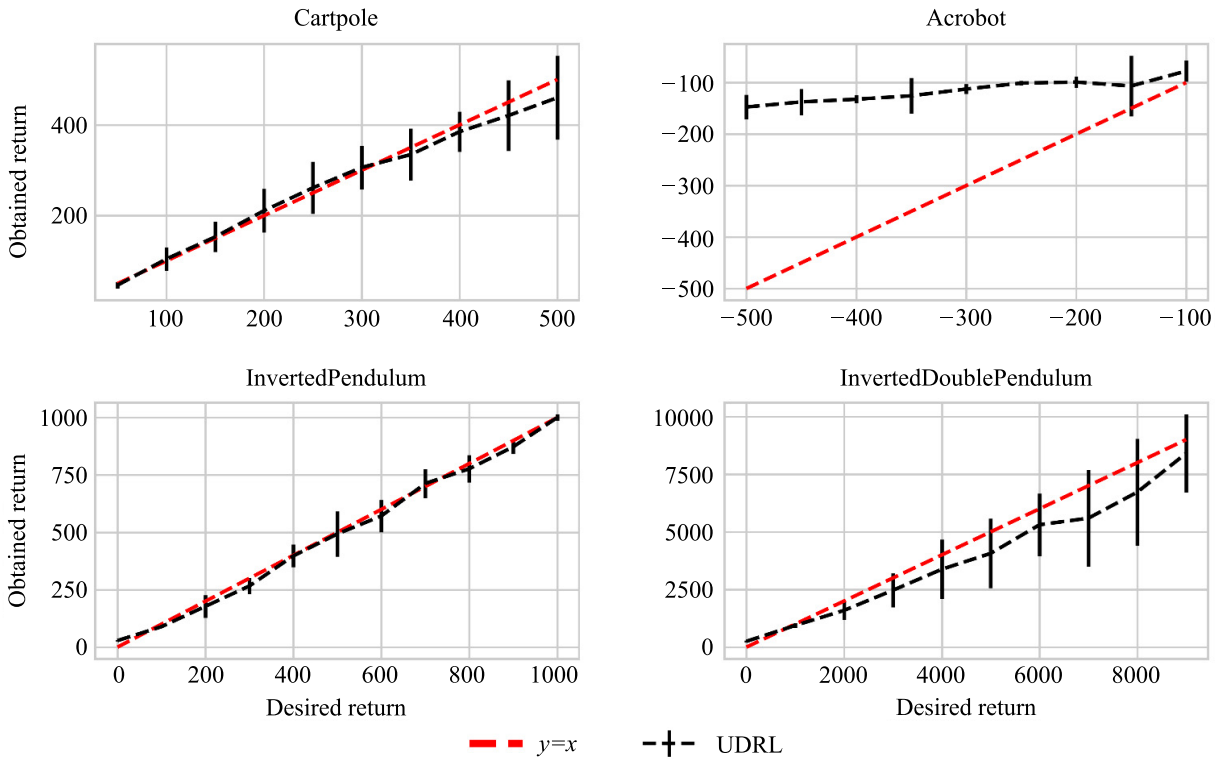

Fig. 8. Test of final performance with different desired returns, the agent can follow the orders.

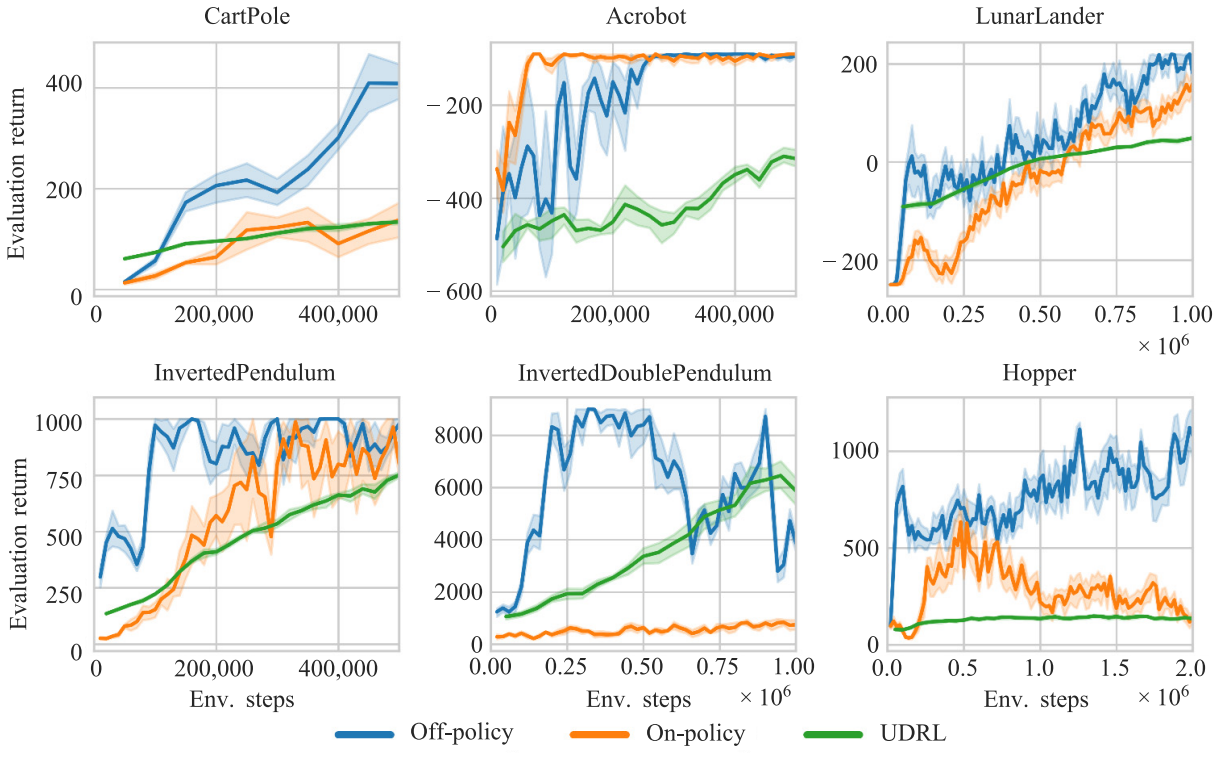

Fig. 9. Comparing upside-down RL with on-policy and off-policy RL.

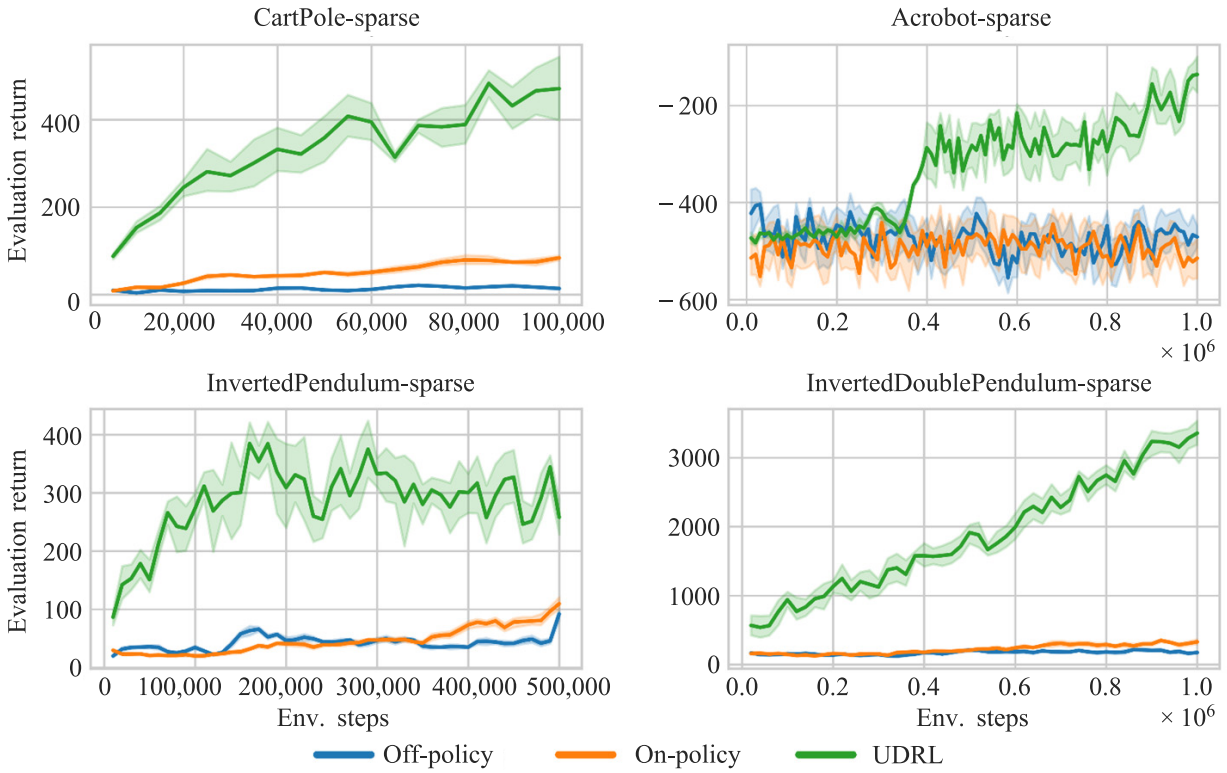

Fig. 10. Comparing upside-down RL with on-policy and off-policy RL in sparse reward environments.

6. Conclusion

In this paper some modifications are proposed to improve sample-efficiency and exploration in upside-down reinforcement learning. Entropy maximization balances a trade-off between exploration and exploitation, noisy layers encourage more exploration by adding parametric noise, and intrinsic reward motivates the agent to search for novel areas. Recurrent networks can model the sequential nature of reinforcement learning and improve performance even in environments where the MDP condition is met. Deep clockwork RNN particularly enhanced the performance because this architecture can recognize sequential patterns with different lengths due to different time clocks. The proportional number of updates instead of fixed number reduces the variance and improves performance in most environments. Furthermore, this hyper-parameter is easier to be tuned in different environments with different lengths. The implementation of all the above-mentioned modifications can significantly outperform the primary algorithm while following the commands.

CRediT authorship contribution statement

Mohammadreza Nakhaei: Writing – review & editing, Writing – original draft, Software, Resources. **Reza Askari Moghadam:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that there's no conflict of interests that could affect the work reported in this paper.

Appendix

Table 1
Upside-down RL hyper-parameters with FC architecture.

Hyper-parameter	Value
Batch size	512
Buffer size	300
Horizon scale	0.01
Intrinsic network learning rate	1e-4
Learning rate	5e-4
Number of episodes in each iteration	150
Number of neurons in hidden layers	[64,64,64]
Number of top episodes	75
Number of updates in each iteration	20
Return scale	0.01
Update ratio (proportional number of updates)	0.2

Table 2
Upside-down RL hyper-parameters with RNN architectures (LSTM, GRU).

Hyper-parameter	Value
Batch size	64
Number of neurons in hidden layers	[64]
Number of recurrent layers	2
RNN input size	16
RNN hidden size	16

Table 3
Upside-down RL hyper-parameters with deep clockwork RNN.

Hyper-parameter	Value
Batch size	64
Hidden neurons for modules (for each layer)	[4, 8]
Number of neurons in hidden layers	[64]
Number of Recurrent Layers	2
Time clocks for modules (for each layer)	[1, 2, 4, 8, 16, 32], [1, 2, 4, 8]

References

[1] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (6) (2017) 26–38.
[2] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Second edition, The MIT Press, 2018.
[3] K.A. ElDahshan, H. Farouk, E. Mofreh, Deep reinforcement learning based video games: A review, in: Proceedings of the 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), IEEE, Piscataway, 2022, pp. 302–309.
[4] M. Wehrmann, N. Zengeler, U. Handmann, Observation time effects in reinforcement learning on contracts for difference, Journal of Risk and Financial Management 14 (2) (2021) 54.
[5] Z. Pan, S. Yin, G. Wen, Z. Tan, Reinforcement learning control for a three-link biped robot with energy-efficient periodic gaits, Acta Mechanica Sinica 39 (2) (2023) 522304.
[6] W. Zhao, J.P. Queralta, T. Westerlund, Sim-to-real transfer in deep reinforcement learning for robotics: A survey, in: Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, Piscataway, 2020, pp. 737–744.
[7] R. Gautron, E.J. Padrón, P. Preux, J. Bigot, O.-A. Maillard, G. Hoogenboom, J. Teigny, Learning crop management by reinforcement: Gym-DSSAT, in: Proceedings of the AIAFS 2023 - 2nd AAAI Workshop on AI for Agriculture and Food Systems, AAAI Press, Washington, 2023.
[8] C.-Y. Yang, C. Shiranthika, C.-Y. Wang, K.-W. Chen, S. Sumathipala, Reinforcement learning strategies in cancer chemotherapy treatments: A review, Computer Methods and Programs in Biomedicine 229 (2023) 107280.
[9] J. Fu, A. Kumar, M. Soh, S. Levine, Diagnosing bottlenecks in deep Q-learning algorithms, in: Proceedings of the 36th International Conference on Machine Learning, PMLR, New York, 2019, pp. 2021–2030.
[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, Washington, 2018, pp. 3207–3214.
[11] J. Schmidhuber, R.K. Srivastava, Upside-down reinforcement learning, US20210089966A1. March, 2021.
[12] R.K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, J. Schmidhuber, Training agents using upside-down reinforcement learning, 2019. https://doi.org/10.48550/arXiv.1912.02877.
[13] M. Fortunato, M.G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al., Noisy networks for exploration, 2017. https://doi.org/10.48550/arXiv.1706.10295.
[14] B. Xin, H. Yu, Y. Qin, Q. Tang, Z. Zhu, Exploration entropy for reinforcement learning, Mathematical Problems in Engineering 1 (2020) 2672537.

[15] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, New York, 2018, pp. 1861–1870.
[16] T. Haarnoja, Controlling robots using entropy constraints, US20220019866A1. January, 2022.
[17] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: Proceedings of the 33rd International Conference on Machine Learning, PMLR, New York, 2016, pp. 1928–1937.
[18] Y. Wang, H. He, X.Y. Tan, Truly proximal policy optimization, in: Proceedings of The 35th Uncertainty in Artificial Intelligence Conference, PMLR, New York, 2020, pp. 113–122.
[19] N. Chentanez, A. Barto, S. Singh, Intrinsically motivated reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems 17 (NIPS 2004), MIT Press, Cambridge, 2005, pp. 1281–1288.
[20] J.-T. Chien, P.-C. Hsu, Stochastic curiosity maximizing exploration, in: Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, Piscataway, 2020, pp. 1–8.
[21] V. Graziano, T. Glasmachers, T. Schaul, L. Pape, G. Cuccu, J. Leitner, J. Schmidhuber, Artificial curiosity for autonomous space exploration, Acta Futura 4 (2011) 41–51.
[22] M.B. Hafez, C. Weber, S. Wermter, Curiosity-driven exploration enhances motor skills of continuous actor-critic learner, in: Proceedings of the 2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), IEEE, Piscataway, 2017, pp. 39–46.
[23] M.B. Hafez, C. Weber, M. Kerzel, S. Wermter, Curious meta-controller: Adaptive alternation between model-based and model-free control in deep reinforcement learning, Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, Piscataway, 2019, pp. 1–8.
[24] G. Khandate, S. Shang, E. Chang, T. Saidi, Y. Liu, S. Dennis, J. Adams, M. Ciocarlie, Sampling-based exploration for reinforcement learning of dexterous manipulation, 2023. https://doi.org/10.48550/arXiv.2303.03486.
[25] T.D. Kulkarni, K.R. Narasimhan, A. Saeedi, J.B. Tenenbaum, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, Curran Associates Inc., New York, 2016, pp. 3682–3690.
[26] H. Ngo, M. Luciw, A. Forster, J. Schmidhuber, Learning skills from play: Artificial curiosity on a Katana robot arm, in: Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), IEEE, Piscataway, 2012, pp. 1–8.
[27] N. Liu, Y. Liu, B. Logan, Z. Xu, J. Tang, Y. Wang, Learning the dynamic treatment regimes from medical registry data through deep Q-network, Scientific Reports 9 (1) (2019) 1495.
[28] D.M. Ramík, K. Madani, C. Sabourin, From visual patterns to semantic description: A cognitive approach using artificial curiosity as the foundation, Pattern Recognition Letters 34 (14) (2013) 1577–1588.
[29] D.M. Ramík, C. Sabourin, K. Madani, Autonomous knowledge acquisition based on artificial curiosity: Application to mobile robots in indoor environment, Robotics Autonomous Systems 61 (12) (2013) 1680–1695.
[30] A. Baranes, P.-Y. Oudeyer, Robust intrinsically motivated exploration and active learning, in: Proceedings of the 2009 IEEE 8th International Conference on Development and Learning, IEEE, Piscataway, 2009, pp. 1–6.
[31] M.G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, R. Munos, Unifying count-based exploration and intrinsic motivation, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, Curran Associates Inc., New York, 2016, pp. 1479–1487.
[32] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, A.A. Efros, Large-scale study of curiosity-driven learning, 2018. https://doi.org/10.48550/arXiv.1808.04355.
[33] M. Fran k, J. Leitner, M. Stollenga, A. Förster, J. Schmidhuber, Curiosity driven reinforcement learning for motion planning on humanoids, Frontiers in Neurorobotics 7 (2014) 25.
[34] T. Hester, P. Stone, Intrinsically motivated model learning for developing curious robots, Artificial Intelligence 247 (2017) 170–186.
[35] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, P. Abbeel, VIME: Variational information maximizing exploration, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, Curran Associates Inc., New York, 2016, pp. 1117–1125.
[36] L. Itti, P. Baldi, Bayesian surprise attracts human attention, Vision Research 49 (10) (2009) 1295–1306.
[37] G. Ostrovski, M.G. Bellemare, A. Van den Oord, R. Munos, Count-based exploration with neural density models, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, PMLR, New York, 2017, pp. 2721–2730.
[38] D. Pathak, P. Agrawal, A.A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, PMLR, New York, 2017, pp. 2778–2787.
[39] J. Schmidhuber, Curious model-building control systems, in: Proceedings of the 1991 IEEE International Joint Conference on Neural Networks, IEEE, Piscataway, 1991, pp. 1458–1463.
[40] J. Schmidhuber, Formal theory of creativity, fun, and intrinsic motivation (1990-2010), IEEE Transactions on Autonomous Mental Development 2 (3) (2010) 230–247.
[41] B.C. Stadie, S. Levine, P. Abbeel, Incentivizing exploration in reinforcement learning with deep predictive models, 2015. https://doi.org/10.48550/arXiv.1507.00814.
[42] M. Yuan, B. Li, X. Jin, W. Zeng, Automatic intrinsic reward shaping for exploration in deep reinforcement learning, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 40531–40554.
[43] Y. Yu, X. Si, C. Hu, J. Zhang, A review of recurrent neural networks: LSTM cells and network architectures, Neural Computation 31 (7) (2019) 1235–1270.
[44] K. Cho, B.V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Stroudsburg, 2014, pp. 1724–1734.
[45] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.
[46] J. Koutník, K. Greff, F. Gomez, J. Schmidhuber, A clockwork RNN, in: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, PMLR, New York, 2014, pp. 1863–1871.
[47] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI gym, 2016. https://doi.org/10.48550/arXiv.1606.01540.
[48] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, Thirtieth AAAI Conference on Artificial Intelligence 30 (1) (2016) 2094–2100.
[49] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2019. https://doi.org/10.48550/arXiv.1509.02971.

Mohammadreza Nakhaei received the M.S. degree in mechatronics engineering from University of Tehran, Tehran, Iran, in 2022. In his dissertation, he focused on sequential modeling and intrinsic motivation in reinforcement learning. He is now a Ph.D. candidate at Robot Learning Lab at Aalto University where he develops new decision making and control methods. His research focuses on representation learning for meta-reinforcement learning and the adaptation to previously unseen tasks.

Reza Askari Moghadam received his M.S. degree in control engineering in 2001 and his Ph.D. degree in electronic engineering in 2007. He worked at the Faculty of New Sciences and Technologies, University of Tehran, Iran. He later joined the Department of Electrical Engineering at the University of Paris-Est Créteil (UPEC) in France. Currently, he is an Associate Professor in the Faculty of Science and Engineering at Sorbonne University, France. His research interests include artificial intelligence, sensors and MEMS devices, and he has published several research papers in these fields.