Hierarchical Reinforcement Learning with Options and United Neural Network Approximation: Volume 1

Chapter · January 2019

DOI: 10.1007/978-3-030-01818-4_45

2 authors, including:

Aleksandr I. Panov
Russian Academy of Sciences
35 PUBLICATIONS **163** CITATIONS

SEE PROFILE

Hierarchical Reinforcement Learning with Options and United Neural Network Approximation

Vadim Kuzmin[1] and Aleksandr I. Panov[2,3(✉)]

[1] National Research University Higher School of Economics, Moscow, Russia
[2] Moscow Institute of Physics and Technology, Moscow, Russia
panov.ai@mipt.ru
[3] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia

Abstract. The "curse of dimensionality" and environments with sparse delayed rewards are one of the main challenges in reinforcement learning (RL). To tackle these problems we can use hierarchical reinforcement learning (HRL) that provides abstraction both on actions and states of the environment. This work proposes an algorithm that combines hierarchical approach for RL and the ability of neural networks to serve as universal function approximators. To perform the hierarchy of actions the options framework is used which main idea is to utilize macro-actions (the sequence of simpler actions). State of the environment is the input to a convolutional neural network that plays a role of Q-function estimating the utility of every possible action and skill in the given state. We learn each option separately using different neural networks and then combine result into one architecture with top-level approximator. We compare the performance of the proposed algorithm with the deep Q-network algorithm (DQN) in the environment where the aim of the magnet-arm robot is to build a tower from bricks.

Keywords: Hierarchical reinforcement learning · Options
Neural network · DQN · Deep neural network · Q-learning

1 Introduction

Most of the real life problems where the reinforcement learning (RL) can be used are prone to exponential growth of the size of stored data with the increase of the problem's scale. This is called the "curse of dimensionality". Also it can happen that the feedback from the environment is delayed and sparse which leads to the long sequence of actions the agent needs to perform in order to get it. This makes it harder for the agent to find and learn that long sequence. Hierarchical reinforcement learning (HRL) aims to contend with the described difficulties performing the abstraction on the set of available actions. Botvinick [3] defines HRL as computational techniques that allow RL to support temporally extend

A. Abraham et al. (Eds.): IITI 2018, AISC 874, pp. 453–462, 2019.
https://doi.org/10.1007/978-3-030-01818-4_45

actions. HRL also tries to decompose the given problem into smaller sub-tasks so that their sequential solution is more effective than the attempt to solve the initial task at once. Especially the decomposition of the problem is important in robotic tasks, in which the robot often has to apply certain sequences of actions. In our work we consider a problem from the Brick world, which is close to real robotic tasks.

Nowadays, artificial neural networks (ANNs) is the most effective tool for image recognition and representation. Their ability to extract useful features from the raw data can be used in robotics and RL in particular. Usually the data from robotic system's sensors is an image of the environment's state and based on it the robot is planning its actions. The capability of neural networks to serve as universal function approximators allows to use them as a value function or a Q-function and in this case the network is called the Q-network. The usage of neural network also dispenses with the necessity of storing all the data explicitly which helps with the dimensionality problem.

The aim of this work is to develop an algorithm of hierarchical reinforcement learning that exploits neural networks. To test the algorithm we simulate the work of a magnet arm robot which goal is to build a tower of required height from bricks. We compare its performance with the deep Q-network algorithm (DQN) [6].

2 Related Work

Hierarchical reinforcement learning has been a field of extensive research efforts for more than 20 years and, as with other disciplines, rapid development of ANNs has opened new perspectives and allowed to solve a bigger amount of tasks. The main trend of the last works in this field is building end-to-end solutions with automatic discovery of profitable hierarchy of actions. We will start with the description of fundamental works in HRL and then move to the latest advances in integration of neural networks into the HRL frameworks.

2.1 Hierarchical Reinforcement Learning

Markov decision process (MDP) is the base of a standard RL problem, while in the HRL setup semi Markov decision process (SMDP) is used because it takes into account that the action can take more than one time unit. More precisely, in SMDP the time of the process being in a particular state is a random variable. SMDP is defined as a tuple $<S, A, T, R, \gamma>$, where S and A are the set of states and actions respectively, $T(s', N|s, a)$ - the transition function (N is a number of time units the action a took, $R(s, a)$ - the reward function, γ - the discount factor needed for the process that can take infinite number of steps. To solve an SMDP means to find an optimal policy (π^*) that maps every environment's state $s \in S$ with the action $a \in A$ so as to maximize the cumulative reward.

Three basic frameworks of HRL can be noted, all of them use the notation of SMDP and initially implied that a programmer sets a hierarchy manually.

The first framework is called "options" [8], according to it the agent can choose between not only basic actions, but also macro-actions (skills) that are defined as a triple $<I, \pi, \beta>$, where $I \subseteq S$ - the set of states where the option can be chosen, $\pi : S \times A \rightarrow [0, 1]$ - the option's policy that for every possible state $s \in I$ and basic action a defines the probability of this action to be taken, $\beta : S \rightarrow [0, 1]$ - the option's termination condition. In its first version the framework operates with the preliminary given options with optimal policies. The policy over options can be learned with the standard RL algorithms but adjusted for SMDP. Further we provide the formalized Q-learning algorithm adjusted for options described in [8].

Algorithm 1. Q-learning on Options with ϵ-greedy exploration

Initialize $Q(s, o_i) = 0 \ \forall o_i : s \in I_i, \forall s \in S$;
for *number of episodes* **do**
  reset the environment;
  observe the state s;
  while *episode in not terminated* **do**
    select an option o with ϵ-greedy from $\{o_i : s \in I_i\}$;
    while *o is not terminated* **do**
      follow the option's policy;
      accumulate the reward, time and observe the state s';
      if *s' terminates the episode* **then**
        break
    observe the option's accumulated reward **r**, the next state s' (where the option terminated) and the time the option took **k**;
    $Q(s, o) \leftarrow (1 - \alpha)Q(s, o) + \alpha[r + \gamma^k \max_{o' \in \{o_i : s' \in I_i\}} Q(s', o')]$;
    if *s' terminates the episode* **then**
      break
    s $\leftarrow s'$

Hierarchies of Machines (HAMs) [2,7] is the second basic approach that constitutes of the set of non-deterministic finite-state machines. They are the programs that set restrictions on the actions the agent can perform in a given state. This is different from the options framework where the choice of action enlarges. Every machine is defined as the set of states, transition function and starting function that determines the initial state of the machine. There are 4 types of machines states: action performs a step in the environment, call is needed to lunch the other machine, choice non-deterministically picks the next state, stop terminates the execution of the current machine and returns back to the last call state. With the set of machines it is possible to reach the hierarchy of any depth by calling one machine through another with the call state. With the HAMs the set of possible policies is bounded and in this case the aim of the agent is to find the right sequence of transitions between the machines states. It is proven that the Cartesian product of the set of the environments states and the set of the machines states defines the SMDP solving which is equivalent to solving the

initial MDP. If the Q-learning idea is used then for the HAMs the Q-function update rule is the following:

$$Q([s_c, m_c], a) \leftarrow (1-\alpha)Q([s_c, m_c], a) + \alpha[r_c + \beta_c \max_{a'} Q([t, n], a')],$$

where s_c - state of the environment on a previous step, m_c - state of the machine on a previous step, t and n - current state of the environment and machine respectively, a - action chosen at the previous step, r_c and β_c - cumulative reward and discount from the last choice state respectively.

The third approach, MAXQ [4], performs the decomposition of the value function for the initial MDP into the set of value functions for subtasks, this set has a tree structure where the solution of the root subtask solves the whole MDP. It is implied that a programmer manually chooses the subtasks the agent has to solve. When the initial value function is decomposed the standard algorithms of RL for SMDP can be used.

In the current work we use the options framework to perform the hierarchy as they are easily interpretable, well studied and represent the most natural way of building the hierarchy.

2.2 Approximation with Neural Networks

It is known that an ANN can approximate complex function with the high precision, therefore, it can be used to replace tabular format of value function and Q-function storage. Furthermore, there are several successful applications of neural networks to HRL.

The work [1] operates exactly with the options framework and relies on the policy gradient theorems. A new option-critic (the critic is a part of the algorithm that evaluated the policy) architecture is proposed and it aims to find options automatically learning them in parallel with the policy over options. Under the assumption that functions representing the options policies and their termination functions are differentiable, the stochastic gradient descent (SGD) is used to perform the optimization. In the Arcade Learning Environment (ALE) a deep convolutional neural network is used to approximate intra-option policies, termination functions and policy over options. Presented approach is an end-to-end solution that requires only the number of options to be learned. Shown results confirm the ability to find interpretive options profitable for playing the testing games.

A combination of options and deep Q-networks is presented in [5]. Two-level hierarchy is proposed where meta-controller chooses sub-goals for options and sends it to the controller which tries to reach that goal operating only basic actions, the controller gets the reward from the internal critic. Meta-controller and controller are deep convolutional neural networks that receive image as an input (plus the sub-goal in case of internal controller) and output the Q-function values. They use SGD to train the networks and the Q-learning update rule extended on options. In the beginning of the process only controller is trained, after some time it is followed by the mutual training process. There is an attempt

to automatically discover options, however, in the experiment with the Atari game "Montezumas Revenge" the problem was not solved to the full extent.

The novel recurrent network architecture is introduced in [9]. It allows to learn macro-actions automatically only interacting with the environment. The architecture consists of two modules: the "action-plan" determines the sequence of elementary actions, the "commitment-plan" decides at what moment the macro-action terminates and the "action-plan" is updated. The difference from the previous models is that instead of planning only one next step this network outputs a matrix where every column is a step starting from the current one and every row is a basic action. With such matrix the network builds a plan for some horizon of steps, follows this plan and updates the matrix to change the macro-action. The experiments are conducted on Atari games where the convolutional network is used to decode the state of the environment. The results showed that using the found macro-actions helped to achieve higher game scores. In the task of text generation LSTM network is used instead of convolutional layers, the experiment resulted in frequent n-grams that also proves the correctness of the model.

Thus, in all the reviewed works there is a tendency for using convolutional and recurrent networks and so-called "managers" controlling the hierarchy.

3 Methodology

As it was mentioned before, the options framework is used to implement the hierarchy. The set of options must be determined by the programmer in advance where every option is a separate Q-network pretrained on a training environment with the standard DQN algorithm [6].

The main challenge of using the non-linear approximator for the Q-function is that convergence can not be guaranteed. But there are techniques able to enhance the stability, they are "experience replay" and usage of "target network". The experience replay implies using a special buffer that stores the transition made by the agent (the current state, the action, the next state, reward and the flag of termination) then every time we sample a batch of transitions that is used to perform the optimization with the ADAM optimizer. Such technique ensures the data used for training is not correlated. We need a separate "target network" that has the same architecture as the main one and copies its weight periodically. It is used to produce the target value for computing the error and also provides the stability of the learning process.

Hereafter we provide the algorithm that uses the framework of options and neural networks as approximators for intra policies and the policy over options. Let us introduce the following notation: s - the environment's state, o - the option, r - the reward, Q_ϕ - the Q-function for the main network, $Q_{\phi'}$ - the Q-function for the target network, ϕ, ϕ' - the weights of the main and target network respectively, y - the target value for the $Q_\phi(s, a)$. See the Algorithm 2.

Algorithm 2. DQN with options and ϵ-greedy exploration

Data: environment, Q_ϕ - network for the Q-function, α - learning rate, γ - discount factor, replay buffer size, batch size, N - frequency of updating the target network, $O = \{o_i\}_{i=1}^{n} =< I_i, \pi_i, \beta_i >_{i=1}^{n}$ - set of options;
reset the environment and observe the state s_i;
last observation := s_i;
for *number of steps* **do**
  save to the buffer the last observation;
  select an option o_i with ϵ-greedy from $\{o_i : s \in I_i\}$;
  while o_i *is not terminated* **do**
    follow the option's policy;
    accumulate the reward, time and update the state s';
    if s' *terminates the episode* **then**
      $done_i$:= True ;
      break
  observe the option's accumulated reward (r_i), the next state (s'_i) (where the option terminated) and the time the option took (k_i);
  save the effect $(o_i, r_i, r_i, done_i, k_i)$ in the state s_i to the buffer;
  last observation := s'_i;
  Perform the training:;
  sample batch $(s_j, o_j, s'_j, r_j, done_j, k_j)$;
  compute the target value: $y_j = r_j + (1 - done_j)\gamma^{k_j} \max_{o'_j} Q_{\phi'}(s'_j, o'_j)$;
  update the wights of the main network using the following gradient:
  $\phi \leftarrow \phi - \alpha \sum_j (\frac{dQ_\phi}{d\phi}(s_j, o_j)(Q_\phi(s_j, o_j) - y_j))$;
  update the weight of the target network every N steps: $\phi' \leftarrow \phi$

4 Experiments

The goal of the experiments is to verify the convergence of the algorithm, check how successfully it overcomes the "curse of dimensionality" and the problem of sparse delay reward.

4.1 Experimental Environment

As a testing environment we have chosen a model of a magnet arm robot that can grab and move bricks. The base step is to mathematically describe the environment, the agent and their interplay, which can be done by defining an MDP. There is a two dimensional space where the agent can perform 6 basic actions: one move for each direction and turning the magnet on or off. The state is represented by the two dimensional matrix where brick is coded by one, turned on magnet - by two, turned off magnet - by three, empty cell - by zero. Such matrix can be used to produce an RGB image representing the state (Fig. 1). The goal of the agent is to build a tower from bricks, the required height is specified. At the beginning of the episode the agent is placed at the top left cell of the grid and the bricks are places from the right to the left. The reward function is set

the way that the agents gets a small negative reward for every action and a large positive reward only when the tower of required height is built and the magnet is off. The built MDP is deterministic as all the transitions between states are executed with the probability equal to one.

Figure 1 represents three different states of the environment with the size 4×3 pixels and 2 bricks. The left image is the starting position, the middle one shows that the magnet is on and the brick is being moved, the right one is the finish state when the tower is built and the magnet is off.

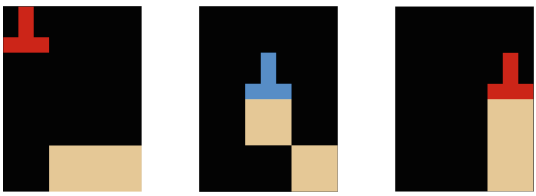


Fig. 1. Magnet arm environment example. Yellow squares are blocks, a red inverted T is an arm with working magnet, a blue inverted T is an arm with idle magnet.

The chosen environment is easily scalable as it is possible to change the size of the grid, the amount of bricks and the required height of the tower. This will allow us to understand how the learning time and convergence changes with the growth of the problem's size. As the reward function is set to give positive feedback only when the tower is built the problem has delayed feedback. This task is also relevant because it can be modelled in 3D to test a real robot.

4.2 Experimental Details

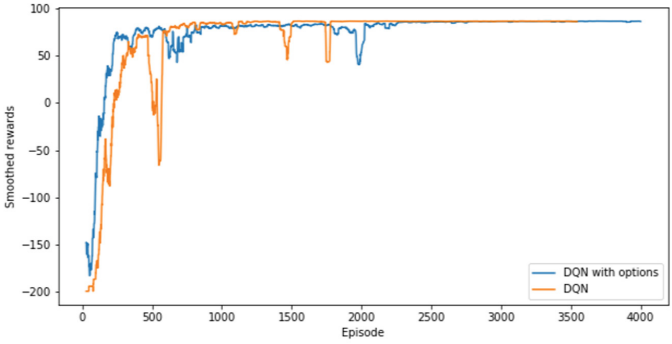
With the described method any amount of options can be utilized and basic actions are treated as options that take exactly one time unit. We have implemented two options, the first one makes the robot to go down and reach the brick or the bound and the second one makes the robot to go down, grab the brick and lift it to the top. Both of this options can be used for different manipulations with bricks.

To train the options' Q-networks we use separate training environments. The first difference of the training environment from the experimental one is their termination condition that tells that the episode is over when the condition of the option is fulfilled. The second difference is that in every new episode the position of the bricks is arbitrary and the agent can be anywhere in the top raw of the matrix. This will allow options to be flexible enough and be initiated from any environment's state.

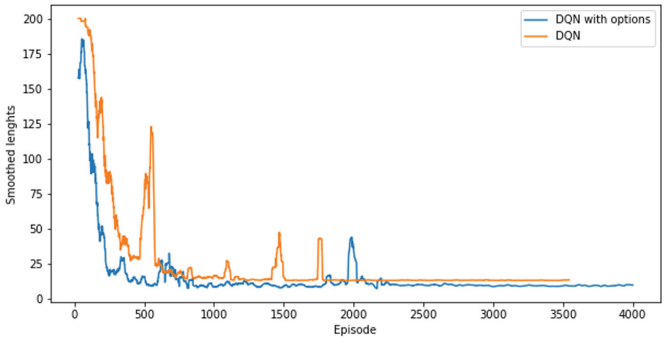
In the experiments we use the buffer with the size of 1 million transitions, the learning starts after 5500 steps and the target network is updated every 200

steps. The baseline is the DQN algorithm that does not include any hierarchy. The neural network architecture is similar to what have been used in [6] for playing Atari games. The input to the network is an RGB image of varying size depending on a problem. The network has three convolutional layers followed by two fully connected layers where the last one has as many neurons as there are actions in the environment including options. The convolutional layers have 32, 64 and 64 neurons respectively with the kernel size 8, 4 and 3 respectively, and the stride 2, 4 and 1 respectively.

We are testing out algorithm on different sizes of the problem starting with the 4×3 grid, 3 cubes and the required height equal to 3. Then move to a larger 6×4 grid with 4 bricks where the robot has to build a tower from all the bricks.



(a)

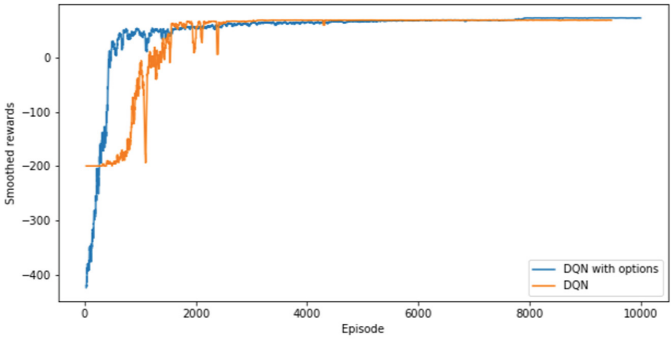


(b)

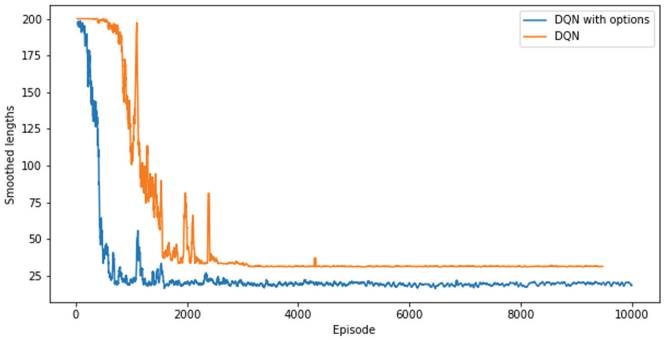
Fig. 2. Learning process in the environment with the size 4×3 and 3 bricks

4.3 Results

The graphs represent the smoothed curves of the learning process. The curves show how the episode's reward or length was changing. The Fig. 2 is the case for the first experimental setup. It can be seen that the algorithm exploiting



(a)



(b)

Fig. 3. Learning process in the environment with the size 6×4 and 4 bricks

options finds the right solution faster, gets the positive reward earlier and needs less actions (the option counts as one action). However, the 4×3 environment is not large enough to see the profit of the options. In this case the agent can handle the process of building a tower without options and the problem of the delayed reward is not that challenging.

The larger problem helps to show that the options can give noticeable acceleration of the learning process. The Fig. 3 show the results for the 6×4 environment with 4 bricks. Here the usage of options allows to outperform the standard DQN significantly. The agent with the options starts to reach the positive episode reward in double-quick time and takes less actions to finish the episode (almost half as much actions as the agent without options).

5 Conclusion and Future Work

Taking into account the derived results it can be stated that using options the agent explores the environment more efficiently and can find the profitable plan of actions faster. The proposed algorithm exploiting the combination of neural networks and the HRL can be used to tackle the "curse of dimensionality" and

delayed rewards. There is also a strong possibility of transfer learning, because the learned options are not specific to any problem in the environment with the magnet arm and bricks.

There is a minor drawback connected to the experimental environment that can lead to the agent ending up without using the options. As the agent gets a small negative reward, taking an option results in a larger negative cumulative reward which leads to the penalty for taking the option. To overcome this issue, we can either change the reward function or eliminate the basic actions being sure that the options are enough to build the tower.

It is evident that because of the same architecture of all the networks lower layers can be shared between them. Therefore, the next step is to develop an architecture that will combine intra policies with the policy over option, joining the network's architectures. Another possible direction of the future work is finding a way of automatic discovery of sub-goals for options.

Acknowledgements. This work was supported by the Russian Science Foundation (Project No. 18-71-00143).

References

1. Bacon, P.-L., Harb, J., Precup, D.: The Option-Critic Architecture. arXiv:1609.05140v2 (2016)
2. Bai, A., Russell, S.: Efficient reinforcement learning with hierarchies of machines by leveraging internal transitions. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. Main track, pp. 1418–1424 (2017)
3. Botvinick, M.M.: Hierarchical reinforcement learning and decision making. Curr. Opin. Neurobiol. **22**, 956–962 (2012)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. arXiv:cs/9905014 (1999)
5. Kulkarni, T.D., Narasimhan, K.R., Saeedi, A., Tenenbaum, J.B.: Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. arXiv:1604.06057 (2016)
6. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529533 (2015)
7. Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference. MIT Press, Cambridge (1998)
8. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. In: Artificial Intelligence (1999)
9. (Sasha) Vezhnevets, A., et al.: Strategic attentive writer for learning macro-actions. In: Proceedings of NIPS (2016)