Chapter 10
A Review of Deep Reinforcement Learning Algorithms and Comparative Results on Inverted Pendulum System

Recep Özalp, Nuri Köksal Varol, Burak Taşci, and Ayşegül Uçar

Abstract The control of inverted pendulum problem that is one of the classical control problems is important for many areas from autonomous vehicles to robotic. This chapter presents the usage of the deep reinforcement learning algorithms to control the cart-pole balancing problem. The first part of the chapter reviews the theories of deep reinforcement learning methods such as Deep Q Networks (DQN), DQN with Prioritized Experience Replay (DQN+PER), Double DQN (DDQN), Double Dueling Deep-Q Network (D3QN), Reinforce, Asynchronous Advanced Actor Critic Asynchronous (A3C) and Synchronous Advantage Actor-Critic (A2C). Then, the cart-pole balancing problem in OpenAI Gym environment is considered to implement the deep reinforcement learning methods. Finally, the performance of all methods are comparatively given on the cart-pole balancing problem. The results are presented by tables and figures.

Keywords Deep Q networks · Deep Q-network with prioritized experience replay · Double deep Q-network · Double dueling Deep-Q network · Reinforce · Asynchronous advanced actor critic asynchronous · Synchronous advantage actor-critic · Inverted pendulum system

R. Özalp · N. K. Varol · A. Uçar (✉)
Department of Mechatronics Engineering, Firat University, 23119 Elazig, Turkey
e-mail: agulucar@firat.edu.tr

R. Özalp
e-mail: rozalp@firat.edu.tr

N. K. Varol
e-mail: 171134109@firat.edu.tr

B. Taşci
Vocational School of Technical Sciences, Firat University, 23119 Elazig, Turkey
e-mail: btasci@firat.edu.tr

G. A. Tsihrintzis and L. C. Jain (eds.), *Machine Learning Paradigms*, Learning and Analytics in Intelligent Systems 18,
https://doi.org/10.1007/978-3-030-49724-8_10

10.1 Introduction

In recent years, the modern control engineering has witnessed to significant developments [1]. PID control is known as the state of the art but it has been still commonly using in the industry. Model predictive control, the state feedback control, adaptive controller, and optimal controller have been developed [1, 2]. Most of these controllers require system dynamics and system modeling. Therefore, the control structure that do not require the system model and the inverse model are searched. The reinforcement learning methods have been shown to be a promising intelligence alternative ones to classical control methods [1, 2].

Reinforcement learning is the third stream of machine learning methods which allow to the agent to find the best action from its own experiences by trial and error [3–7]. Unlike supervised and unsupervised learning, the data set is labeled inexplicitly by reinforcement signal. Thanks to self-learning property of agent, the reinforcement learning methods have been increasing its importance day by day. At many areas such as production control, game theory, finance, communication, autonomous vehicles, robotic, the reinforcement learning methods have been successfully applied [8–15].

Literature includes many reinforcement methods such as tabular Q-learning, Sarsa, actor-critic methods, and policy gradient [4]. After DeepMind introduced AlphaGo, at the game of Go, the reinforcement learning algorithm beat the professional Go players Lee Sedol with the score of 4-1 in 2016 [16], many new reinforcement learning methods based on the Deep Neural Networks (DNNs) have appeared [17–21]. Convolutional Neural Networks (CNNs) and Long Short-Term Memory Network (LSTM) that are the variants of DNNs use large data amounts [22–28]. For example, object detection and recognition algorithms are trained using millions of images such as ImageNet [29]. On the other hand, the methods don't use the hand-crafted features that is difficult to determine at high dimensional state space or don't use any feature extraction method. The advantages increased the application range of the conventional reinforcement learning. The best successful applications consist of the control of mobile and humanoid robots including the sensors such as camera, LIDAR, IMU, and so on [30–36].

In this chapter, Deep Q Networks (DQN) [17, 18], DQN with Prioritized Experience Replay (DQN+PER) [37], Double DQN (DDQN) [20], Double Dueling DQN (D3QN) [22], Reinforce [38], Asynchronous Advanced Actor Critic Asynchronous (A3C) [39] and synchronous Advanced Actor Critic Asynchronous (A2C) [39, 41] are applied on the cart-pole balancing problem on OpenAI GYM environment [42]. The performance of the deep reinforcement learning algorithm in the problem are compared. Some control parameters such as rise time, setting time, max return, and mean reward are evaluated for comparing aim.

The chapter is organized as follows. In Sect. 10.2, the fundamentals of reinforcement learning are introduced and followed by the theory of the deep reinforcement learning algorithms. Section 10.3 describes the cart-pole balancing problem. The results of the simulation with the discussions are given in Sect. 10.4. Section 10.5 summarizes the conclusions.

10.2 Reinforcement Learning Background

Reinforcement learning is designed by using Markov Decision Processes (MDP) described in Sect. 10.2.1 [3, 4]. An agent learns its behavior by trial and error rule and bys interacting with the environment. Agent selects an action of a_t according to the s_t state at time t and then receives the r_t feedback reward signal corresponding to this action. It starts to receive the next state, s_{t+1}. An illustration of this general structure is given in Fig. 10.1.

10.2.1 Markov Decision Process

An MDP is a discrete time stochastic control process for optimizing decision-making under uncertainty [3]. In an MDP, the environment dynamics are defined as the transition probability distribution from one state s to the next state s' after taking a below as:

$$\begin{aligned} &p\big(s_{t+1} = s', r_{t+1} = r|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_1, a_1, r_1, s_0, a_0\big) \\ &\quad = p\big(s_{t+1} = s', r_{t+1} = r|s_t, a_t\big). \end{aligned} \tag{10.1}$$

The expression which is depend on only the present state and action is known as Markov Property. Transition probabilities give the environment information as:

$$\sum_{s_{t+1}\in S}\sum_{r\in R} p(s_{t+1}, r_{t+1}|s_t, a_t) = 1, \forall s_t \in S, a_t \in A \tag{10.2}$$

$$p(s_{t+1}, |s_t, a_t) = \sum_{r\in R} p(s_{t+1}, r_{t+1}|s_t, a_t) \tag{10.3}$$

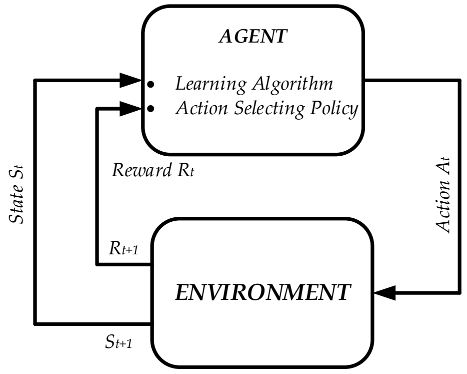

Fig. 10.1 A general reinforcement learning structure [4]

$$r(s_t, a_t) = \sum_{r \in R} r \sum_{s_{t+1} \in S} p(s_{t+1}, r_{t+1} | s_t, a_t). \tag{10.4}$$

An MDP is a five-tuple of S, A, P, R, where S is the space of possible states, A is the space of possible actions, P: S $\times$ A$\times$S $\rightarrow$ [0,1] is the transition probability from one state s_t to the next state s_{t+1}, R: S $\times$ A $\rightarrow \mathbb{R}$ is the reward function specifying the reward r as taking an action from one state s to the next state s_{t+1}, and $\gamma \in [0, 1)$ is an exponential discount factor.

The goal of agent is to maximize the total discounted accumulated return from each state s_t as below:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{10.5}$$

Thanks to the discount factor γ, it is given more importance to short-term rather than long-term reward [4]. γ is chosen about zero to give the significance to the rewards received in the near future while γ is chosen about one for the future rewards.

A policy π is defined as a strategy to choose an action given a state s_t. A deterministic policy is a function converting a state to an action while a stochastic one is a distribution mapping to probabilities actions according to the state.

The expected return value relating to a state s_t under a policy π is given by a value function $V_\pi(s)$

$$V_\pi(s) = \mathbb{E}[R_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right]. \tag{10.6}$$

The value function in (10.5) is easily estimated by using dynamic programming [3] in terms of the form:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \\ &= \sum_{a \in A} \pi(s, a) \sum_{s' \in S} P_{ss'}^a \left[R_{ss'}^a + \gamma V_\pi(s') \right]. \end{aligned} \tag{10.7}$$

Using the Bellman equation [5], the optimal state-action value function, Q-function, following the policy π over both the states and the actions are defined as similar to (10.7) as below:

$$\begin{aligned} Q_\pi(s, a) =& \mathbb{E}_\pi [R_t | s_t = s, a_t = a] \\ =& \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right] \end{aligned}$$

$$= \sum_{s' \in S} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V_{\pi}\left(s'\right) \right]. \tag{10.8}$$

Given the transition probabilities and reward values, the optimal Q-function can be solved recursively by using dynamic programming algorithms [3]. One of the algorithms is the value iteration. In the value iteration, Q-values are first estimated in (10.8) and then the value function is updated by maximizing the Q-value by $V_{\pi}(s_t) = \max_{a_t} Q_{\pi}(s_t, a_t)$. Until the variation between the former and present values of the value function is small for all states, the algorithm is run and the optimal policy is determined [4].

The optimal policy is determined by using dynamic programming methods thanks to the recursive value function. On the other hand, when the transition probabilities and reward values are not previously known, in order to determine to be taken which the action in a state the reinforcement learning is used. In the reinforcement learning, the agent interacts with the environment taking different actions in a state and approximates a mapping from states to the actions.

The reinforcement learning consists of model-based and model free methods. Model-based methods generate a model based on the reduced interactions with the environment whereas model-free methods directly learn from the experiences by via of the interaction with the environment. It does not need to the models of the transition probability and reward function. Q-learning is a kind of model-free reinforcement learning algorithm [6]. For each state and action pair, a Q-value is iteratively estimated as:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \eta \left[r_t + \gamma \left[\max_{a'} Q_t\left(s_{t+1}, a_{t+1} | s_t = s, a_t = a\right) \right] - Q_t(s_t, a_t) \right] \tag{10.9}$$

where η is learning rate.

10.2.2 Deep-Q Learning

Tabular Q-learning generates Q-matrix using state-action pairs. The algorithm provides fine results in not large environments and continuous state-action pairs. In fact, the Q-learning algorithm is considered as the problem of approximation of Q-function at the continuous states. Generally, the feed-forward neural networks (FNNs) [7, 43] are used for constructing Q-function and the gradient descent methods are used at the training stage of the FNNs for finding the optimum solution to this problem. However, FNNs are not sufficient at complex environments with large dimensional and discrete or continuous state-action. Hence, to get rid of this disadvantage, DQNs were proposed [17, 18]. DQNs are constructed by using DNN. CNNs and LSTMs that are kinds of DNNs, were successfully used at many application

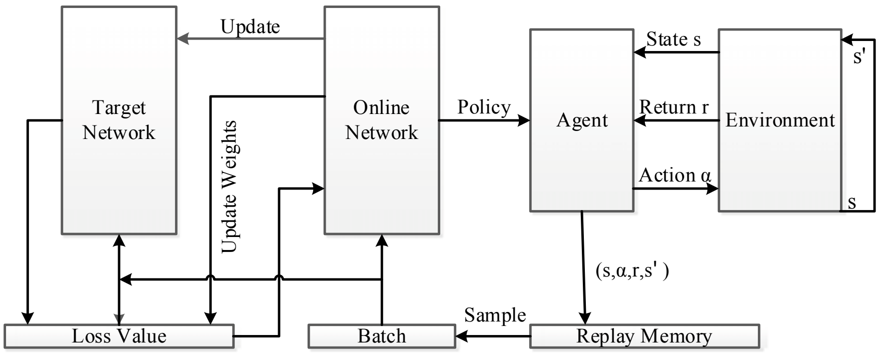

Fig. 10.2 The general framework of DQN

areas without time-consuming feature extraction processing for both large image and sensor inputs [25–36, 44].

Figure 10.2 shows the principles of DQN. In DQN, the network is trained by using the gradient descent method similar to supervised learning [7]. However, the targets are previously unknown. Hence, in order to generate the targets, the network parameters from (i-1) th iteration are first fixed and the target is then approximated by means of the discounted value of the estimated maximum Q-value using the fixed parameters for next state-action pair and the addition of the current reward.

The target is defined as

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}; w_{i-1}) \tag{10.10}$$

where w_i are the network parameters at ith iteration.

The loss function to be optimized is defined as

$$L_i(w_i) = \mathbb{E}_\pi\big[(y_t - Q(s_t, a_t; w_i))^2\big] \tag{10.11}$$

where y_t and $Q_t(s_t, a_t; w_i)$ mean target network output and main network output, respectively. Similar to the supervised learning, the gradient of the loss function is calculated as

$$\nabla_{w_i} L_i(w_i) = \mathbb{E}\big[(y_t - Q(s_t, a_t; w_i))\nabla_{w_i} Q(s_t, a_t; w_i)\big]. \tag{10.12}$$

In DQN, the exploration defines that agent tries out all actions in which previously unknown at the start of training stage. On the other hand, the exploitation express that agent uses the chosen action. If it is maximized discounted total accumulated return, it is called as the acting greedily. DQN is an off-policy method as it learns a greedy policy, i.e., it selects the highest valued action in each state. When an action is re-realized many times, it is expected that the reward for a state-action pair decreases and DQN prefers the action tried the least frequently. However, this exhibits the case of missing some better actions and there is no exploration. This is called exploration-exploitation dilemma. There are two exploration methods as direct and indirect. The direct exploration techniques use the past memory while the indirect ones don't use them. Direct methods have scale polynomials with the size of the state space while undirected method scales in general exponentially with the size of the state space at finite deterministic environments. In direct methods, Shannon information gain is maximized by exploration.

Direct exploration is not used generally in the model-free reinforcement with high dimensional state space [4]. Famous indirect approaches are ϵ-greedy and softmax exploration which is also called as Boltzmann exploration. The ϵ in ϵ-greedy is an adjusted to a constant determining the probability of taking a random action. At the beginning of the training stage, ϵ value is initialized with large value for exploration and then is generally annealed down to near to 0.1 for exploiting. The Boltzmann method uses an action with weighted probabilities instead of taking random action or optimal action. Boltzmann softmax equation is

$$P_t(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_i^n \exp(Q_t(i)/\tau)} \tag{10.13}$$

where τ is the temperature parameter controlling spread of distribution of softmax function. At the beginning of the training stage, all actions are treated equally and then sparsely distributed as in (10.13).

In DQN, the neural networks cause to some instability problems due to the correlations of consecutive observations and small changes between Q-value. In [17, 18], two solutions were proposed: experience replay and frozen target network. In experience replay solution, the generated experience $E_{t=}(s_t, a_t, r_t, s_{t+1})$ by agent is stored in replay memory over many episodes. Mini batches samples are randomly drawn from the replay memory for training the DQN. This achieves uncorrelated samples and updates with small variance.

The updating of parameters at every time step causes lead to oscillations or obstructions in learning. The frozen target network solution [18] proposes that the weights of the target network has kept frozen at most of the time as the networks of the policy network are updated. Later, it is updated to approximate the policy network.

10.2.3 Double Deep-Q Learning

DQN uses same Q values for both selecting and evaluating an action [20]. This may exhibit overestimation of action values generating positive bias at the cases such as inaccuracies and noise. DDQN was proposed to alleviate the limitation of DQN [20]. DDQN uses two separate networks to select and evaluate the action. In Double

DQN, first the action is selected by maximization operation at the online network. Second, Q-value for the selected action are estimated at the target network. The target function of DDQN is written similar to target function in (10.7) of DQN

$$y_t = r_t + \gamma Q_t\left(s_{t+1}, \max_{a_{t+1}} Q_t\left(s_{t+1}, a_{t+1}; w_t^V\right); w_t^T\right) \tag{10.14}$$

where w_t^V and w_t^T are the parameters of value network and target network, respectively.

DQN or DDQN uses an experience replay uniformly sampled from the replay memory. There is not the order of the significance of experience. To give more importance to some experiences, the Prioritized Experience Replay (PER) are proposed in [37]. Some important transitions are replayed. Moore and Atkeson [45] used PER to get rid of the bias and to increase the performance. A solution for PER is given as

$$p = (error + e)^a \tag{10.15}$$

where an error is mapped to a priority. $e > 0$ is a constant and a is in the range [0, 1] and assign with the importance to the error, 0 means that all samples is with equal priority.

10.2.4 Double Dueling Deep-Q Learning

D3QN was proposed to obtain faster convergence than DQN [21]. This architecture has one bottom casing structure and then are separated into two streams for estimating the state value function $V(s_t)$ and the associated advantage function $A(s_t, a_t)$ as in Fig. 10.3. The bottom casing structure is constructed like to DQN and DDQN except for the last fully connected layer. The Q-function is estimated combining two streams [21, 33, 46].

$$Q\left(s_t, a_t; w, w^V, w^B\right) = V\left(s_t; w, w^V\right) + A\left(s_t, a_t; w, w^A\right) \tag{10.16}$$

where w are the parameters of bottom casing structure and w^V and w^A are the parameters of the advantage and value streams with fully connected layers. The Q-function is estimated as

$$Q\left(s_t, a_t; w, w^V, w^B\right) = V\left(s_t; w, w^V\right) + \left(A\left(s_t, a_t; w, w^A\right) - \max_{a_{t+1}} A\left(s_t, a_t; w, w^A\right)\right) \tag{10.17}$$

In order to achieve better stability, the maximum is rescaled with average,

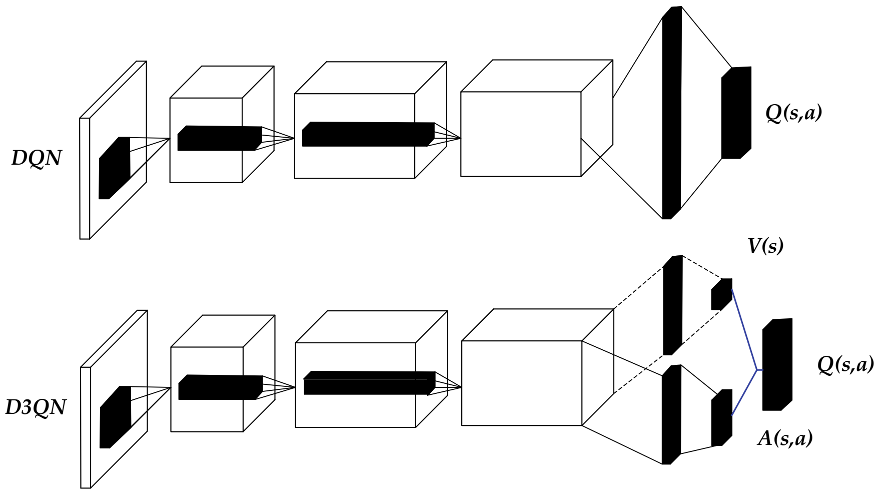

Fig. 10.3 DQN and D3QN network structure [21]

$$Q\left(s_t, a_t; w, w^V, w^B\right) = V\left(s_t; w, w^V\right) + \left(A\left(s_t, a_t; w, w^A\right) - \frac{1}{|\mathcal{A}|}A\left(s_t, a_t; w, w^A\right)\right) \tag{10.18}$$

where $|\mathcal{A}|$ is the number of actions.

10.2.5 *Reinforce*

In contrast to the value-function based methods represented by Q-learning, Policy search methods are based on gradient based approaches [38]. The methods use the gradient of the expected return J for updating the parameters w_i of stochastic policy $\pi^w\left(s_{k,}a_k; w\right)$.

$$w_i = w_i + \eta\nabla_w J \tag{10.19}$$

Reinforce is one of the methods to estimate the gradient $\nabla_w J$ [38]. In this method, likelihood ratio is defined as:

$$\nabla_w B^w(l) = B^w(l)\nabla_w log B^w(l)\ J \tag{10.20}$$

where $B^w(l)$ is the distribution of parameters w.

The expected return for the parameter set w is written as:

$$J^w = \sum_l B^w(l)R(l) \tag{10.21}$$

where $R(l)$ is the return in episode l. $R(l) = \sum_{k=1}^{K} a_k r_k$ where a_k is a weighting factor at the step k. Generally, a_k is selected the equal to the discount factor in (10.5) or 1/K.

$$\nabla_w J^w = \sum_l \nabla_w B^w(l)R(l) = \sum_l B^w(l)\nabla_w log B^w(l)R(l) = \mathbb{E}\{\nabla_w log B^w(l)R(l)\} \tag{10.22}$$

The distribution over a stochastic policy $\pi^w\left(s_{k,}a_k\right)$ is calculated:

$$B^w(l) = \sum_{k=1}^{K} \pi^w\left(s_{k,}a_k\right) \tag{10.23}$$

$$\nabla_w \mathrm{log} J^w = \mathbb{E}\left\{\left(\sum_{k=1}^{K} \nabla_w log\pi^w\left(s_{k,}a_k\right)\right)R(l)\right\}. \tag{10.24}$$

In the application, b_k called as baseline is subtracted from the return term and the result gradient is obtained as:

$$\nabla_w log J^w = \mathbb{E}\left\{\left(\sum_{k=1}^{K} \nabla_w log\pi^w\left(s_{k,}a_k\right)\right)(R(l) - b_k)\right\}. \tag{10.25}$$

10.2.6 Asynchronous Deep Reinforcement Learning Methods

The actor-critic reinforcement algorithms generally consist of two parts [39, 40]. The actor part presents the policy $\pi\left(a_t|s_t; w'_a\right)$ and the critic part presents the value function, $V(s_t; w_c)$ in which w'_a and w_c are the set of the parameters of the actor and the critic, respectively. The actor is responsible of providing the probalities for each action as the critic is responsible of the value for the state. As in Fig. 10.4, the agent chooses an action a_t in state s_t and the behaviour is then reinforceded if the error of the value function is positive.

A3C uses multiple agents. Each agent has its own parameters and a copy of the environment. Asynchronous in A3C emphasis for multiple agents to asynchronously execute in parallel. These agents interact with their own environments. Each of them is controlled by a global network. In A3C, the advantage is defined as

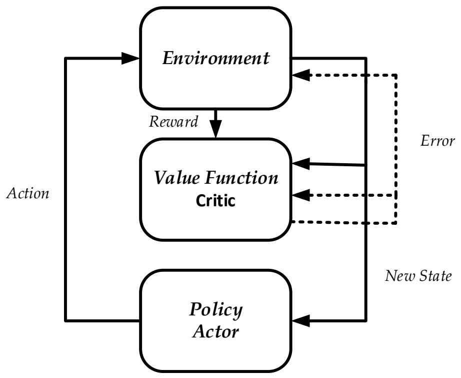

Fig. 10.4 Actor-Critic reinforcement model

$$A(a_t, s_t) = R_t - V(s_t) \tag{10.26}$$

where R_t is the total return at time step t and $V(s_t)$ is the value function called as the baseline. The critic is optimized with respect to the loss:

$$Loss = (R_t - V(s_t; w_c))^2. \tag{10.27}$$

The actor is trained using the gradient in (10.28) including the addition of is the entropy of the policy π, $H\left(\pi\left(s_t; w'_a\right)\right)$ to improve exploration

$$\nabla_{w'_a}\log\pi\left(a_t|s_t; w'_a\right)(R_t - V(s_t; w_c)) + \beta\nabla_{w'_a}H\left(\pi\left(s_t; w'_a\right)\right) \tag{10.28}$$

where $log\pi\left(a_t|s_t; w'_a\right)$ means the log probability for taking an action a_t in the state s_t following the policy π and β is a hyperparameter controlling the strength of the entropy regularization.

Since A3C applies the bootstrapping, the variance is reduced and learning accelerates. A3C is applied at both continuous and discrete action spaces. The feedforward and recurrent network structures can be used in A3C. Generally, A3C uses a CNN with one softmax output and one linear output for the actor and for value function, respectively.

A2C is a synchronous, deterministic kind of A3C [40]. As in Fig. 10.5, in A2C, each agent previously complete its episode and then the policy and value function network are updated. A2C can work more effectively on GPUs and faster than A3C.

10.3 Inverted Pendulum Problem

In this chapter, the problem of the inverted pendulum in the classical control theory is considered. The system consists of a pole anchored on a cart moving in two dimension. The cart can move left or right on a flat surface. The aim of the control

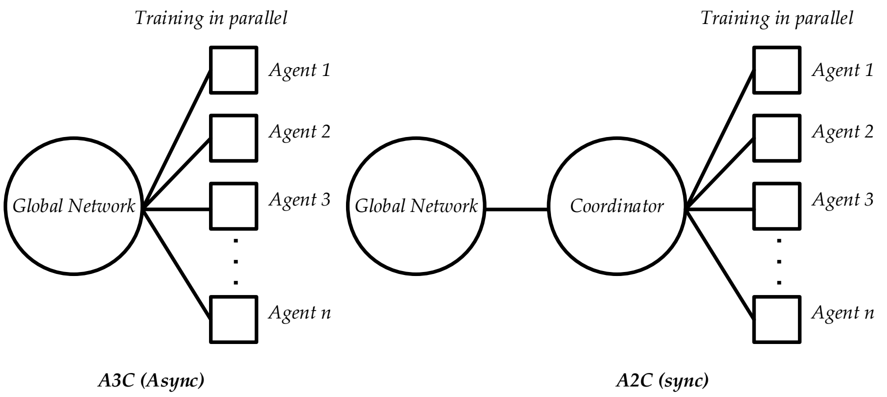

Fig. 10.5 The structure of A3C versus A2C [47]

problem is to balance on the upright position on its mass center of cart the pole. The system is a nonlinear due to the gravity and unstable. Hence, its control is difficult. In order to balance the pole on the cart, the cart has to move by applying a force to the of left and right side of equilibrium point. Figure 10.6 illustrates a setup of the inverted pendulum problem and its parameters.

The system in Fig. 10.6 has the state-space form in

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ \frac{m\left(g\cos(x_2) - l x_4^2\right)\sin(x_2) + u}{M + m\sin^2(x_2)} \\ \frac{g\sin(x_2)}{l} + \frac{\cos(x_2)}{l}\frac{m\left(g\cos(x_2) - l x_4^2\right)\sin(x_2) + u}{M + m\sin^2(x_2)} \end{bmatrix}, \tag{10.29}$$

where $x = \left(x_1\ x_2\ x_3\ x_4\right)$ is the cart position, pendulum angle, cart velocity, and pendulum velocity, M and m mean the masses of the cart and pendulum, respectively, g is the gravitational acceleration, and u is the external force.

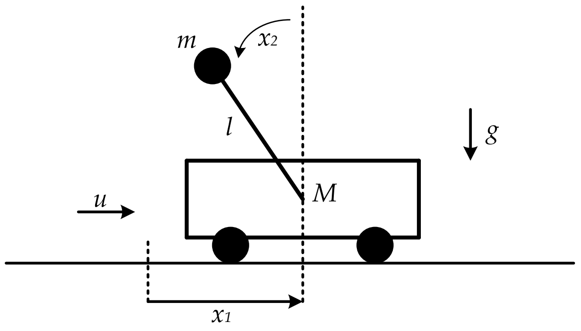

Fig. 10.6 The system of the inverted pendulum [48]

The reinforcement learning does not need to system model or inverse model for control.

10.4 Experimental Results

In this chapter, we used “Cartpole-v1” from OpenAI Gym environment [42]. In this environment, g, M, m, l was determined as 9.8, 1.0.1. 0.5. In the cart-pole balancing problem, it is applied two forces or actions with the magnitude of -1 and 1. The maximum duration of episode is 500.

Each episode is finalized when the pole passes more than 15 degrees from the pivot point, the pole falls down or when the cart moves more than 2.4 units from center. If the cart position and pole angle are not violated, the reward is 1 and the otherwise the reward is 0. We generated all deep reinforcement learning algorithms using Keras and Tensorflow in Python. Especially, the performance of the networks depends on an appropriate setting of hidden layer neuron number. There is no rule for the parameter setting. In this chapter, taking into consideration, we evaluated the network performances of hidden layer neuron number of 24, 32, and 64 since we observed that the training time is too long for bigger numbers and moreover it doesn’t increase the performance also. All network structure and the other hyper parameters of (DQN, DQN with PER, DDQN), D3QN, and (Reinforce, A3C, and A2C) are given in Tables 10.1, 10.2, 10.3, 10.4, 10.5, respectively. In the networks, the optimization method was selected as Adam [49]. Mean Square Error (MSE) and CCE (CSE) were used as the loss function [49]. Rectifier Linear Units (Relu) and softmax activation functions were used [49].

In the experiments, all deep reinforcement learning network structures were evaluated on the cart-pole balancing problem. All experiments were repeated 10 times and their average learning curves were obtained. In order to further evaluate the obtained results, the following performance statistics such as mean return, maximum return, rise time, and settling time were used. In the control performance criteria, the rise time is defined as the episode number required to rise from 10 to 90% of the steady state value, whereas the settling time is defined as the episode number required for the steady state value to reach and remain within a given band.

The results of the networks are presented in Table 10.6. The biggest value of mean return was obtained as 207.12 by using DDQN. On the other hand, the smallest values of rise time and settling time were obtained as 40 and 80 by using DDQN. It was observed that DDQN was sufficient for cart-pole balancing problem in that training speed and the received reward. It is also seen from the figures that the second winner of the race is DQN in that mean reward value, the rise time and the settling time. Figures 10.7 and 10.8 show the learning curves of the networks. The figures show that the obtained maximum return is 500 and all the networks reached to 500. In addition, the learning curves of DQN and DDQN has not oscillation and reached to steady-state value in a fast way.

Table 10.1 The network structure and hyperparameters of DQN, DQN with PER, and DDQN

Hyperparameter	Value
Discount factor	0.99
Learning rate	0.001
Epsilon	1.0
Epsilon decay	0.999
Epsilon min	0.01
Batch size	64
Train start	1000
Replay memory	2000
a	0.6
e	0.01
Network structure	
Hidden layer	2
Hidden activation	Rectifier Linear Units
Output activation	Linear
Output number	2
Optimizer method	Adam
Loss	MSE

Table 10.2 The networks structure and hyperparameters of Reinforce

Hyperparameter	Value
Discount factor	0.99
Learning rate	0.001
Network structure	
Hidden layer number	2
Hidden activation	Relu
Output activation	Softmax
Output number	2
Optimizer method	Adam
Loss	CCE

In the experiments, it was observed that the performance of A3C and A2C especially affected from the weight initialization and provided different results at each run although it was tried the different weight initialization methods such as Xavier and He Normal [49]. As can be seen from Table 10.6, the best hidden neuron number of DQN, DQN with PER, DDQN, D3QN, Reinforce, A3C, and A2C were obtained as 32, 24, 32, 32, 64, 32, and 64.

Table 10.3 The networks structure and hyperparameters of D3QN

Hyperparameter	Value
Discount factor	0.95
Learning rate	2.5e-4
Epsilon decay	0.99
Replay memory	2000
Tau	1
Network structure	
Hidden layer	2
Hidden activation	Relu
Output activation	Linear
Output number	2
Optimizer method	Adam
Loss	MSE

Table 10.4 The networks structure and hyperparameters of A3C

Hyper parameter	Value
Discount factor	0.99
Actor learning rate	0.001
Critic learning rate	0.001
Threads	8
Network structure of actor part	
Hidden layer number	2
Hidden activation	Relu
Output activation	Softmax
Output number	2
Optimizer method	Adam
Loss	CCE
Network structure of critic part	
Hidden layer/Node number	2
Hidden activation	Relu
Output activation	Linear
Output number	1
Optimizer method	Adam
Loss	MSE

10.5 Conclusions

In this chapter, the control problem of cart-pole balancing was taken into consideration. The main aim of the proposed chapter is to review deep reinforcement learning

Table 10.5 The networks structure and hyperparameters of A2C

Hyperparameter	Value
Discount factor	0.99
Actor learning rate	0.001
Critic learning rate	0.005
Network structure of actor part	
Hidden layer number	2/24
Hidden activation	Relu
Output activation	Softmax
Output number	2
Optimizer method	Adam
Loss	CCE
Network structure of critic part	
Hidden layer/node number	2
Hidden activation	Relu
Output activation	Linear
Output number	1
Optimizer method	MSE

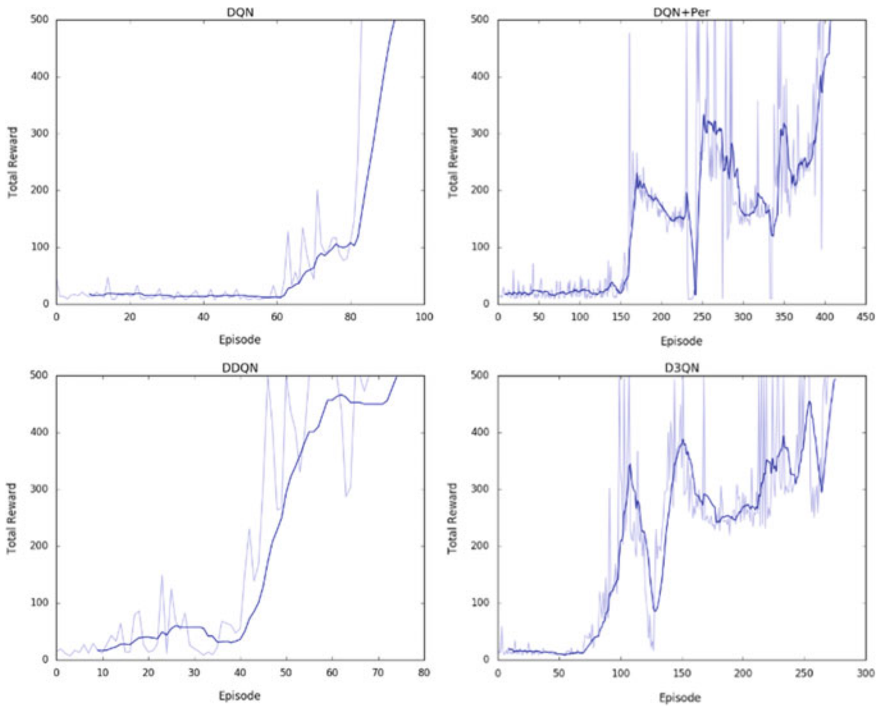

Fig. 10.7 Experimental results of DQN, DQN with Per, DDQN, D3QN

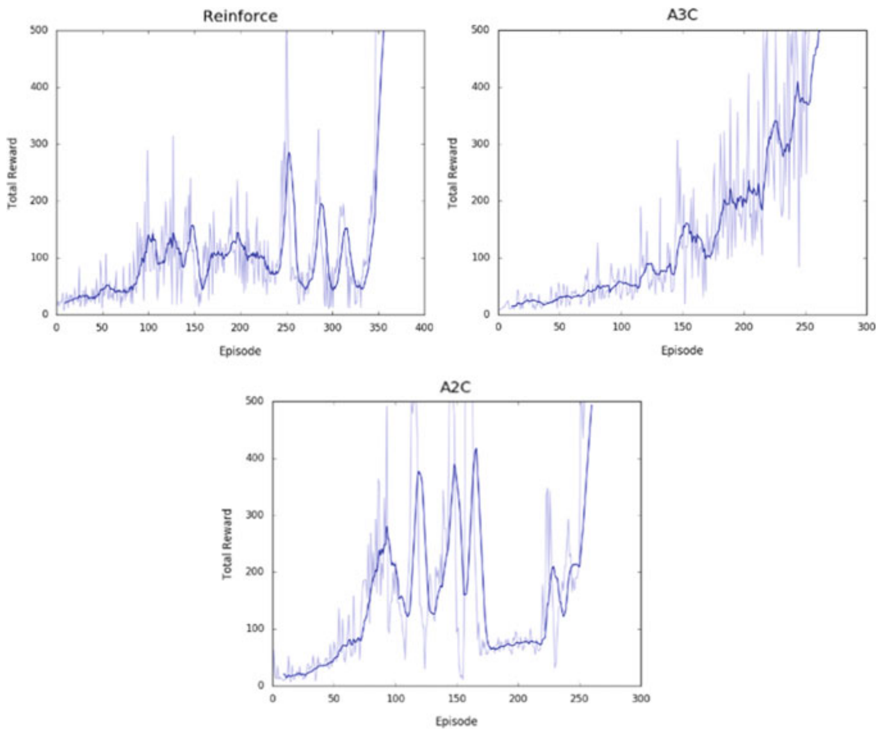

Fig. 10.8 Experimental results of Reinforce, A3C, A2C

methods in the literature and to illustrate the performances of the methods on the cart-pole balancing problem.

This chapter firstly describes the state of the art deep reinforcement learning methods such as DQN, DQN with PER, DDQN, D3QN, Reinforce, A3C, and A2C. Secondly the different network structures are constructed. The cart-pole balancing problem in the OpenAI Gym Environment is finally solved by using the generated methods in Keras.

The effectiveness of the deep reinforcement learning methods is shown by means of the accumulated reward, average reward, max reward, rise time, and settling time. Experimental results demonstrated that the DDQN performed significantly well in the control. It was observed that DDQN achieved the best values of the mean return, rise time and settling time for this cart-pole balancing problem, although it is known that the methods such as Reinforce, A3C, and A2C have fast and high accuracy. Meanwhile, the experimental result showed that the network structure is important for excellence performance.

The future work will focus on complex problems to test the performance of the deep reinforcement learning methods including both data and image inputs.

Table 10.6 Performance comparison of DQN, DQN with Per, DDQN, D3QN, Reinforce, A3C, A2C as to mean reward, max reward, rise time, and settling time

Method	Hidden neuron number	Mean reward	Max reward	Rise time	Settling time
DQN	24	112.64	500	114	140
	32	86.79	500	83	100
	64	179.02	500	141	180
DQN + Per	**24**	137.47	500	232	300
	32	147.57	500	161	430
	64	206.34	500	250	300
DDQN	24	140.30	500	108	180
	32	**207.12**	500	**46**	**80**
	64	191.64	500	99	180
D3QN	24	122.02	500	120	320
	32	201.04	500	99	280
	64	244.09	500	125	400
Reinforce	24	126.43	500	364	450
	32	120.02	500	293	450
	64	102.50	500	250	355
A3C	24	156.62	500	257	430
	32	191.50	500	244	255
	64	161.17	500	240	500
A2C	24	130.35	500	202	440
	32	101.45	500	191	379
	64	179.44	500	78	260

Acknowledgements This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) grant numbers 117E589. In addition, GTX Titan X Pascal GPU in this research was donated by the NVIDIA Corporation.

References

1. K. Ogata, Y. Yang, *Modern Control Engineering*, vol. 4 (London, 2002)
2. A.E. Bryson, *Applied Optimal Control: Optimization, Estimation and Control* (Routledge, 2018)
3. M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley, 2014
4. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 2018)
5. K. Vinotha, Bellman equation in dynamic programming. Int. J. Comput. Algorithm **3**(3), 277–279 (2014)
6. C.J. Watkins, P. Dayan, Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
7. S.S. Haykin, *Neural Networks and Learning Machines* (Prentice Hall, New York, 2009)

8. P. Abbeel, A. Coates, M. Quigley, A.Y. Ng, An application of reinforcement learning to aerobatic helicopter flight, in *Advances in Neural Information Processing Systems* (2007), pp. 1–8)
9. Y.C. Wang, J.M. Usher, Application of reinforcement learning for agent-based production scheduling. Eng. Appl. Artif. Intell. **18**(1), 73–82 (2005)
10. L. Peshkin, V. Savova, Reinforcement learning for adaptive routing, in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, vol. 2 (IEEE, 2002), pp. 1825–1830. (2002, May)
11. X. Dai, C.K. Li, A.B. Rad, An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. IEEE Trans. Intell. Transp. Syst. **6**(3), 285–293 (2005)
12. B.C. Csáji, L. Monostori, B. Kádár, Reinforcement learning in a distributed market-based production control system. Adv. Eng. Inform. **20**(3), 279–288 (2006)
13. W.D. Smart, L.P. Kaelbling, Effective reinforcement learning for mobile robots, in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 4 (IEEE, 2002), pp. 3404–3410. (2002, May)
14. J.J. Choi, D. Laibson, B.C. Madrian, A. Metrick, Reinforcement learning and savings behavior. J. Financ. **64**(6), 2515–2534 (2009)
15. M. Bowling, M. Veloso, *An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning* (No. CMU-CS-00-165). (Carnegie-Mellon University Pittsburgh Pa School of Computer Science, 2000)
16. D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, Mastering the game of Go with deep neural networks and tree search. Nature, **529**(7587), 484 (2016)
17. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning (2013). arXiv:1312.5602
18. V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
19. H.V. Hasselt, Double Q-learning, in *Advances in Neural Information Processing Systems* (2010), pp. 2613–2621
20. H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in *Thirtieth AAAI Conference on Artificial Intelligence* (2016, March)
21. Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning (2015). arXiv:1511.06581
22. J. Sharma, P.A. Andersen, O.C. Granmo, M. Goodwin, Deep q-learning with q-matrix transfer learning for novel fire evacuation environment (2019). arXiv:1905.09673
23. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
24. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1725–1732
25. T.N. Sainath, O. Vinyals, A. Senior, H. Sak, Convolutional, long short-term memory, fully connected deep neural networks, in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2015), pp. 4580–4584. (2015, April)
26. O. Abdel-Hamid, A.R. Mohamed, H. Jiang, G. Penn, Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition, in *2012 IEEE International Conference on Acoustics, Speech And Signal Processing (ICASSP)* (IEEE, 2012), pp. 4277–4280. (2012, March)
27. A. Uçar, Y. Demir, C. Güzeliş, Moving towards in object recognition with deep learning for autonomous driving applications, in *2016 International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)* (IEEE, 2016), pp. 1–5. (2016, August)
28. H.A. Pierson, M.S. Gashler, Deep learning in robotics: a review of recent research. Adv. Robot. **31**(16), 821–835 (2017)

29. L. Fei-Fei, J. Deng, K. Li, ImageNet: constructing a large-scale image database. J. Vis. **9**(8), 1037–1037 (2009)
30. A.E. Sallab, M. Abdou, E. Perot, S. Yogamani, Deep reinforcement learning framework for autonomous driving. Electron. Imaging **2017**(19), 70–76 (2017)
31. Y. Zhu, R. Mottaghi, E. Kolve, J.J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, Target-driven visual navigation in indoor scenes using deep reinforcement learning, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), pp. 3357–3364. (2017, May)
32. S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**(1), 1334–1373 (2016)
33. L. Xie, S. Wang, A. Markham, N. Trigoni, Towards monocular vision based obstacle avoidance through deep reinforcement learning (2017). arXiv:1706.09829
34. X.B. Peng, M. van de Panne, Learning locomotion skills using deeprl: does the choice of action space matter?, in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (ACM, 2017), p. 12. (2017, July)
35. G. Kahn, A. Villaflor, B. Ding, P. Abbeel, S. Levine, Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation, in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 1–8. (2018, May)
36. T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, Deep q-learning from demonstrations, in *Thirty-Second AAAI Conference on Artificial Intelligence* (2018, April)
37. T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay (2015). arXiv:1511.05952
38. R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3–4), 229–256 (1992)
39. V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in *International Conference on Machine Learning*, pp. 1928–1937. (2016, June)
40. A.V. Clemente, H.N. Castejón, A. Chandra, Efficient parallel methods for deep reinforcement learning (2017). arXiv:1705.04862
41. R.R. Torrado, P. Bontrager, J. Togelius, J. Liu, D. Perez-Liebana, Deep reinforcement learning for general video game AI, in *2018 IEEE Conference on Computational Intelligence and Games (CIG)* (IEEE, 2018), pp. 1–8. (2018, August)
42. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym (2016). arXiv:1606.01540
43. Y. Demir, A. Uçar, Modelling and simulation with neural and fuzzy-neural networks of switched circuits. COMPEL Int. J. Comput. Math. Electr. Electron. Eng. **22**(2), 253–272 (2003)
44. Ç. Kaymak, A. Uçar, A Brief survey and an application of semantic image segmentation for autonomous driving, in *Handbook of Deep Learning Applications* (Springer, Cham, 2019), pp. 161–200
45. A.W. Moore, C.G. Atkeson, Memory-based reinforcement learning: efficient computation with prioritized sweeping, in *Advances in Neural Information Processing Systems* (1993), pp. 263–270
46. R. Özalp, C. Kaymak, Ö. Yildirum, A. Uçar, Y. Demir, C. Güzeliş, An implementation of vision based deep reinforcement learning for humanoid robot locomotion, in *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)* (IEEE, 2019), pp. 1–5 (2019, July)
47. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html. Accessed 9 Sept 2019
48. C. Knoll, K. Röbenack, Generation of stable limit cycles with prescribed frequency and amplitude via polynomial feedback, in *International Multi-Conference on Systems, Signals & Devices* (IEEE, 2012), pp. 1–6. (2012, March)
49. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. (MIT Press, 2016)