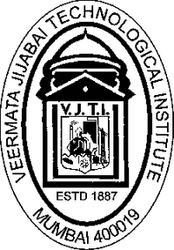
**An Extensive study of Deep Reinforcement Learning for Robotic control tasks**

Chetan Borse, Harshvardhan Kanthode.

Under the supervision of,

**Prof. A.S. Rao**



**DEPARTMENT OF PRODUCTION ENGINEERING**

**VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE,MUMBAI**

**May, 2019.**

**Acknowledgement**

We would like to express our gratitude and appreciation towards **Prof. A.S Rao**, Assistant Professor, Department of Mechanical Engineering, VJTI for granting us the opportunity to work in this active research field, and for being always available to entertain any discussions.

We are obliged to take this opportunity to sincerely thank the **American Society of Mechanical** **Engineers** (ASME), VJTI for introducing us to the project idea and making it possible to embark on this project.

CHETAN BORSE (171040074),

HARSHVARDHAN KANTHODE (171050032),

Department of Production Engineering,

VJTI, Mumbai

**ABSTRACT**

Reinforcement learning is an active research field in the vast domain of Machine Learning and Artificial Intelligence.

Recent advancements in computational power have allowed us to solve many of the image processing and recognition tasks in real-time using Supervised Learning. Using parallel computing, learning can be achieved even faster. However this requires an extensive knowledge of the environment as well as the task in hand, i.e. curating the data and their corresponding true labels becomes of paramount importance before any kind of learning is achieved.

For classical control tasks, this is not feasible as training an agent in an environment such as robotic manipulation is not possible manually for each position in the space. Instead, Reinforcement learning, a subset of Unsupervised learning is used to tackle these control tasks.

We review various Model-free algorithms based on the Actor-critic Deep-Q networks that can operate over continuous action spaces in both grid-world and real-world environments.

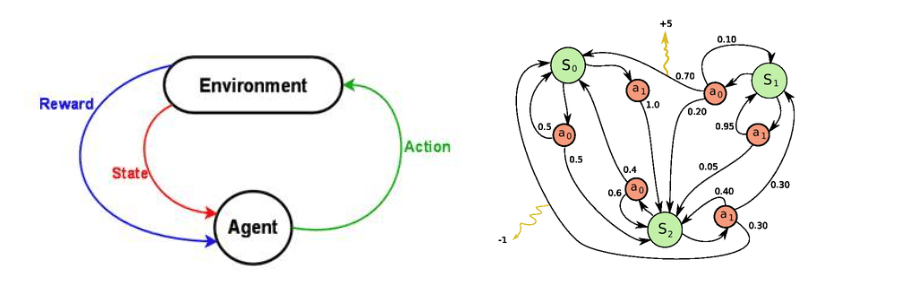
Simulation lets us show the procedures that we adopted to precisely estimate the algorithms hyper-parameters and to correctly design good policies. Real-world experiments let show that our policies, if correctly training on simulation, can be transferred and executed in a real environment with almost no changes.

**1. INTRODUCTION**

Reinforcement learning lies somewhere in between supervised and unsupervised learning. Whereas in supervised learning one has a target label for each training example and in unsupervised learning one has no labels at all, in reinforcement learning one has sparse and time-delayed labels – the rewards. Based only on those rewards the agent has to learn to behave in the environment.

Suppose you are an ***agent***, situated in an ***environment*** (e.g. Any classic/retro Video game). The environment is in a certain ***state*** (e.g. location of the paddle, location and direction of the ball, existence of every brick and so on). The agent can perform certain ***actions***in the environment (e.g. move the object to the left or to the right). These actions sometimes result in a **reward** (e.g. increase in score). Actions transform the environment and lead to a new state, where the agent can perform another action, and so on. The rules for how you choose those actions are called **policy**. The environment in general is *stochastic*, which means the next state may be somewhat random (e.g. when you lose a ball and launch a new one, it goes towards a random direction).

Modern Robots require adaptability to dynamic or fluctuating workspaces. They should be equipped to adapt the motion of the joints to the disturbances present in the environment.

 **Left: Reinforcement learning, Right: Markov-decision process**

We can start by thinking about the control problem of building a walking robot. Cameras and other haptic sensors can be used for visual feedback, which gives an estimation of the state the bot is in, each time an action is taken. These inputs provide control to the robot.

New motions, path planning and trajectory mapping are tasks for a manipulator being used in industries like Manufacturing and Assembly. These challenges are difficult to resolve by the traditional Robot Control approach.

Reinforcement Learning (RL) has been employed as a replacement for this purpose. However the large number of Joints and Links in an industry-grade robot require a large state-space, which accept a voltage value to their motors mapping the torque and speed values to suitably position the end-effector. The torque angles require an infinite Action space which is difficult for any algorithm to converge on within a limited amount of training steps.

1.1 **The Environment**

A good RL implementation requires a deep understanding of the set up of the problem. The *Environment* is everything surrounding the *Agent* and constitutes of every parameter which isn’t being optimized by the learning algorithm. For example - An environment for a Bipedal Robot would be anything excluding the physical skeleton of the robot. When the Agent generates an action (outputting Torque values for motors), the environment returns the Reward and the Observations (Joint angles and position of end-effector.

There are essentially 2 types of RL agents and environments – Model Free, and Model based.

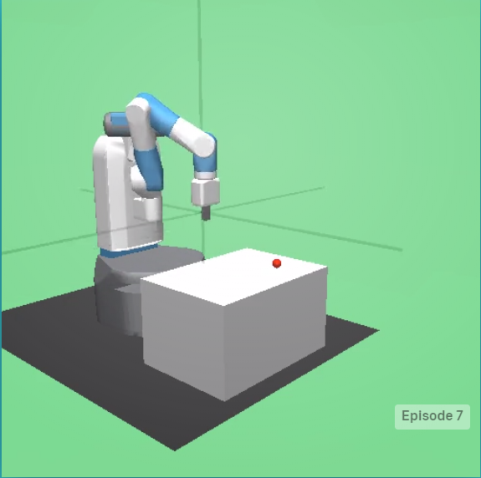
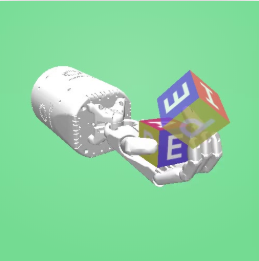
Model-free models focus on exploring the entire state-space rather than relying on existing knowledge of the useless states. Since Model-free models are equipped to deal with rapid changes to the environment, they are preferred over Model-based. We will focus on the former for our control problem.

However, Model free systems require a much more learning and training of the agent, and Sample Inefficiency is also one of the main challenges. Hence we use Simulations of the hardware on Software before deploying it over to the real-life counterpart.

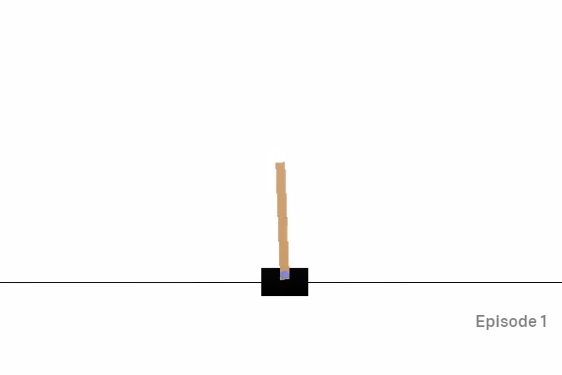
Simulation also allows us to include conditions that are hard to test within the real world, like Air resistance and friction.

OpenAI Gym provides a set of rendered environments within their library,

Our Environment requires a 3-axis robot placed beside a tabletop with the end effector positioned so as to reach any target point on the table, allowing for a limited movement on a 2d plane.

 ****

**Fig 1: FetchReach-v1 Fig 2: HandManipulateBlock-v0**

****

**Fig 3 : CartPole-v1**

1.2 **Reward and its shaping**

1.3 **The Policy**

A policy is essentially a method or rule of deciding actions based on the states of the agent.

 At present, the two most popular classes of reinforcement learning algorithms are **Q Learning** and **Policy Gradients**. Q learning is a type of value iteration method aims at approximating the Q function, while Policy Gradients is a method to directly optimize in the action space.

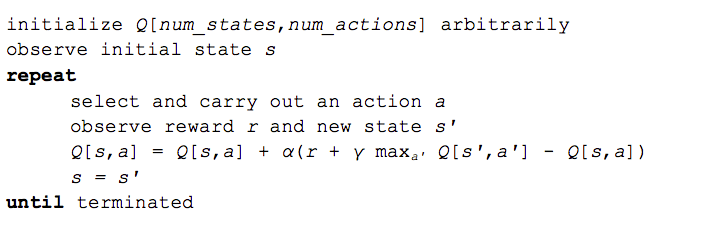
* **Q-learning:**

q learning example

The Q-function represents the maximum discounted future reward for an action ‘a’ done in a state ‘s’. It represents the ‘quality’ of that action in the near future of the agent.

q learning algorithm

Here **‘Π’** represents the policy. It is optimised based on the discounted rewards of the future as well as the current reward obtained.

 **Fig : Q –learning Algorithm using Bellman equation**

The maxa’ Q[s’, a’] that we use to update Q[s, a] is only an approximation and in early stages of learning it may be completely wrong. However the approximations get more and more accurate with every iteration and it has been shown, that if we perform this update enough times, then the Q-function will converge and represent the true Q-value.

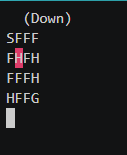
****

Fig 4 : FrozenLake text game

In case of 16 observations (8x8 state matrix) and 4 actions (moves) (Frozen-lake game) the Q-matrix is a manageable matrix. Imagine you got a full Atari game screen of pixels as an observation and it becomes quickly visible the Q-matrix solution will not cope. Also the Q-learning agent does not have the ability to estimate value for unseen states, it has no clue which action to take and goes back to random action as best.

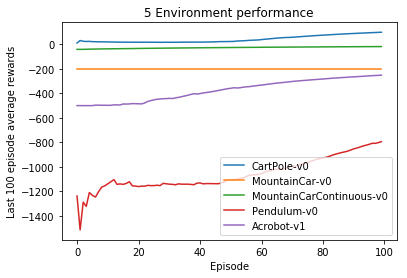
To deal with these problems, *Deep Q-Network (DQN)* removes the two-dimensional Q-matrix by introducing a Neural Network. So it leverages a Neural Network to estimate the Q-value function. The input for the network is the current game state, while the output is the corresponding Q-value for each of the actions.

After the first publication of DQN many Deep Reinforcement Learning algorithms have been invented/tried, Some main ones in chronological order:

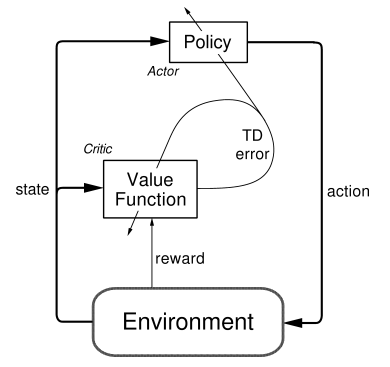
* *DQN,*
* *Double DQN,*
* *Duelling DQN,*
* *Deep Deterministic Policy Gradient,*
* *Continuous DQN (CDQN**or NAF) ,*
* *A2C/A3C,*
* *Proximal Policy Optimization Algorithms, etc*
* **Policy gradients:**

 One of the subsequent challenges that the reinforcement learning community faced was figuring out how to deal with continuous action spaces. This is a significant obstacle, **since most interesting problems in robotic control, fall into this category.**

Building off the prior work on Deterministic Policy Gradients, **Google Deepmind** have produced a ***policy-gradient actor-critic***algorithm called Deep Deterministic Policy Gradients (DDPG) that is ***off-policy*** and ***model-free***, and that uses some of the deep learning tricks that were introduced along with Deep Q-Networks



OUR APPROACH:



**Forward Kinematics:**

Forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters.

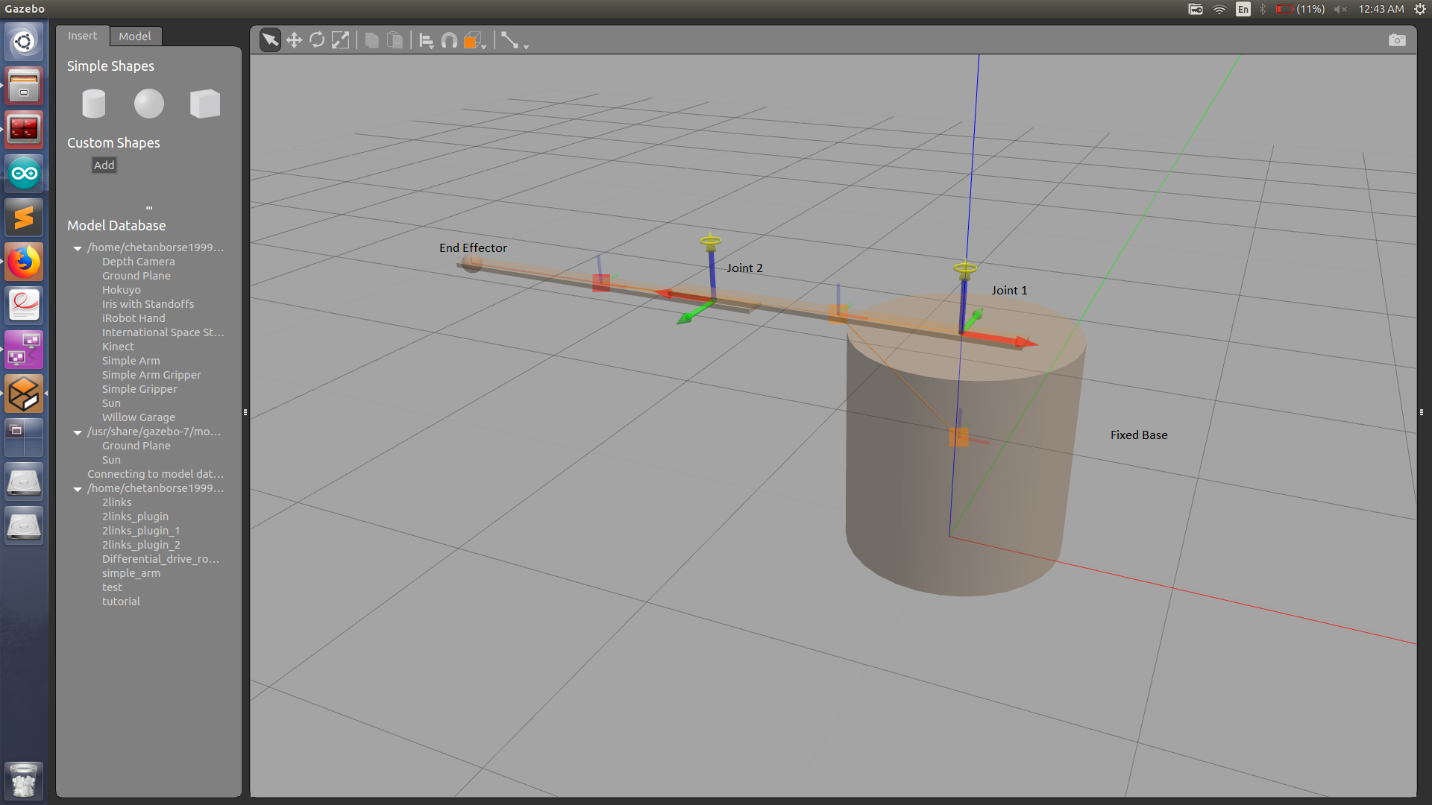
For our purpose, we have created the above model. The spherical element at the end of the manipulator is assumed to be end-effector. The end effector, the two joint and the base have a frame of reference. Cartesian coordinates, expressed as vectors, can be calculated with reference to base by means of a transformation matrix. This kinematic equation is written as a function of joint angles. Thus we can calculate the position of end effector in space verify it with the simulation.

3. **Tools used**

**Gazebo Sim:**

Gazebo is a well-designed simulator makes it possible to rapidly test algorithms, design robots, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It is also open-sourced and has good community support.

We have used Gazebo to simulate a 2 DOF robotic manipulator on a fixed cylindrical base.



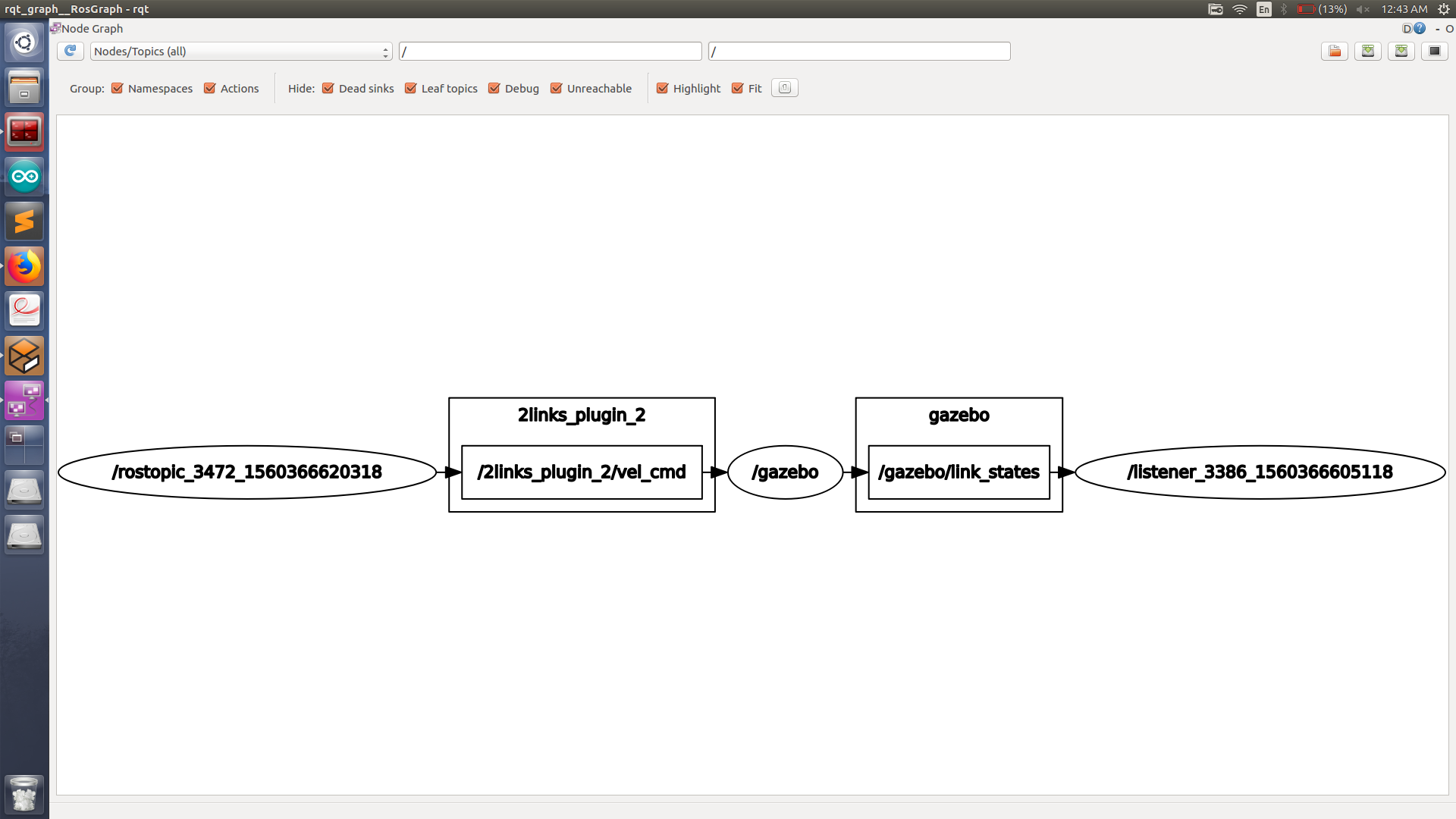
**Plugins:**

In Gazebo, actuation is done by the means of plugins. Plugins are written C++ which are can be used to control joints. For our required purpose, plugin was made such that, it accepted two angles as input (in radians) and writes those angles to the respective joints as specified in the code.

**ROS integration:**

Robot Operating System (ROS) is robotics middleware (i.e. collection of software frameworks for robot software development). It provides services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

In ROS, packages contain application-related code, libraries, message formats, etc. For our aim, we have created a customized ROS packages which consists of necessary libraries and messages.

****

STOCASTIC VS DETERMINISTIC MODELS

A system is a system. This is neither deterministic nor stochastic. However, if we want describe the *development* of a (dynamic) system, we use a *model*, and such a model (description) can be deterministic or stochastic.

A deterministic model can eventually be given as a mathematic formula or equation (or a set of equations, e.g. differential equations). It allows us to assume we know everything (relevant) that happens in the system and that this is correctly specified in the formula(s). For any set of parameters the entire history (past and future) of the system is thus "known", as we can directly evaluate the formula(s) for any given time-point (that can be practically quite demanding, even impossible, but we look at the principles here).

A stochastic model is used if we can not (or don't want to) model quantitative relationships between the components of the system but instead can (or want to) give only probabilities for some events happening during some (usually short) periods of time. Having some starting values we can find probabilities of the system being in diffenet possible future states.

\begin{figure*}
\centerline{\psfig{figure=jit.ps,height=2in}}\par\rule{\textwidth}{.2mm}
\end{figure*}

**Figure:** Learned value function over 25 irregularly spaced nodes, after nine iterations. The circles show the value function at the nodes, and the surface shows a smoother approximation by means of kernel regression.

1. Sutton, Richard S., and Andrew G. Barto. [Reinforcement learning: An introduction](http://www.incompleteideas.net/book/the-book.html).
2. Silver, et al. [Deterministic Policy Gradients](http://jmlr.org/proceedings/papers/v32/silver14.pdf)
3. Mnih, et al. [Human-level control through deep reinforcement learning](http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html)
4. Lillicrap, et al. Continuous control with Deep Reinforcement Learning

References

* Chris Watkins, Learning from Delayed Rewards, Cambridge, 1989 ([thesis](http://www.cs.rhul.ac.uk/home/chrisw/new_thesis.pdf))
* Awesome Reinforcement Learning repository, <https://github.com/aikorea/awesome-rl>
* J. Kober, J. Andrew (Drew) Bagnell, and J. Peters, “Reinforcement Learning in Robotics: A Survey,” International Journal of Robotics Research, July, 2013. ([pdf](http://www.ias.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf))
* Marc Peter Deisenroth, Gerhard Neumann and Jan Peters, A Survey on Policy Search for Robotics ([pdf](https://spiral.imperial.ac.uk/bitstream/10044/1/12051/4/2300000021-Deisenroth-Vol2-ROB-021_published.pdf))
* Sutton, Richard S., and Andrew G. Barto. [Reinforcement learning: An introduction](http://www.incompleteideas.net/book/the-book.html).
* Silver, et al. [Deterministic Policy Gradients](http://jmlr.org/proceedings/papers/v32/silver14.pdf)
* Mnih, et al. [Human-level control through deep reinforcement learning](http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html)
* Lillicrap, et al. Continuous control with Deep Reinforcement Learning