Background

**Machine Learning**

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

In this work we will work on a branch of machine learning call "Reinforcement Learning" (aka RL).

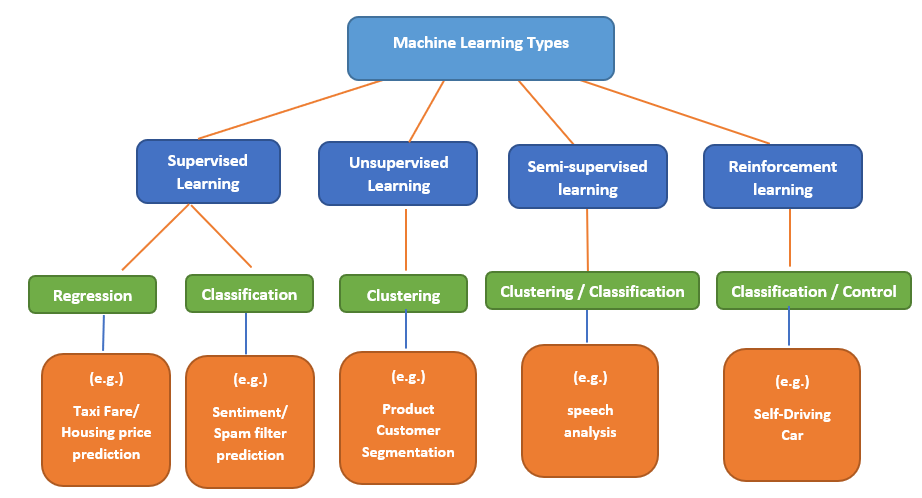


Figure 1- ML categories

**Reinforcement Learning**

When it comes to explaining machine learning to those not concerned in the field, reinforcement learning is probably the easiest sub-field for this challenge. RL it’s like teaching your dog (or cat if you live your life in a challenging way) to do tricks: you provide goodies as a reward if your pet performs the trick you desire, otherwise, you punish him by not treating him, or by providing lemons. Dogs really hate lemons.

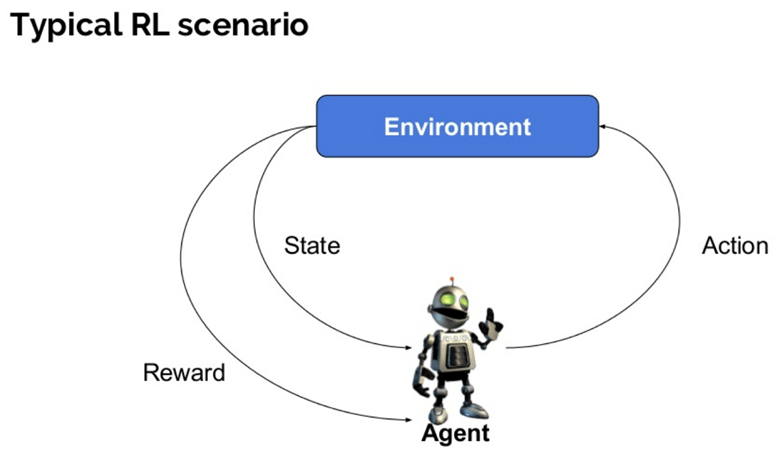


Figure 2- RL scheme

Here are some basic terms we need to use in order to continue:

* **Agent:** It is an assumed entity which performs actions in an environment to gain some reward (e.g. a dog).
* **Environment (e):** A scenario that an agent has to face (e.g. doggy playground).
* **Reward (R):** An immediate return given to an agent when he or she performs specific action or task (e.g. goodies for the dog, or lemons). The reward depends on the current state, and action: .
* **State (s):** State refers to the current situation the environment returns (e.g. dog's positions in time, weather, owner's mood etc.).
* **Observation (o)**: some partial data from the state, as observed by the agent (e.g. dog's view of the world).
* **Action (a)**: a move that the agent can make inside the environment (e.g. do can run, flip, jump…)
* **Policy (π):** It is a strategy which the agent applies to decide the next action based on the current observation (e.g. when to run, when to jump, when to flip…). The policy can be either deterministic or stochastic.
* **Value (V):** It is the expected long-term return with discount, as compared to the short-term reward. The value function takes the current 'state' as parameter.
* **Discount ():** the 'importance' of future rewards compared to current rewards.
* **Q value or action value (Q):** Q value is quite similar to value. The only difference between the two is that it takes an additional parameter as a current action. The connection between Q value and Value is:

In RL, the scenario is modeled as a Markov Decision Process (MDP):

* is the probability of transition (at time ) from state to state under action .
* is the immediate reward after transition from to with action .

A RL agent interacts with its environment in discrete time steps. At each time t, the agent receives an observation , which typically includes the reward . It then chooses an action from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state and the reward , associated with the transition , is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can (possibly randomly) choose any action as a function of the history.

Question – we said that a machine learning algorithm learns from **data**. What data is used here?

Answer – A RL algorithm, will often use some "trial and error" of the agent. The agent will run around in the environment, try different actions, and get different rewards. For every timestep the RL algorithm will save . This will be the data in each the algorithm learns from.

There are many ways and algorithms used to solve this problem, but we will discuss the model free method of DDPG.

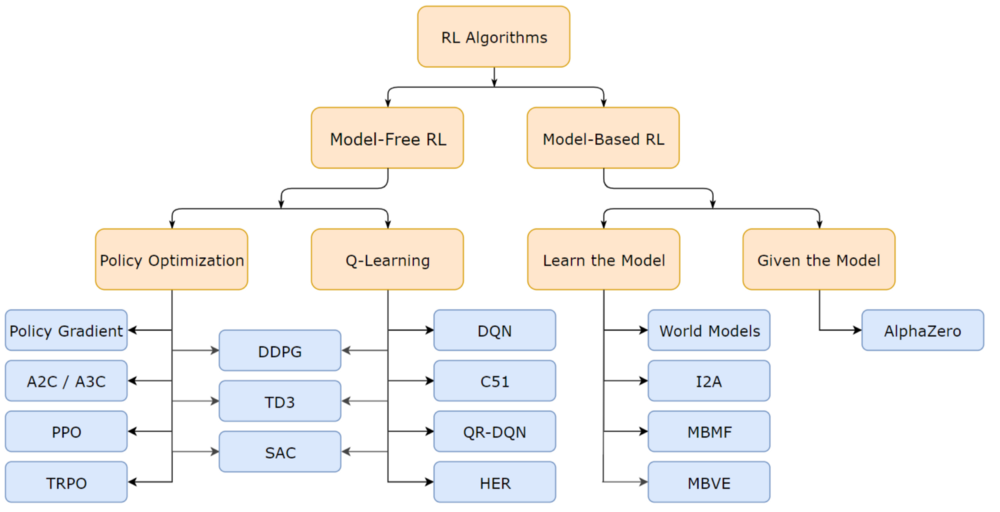


Figure 3- RL algorithm types

**DDPG -**

“Continuous Control With Deep Reinforcement Learning” (Lillicrap et al, 2015)

Deep Deterministic Policy Gradient (DDPG) is a **model-free** algorithm which concurrently learns a **Q-function** and an **optimal policy** using a **Deep Neural Network**. It uses **off-policy** data and the **Bellman equation** to learn the Q-function, and uses the Q-function to learn the **deterministic policy**. DDPG is used for **Continuous Control**.

Now let's go through these terms one by one:

* **Model free** - mean that the RL algorithm doesn't need an explicit statistical model of the environment (don't need to know ). Instead, DDPG tries to model the Q-value.
* **Continuous Control** – means that the actions a, are a continuous value (also - making them differentiable).
* **Off-policy -** basically means that the data (saved ) the algorithm uses in the learning process, does not necessarily come from the current policy the agent uses.
* **Deterministic Policy** – mean that the policy function is deterministic
  + The Trajectory (– a sequence of states and actions observed in the environment:

.

* + Since is deterministic, .
* **Optimal Policy ():**
  + The Return (R) – the overall discounted reward, determined by states and actions in time.
  + The optimal policy () – the policy function that maximizes the expected Return for every state.
* **Q-function in DDPG:**
  + The Optimal Q-value function () - gives the expected return if you start in state , take an arbitrary action , and then forever after act according to the optimal policy in the environment.

* **Bellman equation -** the Q-function obeys a special self-consistency equation called the Bellman equation. The basic idea behind the Bellman equations is this:

The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

Where s' is the next state, a' is the best next action (since is optimal), and represents the probability of s' to be the next state of the environment, given s and a. . The proof for the bellman equation comes from the fact that we are dealing with an MDP (without it, everything falls apart).

* **Deep Neural Network** **(DNN)**- an artificial neural network with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output. Each layer has a set of weights (parameters ) that are tuned (using optimization methods). After tuning the weights, a DNN can represent extremely complex mathematical functions. In DDPG, the algorithm uses a DNN to represent the optimal policy function and Q-function.

The Idea of DDPG is too 'learn' concurrently both and , when 'learn' means to tune the DNN's parameters . When the networks are trained, the agent can use as the policy to act of – thus solving the RL problem (maximizing its reward).

DDPG Implementation:

The DDPG architecture is said to be an "Actor Critic" model. The Actor is (deciding which actions to take) and the Critic is (deciding is the actor's action was good or not).

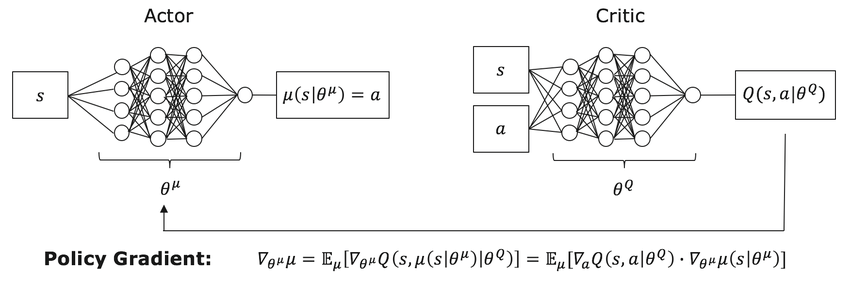


Figure 4 - DDPG architecture

**Learning :**

The Bellman equation is the starting point for learning an approximator to . Suppose the approximator is a neural network , with parameters , and that we have collected a set of transitions (where indicates whether state is terminal). We can set up a mean-squared Bellman error (MSBE) function, which tells us roughly how closely comes to satisfying the Bellman equation:

The Critic parameters are learned by: .

The Actor parameters are learned by

**Implementation Tricks:**

* Replay Buffers - All standard algorithms for training a deep neural network to approximate make use of an experience replay buffer. This is the set of previous experiences. In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it may not always be good to keep everything. If you only use the very-most recent data, you will overfit to that and things will break; if you use too much experience, you may slow down your learning. This may take some tuning to get right.
* Target Networks - Q-learning algorithms make use of target networks. The term

is called the target, because when we minimize the MSBE loss, we are trying to make the Q-function be more like this target. Problematically, the target depends on the same parameters we are trying to train: . This makes MSBE minimization unstable. The solution is to use a set of parameters which comes close to , but with a time delay—that is to say, a second network, called the target network, which lags the first. The parameters of the target network are denoted .

In DDPG-style algorithms, the target network is updated once per main network update by 'soft update'

where is a hyperparameter between 0 and 1 (usually close to 0). (This hyperparameter is called TAU in our code).

* Exploration vs. Exploitation - DDPG trains a deterministic policy in an off-policy way. Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make DDPG policies explore better, we add noise to their actions at training time. The authors of the original DDPG paper recommended time-correlated OU noise. To facilitate getting higher-quality training data, we reduce the scale of the noise over the course of training.

At test time, to see how well the policy exploits what it has learned, we do not add noise to the actions. (in our code we implemented this in our class OUNoise).

**MARL and MADDPG –**

* **MARL** - Multi-Agent Reinforcement Learning (MARL) is a growing and interesting field of RL.

As we know, RL has been applied to solve challenging problems like gameplaying, robotics, industrial applications and more. In the sections above we mentioned single agent domains. however, there are a number of important applications that involve interaction between multiple agents where RL agents co-evolve together. For example, multi-robot control, the discovery of communication and language, multiplayer games, and even analysis of social dilemmas - all operate in a multi-agent domain. Successfully scaling RL to environments with multiple agents is crucial to building artificially intelligent systems that can productively interact with humans and each other.

* **Challenges of MARL** - Unfortunately, traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments. One issue is that each agent’s policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent (in a way that is not explainable by changes in the agent’s own policy).

Some explanation – when an agent "plays" around in the environment he "thinks" that his actions (and his actions alone) will lead to the rewards he gets along the way. From that perspective, an agent can't learn anything if his rewards are not solely depended on his actions. For example, if a tennis player plays a game thinking that he's playing against a brick wall – he will be very confused when the wall gets better every game.

This presents learning stability challenges and prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning. Other RL methods, on the other hand, usually exhibit very high variance when coordination of multiple agents is required. In general, it is hard to train multiple (sometimes competing) agent, as evidenced by the notorious instability of adversarial training methods.

* **MADDPG** – one of the algorithms used to face these issues is the Multi-Agent Deep Deterministic Policy Gradient (MADDPG), presented in:

*Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-criticfor mixed cooperative-competitive environments. Inthe Annual Conference on Neural Informa-tion Processing Systems (NIPS).*

<https://arxiv.org/abs/1706.02275>

They used an extension to the Markov decision processes (MDPs), called Markov Games.

These Markov games, separate the State and Action into partial observations and actions that each agent takes. The architecture of the MADDPG agent is a set of multiple agents and critics (one for each agent), this opposed to other ideas of a central critic for all agents. Each critic receives as parameters all actions and partial observations from all of the agents, and that way the agents can learn better. Using the previous example, now the tennis players will not think of the opponent as a static wall – but as another player which gets better at the same time. The neural networks used in the MADDPG algorithm look as follows:

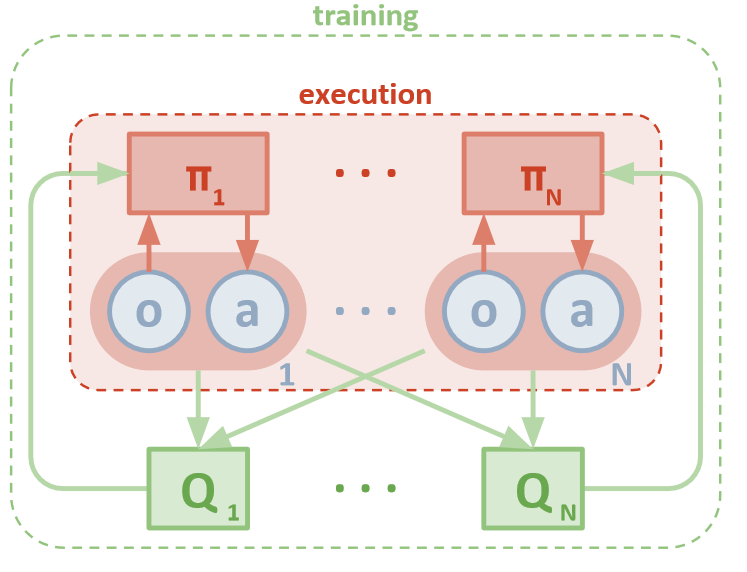
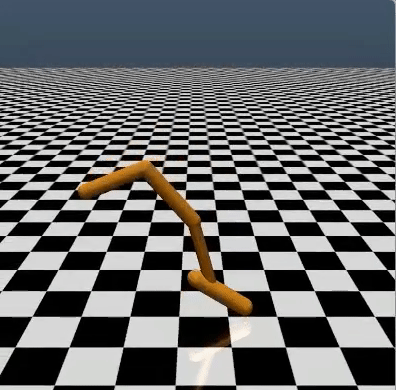


Figure 5 - MADDPG architecture

**Reinforcement Learning Environments**

Modeling real life problems is hard. Especially when the problems we're dealing with are physical. As we wrote, Reinforcement Learning is a study of an agent learning through interactions with the environment. You can't have a computer learn to drive a real car, At least not at the beginning… Even learning to move a robotic arm can be challenging because of the financial cost of owning a robot. Thus, to test and compare results of different reinforcement learning algorithms, we need testbed environments. By far the most commonly used testbed has been 57 Atari 2600 games. However, different environments require different exploration schemes and different algorithms. Thus, it is important to have various environments. Here are some famous RL environments:

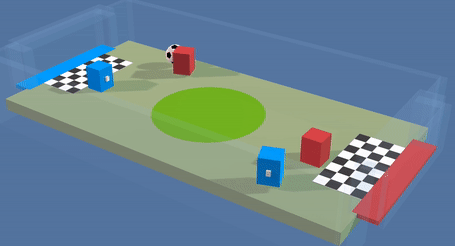
* **OpenAI Gym** – Perhaps the most famous toolkit for developing and comparing reinforcement learning algorithms. The gym library is a collection of environments that makes no assumptions about the structure of your agent. Gym comes with a diverse suite of environments, ranging from classic video games such as [Atari 2600](https://www.endtoend.ai/envs/gym#atari-2600) and [continuous control tasks](https://www.endtoend.ai/envs/gym#mujoco).

Figure 6 - OpenAI

* **The Unity Machine Learning Agents Toolkit** (ML-Agents) is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents.
* **PySC2** - [DeepMind](http://deepmind.com/)’s Python component of the StarCraft II Learning Environment (SC2LE). It exposes [Blizzard Entertainment](http://blizzard.com/)’s [StarCraft II Machine Learning API](https://github.com/Blizzard/s2client-proto) as a Python RL Environment. This is a collaboration between DeepMind and Blizzard to develop StarCraft II into a rich environment for RL research. PySC2 provides an interface for RL agents to interact with StarCraft 2, getting observations and sending actions.

Figure 7- Unity

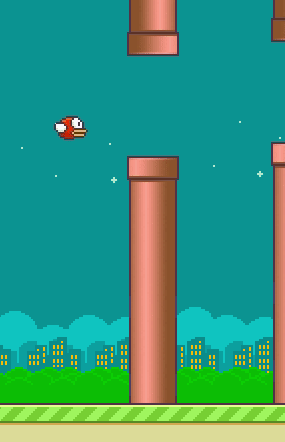
* **PyGame** **Learning Environment (PLE)** - A learning environment, mimicking the Arcade Learning Environment interface, allowing a quick start to Reinforcement Learning in Python. The goal of PLE is allow practitioners to focus design of models and experiments instead of environment design.

Figure 8 - PySC2

Figure 9 - PyGame

* **MARL environments** - even though there are many RL environments, there are not many built for multi-agent scenario. Especially environments that are modular in the sense of the number of agents. The well-known existing environments are the unity tennis environment, and OpenAI's "Multi-Agent Particle Environment" (which was used in the MADDPG paper). These environments are either not scalable (tennis) or not very interesting (particles).

Figure 10 - OpenAI Particle enviornments as used in MADDPG article

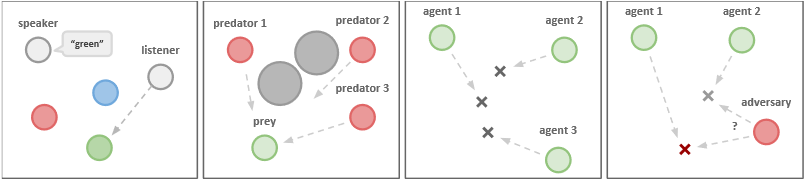
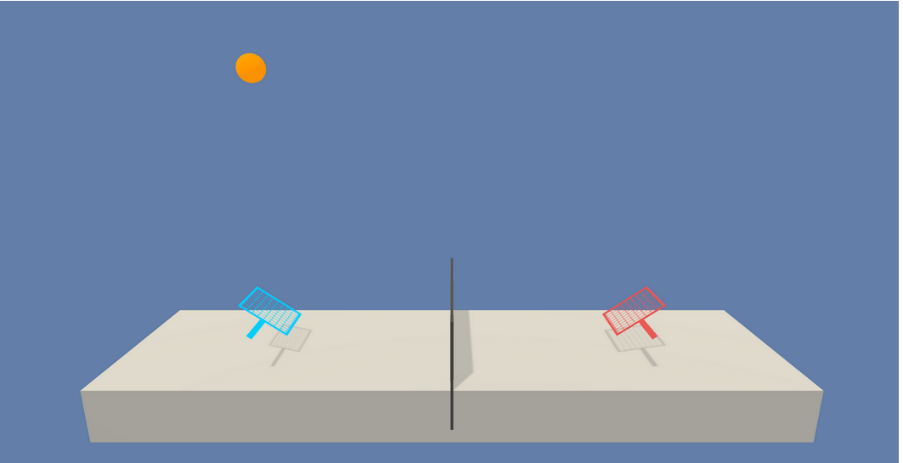


Figure 11 - Tennis in Unity

Links:

<https://www.codeproject.com/Articles/5245488/Introduction-to-Machine-Learning-and-ML-NET-Part-1>

<https://www.guru99.com/reinforcement-learning-tutorial.html>

<https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/>

<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>

<https://towardsdatascience.com/q-learning-54b841f3f9e4>

<https://en.wikipedia.org/wiki/Reinforcement_learning>

<https://en.wikipedia.org/wiki/Q-learning>

<https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c>

<https://www.endtoend.ai/envs/>

<https://expertsystem.com/machine-learning-definition/>

<https://spinningup.openai.com/en/latest/algorithms/ddpg.html#>

<https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#the-optimal-q-function-and-the-optimal-action>

<https://www.researchgate.net/figure/Basic-structure-of-the-DDPG-actor-critic-agent-Actor-with-parameters-th-m-state-s-as_fig3_322879739>

<https://arxiv.org/pdf/1706.02275.pdf>

https://www.endtoend.ai/envs/