Reinforcement Learning

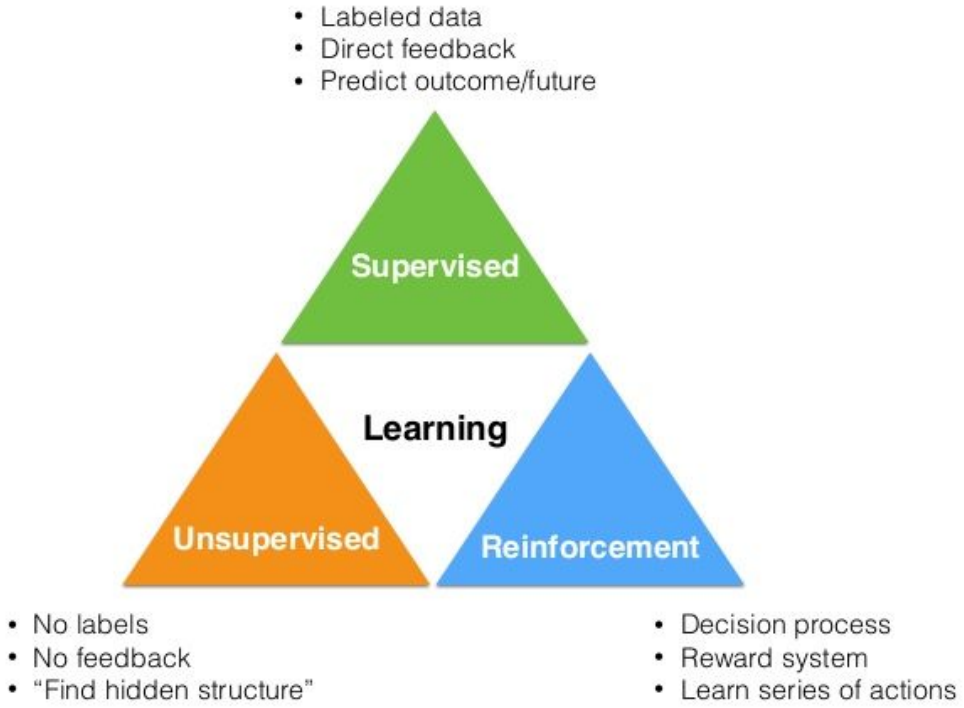
ML/DL Sharing Group
25/01/2018

Adrián Portabales Goberna <adrianportabales@gmail.com>
Pablo Dago Casas <pdago@gradiant.org>

Contents

- Introduction
  - What is RL?
  - Policy
  - Evaluating actions
  - How do we learn?
- Other methods
  - https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

What is RL?


• Labeled data
• Direct feedback
• Predict outcome/future
Supervised
Learning
Unsupervised
Reinforcement
• No labels
• No feedback
• "Find hidden structure"
• Decision process
• Reward system
• Learn series of actions

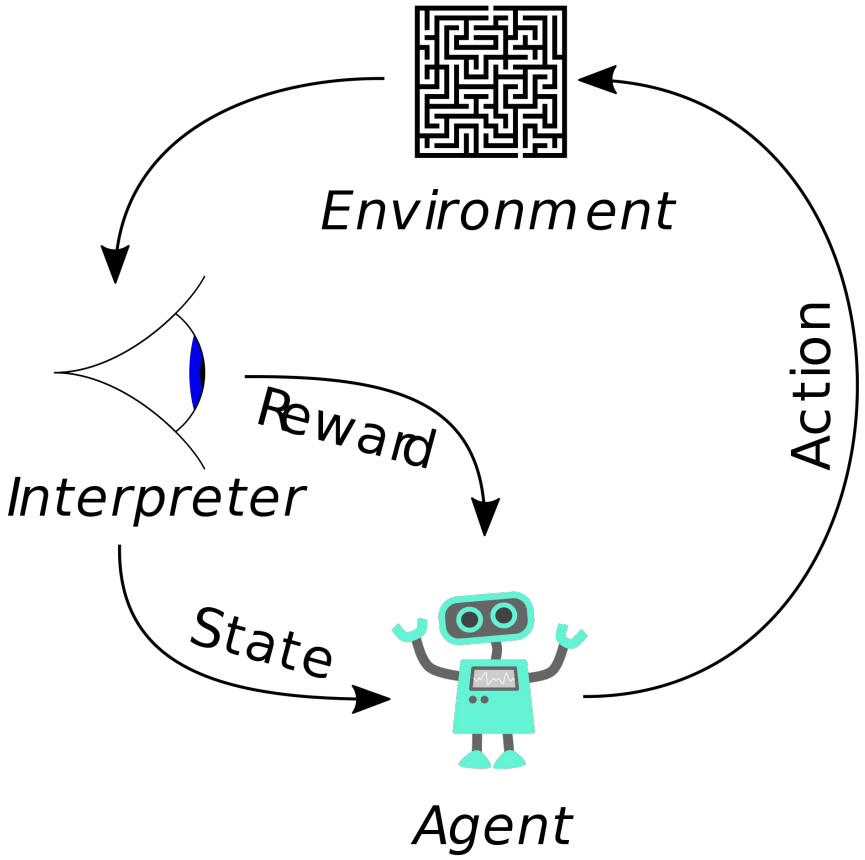
What is RL?

Agent observes the ***environment*** and takes ***actions***, obtaining the corresponding ***reward***

Environment: an image, signals from sensors, ...

Rewards are not always obtained immediately, can be positive or negative

The objective of the agent is to maximize the rewards



Cart pole example

- Environment: speed, acceleration, angle and position of the pole
- Actions: step right or left
- Reward: +1 every step the pole is upright (+- 5°)*

You lose if the pole is +-15° or out of the screen

Policy

Policy is the algorithm the agent uses to determine its actions

- Deterministic:
  - Angle > 0 → Move right
  - Angle < 0 → Move left
- Random
- Genetic algorithms
- Neural networks
- ...

Evaluating actions

- Rewards are the instrument we have to evaluate the policy
- How do we assign a value to an action?
  - We don't know the optimal action beforehand (supervised learning)
  - We rely on rewards:
    - Might be sparse → Not all actions cause a reward
    - Might be delayed → We might see an action is "good" after a chain of actions

Discounted rewards: To evaluate an action we use all the rewards that come after it (with a discount factor)

How do we learn? - Policy gradients

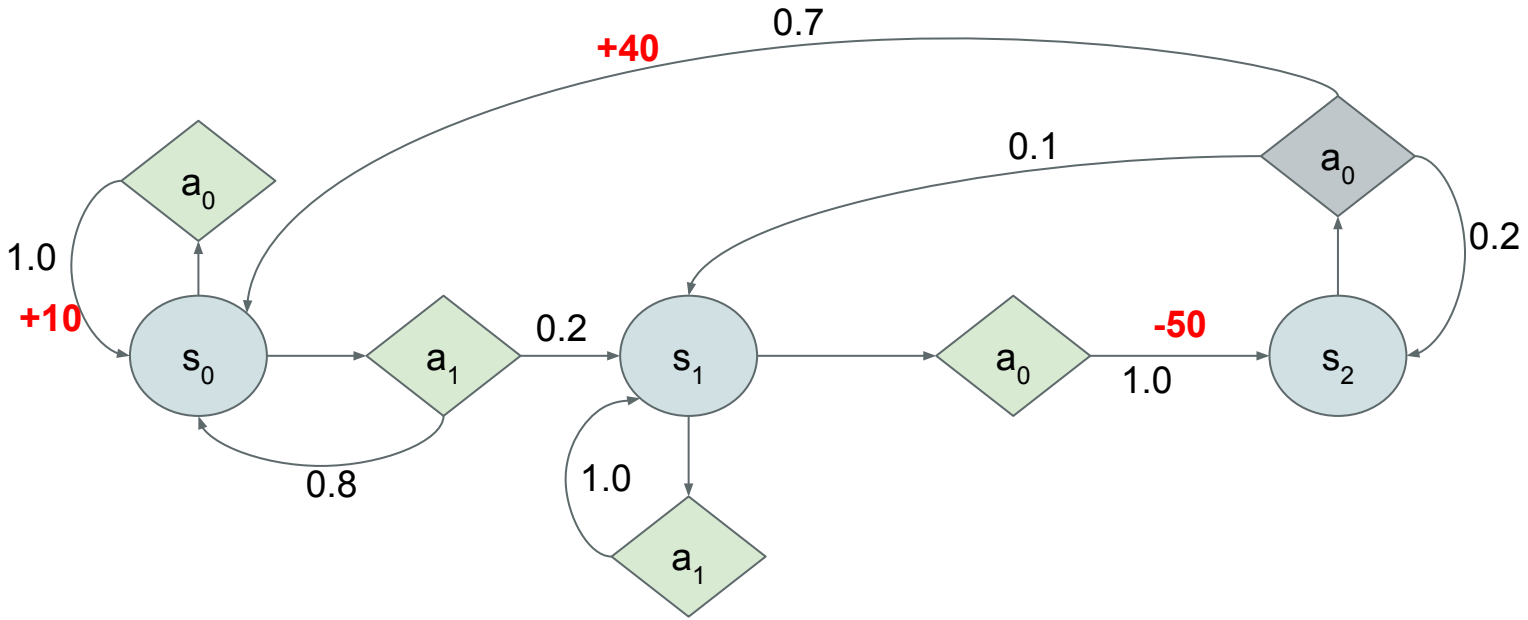
Optimize the parameters of the policy by following the gradients to optimize rewards

1. Let the policy run and compute the gradients of the rewards
2. After some executions, compute the score of the performed actions
3. Apply the gradients according to the scores: more/less likely actions

We are optimizing the policy to increase rewards

How do we learn? - Q-learning

Markov decision process



How do we learn? - Q-learning

Bellman optimality equation to compute the value of each state:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s' + \gamma V^*(s'))]$$

Can be computed iteratively:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s' + \gamma V_k(s'))]$$

This way we know the value of the states, but how do we evaluate the actions?

Q-value:

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s' + \gamma \max_{a'} Q_k(s', a'))]$$

How do we learn? - Q-learning

Which are the transition probabilities?

Which are the rewards?

We don’t know, so we must experience and estimate

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha[r + \gamma \max_{a'} Q_k(s',a'))]$$

Exploration policies:

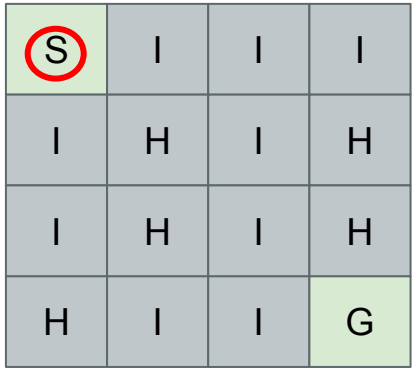
- greedy
- ε-greedy
- random
- ...

How do we learn? - Q-learning

Frozen lake example:

- 16 states
- Reward for getting to the goal
- Lose if you step on holes

Example in notebook



How do we learn? - Q-learning

What happens if we change the board or with unseen states? (generalization)

What happens with a larger number of states and/or actions?

We must use approximate Q-learning: learning how to derive the Q-values given the current state/status

How do we learn?

Cart pole example

- Environment: speed, acceleration, angle and position of the pole
- Actions: Move right or left
- Reward: +10 every step the pole is upright (+- 5°)*

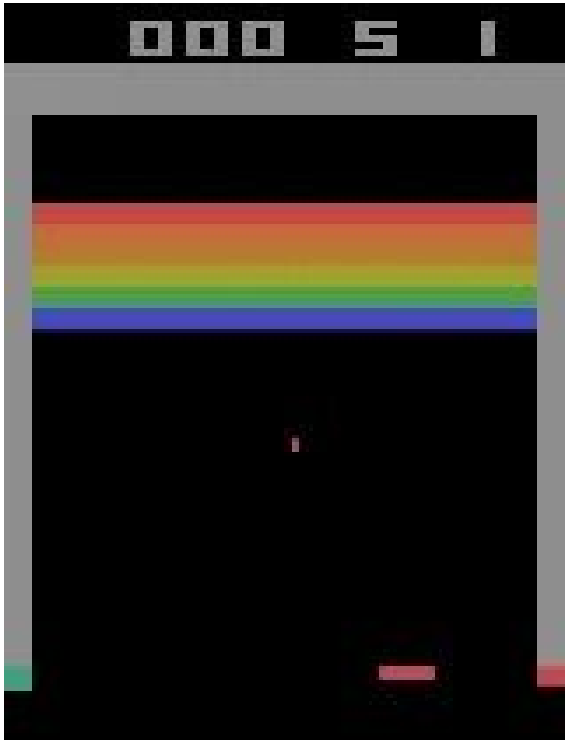
Example in notebook

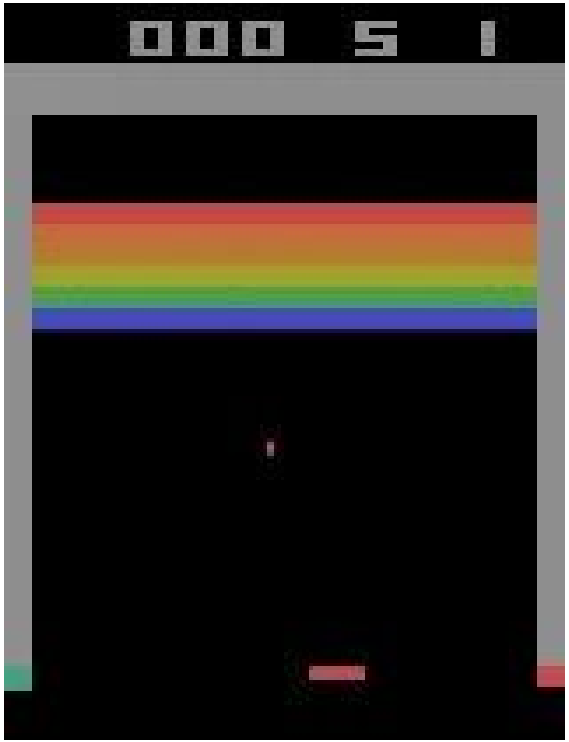
You lose if the pole is +-15° or out of the screen

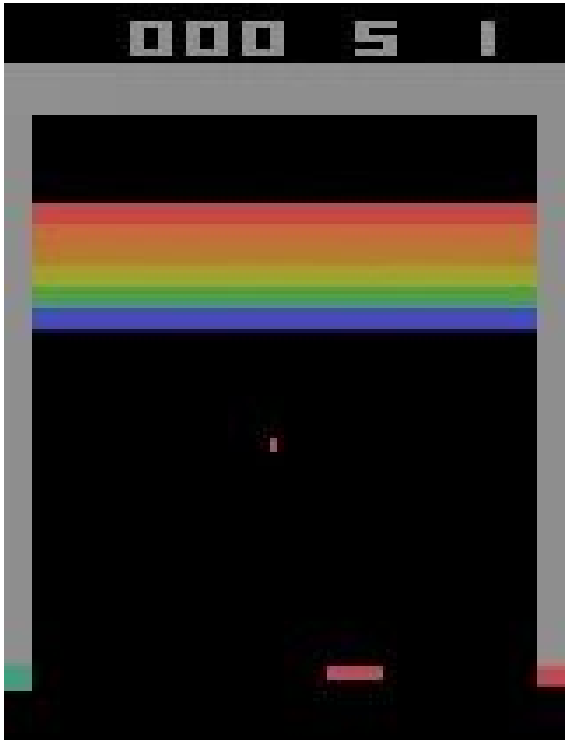
Applications

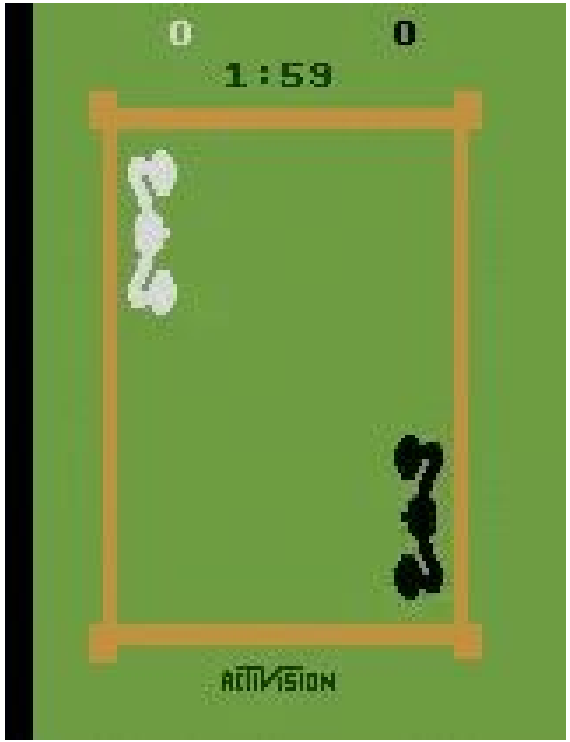
- Videogames
- Smart thermostat
- Hyperparameter selection
- Pharmacology
- ...

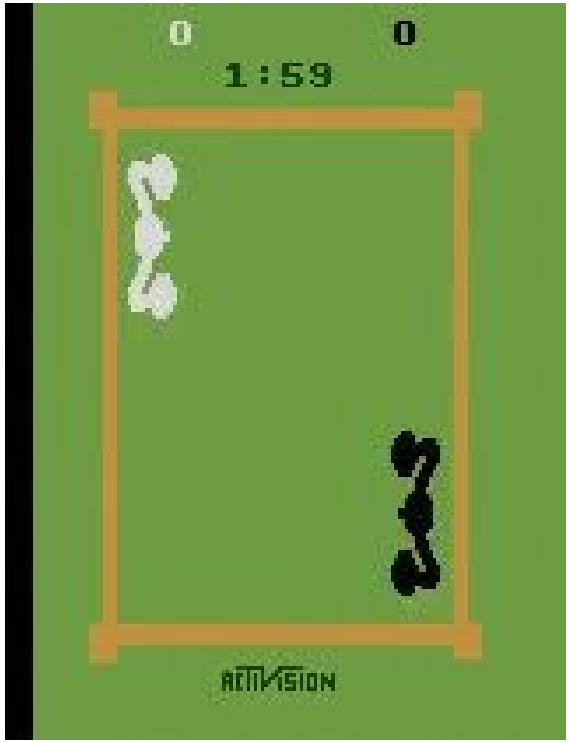
Parte 2

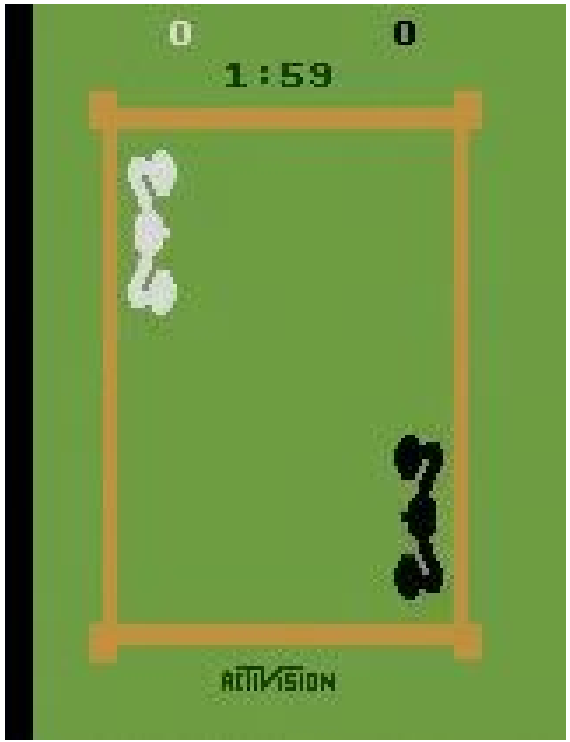

000 5 1

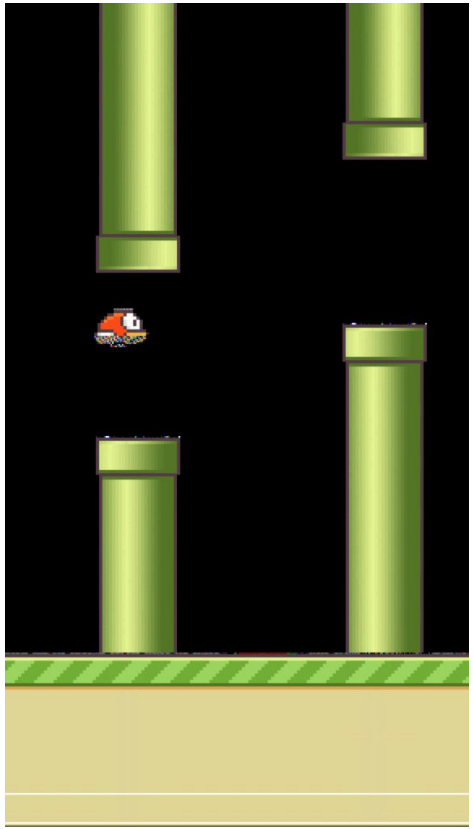

000 5 1


000 5 1


0
0
1:59
ACTIVISION


0
0
1:59
ACTIVISION


0
0
1:59
ACTIVISION



Deep Q Network

Reality

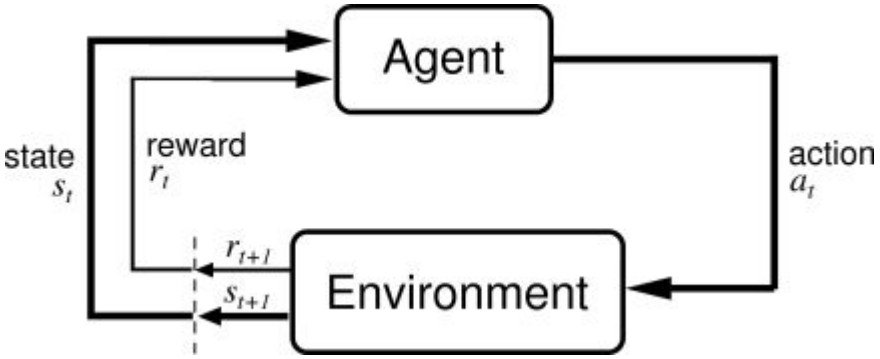
1. The data are highly correlated
2. The target values are not robust
3. The anomalies in rewards make the value function freaks out

DeepMind, save us

1. Different NN for Q and Q_target
2. Estimate Q_target using past experiences
3. Update Q_target every n steps
4. Reward between -1 y 1

Deep Q Network (DQN) - Atari

- DQN = Q-Learning + Deep Network
- Uses NN for approximation of non-lineal function
- Episodic tasks
  - Episodic Task
  - Continuous Task
- Discrete action space
- Multi-Frame as state



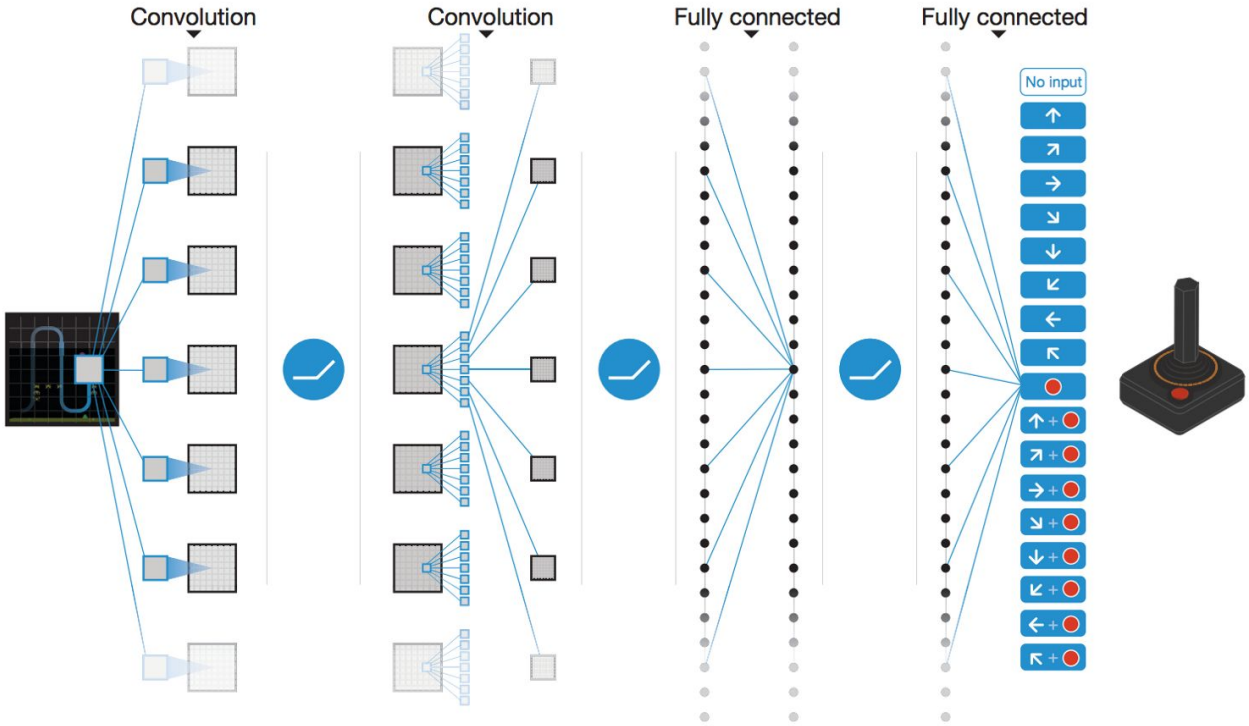
Deep Q Network (DQN) - Atari

- State:
  - 210x160x3 -> 210x160x1 -> 84x84x1
  - 4x Stack

- If will be a Q-Table: $256^{84x84x4} \approx 10^{67970}$ rows

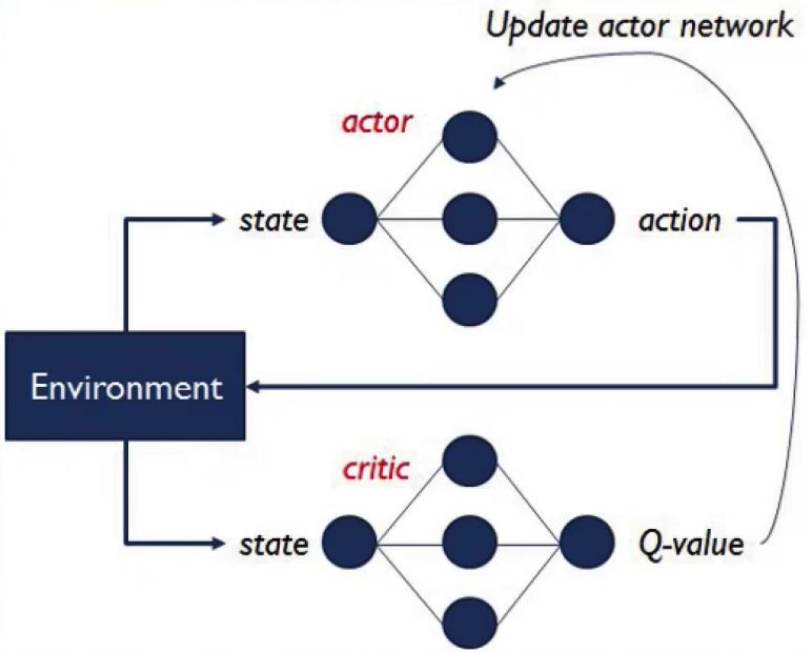
Deep Q Network (DQN) - Atari

- Architecture



Deep Q Network (DQN) - Atari

- Experience Replay / Replay Buffer
  - <state,action,reward,next state>



Exploration vs Exploitation

Exploration: Gather more information

VS

Explotación: Make the best decision given current information

Exploration vs Exploitation

Exploration: Gather more information

VS

Explotación: Make the best decision given current information

ε-greedy: Explore through random actions with ε probability and decay to a low value as training progress (exploitation)

OpenAI + Baselines

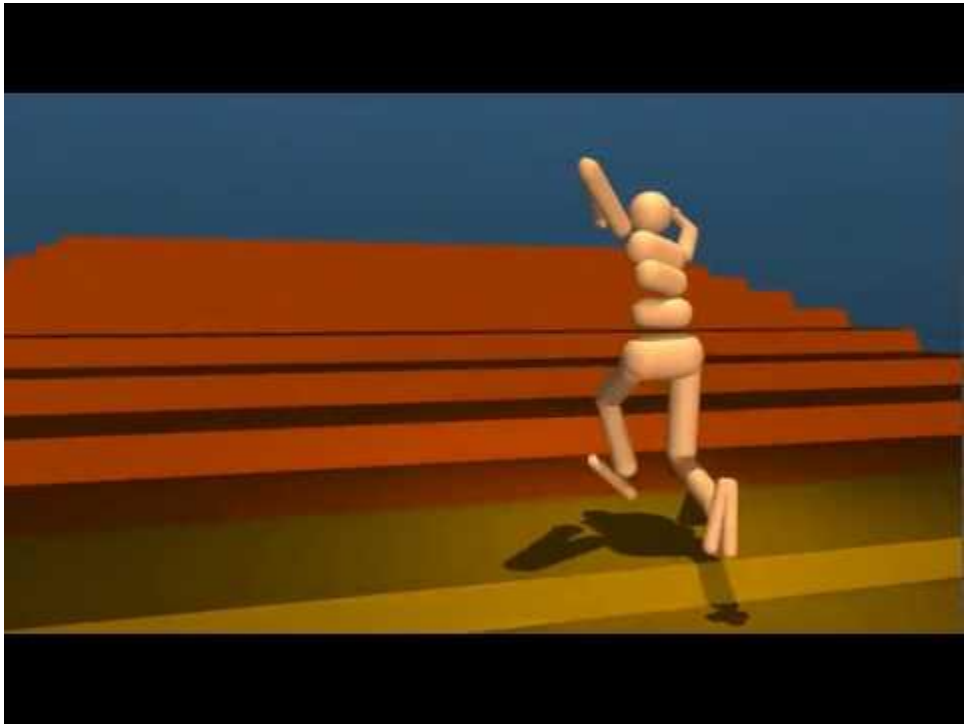
- OpenAI - gym
  - Library of environments
    - Pong
    - Doom
    - ...
  - Same API
  - Results benchmarking

- Baselines
  - OpenAI project
  - Implementations of reinforcement learning algorithms

Demo

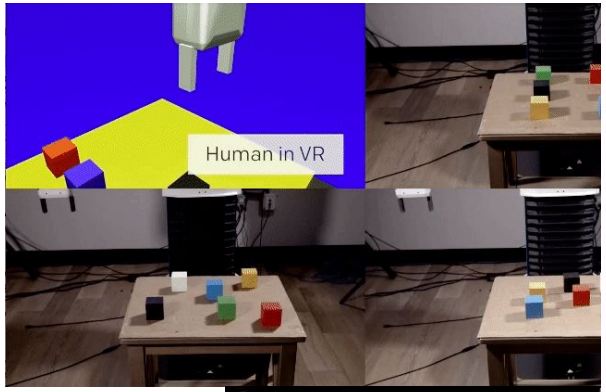
DQN is only the first step

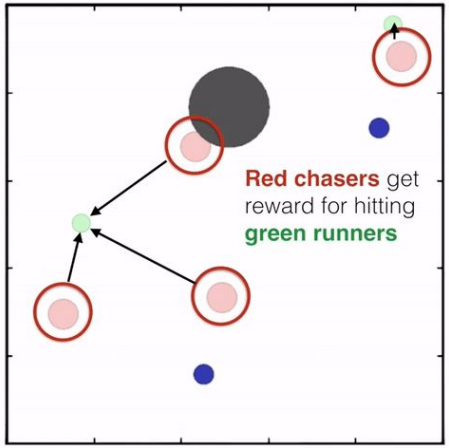
- Deep Q-Network (Mnih et al., 2015)
- Double DQN (Hasselt et al., 2016)
- Normalized Advantage Function (Gu et al., 2016)
- (Persistent) Advantage Learning (Bellemare et al., 2016)
- Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016)
- SVG(0) (Heese et al., 2015)
- Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016)
- Asynchronous N-step Q-learning (Mnih et al., 2016)
- Actor-Critic with Experience Replay (Wang et al., 2017)
- DQN results: https://github.com/openai/baselines-results/blob/master/dqn_results.ipynb

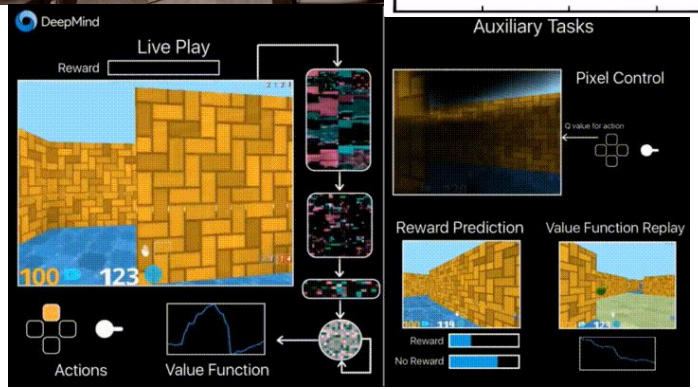


Multi-agent Reinforcement Learning

- Self-play
- Coordination
- Transfer Learning
- Joint action learning
- Learning To Communicate
- Cooperation and competition
- Inverse Reinforcement Learning

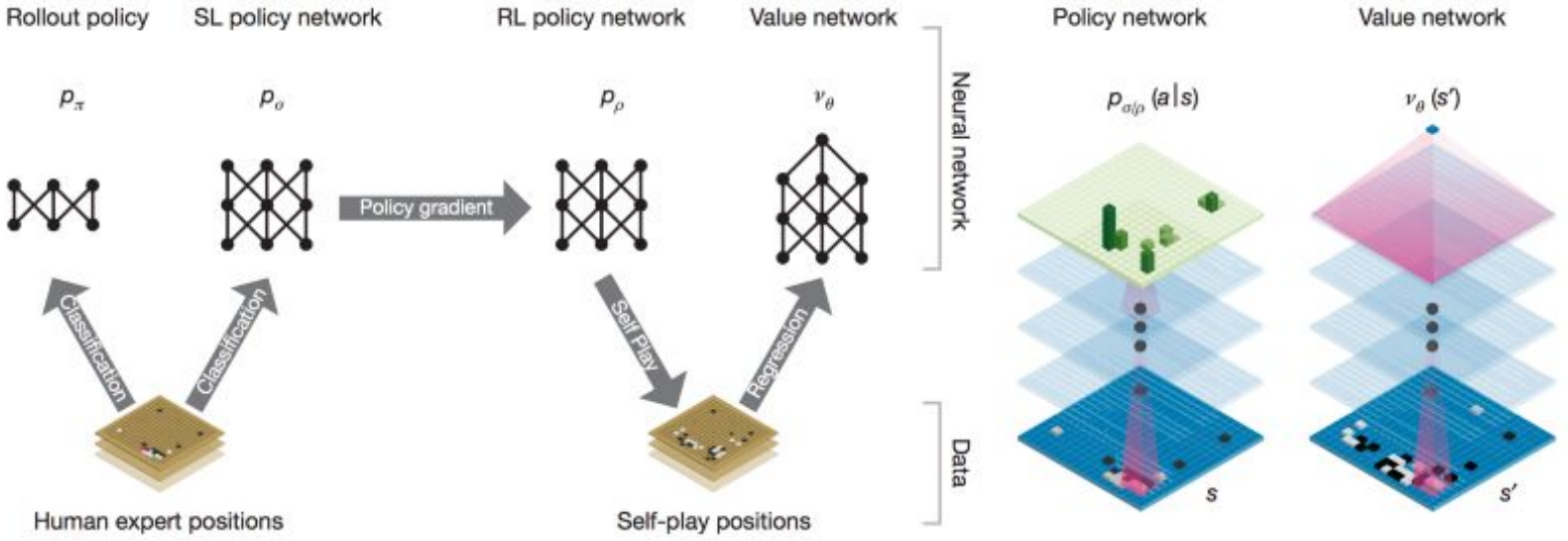






AlphaGo - AlphaGo Zero

- Reducing “action candidates” (Breadth Reduction)
- Board Evaluation (Depth Reduction)
- Looking ahead (Monte Carlo Search Tree)



AlphaGo - AlphaGo Zero

- AlphaGo Zero only uses the black and white stones from the Go board as its input, whereas previous versions of AlphaGo included a small number of hand-engineered features.
- It uses one neural network rather than two. Earlier versions of AlphaGo used a “policy network” to select the next move to play and a ”value network” to predict the winner of the game from each position. These are combined in AlphaGo Zero, allowing it to be trained and evaluated more efficiently.
- AlphaGo Zero does not use “rollouts” - fast, random games used by other Go programs to predict which player will win from the current board position. Instead, it relies on its high quality neural networks to evaluate positions.

Cool, right?

Uber Please

Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such Vashisht Madhavan Edoardo Conti Joel Lehman Kenneth O. Stanley Jeff Clune

Uber AI Labs

{felipe.such, jclune}@uber.com

"Includes results for random search on Atari. Surprisingly, on some games random search substantially outperforms DQN, A3C, and ES, although it never outperforms the GA."

Uber Plea

“Includes results for ran *bstantially outperforms*

Reinforcement Learning

ML/DL Sharing Group
25/01/2018

Adrián Portabales Goberna <adrianportabales@gmail.com>
Pablo Dago Casas <pdago@gradiant.org>