Reinforcement Learning

Learning to play games…
and live in a complex world

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com

sli.do
#DeepLearning

Table of Contents

Reinforcement Learning

Main points

OpenAI Gym

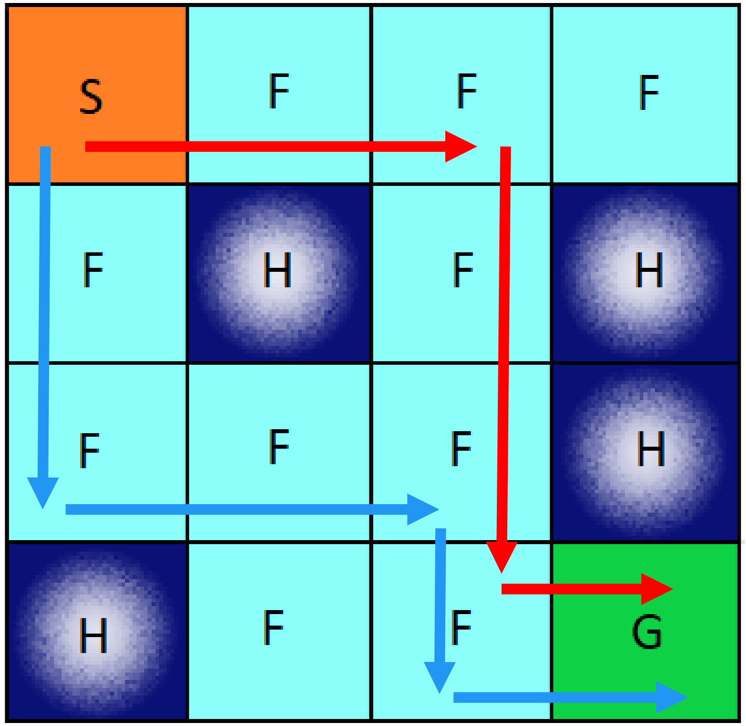
- Install the Python library

```
conda install -c powerai gym
```

- Usage

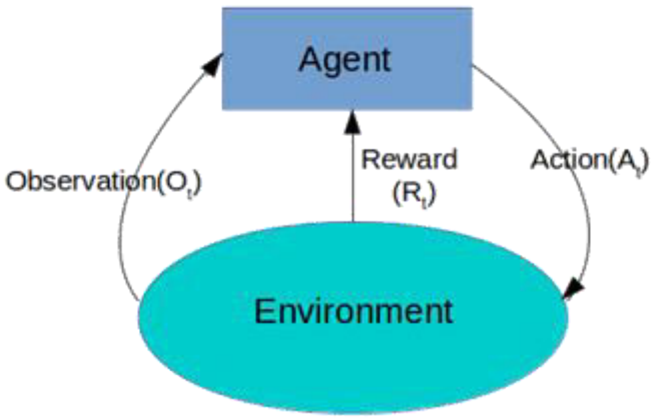
```
import gym
environment = gym.make("FrozenLake-v0")
```

- Goal: Reach cell G
  - [Environment description]
  - Slight complication:
    you don't always go in the direction you're trying to go



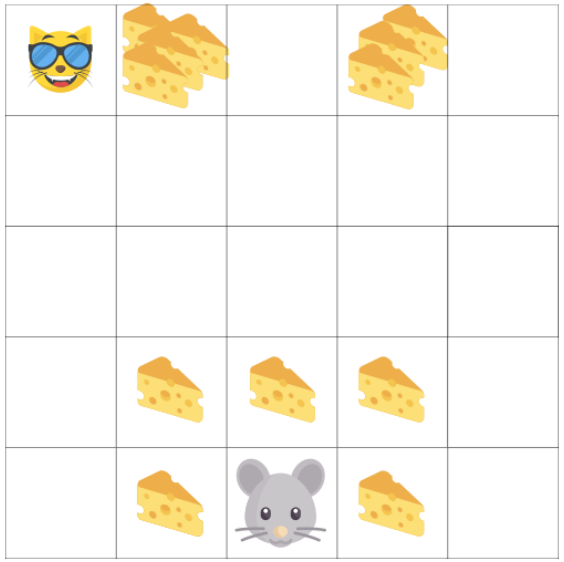
Reinforcement Learning

- Supervised or unsupervised?
  - Feedback system (reward)
  - Sequential learning (time-dependent process)
  - No supervisor; trial and error
  - The agent influences the environment
- Learning process
  - Similar to how children learn
  - Agent learns from environment by performing actions and taking rewards (positive / negative)
- RL loop
  - Observe state $S_i \in S$, S – "state space"
  - Take action $A_i \in A$, A – "action space"
  - Receive reward R_i, update state to S_{i+1}



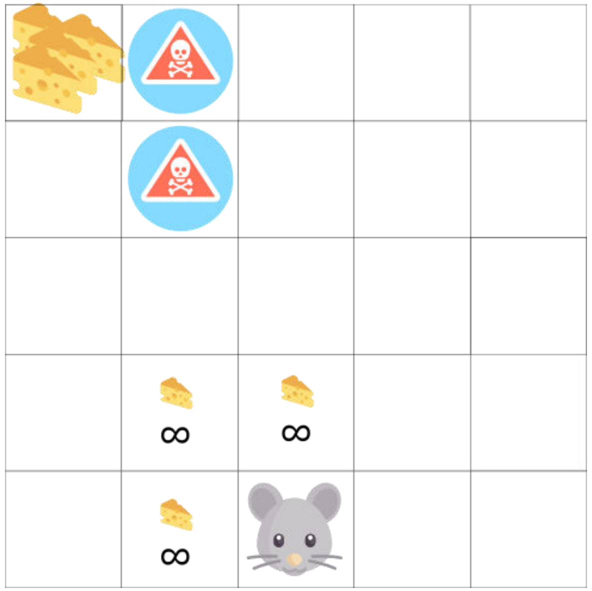
Reinforcement Learning (2)

- Goal: maximize the cumulative reward G
- What doesn't work (every time)
  - Greedy search
  - Hand-coded heuristics
    - I.e. simply sum the rewards at each time step
- Tradeoff: instant gratification vs. later rewards
  - Discount each reward by $\gamma \in [0; 1)$
    - $\gamma \approx 0 \Rightarrow$ short-term rewards get a bigger weight (nearest cheese); and vice versa
  - $G_t = R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots = \sum \gamma^k R_{k+(t+1)}$ at time t



Exploration / Exploitation Tradeoff

- Tradeoff
  - Exploitation: Exploit known information to maximize R
  - Exploration: Find out more information about the environment
- Problem
  - Infinite amount of small cheese vs. one large piece
- Different approaches to avoid this
  - This is the main reason that greedy algorithms cannot perform too well on real-life problems
- Types of RL algorithms
  - Value-based: maximize expected reward V
  - Policy-based: optimize a function $a = \pi(s)$
    - Action = policy with given state

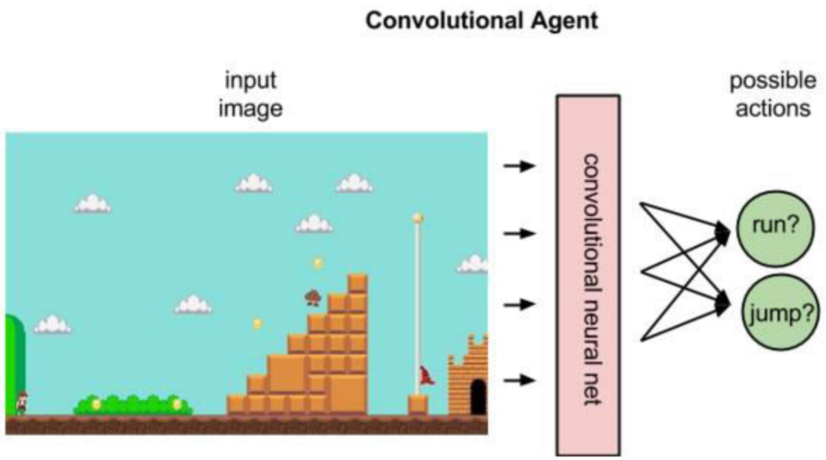


Q-Learning

- Value-based approach
  - Given the state and actions, take the most optimal one
  - Frozen lake: 16 states (cells that you can be in), 4 actions
  - Q-table: 16×4 grid
  - Equation: $Q(S_t, A_t) = (1 - \alpha) Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, A) \right)$
    - α – learning rate
- Training
  - Update the values in the Q-table by playing a lot of games
- How about a different game?
  - Example: 10 000 states, 10 actions
    - Tables quickly become exponentially big
    - We need a lot of games

Deep Q-Learning

- Solution: use an NN as a function approximator
  - Doesn't even need to be recurrent!
- Loss / cost function
  - MSE: $J = \sum(\tilde{Q} - Q)^2$
- Output: $\tilde{Q} = \mathbb{R}^A$
- For games where the state is a screen image, it's useful to add convolutional layers at the beginning

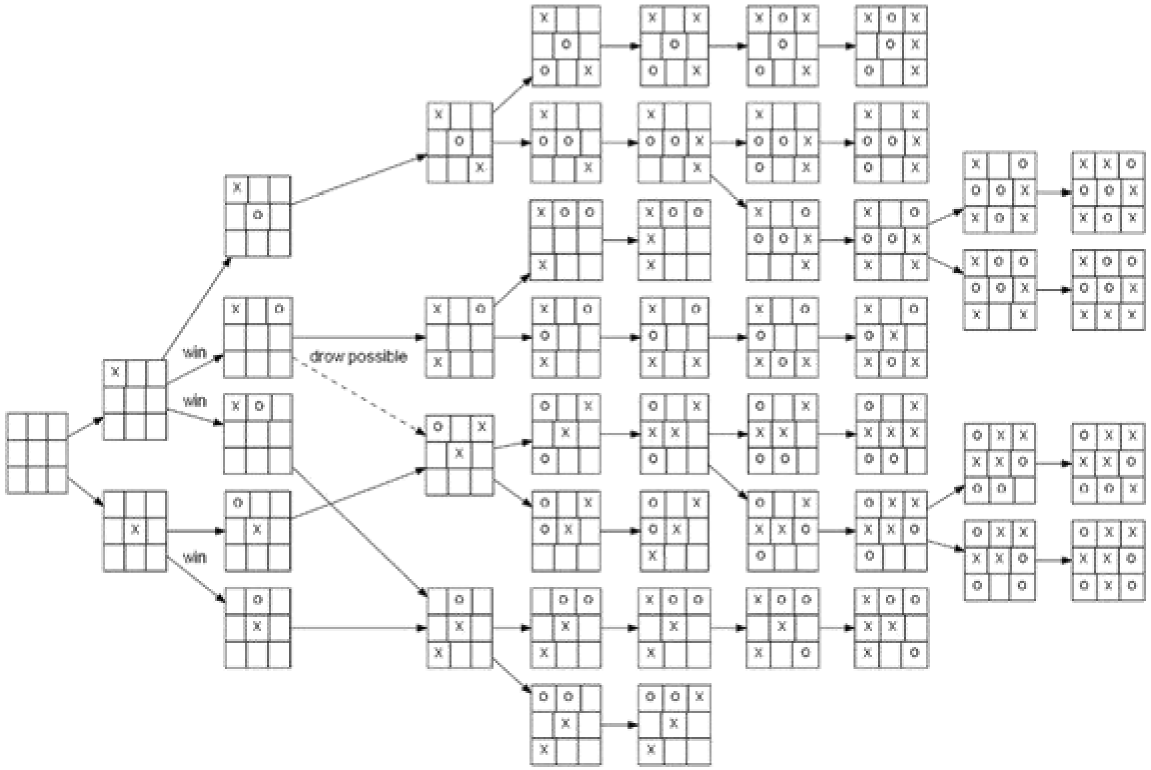


Playing Games

AlphaGo and its variations

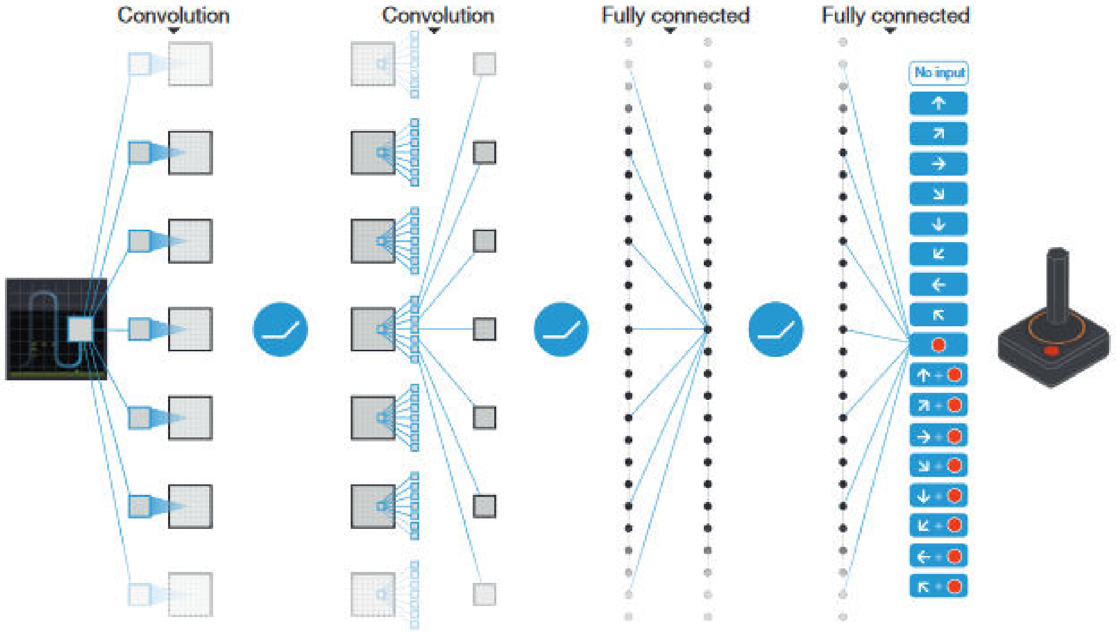
Two-player Games

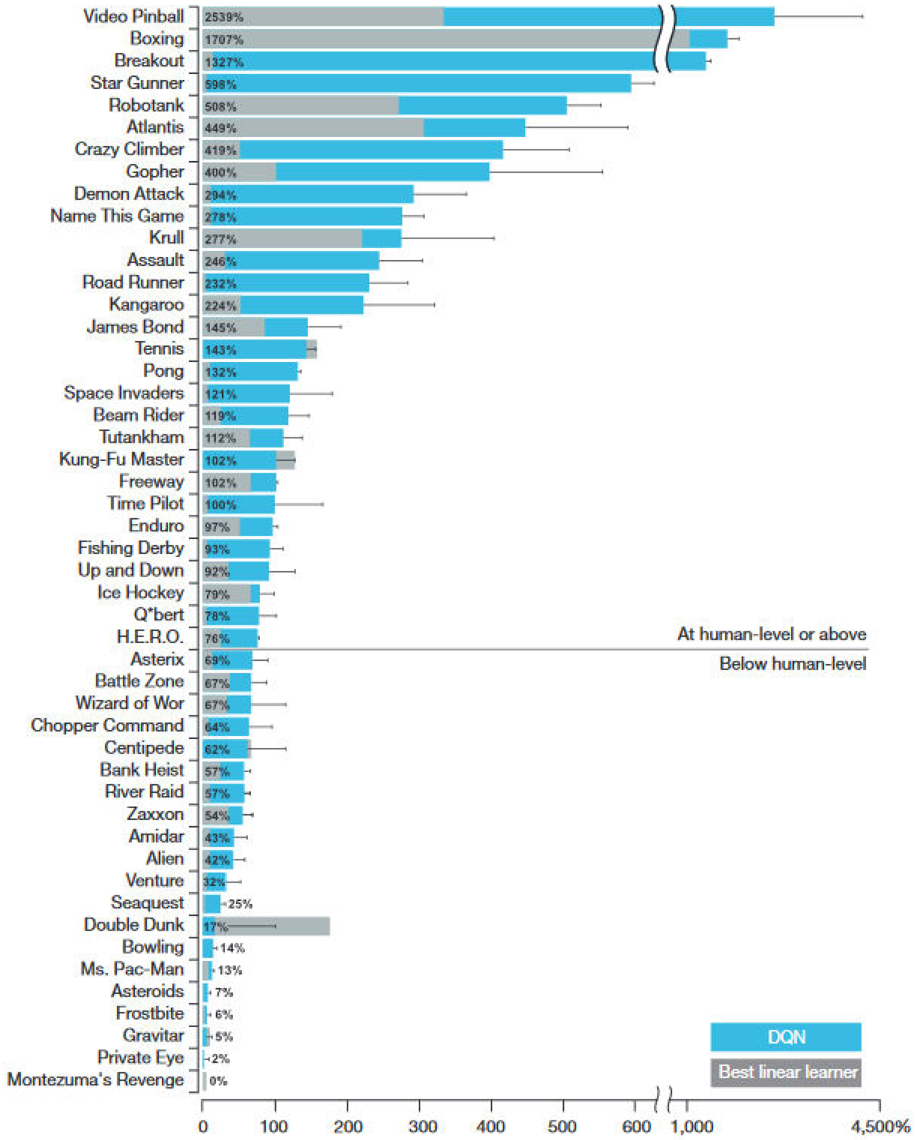
- Many different approaches to two-player games
  - We already saw how GANs learn: minimax algorithm
  - Idea: maximize your own reward while minimizing the opponent's reward
  - For small games, this is viable
    - Chess: $\sim 10^{120}$ nodes
      - Atoms in the Universe: $\sim 10^{80}$
  - One optimization: build the tree for, say 4 moves in advance
    - Exploration / exploitation tradeoff
  - A variant: simulate
    - Monte Carlo tree search



Deep-Q Networks

- Mnih et al., 2013; Mnih et al., 2015





AlphaGo

- Experience replay
  - Uses mini-batches to update Q values
  - Prevents overfitting (the network tends to play similar games)
- Target network
  - Doesn't update NN parameters (red rectangle) every step
    - Because they are unstable

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

  - Instead, updates them every 1000 steps
- Clipping rewards
  - Different games have different reward ranges; clip $R \in [-1; 1]$
- Skipping frames
  - Humans don't perform at 60fps $\Rightarrow$ we can go away with a smaller NN
  - Uses 4 frames at a time

AlphaGo (2)

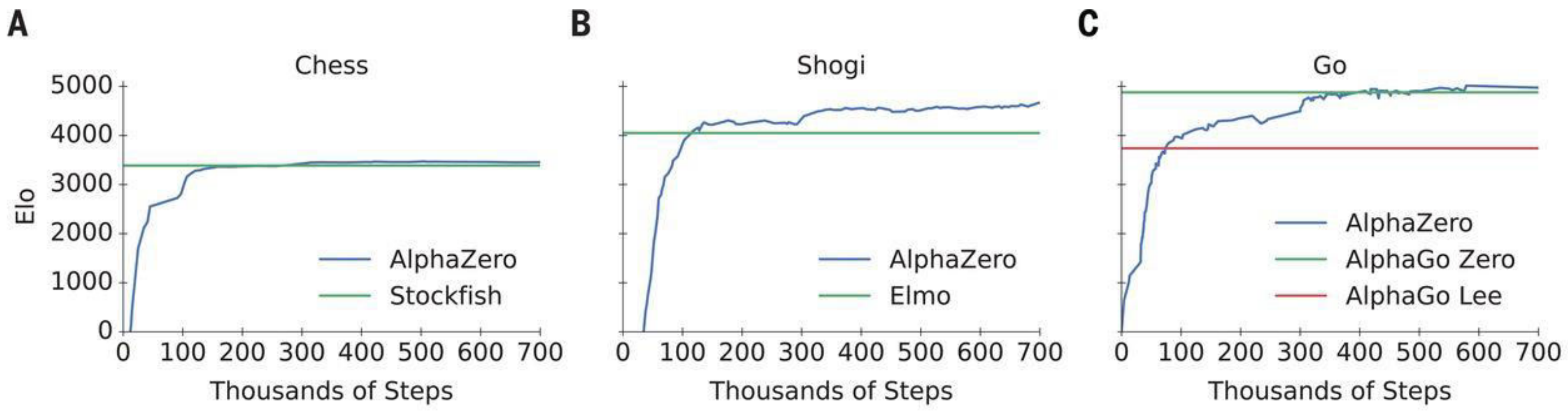
- Performance w.r.t. experience replay / target network

Replay	◯	◯	×	×
Target	◯	×	◯	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

- Results (2015)

	Breakout	R. Raid	Enduro	Sequest	S. Invaders
DQN	**316.8**	**7446.6**	**1006.3**	**2894.4**	**1088.9**
Naive DQN	3.2	1453.0	29.1	275.8	302.0
Linear	3.0	2346.9	62.0	656.9	301.3

- Silver et al., Science, 07.12.2018



Applications of DQNs

- Snake
- Small board games (AlphaToe)
- CNNs for OpenAI Gym
- LSTMs with attention
- DeepMind's DQN papers
  - Silver et al., 2017: AlphaGo Zero
- A3C algorithm, tensorflow

State of RL

- Key RL papers
  - Pay attention to " 12. Reproducibility, Analysis, and Critique "
- Notes on important papers
- NeurIPS 2019
- Overview of deep RL algorithms (Ivanov, 2019)
- Some interesting applications
  - Painting like a human (RL-style GAN), Huang et al., 2019
  - Traffic control (Guo & Wang, 2019)
  - Molecular dynamics (Zhou et al., 2019)
  - Recommenders (online ads), Zhao et al., 2019

"Specification gaming"

- Source
  - The hardest step in optimization is to choose the correct reward function
  - A wrongly or poorly chosen reward tends to create algorithms which cheat
- Examples
  - Creatures bred for speed grow really tall and generate high velocities by falling over
  - Simulated pancake making robot learned to throw the pancake as high in the air as possible in order to maximize time away from the ground
  - Agent kills itself at the end of level 1 to avoid losing in level 2
  - Self-driving car rewarded for speed learns to spin in circles
  - Agent pauses the game indefinitely to avoid losing

Summary

- Problem description
- Approaches
- Deep-Q networks
- AlphaGo
- "Specification gaming"

Questions?