Machine Learning Graz

TU Graz
Graz University of Technology

An Introduction to Reinforcement Learning

Anand Subramoney

anand [at] igi.tugraz.at

Institute for Theoretical Computer Science, TU Graz

http://www.igi.tugraz.at/

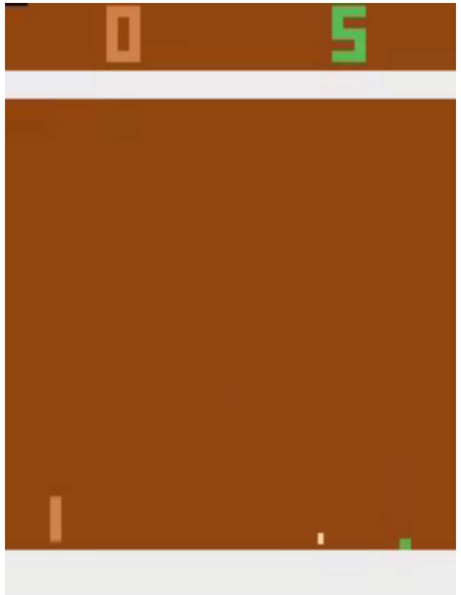
Machine Learning Graz Meetup
12th October 2017

Outline

- Introduction
- Value estimation
- Q-learning
- Policy gradient
- DQN
- A3C

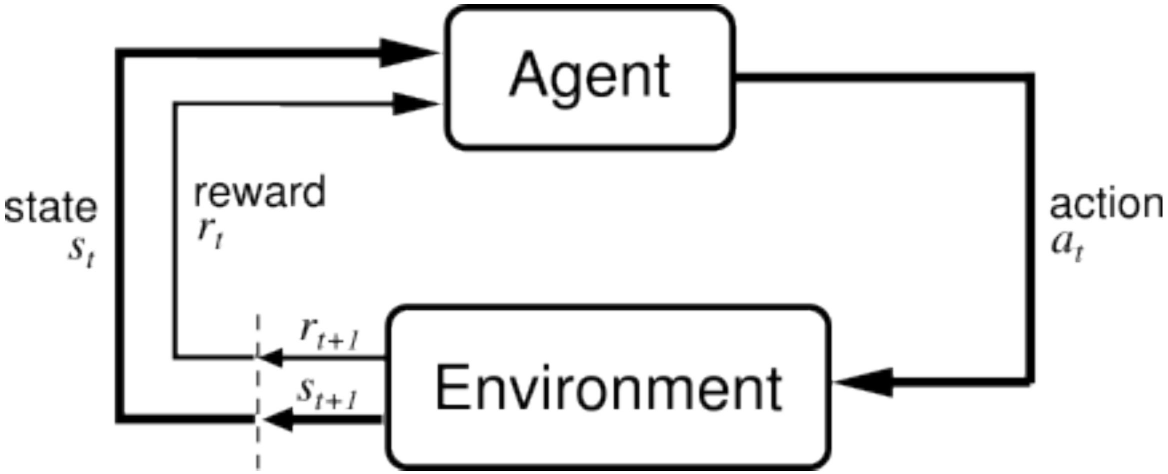
What is Reinforcement Learning?

- Learning an agent while **interacting** with the environment
- The agent receives a “**reward**” for each action it takes
- The goal of the agent is to **maximize the reward** it receives
- The agent is not told what the ”right” action is. i.e. it is not supervised



Notation

- The state of the environment is s_t at time t
  - Examples of state: the (x, y) coordinates, image pixels etc.
- At each time step t, the agent takes action a_t (knowing s_t)
  - Examples of action: Move right/left/up/down, acceleration of car etc.
- Then the agent gets a reward r_t
  - Could be 0/1 or points in the game
- The agent plays for one "episode"
  - Called "episodic" RL
  - E.g. one game until it wins/loses etc.
  - Non-episodic also possible



Notation

- Model: $\mathcal{P}_{ss'}^{a} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$
  - What is the next state given the current state and action taken?
  - The environment can be stochastic, in which case this is a probability distribution

- Reward: $\mathcal{R}_{ss'}^{a} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$
  - Expected value of reward when going from one state to another taking a certain action
  - In the most general case, the reward is not deterministic

Policy

- The agent has a certain mapping between state and action
- This is called the **policy** of the agent
- Denoted by $\pi(s, a)$
  - In the stochastic case, it's the probability distribution over actions at a given state $\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathrm{P}(\boldsymbol{a}_t | \boldsymbol{s}_t)$

The goal of reinforcement learning

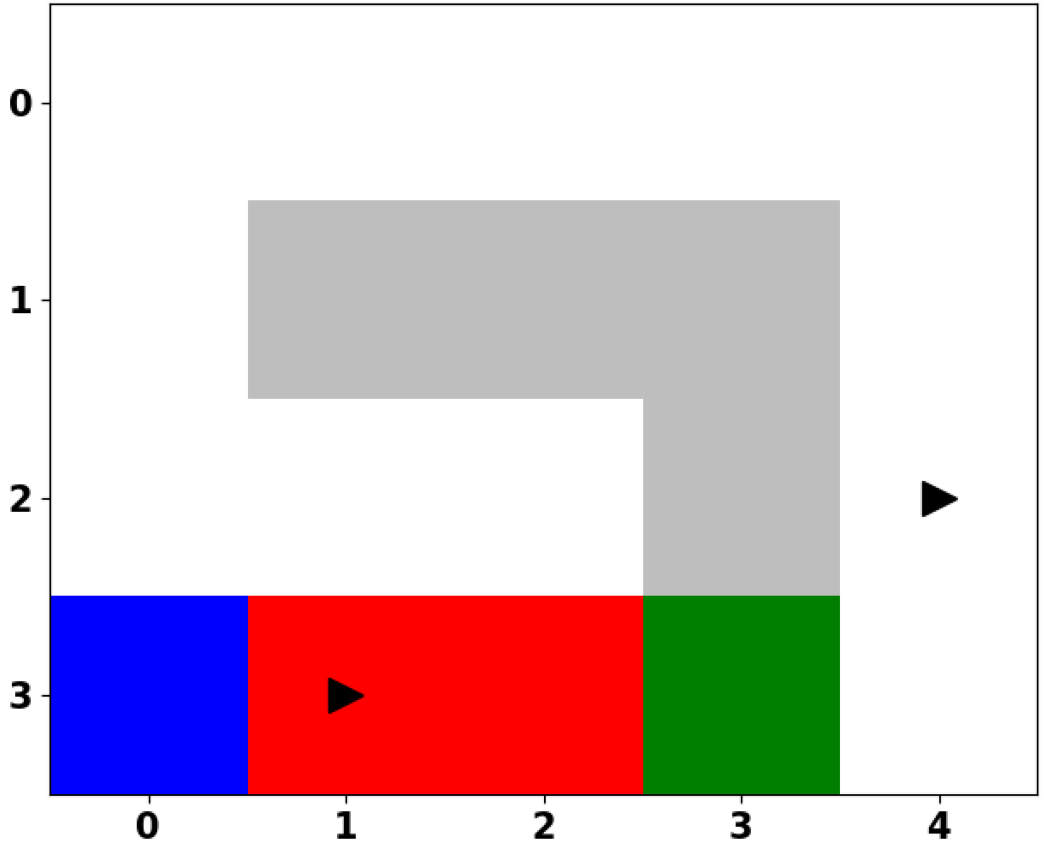
- Is to find a policy that maximizes the total expected reward
  - also called the "return"

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- In an episode
- γ is called the "discounting factor"

- Small γ produces shortsighted, large γ far-sighted policies.
- R is always finite if $\gamma < 1$ and the local rewards r are from a bounded set of numbers.

Example environment



The agent receives -0.001 reward every step. When it reaches the goal or a pit, it obtains rewards of +1.0 or -1.0 resp. and the episode is terminated.

The goal of reinforcement learning

- How can the agent quantify the desirability of intermediate states (where no, or no relevant reward is given)?

- The difficulty is, that the desirability of intermediate states depends on:
  - The concrete selection of actions AFTER being in such an intermediate state,
  - AND on the desirability of subsequent intermediate states.

- The value function allows us to do this

The value function

- Defined as:
  - $V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$
- The value of a state s is the expected return starting from that state s and following policy π
- Satisfies the Bellman equations

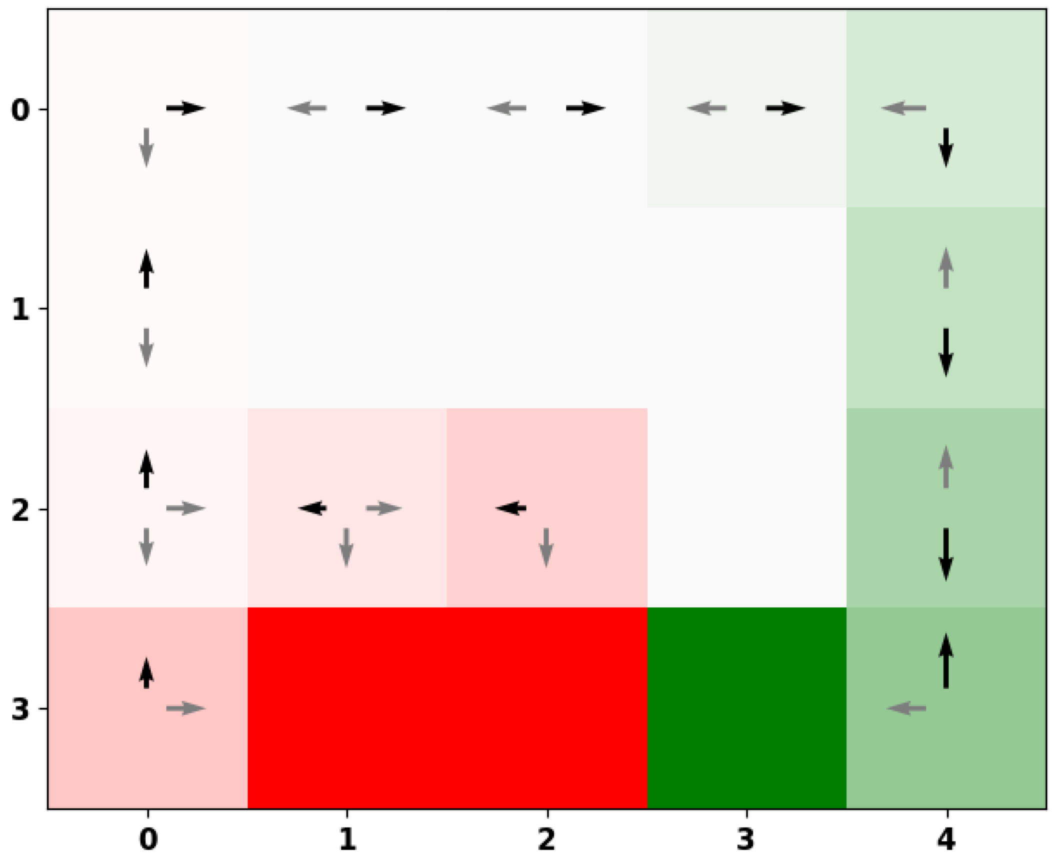
Bellman equation for V^{π} :

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V^{\pi}(s') \right]$$

– a system of $|S|$ simultaneous linear equations

Note that it's a recursive formulation of the value function

Example value function



Calculating the value function

- If the model $\mathcal{P}_{ss'}^{a}$ and reward $\mathcal{R}_{ss'}^{a}$ are known, calculate $V^{\pi}(s)$ using iterative policy evaluation.

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^{a} [\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^{\pi}$

http://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Why value function?

- There exists a natural partial order on all possible policies:

$$\pi' \geq \pi \; if \; and \; only \; if \; V^{\pi'}(s) \geq V^{\pi}(s) \; for \; all \; s \in S$$

- **Definition:** A policy π' is called optimal if $\pi' \geq \pi$ for all policies π

- Existence of at least one optimal policy is guaranteed, and they satisfy Bellman Optimality equations.

The action-value function

- Defined as:
  - $Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\,, a_t = a\}$

- This is called the "Q function"
- The value of taking action a in state s following policy π thereafter
- Also satisfies the Bellman equations

$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \middle| s_t = s, a_t = a \right\} \\
&= \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V^{\pi}(s') \right]
\end{aligned}
$$

Finding an optimal policy

- Define a new policy π' that is greedy with respect to V^{π}
- For all states s: $\pi' = argmax_a Q^{\pi}(s, a)$
- This policy satisfies $Q^{\pi}\left(s, \pi'(s)\right) \geq V^{\pi}(s)$
- Can be shown that:
  - $\pi' \geq \pi$ for $\gamma < 1$
  - Eventually converges to an optimal policy

- This works only if $V^{\pi}(s)$ can be calculated

Other ways to calculate V/Q

- Monte-carlo policy evaluation
  - Sample one episode and update the value function for each state
  - $V(s_t) \longleftarrow V(s_t) + \alpha\left(R_t - V(s_t)\right)$
  - Asymptotically converges to the true value function

- Temporal Difference (TD) Learning
  - For each step of each episode:
  - Take action a, observe reward r_{t+1} and next state s_{t+1}
  - $V(s_t) \longleftarrow V(s_t) + \alpha(\underbrace{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}_{\text{Temporal Difference}})$

Learning Q-function (SARSA)

- Q can be used to define a policy
  - take action a = $argmax_a Q(s, a)$ at every state with probability $1 - \epsilon$
  - With probability ϵ take a random action (exploration)

- Use temporal difference learning to learn Q-function
  - For each step of each episode:
  - Take action a, observe reward $r_{t+1}$and next state s_{t+1}
  - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}\ ) \ \ - Q(s_t, a_t))$

- $a_{t+1}$for learning can be used from this policy
- Called SARSA

Q-learning

- Use temporal difference learning to learn Q-function
  - For each step of each episode:
  - Take action a, observe reward r_{t+1} and next state s_{t+1}
  - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q\,(s_{t+1}, a\,) \;\; - Q(s_t, a_t))$

- Q-learning requires for convergence to the optimal policy that rewards are sampled for each pair (s, a) infinitely often.

- http://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Function approximation

- The Q-function can be approximated with a neural network (or any other function approximator)
- The targets for the network would be

$$r_{t+1} + \gamma \max_a Q\,(s_{t+1}, a\,)$$

- Train the neural network with backpropagation

The goal of reinforcement learning (repeated)

- Is to find a policy that maximizes the total expected reward
  - also called the "return"

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- γ is called the "discounting factor"

- Small γ produces shortsighted, large γ far-sighted policies.
- R is always finite if $\gamma < 1$ and the local rewards r are from a bounded set of numbers.

Policy Gradient

- Why not learn the policy directly?
- Define cost function as the total expected reward:

$$J(\theta) = E\left\{\sum_{k=0}^{H} a_k r_k\right\} = E\{r(\tau)\}$$

  - a_k is some discounting factor
  - r_k is reward at step k
  - τ is a trajectory and $r(\tau)$ = $\sum_{k=0}^{H} a_k r_k$

- Learn this using gradient ascent:

$$\theta_{t+1} = \theta_t + \eta \nabla_\theta J(\theta)$$

- Problems?
  - Cannot calculate gradient of J

Policy Gradient

- It is possible to empirically estimate the gradient (Williams 1992)

$$\nabla_\theta J(\theta) = E\{\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)\}$$

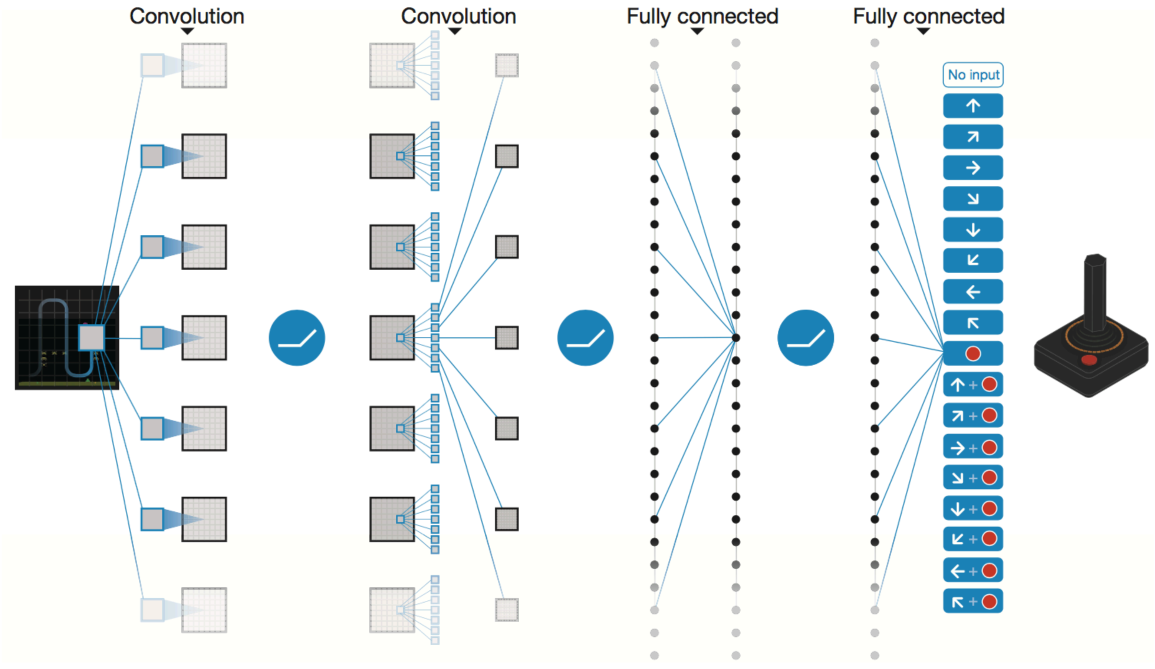
$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ (R_t - b)$$

- Uses the log-likelihood trick (or REINFORCE trick)
- Baseline is used to reduce variance of gradient estimator
- Baseline doesn’t introduce bias
- DEMO

DQN and A3C

DQN

- Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
- Uses a deep neural network to learn the Q-values

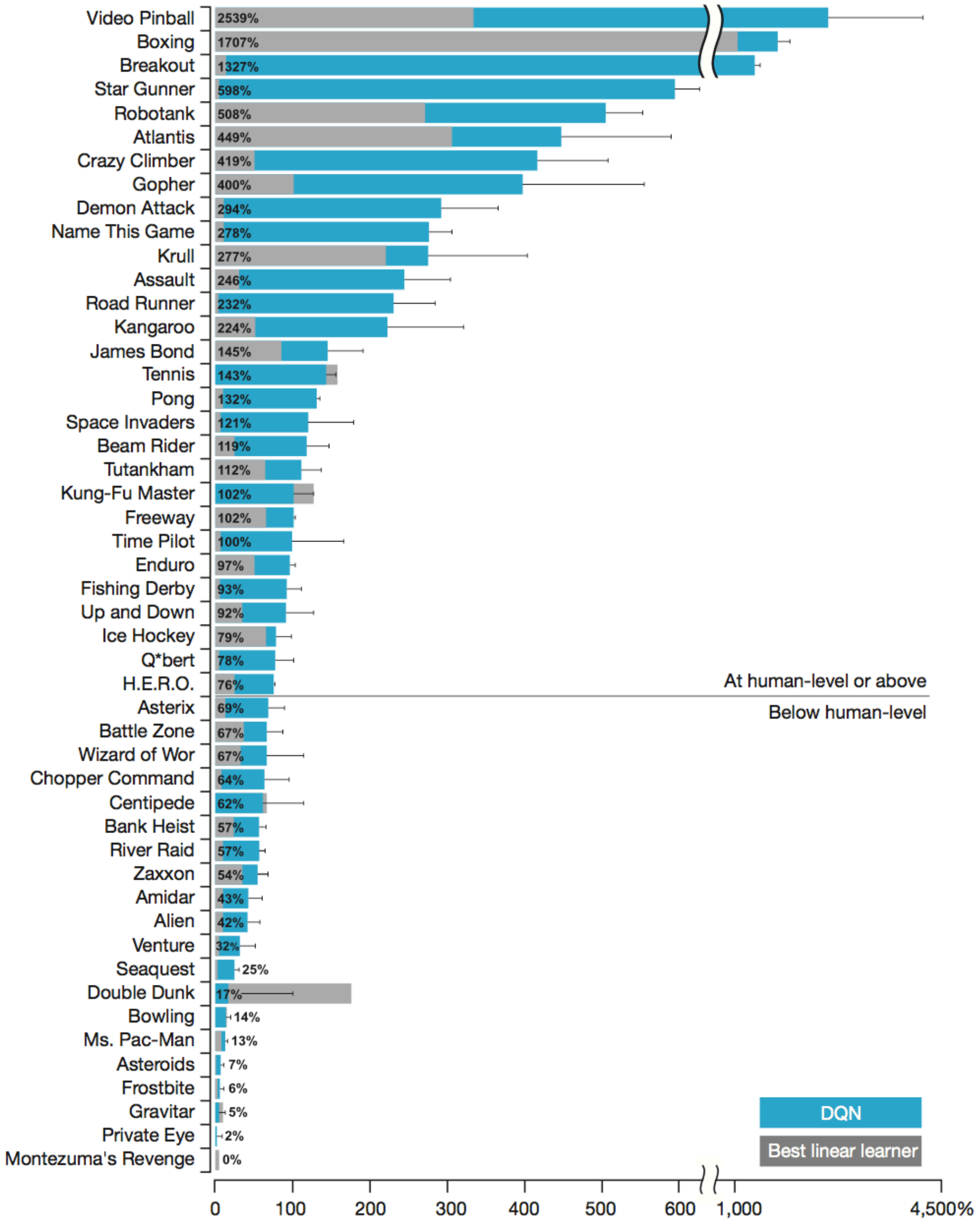


DQN: Two key ideas

- Episode replay:
  - Store earlier steps and apply Q-learning updates in random batches from this memory
- Update policy network only once every C steps

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i) \right)^2 \right]$$

DQN



Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

A3C

- Mnih, V. *et al.* Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]* (2016).

- A3C: Asynchronous Advantage Actor Critic
- Uses policy gradient with a baseline that is the value function

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \underbrace{\log \pi_\theta(a_t|s_t)}_{\text{Actor}} \; \underbrace{(R_t - \overbrace{V(s_t)}^{\text{Critic}})}_{\text{Advantage}}$$

A3C

Game	DQN	Gorila	Double	Dueling	Prioritized	A3C FF, 1 day	A3C FF	A3C LSTM
Alien	570.2	813.5	1033.4	**1486.5**	900.5	182.1	518.4	945.3
Amidar	133.4	189.2	169.1	172.7	218.4	**283.9**	263.9	173.0
Assault	3332.3	1195.8	6060.8	3994.8	7748.5	3746.1	5474.9	**14497.9**
Asterix	124.5	3324.7	16837.0	15840.0	**31907.5**	6723.0	22140.5	17244.5
Asteroids	697.1	933.6	1193.2	2035.4	1654.0	3009.4	4474.5	**5093.1**
Atlantis	76108.0	629166.5	319688.0	445360.0	593642.0	772392.0	**911091.0**	875822.0
Bank Heist	176.3	399.4	886.0	**1129.3**	816.8	946.0	970.1	932.8
Battle Zone	17560.0	19938.0	24740.0	**31320.0**	29100.0	11340.0	12950.0	20760.0
Beam Rider	8672.4	3822.1	17417.2	14591.3	**26172.7**	13235.9	22707.9	24622.2
Berzerk			1011.1	910.6	1165.6	**1433.4**	817.9	862.2
Bowling	41.2	54.0	**69.6**	65.7	65.8	36.2	35.1	41.8
Boxing	25.8	74.2	73.5	**77.3**	68.6	33.7	59.8	37.3
Breakout	303.9	313.0	368.9	411.6	371.6	551.6	681.9	**766.8**
Centipede	3773.1	**6296.9**	3853.5	4881.0	3421.9	3306.5	3755.8	1997.0
Chopper Comman	3046.0	3191.8	3495.0	3784.0	6604.0	4669.0	7021.0	**10150.0**
Crazy Climber	50992.0	65451.0	113782.0	124566.0	131086.0	101624.0	112646.0	**138518.0**
Defender			27510.0	33996.0	21093.5	36242.5	56533.0	**233021.5**
Demon Attack	12835.2	14880.1	69803.4	56322.8	73185.8	84997.5	113308.4	**115201.9**
Double Dunk	-21.6	-11.3	-0.3	-0.8	**2.7**	0.1	-0.1	0.1
Enduro	475.6	71.0	1216.6	**2077.4**	1884.4	-82.2	-82.5	-82.5
Fishing Derby	-2.3	4.6	3.2	-4.1	9.2	13.6	18.8	**22.6**
Freeway	25.8	10.2	**28.8**	0.2	27.9	0.1	0.1	0.1
Frostbite	157.4	426.6	1448.1	2332.4	**2930.2**	180.1	190.5	197.6
Gopher	2731.8	4373.0	15253.0	20051.4	**57783.8**	8442.8	10022.8	17106.8
Gravitar	216.5	**538.4**	200.5	297.0	218.0	269.5	303.5	320.0
H.E.R.O.	12952.5	8963.4	14892.5	15207.9	20506.4	28765.8	**32464.1**	28889.5
Ice Hockey	-3.8	-1.7	-2.5	-1.3	**-1.0**	-4.7	-2.8	-1.7
James Bond	348.5	444.0	573.0	835.5	**3511.5**	351.5	541.0	613.0
Kangaroo	2696.0	1431.0	**11204.0**	10334.0	10241.0	106.0	94.0	125.0
Krull	3864.0	6363.1	6796.1	8051.6	7406.5	**8066.6**	5560.0	5911.4
Kung-Fu Master	11875.0	20620.0	30207.0	24288.0	31244.0	3046.0	28819.0	**40835.0**
Montezuma's Revenge	50.0	**84.0**	42.0	22.0	13.0	53.0	67.0	41.0
Ms. Pacman	763.5	1263.0	1241.3	**2250.6**	1824.6	594.4	653.7	850.7
Name This Game	5439.9	9238.5	8960.3	11185.1	11836.1	5614.0	10476.1	**12093.7**
Phoenix			12366.5	20410.5	27430.1	28181.8	52894.1	**74786.7**
Pit Fall			-186.7	-46.9	**-14.8**	-123.0	-78.5	-135.7
Pong	16.2	16.7	**19.1**	18.8	18.9	11.4	5.6	10.7
Private Eye	298.2	**2598.6**	-575.5	292.6	179.0	194.4	206.9	421.1
Q*Bert	4589.8	7089.8	11020.8	14175.8	11277.0	13752.3	15148.8	**21307.5**
River Raid	4065.3	5310.3	10838.4	16569.4	**18184.4**	10001.2	12201.8	6591.9
Road Runner	9264.0	43079.8	43156.0	58549.0	56990.0	31769.0	34216.0	**73949.0**
Robotank	58.5	61.8	59.1	**62.0**	55.4	2.3	32.8	2.6
Seaquest	2793.9	10145.9	14498.0	37361.6	**39096.7**	2300.2	2355.4	1326.1
Skiing			-11490.4	-11928.0	**-10852.8**	-13700.0	-10911.1	-14863.8
Solaris			810.0	1768.4	**2238.2**	1884.8	1956.0	1936.4
Space Invaders	1449.7	1183.3	2628.7	5993.1	9063.0	2214.7	15730.5	**23846.0**
Star Gunner	34081.0	14919.2	58365.0	90804.0	51959.0	64393.0	138218.0	**164766.0**
Surround			1.9	**4.0**	-0.9	-9.6	-9.7	-8.3
Tennis	-2.3	-0.7	-7.8	**4.4**	-2.0	-10.2	-6.3	-6.4
Time Pilot	5640.0	8267.8	6608.0	6601.0	7448.0	5825.0	12679.0	**27202.0**
Tutankham	32.4	118.5	92.2	48.0	33.6	26.1	**156.3**	144.2
Up and Down	3311.3	8747.7	19086.9	24759.2	29443.7	54525.4	74705.7	**105728.7**
Venture	54.0	**523.4**	21.0	200.0	244.0	19.0	23.0	25.0
Video Pinball	20228.1	112093.4	367823.7	110976.2	374886.9	185852.6	331628.1	**470310.5**
Wizard of Wor	246.0	10431.0	6201.0	7054.0	7451.0	5278.0	17244.0	**18082.0**
Yars Revenge			6270.6	**25976.5**	5965.1	7270.8	7157.5	5615.5
Zaxxon	831.0	6159.4	8593.0	10164.0	9501.0	2659.0	**24622.0**	23519.0

Resources

- Book: **Reinforcement Learning** An Introduction, *Richard Sutton and Andrew Barto*
  - Available online on Andrew Barto's website: http://www.incompleteideas.net/sutton/book/the-book-1st.html
- Course: Autonomously Learning Systems IGI TU Graz
  - 2016 website: http://www.igi.tugraz.at/lehre/Autonomously_learning_systems/WS16/
  - Next course in 2018
  - Lecture slides available there
- DQN: https://deepmind.com/research/dqn/
- OpenAI Gym: https://gym.openai.com/envs
- Deep Reinforcement Learning: Pong from Pixels (Andrej Karpathy): https://karpathy.github.io/2016/05/31/rl/
- Book: **Deep Learning**, *Ian Goodfellow, Yoshua Bengio and Aaron Courville*
  - *Available online: http://www.deeplearningbook.org*
- *RLPy: https://rlpy.readthedocs.io/en/latest/ (python 2.7 only)*