

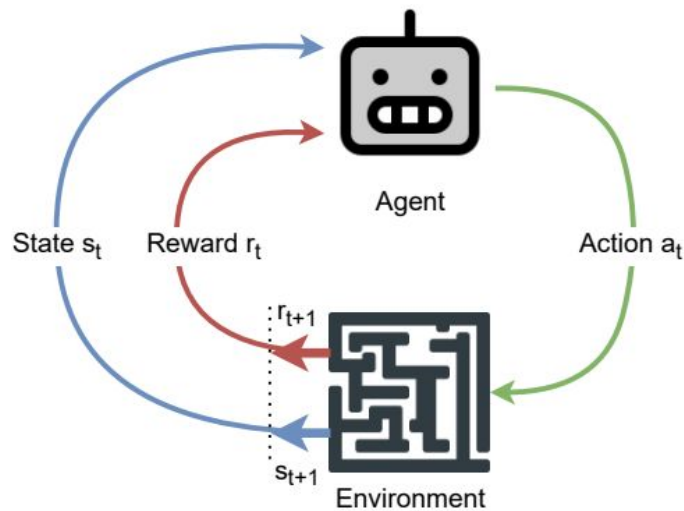
Deep Learning

Lecture 9

somehow intersects with a good (theory) RL course at HSE University (http://wiki.cs.hse.ru/Reinforcement_learning_2022_2023)
and with a good RL course at AI Masters (<https://ozonmasters.ru/reinforcementlearning>)

Lecturer: Daniil Tiapkin (<https://d-tiapkin.github.io/>)

Reinforcement Learning: previous lecture



Small remark: Why we care about RL?

Solving games is good, but why we need it in a real life?

- Solving VERY hard combinatorial problems:

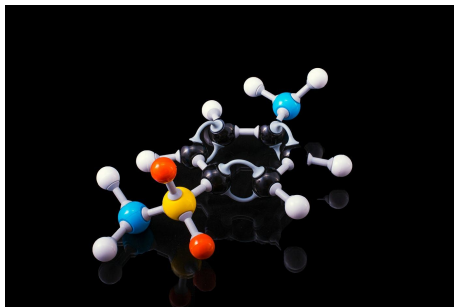
Lazic et al. “Data center cooling using model-predictive control”,
<https://research.google/pubs/pub47493/>

- Reinforcement Learning from Human Feedback

Everybody knows about ChatGPT!

- Generative Flow Networks

very recent preprint: <https://arxiv.org/abs/2310.12934>
tldr: complex molecule generation (drug discovery)
via a specific type of RL algorithms

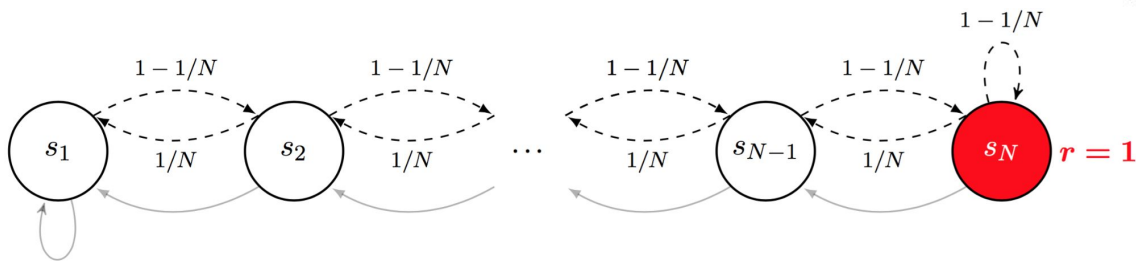
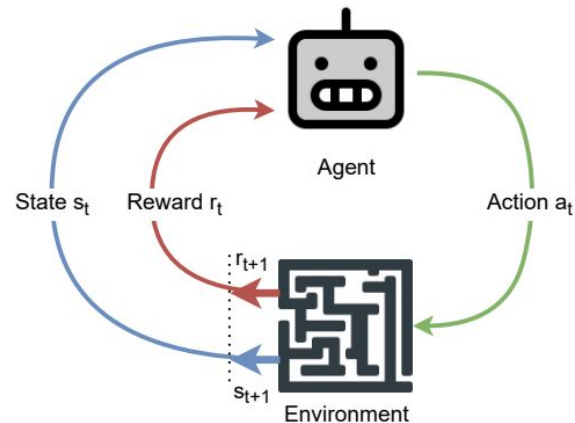


Recap: Markov Decision Process

Markov Decision Process is a 5-tuple (S, A, P, R, γ)

- S - state space;
- A - action space;
- $P(s' | s, a)$ - transition probability kernel;
- $R(s, a)$ - reward distribution (with a mean reward $r(s, a)$);
- γ - discounting factor

Policy π : rule to choose next action given current state



MDP Example: Chain

Recap: Value function and Q-function

Goal of the agent: find a policy π that maximizes the expected sum of rewards:

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \quad Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right],$$

$$r_t \sim R(\cdot | s_t, a_t), s_{t+1} \sim P(\cdot | s_t, a_t), a_t \sim \pi(\cdot | s_t).$$

A policy that attains maximum for each states is called *optimal*.

Recap: Bellman equations

Idea: Q-value of the optimal policy could be written as a solution to a large system of equations:

$$Pf(s, a) = \mathbb{E}_{s' \sim P(s, a)}[f(s')].$$

$$\begin{aligned} Q^\pi(s, a) &= r(s, a) + \gamma PV^\pi(s, a), & Q^\star(s, a) &= r(s, a) + \gamma PV^\star(s, a), \\ V^\pi(s) &= \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi(a, s) & V^\star(s) &= \max_{a \in \mathcal{A}} Q^\star(s, a) \end{aligned}$$

In particular, it implies that it is enough to solve the system of equations (see the right)!

Q: How to solve it?

Value Iteration

We want: some iterative solver for optimal Bellman equations

Approach: treat as a fixed point!

Algorithm 2 Value Iteration for discounted MDPs

Input: MDP $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, the immediate reward function r , iteration budget T ;

Initialize: $Q^0(s, a) = 0$, $V^0(s) = 0$;

for $t = 1, \dots, T$ **do**

$$Q^t(s, a) := r(s, a) + \gamma P V^{t-1}(s, a) \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A};$$

$$V^t(s) := \max_{a \in \mathcal{A}} Q^t(s, a) \quad \forall s \in \mathcal{S};$$

$$\pi^t(s) := \arg \max_{a \in \mathcal{A}} Q^t(s, a) \quad \forall s \in \mathcal{S}.$$

end for

Output: approximately optimal policy π^T .

Q: What are practical problems of the presented algorithm?

Problems of Value Iteration

1. The model is unknown, we cannot compute expectations!



2. Even if we can compute expectations, typically is it impossible to store all Q-values!

Let's focus on that!

Small remark: small finite MDPs sometimes are called “tabular” because Q-values are stored in a table in this case

A way to practical algorithm...

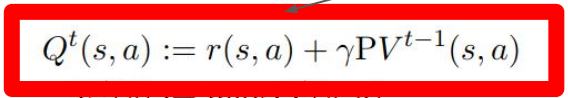
Idea: reformulate VI as a single optimization problem.

Algorithm 2 Value Iteration for discounted MDPs

Input: MDP $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, the immediate reward function r , iteration budget T ;

Initialize: $Q^0(s, a) = 0$, $V^0(s) = 0$;

for $t = 1, \dots, T$ **do**


$$Q^t(s, a) := r(s, a) + \gamma P V^{t-1}(s, a)$$

$$V^t(s) := \max_{a \in \mathcal{A}} Q^t(s, a)$$

$$\pi^t(s) := \arg \max_{a \in \mathcal{A}} Q^t(s, a)$$

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A};$$

$$\forall s \in \mathcal{S};$$

$$\forall s \in \mathcal{S}.$$

end for

Output: approximately optimal policy π^T .

How to rewrite this line as an optimization problem?

Rewrite one line a little bit...

We have: $Q^t(s, a) = r(s, a) + \gamma P V^{t-1}(s, a)$

But everything here is just expectations, let us rewrite it like that:

$$Q^t(s, a) = \mathbb{E}_{r \sim R(s, a), s' \sim P(s, a)} [r + \gamma V^{t-1}(s')]$$

Computation of expectation could be rewritten as a least-squares problem!

$$Q^t(s, a) = \arg \min_{q \in \mathbb{R}} \mathbb{E}_{r \sim R(s, a), s' \sim P(s, a)} \left[(r + \gamma V^{t-1}(s') - q)^2 \right]$$

How to aggregate all the state-action pairs?

Now, for each state-action pair we have to solve separate problem:

$$Q^t(s, a) = \arg \min_{q \in \mathbb{R}} \mathbb{E}_{r \sim R(s, a), s' \sim P(s, a)} \left[\left(r + \gamma V^{t-1}(s') - q \right)^2 \right]$$

Q: How to make generate a single problem?

Least Squares Value Iteration (LSVI)

Let us define a distribution over all state-action pairs D such that $D(s,a) > 0$ for all s,a .

Then we can rewrite one iteration as follows:

$$Q^t = \arg \min_{q: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}} \mathbb{E}_{(s,a) \sim D} \mathbb{E}_{r \sim R(s,a), s' \sim P(s,a)} \left[\left(r + \gamma V^{t-1}(s') - q(s,a) \right)^2 \right]$$

Or, alternatively

$$Q^t = \arg \min_{q: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}} \mathbb{E}_{(s,a) \sim D} \mathbb{E}_{r \sim R(s,a), s' \sim P(s,a)} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q^{t-1}(s', a') - q(s,a) \right)^2 \right]$$

The final algorithm is called *Least-Squares Q-Value Iteration* (also known as *Fitted Q-Value iteration*)

How to make LSVI practical?

Step 1. Replace optimization over all functions with an optimization over some functional class (for example, deep neural networks)

$$\theta^t = \arg \min_{\theta \in \mathbb{R}^p} \mathbb{E}_{(s,a) \sim \mathcal{D}} \mathbb{E}_{r \sim R(s,a), s' \sim P(s,a)} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta^{t-1}}(s', a') - Q_{\theta}(s, a) \right)^2 \right],$$

Step 2. Replace expectation with Monte-Carlo estimate:

$$\mathcal{B} = \{(s_k, a_k, r_k, s'_k)\}_{k=1}^M, \quad (s_k, a_k) \sim \mathcal{D}, r_k \sim R(s_k, a_k), \quad s'_k \sim P(s_k, a_k)$$

$$\theta^t = \arg \min_{\theta \in \mathbb{R}^p} \frac{1}{M} \sum_{k=1}^M \left[\left(r_k + \gamma \max_{a' \in \mathcal{A}} Q_{\theta^{t-1}}(s'_k, a') - Q_{\theta}(s_k, a_k) \right)^2 \right]$$

How to make LSVI practical?

Step 1. Replace optimization over all functions with an optimization over some functional class (for example, deep neural networks)

$$\theta^t = \arg \min_{\theta \in \mathbb{R}^p} \mathbb{E}_{(s,a) \sim \mathcal{D}} \mathbb{E}_{r \sim \mathcal{R}(s,a), s' \sim \mathcal{P}(s,a)} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta^{t-1}}(s', a') - Q_{\theta}(s, a) \right)^2 \right],$$

Step 2. Replace expectation with Monte-Carlo estimate:

$$\mathcal{B} = \{(s_k, a_k, r_k, s'_k)\}_{k=1}^M, \quad (s_k, a_k) \sim \mathcal{D}, r_k \sim \mathcal{R}(s_k, a_k), \quad s'_k \sim \mathcal{P}(s_k, a_k)$$

$$\theta^t = \arg \min_{\theta \in \mathbb{R}^p} \frac{1}{M} \sum_{k=1}^M \left[\left(\overbrace{r_k + \gamma \max_{a' \in \mathcal{A}} Q_{\theta^{t-1}}(s'_k, a')}^{\text{target}} - Q_{\theta}(s_k, a_k) \right)^2 \right]$$

↑
target network

(almost) Deep Q-Network

Final Loss function (TD-loss):

The diagram shows the TD-loss function $\mathcal{L}_{TD}(\theta, \bar{\theta}, s, a, r, s')$ with three arrows pointing to its arguments: 'online network params' points to θ , 'transitions' points to s, a, r, s' , and 'target network params' points to $\bar{\theta}$.

$$\mathcal{L}_{TD}(\theta, \bar{\theta}, s, a, r, s') = \left(Q_{\theta}(s, a) - r - \gamma Q_{\bar{\theta}}(s', \arg \max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s', a)) \right)^2.$$

- Current estimate of optimal policy – $\arg \max Q(s, a)$
- Update SGD because there can be too much transition data;
- Instead of computation of an exact solution to next parameter, let's perform 1000 stochastic gradient updates, and then update target network with a parameters for new online network;

Q: Do you see something strange?

Does we have just reduced RL to supervised learning?

Simple answer: **NO.**

- We do not have an access to a dataset **B** and we have to collect it online;
- Next we will understand how to collect it by online interactions.

More complex answer: **Sometimes yes, but not really.**

- There exists a paradigm of *Offline Reinforcement Learning*;
- Has its own challenges connected to distributional shift;
- However, after offline learning algorithm usually works with online interactions (Offline-to-Online).

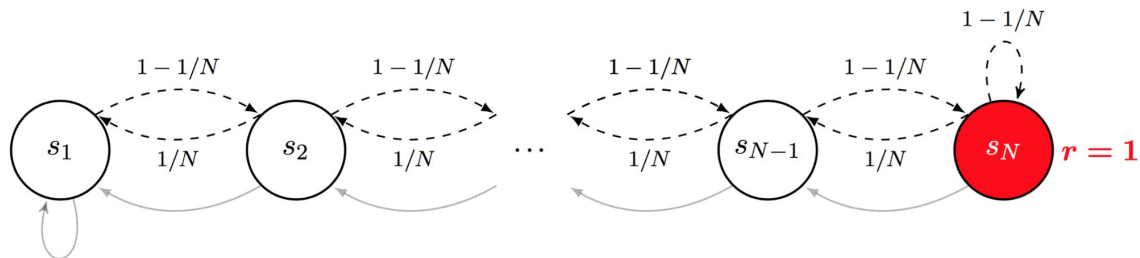
Exploration

- We want to collect a dataset of experiences

$$\mathcal{B} = \{(s_k, a_k, r_k, s'_k)\}_{k=1}^M, \quad (s_k, a_k) \sim \mathcal{D}, r_k \sim R(s_k, a_k), \quad s'_k \sim P(s_k, a_k)$$

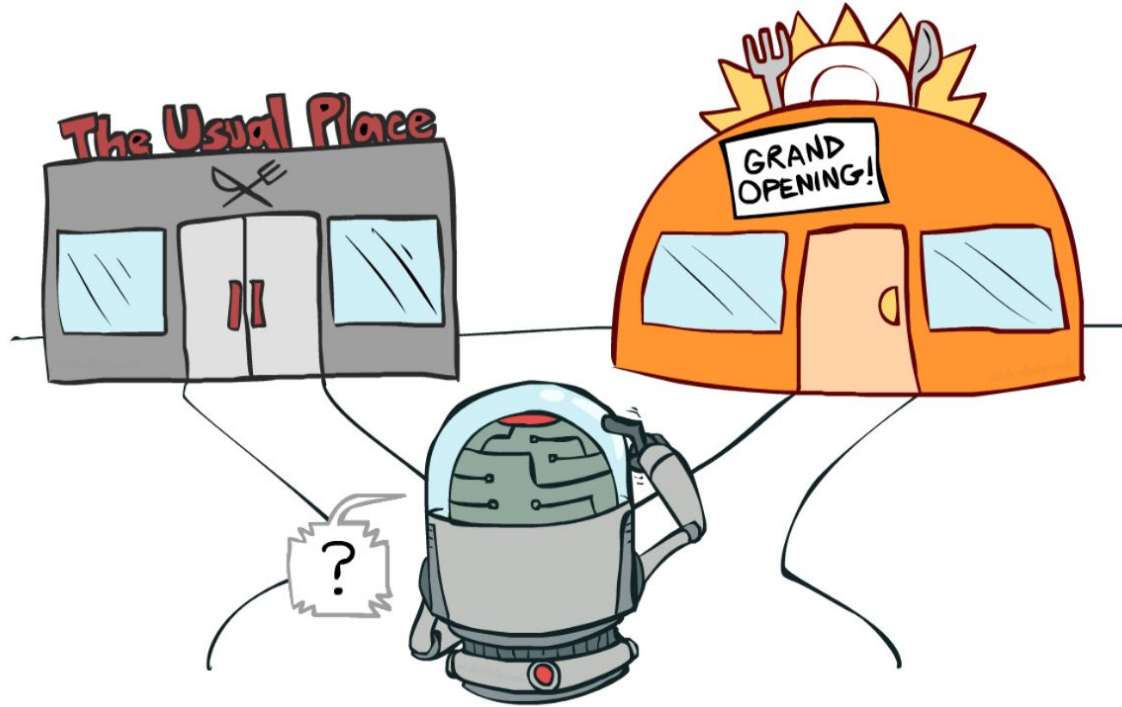
- *Simple way:* take all actions uniformly!

It is not really good:



Important: we have a good signal to rather good states – estimate of optimal policy (greedy policy w.r.t Q-value)!

Exploration-Exploitation Tradeoff

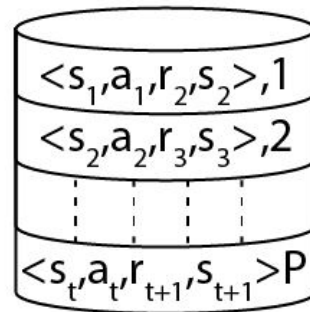


Two ideas:

1. **ϵ -greedy exploration:** exploit good “learning signal” and save a required exploration property

$$\pi(s) = \begin{cases} \arg \max_{a \in \mathcal{A}} Q_{\theta}(s, a) & \text{with probability } 1 - \epsilon \\ \mathcal{U}\text{nif}(\mathcal{A}) & \text{with probability } \epsilon \end{cases}$$

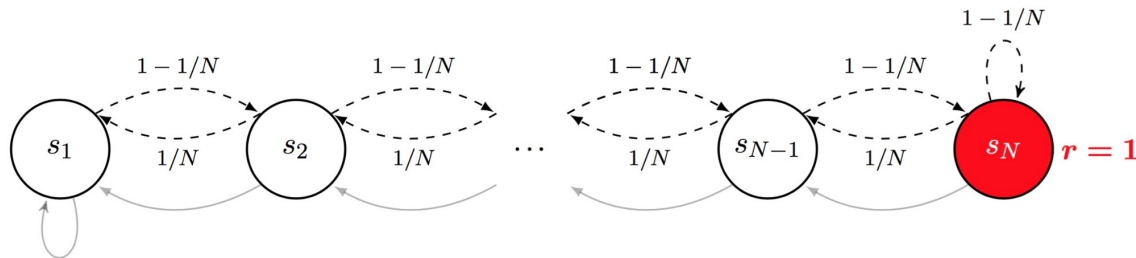
2. **Experience Replay:** collect transitions into some storage and remove the oldest one.
 - a. reuses old experiences;
 - b. new experiences have relatively large probability to be sampled.



This way of learning is called “off-policy”: the current estimate of Q-value is learnt from previous “outdated” data, not the current trajectories

About ϵ -greedy exploration

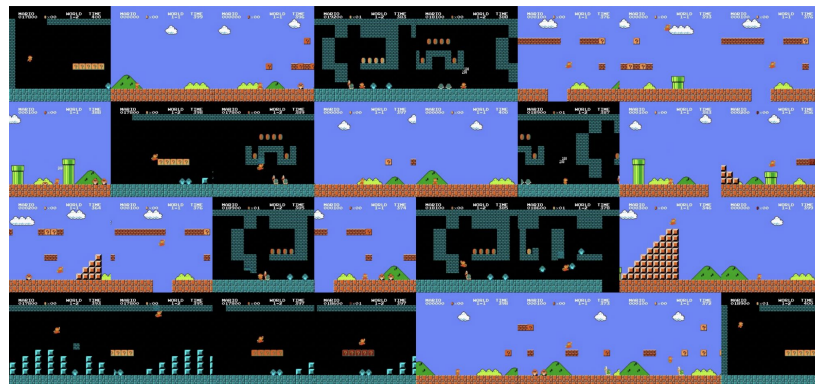
It is very INEFFICIENT way to do exploration: for very simple chain MDP you need exponential number of steps on average!



But it is rather simple and works relatively good in practice.

Remark: there are a lot of different ways to do better exploration, still active research area!

Key property of replay buffer: decorrelation



Source: RL course at AI Masters (<https://ozonmasters.ru/reinforcementlearning>)

Final algorithm – Deep Q-Network

Final Loss function (TD-loss):

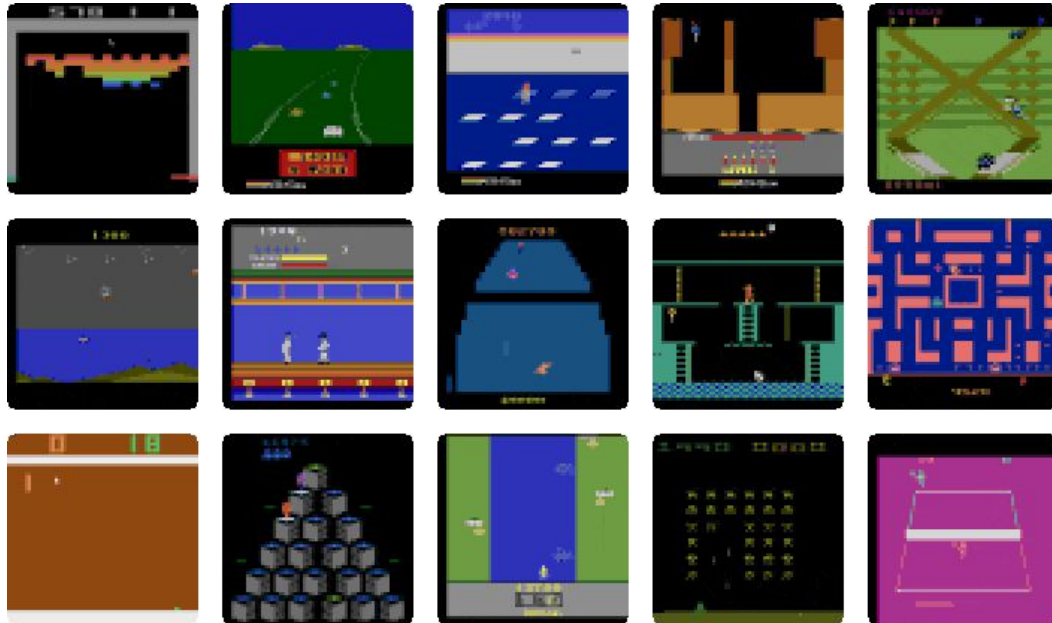
The diagram illustrates the TD-loss function $\mathcal{L}_{TD}(\theta, \bar{\theta}, s, a, r, s')$. Three arrows point to the function: 'online network params' points to θ , 'transitions' points to the boxed input s, a, r, s' , and 'target network params' points to $\bar{\theta}$. The function is defined as the squared difference between the current Q-value and the bootstrapped target Q-value.

$$\mathcal{L}_{TD}(\theta, \bar{\theta}, s, a, r, s') = \left(Q_{\theta}(s, a) - r - \gamma Q_{\bar{\theta}}(s', \arg \max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s', a)) \right)^2.$$

- To optimize this loss, a batch of transitions is sampled uniformly from the replay buffer;
- Replay buffer is filled during interactions;
- Interactions with environment use ϵ -greedy exploration strategy;
- Target network is updated every T steps, where T is a tuning parameter (1000, 5000, ...)
- Be careful with ends of episodes!

How it looks in practice?

Deep RL Benchmark: Atari Games



source: <https://blog.research.google/2021/02/mastering-atari-with-discrete-world.html>

Preprocessing

Action selection frequency:

- Framestack;
- Frameskip;
- MaxAndSkip;
- (sometimes) Sticky actions;

Atari-specific preprocessing:

- EpisodicLife;
- FireReset;

Standard tricks:

- Crop image;
- Rescale (often to 84x84);
- Grayscale;

Reward preprocessing:

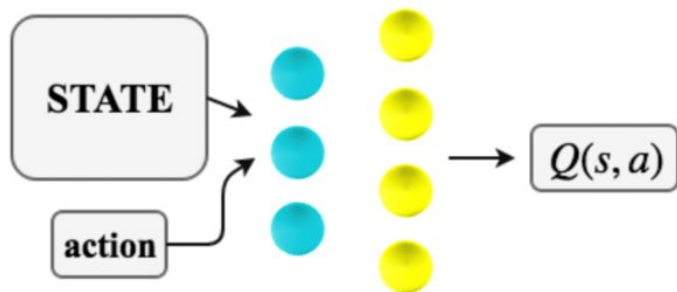
(!) Clip reward to $\{-1, 0, 1\}$;

Important !

No hyperparameter tuning for each specific game!

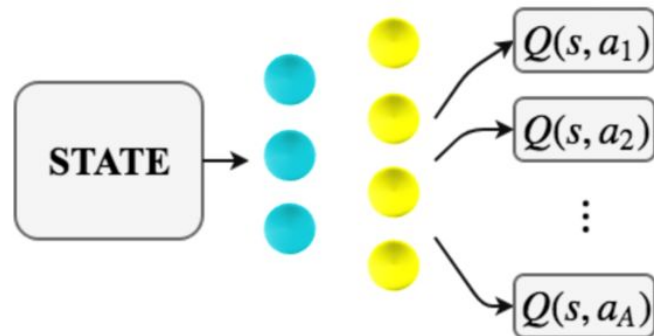
Architecture

Option 1



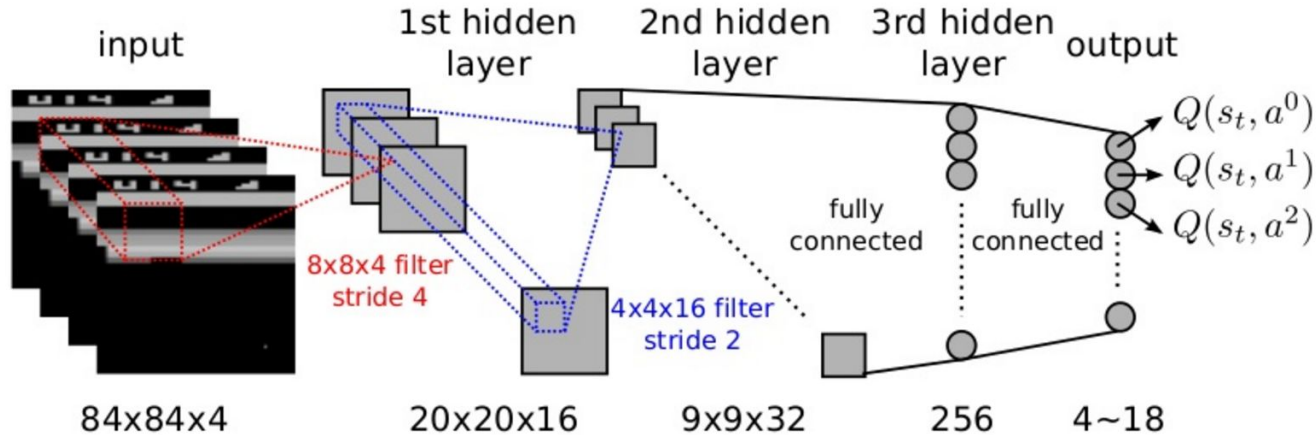
× $\operatorname{argmax}_a Q(s, a, \theta)$ — expensive!

Option 2



✓ $\operatorname{argmax}_a Q(s, a, \theta)$ — one forward pass.

Architecture



Source: https://leonardoaraujosantos.gitbook.io/artificial-intelligence/artificial_intelligence/reinforcement_learning/deep_q_learning

Q: Why networks in DEEP RL is not so deep?

Q: What are problems with use of batchnorm and dropout?

Overfitting in RL

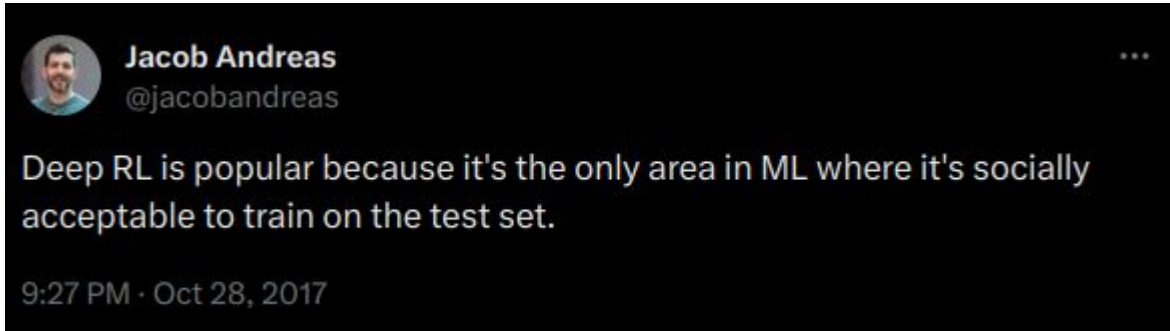
Problem: RL networks tends to overfit very quickly due to highly correlated data (even replay buffer cannot help enough).

To generate good training data, RL agent should behave good enough early!

This problem is much deeper than you might think: **Plasticity loss**

<https://arxiv.org/abs/2305.15555>

Test / train in RL?



Test / train in RL: Procgen benchmark

<https://github.com/openai/procgen>

Collection of generated levels for arcade games
to study generalization of RL agents



Recap

- (recap) Markov Decision Process
- (recap) Bellman Equations
- (recap) Value Iteration
- Least-Squared Value Iteration
- Exploration-Exploitation Tradeoff
- Experience Replay (Replay Buffer)
- Deep Q-Network (DQN)
- Atari & Procgen Benchmarks