DeepMind

Deep Q-Networks and Its Variants

Bilal Piot and Corentin Tallec

01/04/2021

DeepMind

Programme of the talk

Programme of the talk

- **Part 1: DQN Deep Q-Networks**
  - Context and History:
    - Context
    - The milestones: from Value Iteration to DQN
    - Environment and Results
  - Theory:
    - Reminder of the Value Iteration algorithm
    - Approximate Value Iteration
    - Neural Fitted-Q algorithm
    - From Neural Fitted-Q to DQN
  - Practice:
    - Overview of a DQN Implementation

Programme of the talk

- **Part 2: DQN and Its Variants**
  - DQN and its variants: an overview of the literature:
    - Algorithmic Improvements:
      - DDQN
      - Prioritized Replay
      - Distributional RL
    - Architectural Improvements:
      - Dueling DQN
      - Distributed Agents
    - Memory:
      - Working Memories
      - Episodic Memories
    - Exploration:
      - Never Give Up Agent
    - Meta Controllers:
      - Bandit
      - Meta-gradient RL
    - Agent57: Combining all the known improvements.

DeepMind

Part 1: DQN
Deep Q-Networks

DeepMind

Context and History

Context

- **Control Theory:**
  - Aims at guiding “safely” and “rapidly” a dynamical system to a desired state.
  - Has been one of the major advances in engineering in the 20th century:
    - Aviation
    - Manufacturing
    - Electronics
    - Energy
  - Works extremely well when those conditions are met:
    - Knowledge of the state variables (features)
    - Knowledge of their dynamics

- **Reinforcement Learning (RL):**
  - A general paradigm for control theory when the model of the world is unknown.
  - Has the potential to tackle complex control theory problems:
    - High dimensional state-action spaces
    - Partial observability
  - Deep Q-Network was one of the papers that reignited the interest in RL as a general solution to control theory.

Milestones: From Value Iteration to DQN

- **Value Iteration in stochastic games (Shapley 1953) and Markov decision processes (Bellman 1957):**
  - Computes the optimal value of an MDP.
  - The proof relies on fixed-point theory: Banach contraction theorem (1922).

- **Approximate Value Iteration:**
  - Bounds in infinity norm: Bertsekas and Tsitsiklis (1996)
  - Bounds in Lp norm: Munos (2007)

- **Fitted-Q Algorithms:**
  - Fitted-Q with random forests: Ernst (2005)
  - Neural fitted-Q: Riedmiller (2005)

- **Atari as an environment:**
  - ALE: Bellemare (2012)

- **Deep Q-Network:**
  - Scaling Neural fitted-Q to Atari games: Mnih (2013)

The Environment: ALE

The Arcade Learning Environment (ALE) is the environment chosen for DQN:

- Around 50 Atari games

- Raw observations:
  - RAM: 128 bytes (0-255)
  - 2D-RGB image: 160x210x3, 100 800 bytes

- Raw actions: 18 discrete actions (0-17)

- Rewards: deltas of the score of the game

- Frequency: 60 Hz

- Length of a game: 30 minutes or more

- Why is it interesting:
  - Huge state space
  - No canonical meaningful features
  - Very long optimization horizon

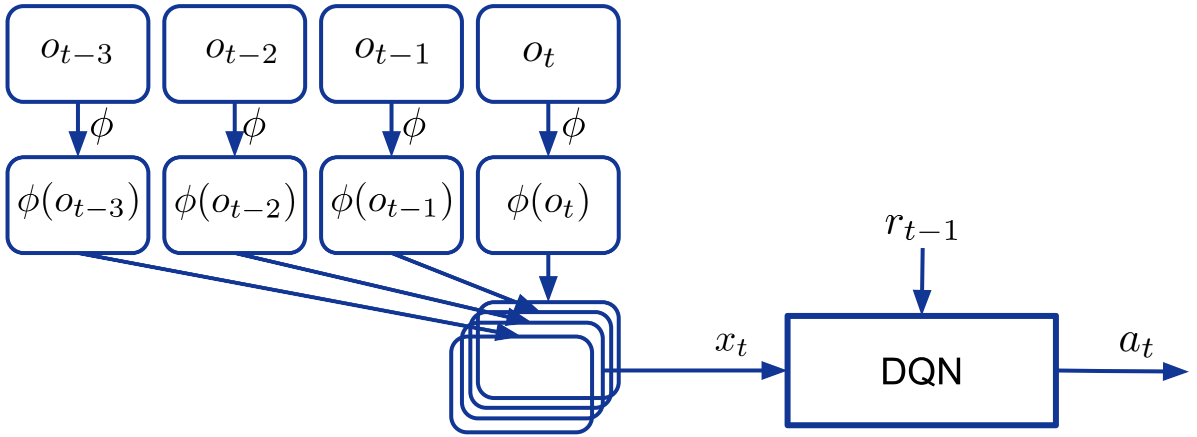
Preprocessing:

The algorithm DQN is almost end to end:

Environment:
Image: [160, 210, 3]
RAM: 128 bytes

Preprocessing:
Image: [84, 84, 1]

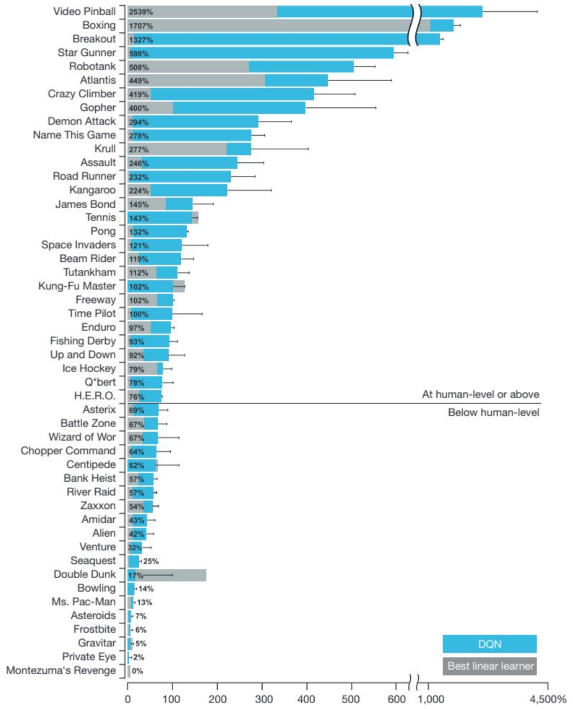
Stacking:
Input: [84, 84, 4]



Output: (0-17)

- Preprocessing ϕ of an observation is done in this order (Reduce the computation):
  - No RAM state in the observation, keep only the image: [160, 210, 3]
  - Take the maximum for each pixel of image at time t and t-1: [160, 210, 3]
  - Extract the luminance from the RGB: [160, 210, 1]
  - Downscale the image to 84x84: [84, 84, 1]
- Stacking of the 4 previous preprocessed observation (Reduce the partial observability): [84, 84, 4]
- The stack of the preprocessed observation is the input of the DQN algorithm.
- The output of DQN is the action at 15 Hz frequency which means that there is an action repeat of 4.

The results:



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%

DeepMind

Theory

Reminder of the Value Iteration (VI) Algorithm: Theoretical setting.

We consider the control problem in a finite Markov Decision Process

- **States:** $x \in \mathcal{X}$
- **Actions:** $a \in \mathcal{A}$
- **Reward function:** $R(x,a), \quad R \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$
- **Transition kernel:** $P(y|x,a), \quad P \in \Delta_{\mathcal{X}}^{\mathcal{X}\times\mathcal{A}}$
- **Discount factor:** $\gamma \in]0,1[$

We are looking for a stationary policy $\pi(a|x), \quad \pi \in \Delta_{\mathcal{A}}^{\mathcal{X}}$

that maximises the expected discounted sum of future rewards represented by the state-action value function:

$$Q^\pi(x,a) = \mathbb{E}_{\tau\sim\mathcal{T}_{x,a,\pi}}\left[\sum_{n\geq 0}\gamma^n R(X_n, A_n)\right]$$

where $\mathcal{T}_{x,a,\pi}$ **is the distribution over trajectories** $\tau = (X_n, A_n)_{n\in\mathbb{N}}$ **following policy** π **and starting from** $(X_0, A_0) = (x,a)$.

Reminder of the Value Iteration (VI) Algorithm: Formulation

Let us define the optimal Bellman operator :

$$[T^\star Q](x,a) = R(x,a) + \gamma \sum_{y\in\mathcal{X}} P(y|x,a) \max_{b\in\mathcal{A}} Q(y,b)$$

This operator is a contraction, therefore $\exists! Q^\star$ **such that** $Q^\star = T^\star Q^\star$

In addition, $Q^\star$ **corresponds to the maximum of the state-action value function:** $Q^\star = \max_\pi Q^\pi$

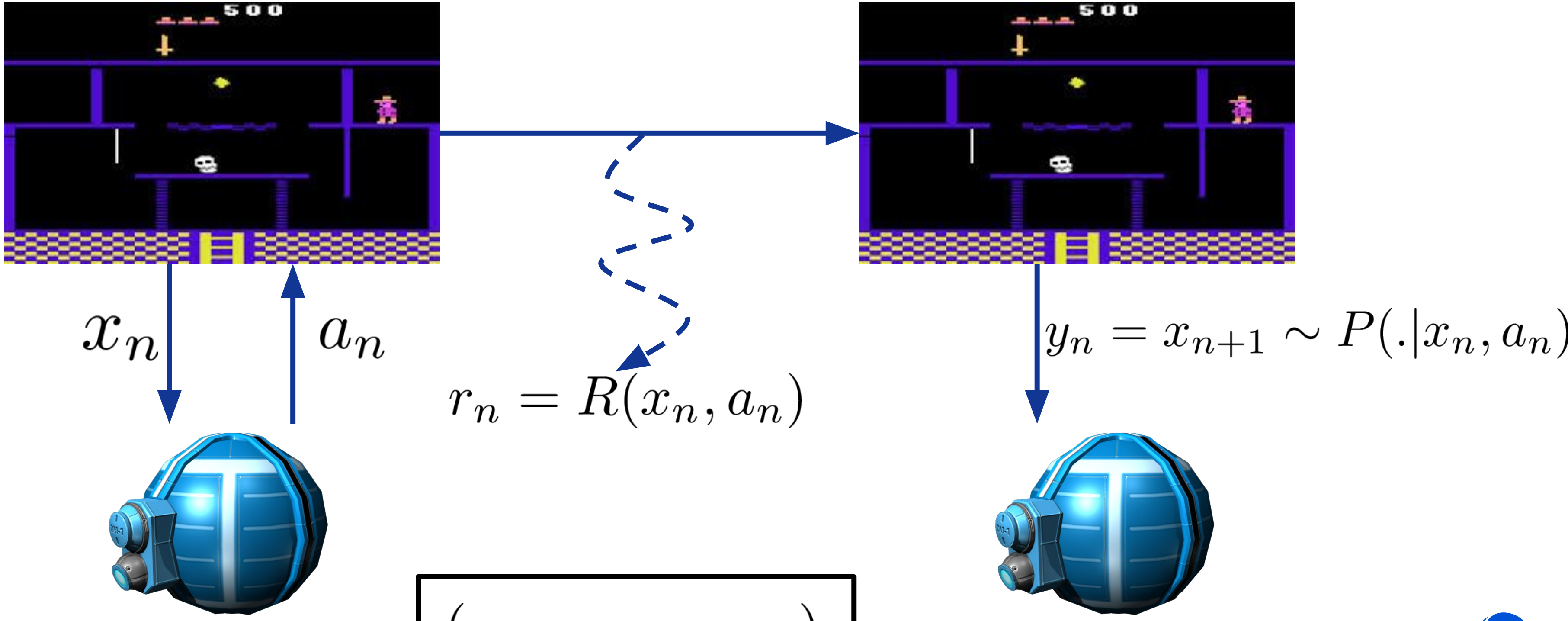
To compute $Q^\star$ **one can follow the contracting discrete scheme called Value Iteration:**

Initialisation: $Q_0 \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$

VI recurrence: $\forall k \geq 0, \quad Q_{k+1} = T^\star Q_k$

$$\lim_{k\to\infty} Q_k = Q^\star$$

Real-world setting: Interactive Environments



$$\boxed{(x_n, a_n, r_n, y_n)}$$

From Value Iteration to Approximate Value Iteration

PROBLEM: Applying the VI recurrence $Q_{k+1} = T^\star Q_k$ is impossible in a real-world setting:

- State-action spaces can be too large!
- Dynamics are not fully known!
- States and actions can't be trivially collected, they need to be reached by a non trivial policy (**locality problem**).

However, the optimal Bellman operator can be evaluated on a known dataset of already visited transitions:

$$\mathcal{T} = (x_n, a_n, r_n = R(x_n, a_n), y_n \sim P(.|x_n, a_n))_{1 \le n \le N}$$

The evaluations (also called targets) $t_n = r_n + \gamma \max_{b \in \mathcal{A}} Q(y_n, b)$ are unbiased estimates of $[T^\star Q](x_n, a_n)$

Therefore, we can build the following regression dataset: $D(\mathcal{T}, T^\star Q) = \{(x_n, a_n), t_n\}_{1 \le n \le N}$

$$\begin{tikzcd} D(\mathcal{T}, T^\star Q) \arrow[r] & \boxed{\text{Regression Algorithm}} \arrow[r] & \boxed{\hat{T}^\star Q} \end{tikzcd}$$

$$\epsilon_{T^\star Q} = \hat{T}^\star Q - T^\star Q$$

Reminder on Regression

In regression, we have:

- a set of points $\mathcal{T} = \{x_n \in \mathcal{X}\}_{1 \leq n \leq N}$
- and noisy but unbiased estimates $\{t_n \in \mathbb{R}\}_{1 \leq n \leq N}$ also called targets of a function $f \in \mathbb{R}^{\mathcal{X}}$

The goal is to retrieve this function with minimal error. More precisely:

- $t_n = f(x_n) + \eta(x_n)$ with $\mathbb{E}[\eta(x_n)] = 0$

From the dataset $D(\mathcal{T}, f) = \{x_n, t_n\}_{\{1 \leq n \leq N\}}$ a regression algorithm output a function $\hat{f} \in \mathbb{R}^{\mathcal{X}}$.

The regression error is defined as $\epsilon_f = \hat{f} - f$. One instantiation of a regression algorithm is:

$$\boxed{\hat{f} = \operatorname*{argmin}_{h \in \mathcal{F}} \sum_{n=1}^{N} \mathcal{L}(h(x_n), t_n)}$$

with regression loss $\mathcal{L} = (.)^2$ and functional space $\mathcal{F} \subset \mathbb{R}^{\mathcal{X}}$

Approximate Value Iteration step seen as a regression

Regression Notations	Approximate VI step Notations
$f \in \mathbb{R}^{\mathcal{X}}$	$T^\star Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$
$\mathcal{T} = \{x_n \in \mathcal{X}\}_{1\leq n\leq N}$	$\mathcal{T} = \{x_n, a_n, r_n, y_n\}_{1\leq n\leq N}$
$t_n = f(x_n) + \eta(x_n)$	$t_n = r_n + \gamma \max_{b\in\mathcal{A}} Q(y_n, b) = T^\star Q[x_n, a_n] + \eta(x_n, a_n)$
$D(\mathcal{T}, f) = \{x_n, t_n\}_{\{1\leq n\leq N\}}$	$D(\mathcal{T}, T^\star Q) = \{(x_n, a_n), t_n\}_{1\leq n\leq N}$
$\hat{f} \in \mathbb{R}^{\mathcal{X}}$	$\hat{T}^\star Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$
$\epsilon_f = \hat{f} - f$	$\epsilon_{T^\star Q} = \hat{T}^\star Q - T^\star Q$

Approximate Value Iteration (AVI): Formulation

Initialisation: $Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$

AVI recurrence: $\forall k \geq 0, \quad Q_{k+1} = \hat{T}^\star Q_k$

The AVI recurrence step consists in two steps:

1. Build a regression dataset:
    a. **Collect** a dataset of transitions: $\mathcal{T} = (x_n, a_n, r_n = R(x_n, a_n), y_n \sim P(.|x_n, a_n))_{1 \leq n \leq N}$
    b. **Compute** unbiased estimates of the optimal Bellman operator: $t_{n,k} = r_n + \gamma \max_{b \in \mathcal{A}} Q_k(y_n, b)$
    c. **Create** the regression dataset: $D(\mathcal{T}, T^\star Q_k) = \{(x_n, a_n), t_{n,k}\}_{1 \leq n \leq N}$
2. Apply a regression algorithm of your choice:

$$\begin{tikzcd} D(\mathcal{T}, T^\star Q_k) \arrow[r] & \boxed{\text{Regression Algorithm}} \arrow[r] & \hat{T}^\star Q_k \end{tikzcd}$$

$$\epsilon_k = \hat{T}^\star Q_k - T^\star Q_k$$

Approximate Value Iteration (AVI): Bounds

Let $\epsilon = \sup_{k \in \mathbb{N}} \|\epsilon_k\|_\infty$ be the supremum in infinite norm of the regression errors and let us define the greedy policy:

$$\pi_k(x) = \underset{a \in \mathcal{A}}{\mathrm{argmax}} Q_k(x, a)$$

then, we have the following bound:

$$\limsup_{k \to \infty} \|Q^\star - Q^{\pi_k}\|_\infty \leq \frac{2\gamma\epsilon}{(1-\gamma)^2}$$

This is a bound in infinite norm, for tighter bounds with other norms see:

- Munos 2007: https://hal.inria.fr/inria-00124685/document
- Scherrer 2014: https://hal.inria.fr/hal-01091341/document

Neural Fitted-Q: Intro and Notations

Neural Fitted-Q (Riedmiller 2005) is an instantiation of AVI where the regression algorithm have the following properties:

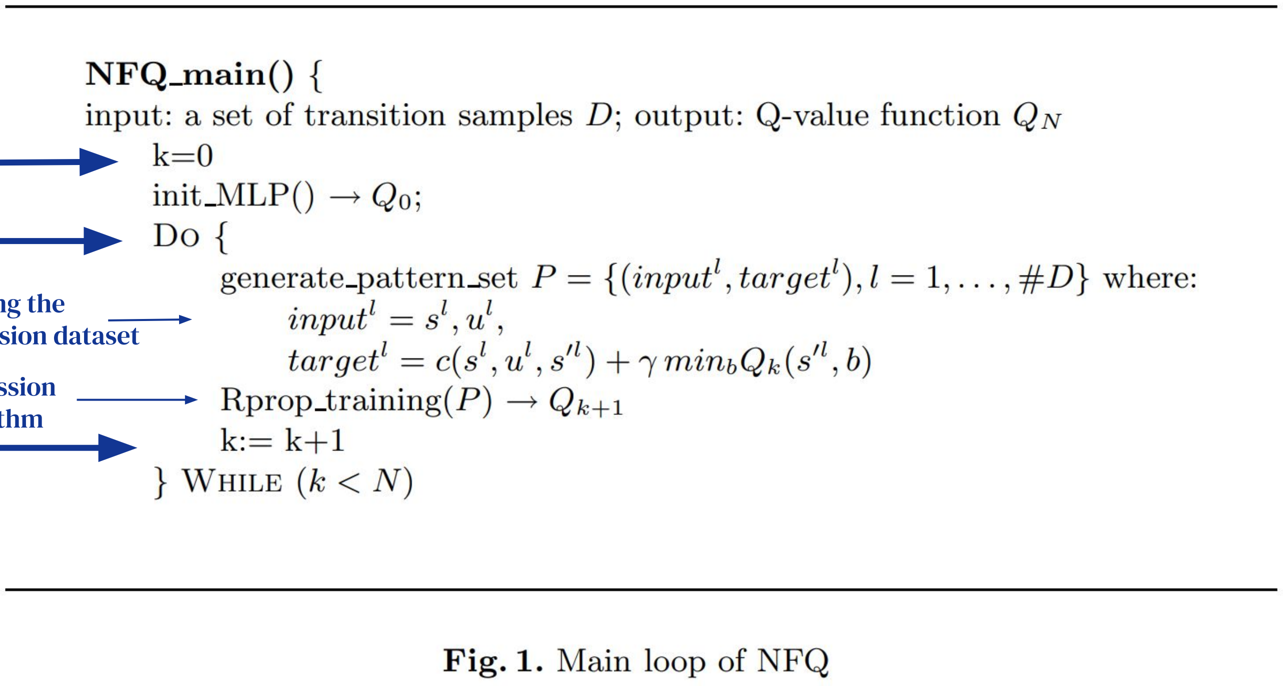
- The functional regression space is parameterized by a neural network.
- The loss function is a squared-like loss.
- The optimizer is SGD-based (Rprop for the original but could be Adam).
- The data used for regression was often a fixed batch of data.

$$\mathcal{F}_\theta = \{Q_\theta | \theta \in \mathbb{R}^{\mathcal{N}}\} \qquad Q_k = Q_{\theta_k}$$

Initialisation: $$Q_{\theta_0} \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$$

Neural Fitted-Q recurrence: $$\forall k \geq 0, \quad Q_{\theta_{k+1}} = \hat{T}^\star Q_{\theta_k}$$

Neural Fitted-Q: original pseudo-code.



Initialisation:

Neural Fitted-Q Recurrence:

Building the regression dataset

Regression algorithm

```
NFQ_main() {
input: a set of transition samples $D$; output: Q-value function $Q_N$
    k=0
    init_MLP() $\rightarrow Q_0$;
    Do {
        generate_pattern_set $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:
            $input^l = s^l, u^l,$
            $target^l = c(s^l, u^l, s'^l) + \gamma \, min_b Q_k(s'^l, b)$
        Rprop_training($P$) $\rightarrow Q_{k+1}$
        k:= k+1
    } WHILE $(k < N)$
```

Fig. 1. Main loop of NFQ

Neural Fitted-Q: an overview.

The Neural Fitted-Q recurrence follows the same pattern as the AVI one:

1. **Build the regression dataset:**
    a. **Collect** a dataset of transitions: $\mathcal{T} = (x_n, a_n, r_n = R(x_n, a_n), y_n \sim P(.|x_n, a_n))_{1 \leq n \leq N}$
    b. **Compute** unbiased estimates of the optimal Bellman operator: $t_{n,k} = r_n + \gamma \max_{b \in \mathcal{A}} Q_{\theta_k}(y_n, b)$
    c. **Create** the regression dataset: $D_k = D(\mathcal{T}, T^\star Q_{\theta_k}) = \{(x_n, a_n), t_{n,k}\}_{1 \leq n \leq N}$
2. **Apply regression with squared loss and optimize it with an SGD-based optimizer:**

$$\mathcal{L}\left(\theta, D_k\right) = \mathbb{E}_{(x,a),t \sim D_k}\left[\left(Q_\theta(x,a) - t\right)^2\right]$$

$$\theta_{k+1} = \underset{\theta \in \mathbb{R}^{\mathcal{N}}}{\mathrm{Optiargmin}} \quad \mathcal{L}(\theta, D_k)$$

Neural Fitted-Q: in details.

Here we show how we compute in details the regression $\theta_{k+1} = \underset{\theta \in \mathbb{R}^{\mathcal{N}}}{\text{Optiargmin}} \quad \mathcal{L}(\theta, D_k)$

The expected squared loss $\mathcal{L}(\theta, D_k)$ **will be approximated by** $\mathcal{L}(\theta, \mathcal{B}_k)$ **where** $\mathcal{B}_k$ **is a batch of data.**

Then, the optimization will consists of a fixed number of SGD-like steps performed by an optimizer:

1. **Initialisation:** $\theta = \theta_k$
2. **For** $0 \leq i \leq I$ **:**
   a. Draw uniformly a batch of transitions from the dataset: $(x_j, a_j, r_j, y_j)_{1 \leq j \leq B} \sim \mathcal{T}$
   b. Compute the targets: $t_{j,k} = r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta_k}(y_j, b)$
   c. Form the regression batch: $\mathcal{B}_k = \{(x_j, a_j), t_{j,k}\}_{1 \leq j \leq B}$
   d. Compute the loss: $\mathcal{L}(\theta, \mathcal{B}_k) = \frac{1}{B} \sum_{j=1}^{B} \left[(Q_\theta(x_j, a_j) - t_{j,k})^2 \right]$
   e. Take a SGD-like step: $\theta \leftarrow \theta - \alpha \, \text{Optimizer}(\nabla_\theta \mathcal{L}(\theta, \mathcal{B}_k))$
3. **Update the parameters:** $\theta_{k+1} = \theta$

From Neural Fitted-Q to DQN

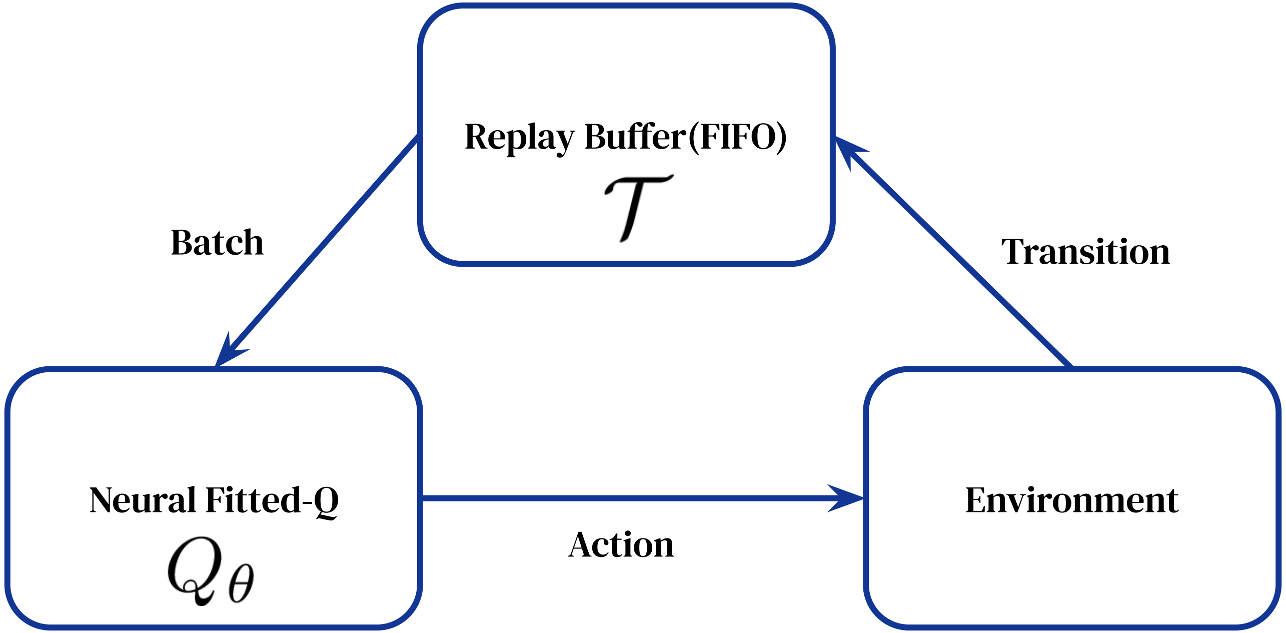
From Neural Fitted-Q to DQN, only some small changes but a 8 year gap:

- **Data collection:** going from a batch dataset to a replay buffer filled by the online policy. Acting and Learning are done simultaneously.
- **Architecture:** Bigger neural network architecture.
- **Optimization:** Using RMSprop.

Some vocabulary introduced by DQN but the underlying concepts were already existing:

- **Online network:** The neural network that optimizes the loss. Q_θ
- **Target network:** The neural network of the previous AVI iteration with fixed weights. Q_{θ_k}
- **Online policy:** The policy that collects the data. π_θ
- **Replay buffer:** The collections of transitions from which the batches are samples. $\mathcal{T}$
- **Update period:** How many steps of gradients are taken before going to the next AVI iteration. I

DQN: an overview



DQN: the data collection (Acting).

In DQN, the data is collected via the epsilon-greedy online policy:

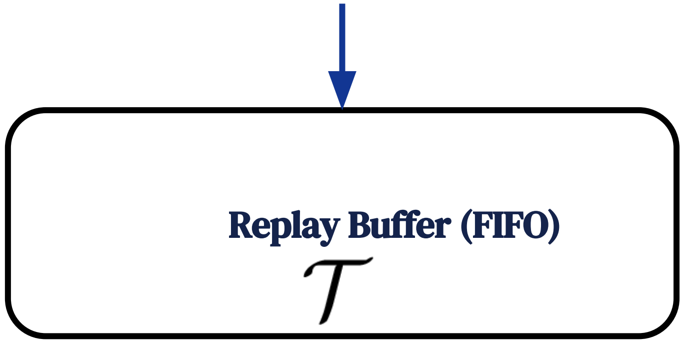
Greedy-policy: $\pi_\theta(x) = \underset{a \in \mathcal{A}}{\text{argmax}} Q_\theta(x, a)$

Uniform policy: π_U

Epsilon-greedy policy: $\pi_{\theta,\epsilon} = (1-\epsilon)\pi_\theta + \epsilon\pi_U$

Start an episode at x_0 and collect an episode following the policy $\pi_{\theta,\epsilon}$:

$$(x_n, a_n \sim \pi_{\theta,\epsilon}(x_n), r(x_n, a_n), y_n = x_{n+1} \sim P(.|x_n, a_n))_{1 \leq n \leq H}$$



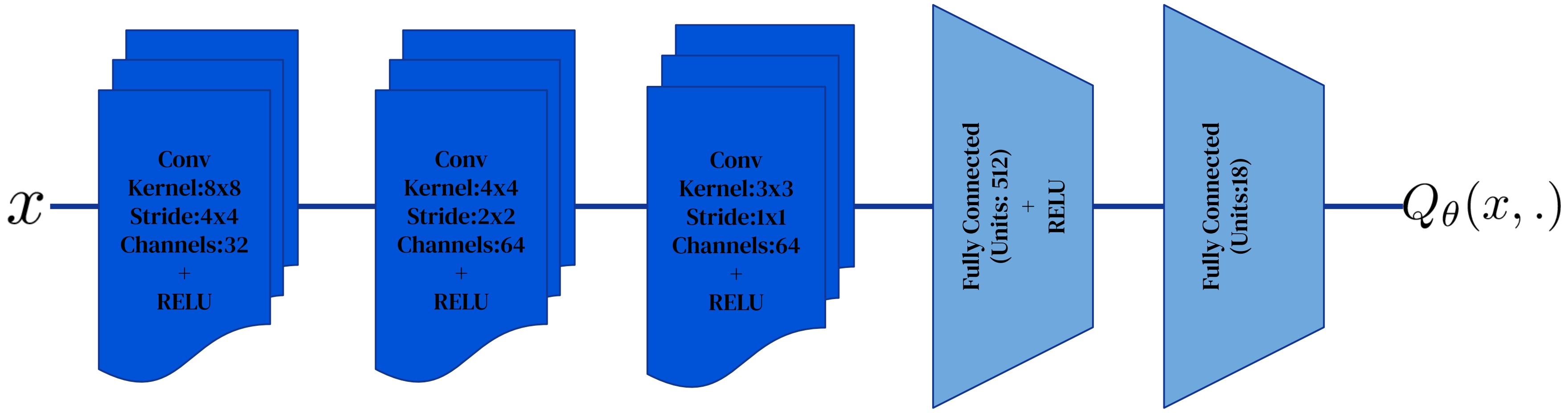
DQN: the regression step (Learning).

Here we show how we compute in details the regression $\theta_{k+1} = \underset{\theta \in \mathbb{R}^{\mathcal{N}}}{\text{Optiargmin}} \quad \mathcal{L}(\theta, D_k)$

Then, the optimization will consists of a fixed number of SGD-like steps performed by an optimizer:

1. **Initialisation:** $\theta = \theta_k$
2. **For** $0 \leq i \leq I$ **:**
   a. Draw uniformly a batch of transitions from the replay: $(x_j, a_j, r_j, y_j)_{1 \leq j \leq B} \sim \mathcal{T}$
   b. Compute the targets: $t_{j,k} = r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta_k}(y_j, b)$
   c. Form the regression batch: $\mathcal{B}_k = \{(x_j, a_j), t_{j,k}\}_{1 \leq j \leq B}$
   d. Compute the loss: $\mathcal{L}(\theta, \mathcal{B}_k) = \frac{1}{B} \sum_{j=1}^{B} \left[(Q_\theta(x_j, a_j) - t_{j,k})^2 \right]$
   e. Take a SGD-like step: $\theta \leftarrow \theta - \alpha \, \text{Optimizer}(\nabla_\theta \mathcal{L}(\theta, \mathcal{B}_k))$
3. **Update the parameters:** $\theta_{k+1} = \theta$

DQN: Architecture.



DeepMind

Practice

DQN Zoo Codebase

Material:

- The DQN Zoo codebase

Information:

- The codebase is open source and developed by DeepMind (John Quan and Georg Ostrovski)
- The code is in Python, JAX, Haiku and Rlax.
- It tries to reproduce results of DQN and some of its variants on Atari:
    - DQN
    - DDQN
    - Prioritized Experience Replay
- Can be installed on a machine with a single GPU
- Comes with a run function and plotting tools
- Each agent comes with:
    - A class describing the agent
    - A run function
    - A test function

Inspecting the DQN code

Questions:

- In which file the learning/interaction loop is implemented?
- In which file the DQN agent functions are defined?
- In which library can I find the DQN loss?
- How the networks parameters are updated?
- How is the replay implemented?
- Where is the preprocessing done?

Exercice

From the DQN code write its pseudo code.

DeepMind

Part 2: DQN and its Variants

DeepMind

DQN and its variants: an overview of the literature

Improvements to DQN

Since 2015, several improvements have been made:

- **Algorithmic improvements:**
  - Double DQN
  - Prioritized replay
  - Distributional RL

- **Architectural improvements:**
  - Dueling architecture
  - Distributed setting

- **Memory additions:**
  - Working memories: LSTM, GRU
  - Episodic memory

- **Exploration mechanisms:**
  - An entire literature on this topic has been developed since the 90's

- **Meta-controller:**
  - Bandits
  - Meta-gradients
  - Population-based training

Double DQN (DDQN)

Material:

- Double Q Learning paper
- Double DQN paper

The Problem: Q Learning has been shown to overestimate its targets, because it uses a single estimator for estimating the Q values and choosing the maximum over actions.

The Solution: To overcome this, Double Q Learning uses a double estimator technique.

The double estimator technique disentangle the estimation of the Q values from the choice of the maximum over the actions:

- The target network is used for estimation.
- The online network for choosing the greedy action.

$t_{j,k} = r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta_k}(y_j, b)$ becomes $t_{j,k} = r_j + \gamma Q_{\theta_k}(y_j, a^\star_{j,\theta})$ where $a^\star_{j,\theta} = \operatorname{argmax}_{b \in \mathcal{A}} Q_\theta(y_j, b)$

Prioritized Experience Replay

Material:

- [Prioritized Experience Replay paper]

The Idea: Prioritize transitions with high TD errors and weight those transitions adequately to eliminate the introduced bias.

TD error: $\delta_{j,k} = r_j + \gamma Q_{\theta_k}(y_j, a^\star_{j,\theta}) - Q_\theta(x_j, a_j)$

Priority: $p_j = |\delta_{j,k}| + \epsilon$

Probability of being selected: $P_j = \frac{p_j^\alpha}{\sum_n p_n^\alpha}$

Loss reweighting: $w_j = \frac{(NP_j)^{-\beta}}{\max_n w_n}$

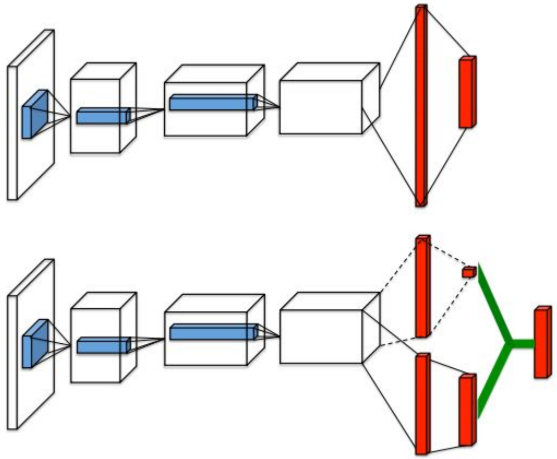
Dueling Architecture

Material:

- Dueling network architecture

The Idea: Decompose the Q value into a state-dependent part and a state-action dependent part.

The effect: This allows to share the state-dependent estimations (good for actions that are less chosen), focuses in estimating the state-action dependent part (good for relative ranking of the actions).



$$Q_\theta(s,a) = V_{\theta_V}(s) + A_{\theta_A}(s,a) - \frac{1}{|\mathcal{A}|}\sum_{b\in\mathcal{A}} A_{\theta_A}(s,b)$$

Distributional RL

Material:

- Categorical Distributional RL paper
- Implicit Quantile Network paper
- Dopamine Blog

The Idea:

- Learn the distribution of the discounted return.
- Still act according the expected discounted return.

Why it works:

- Learning the full distribution is a natural auxiliary task for better representation learning.

Distribution of returns of a given policy: $Z^{\pi}(x, a) = \sum_{n \geq 0} \gamma^n R(X_n, A_n)$

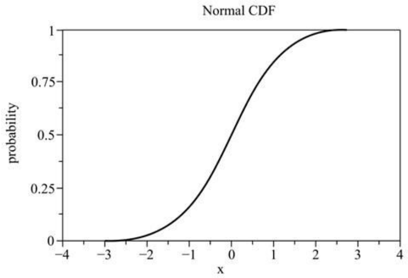
Distributional Bellman Operator: $\mathcal{T}^{\star} Z(x, a) = R(x, a) + \gamma Z(Y, a^{\star})$

$$a^{\star} = \operatorname*{argmax}_{b \in \mathcal{A}} \mathbb{E}[Z(Y, b)]$$

Distributional RL in a nutshell!

To learn a distribution of a real random variable you need to learn the cumulative distribution function:

$$F_X(x) = P(X \leq x)$$



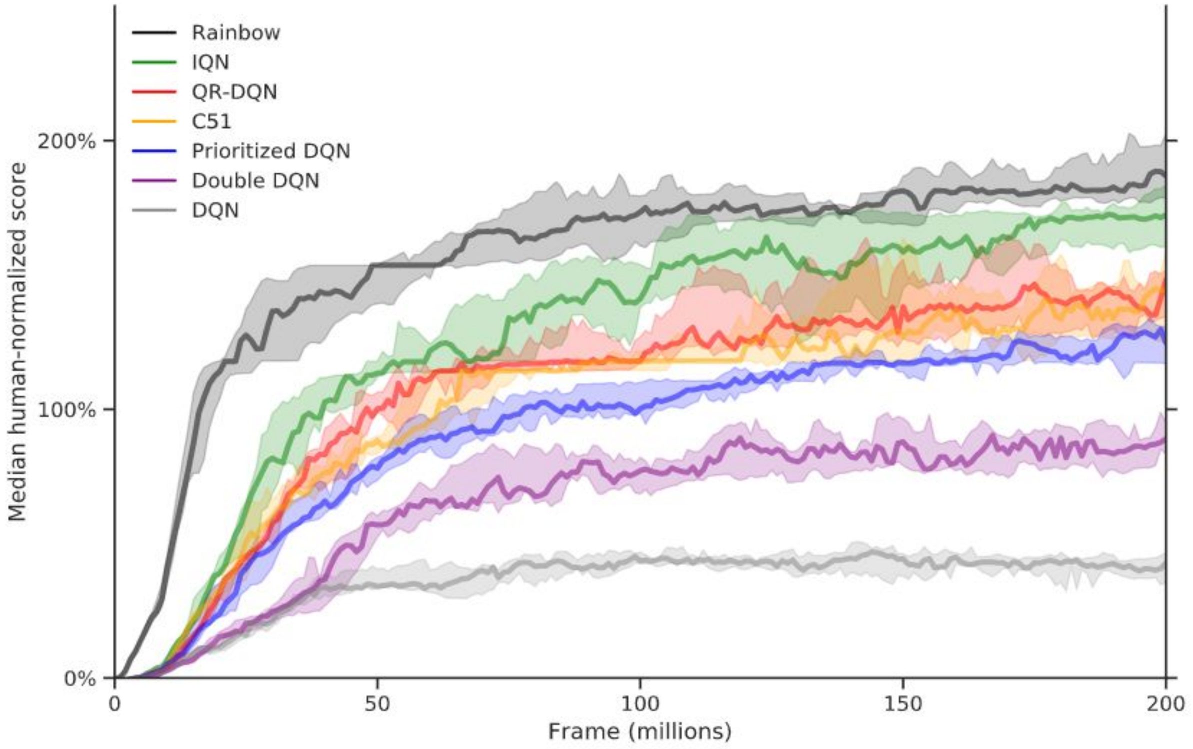
Categorical approach: Learning the probabilities by counting!

$$\mathbb{E}[1_{X \leq x}] = P(X \leq x)$$

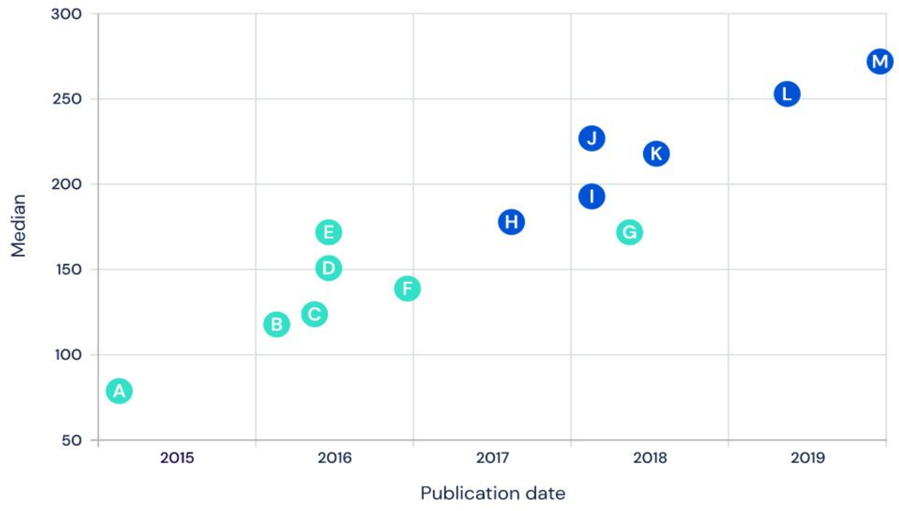
Quantile approach: Learning the quantiles with quantile regression!

$$Q_X(\tau) = F_X^{-1}(\tau) = \inf\{x : F_X(x) \geq \tau\}$$

Results for DQN-based non-distributed agents



Results for DQN-based non-distributed agents



Classic Deep RL

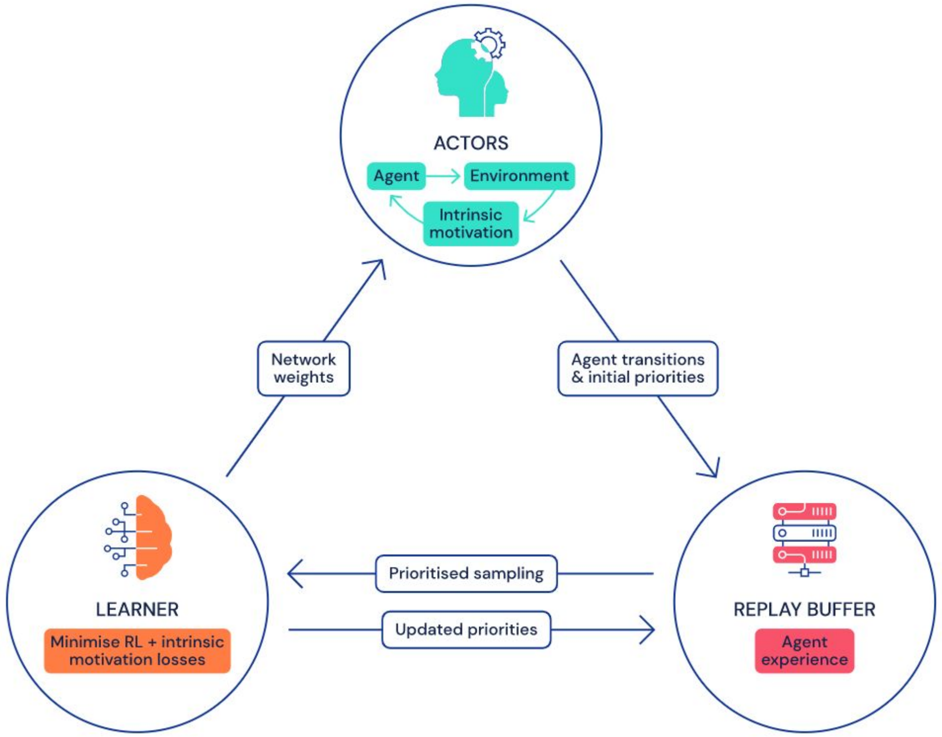
- A DQN (Mnih et al.)
- B DDQN (van Hasselt et al.)
- C Prioritised DQN (Schaul et al.)
- D Dueling (Wang et al.)
- E Prioritised Dueling (Wang et al.)
- F Bootstrapped DQN (Osband et al.)
- G NoisyNet Dueling (Fortunato et al.)

Distributional RL

- H C51 (Bellemare et al.)
- I QR-DQN (Dabney et al.)
- J Rainbow (Hessel et al.)
- K IQN (Dabney et al.)
- L C51-IDS (Nikolov et al.)
- M FQF (Yang et al.)

Distributed Setting

The Idea: Decoupling the learning from the data collection.



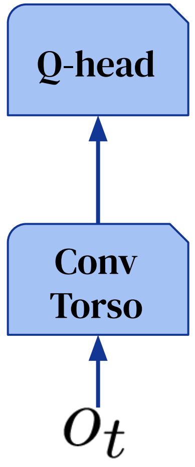
Working Memory

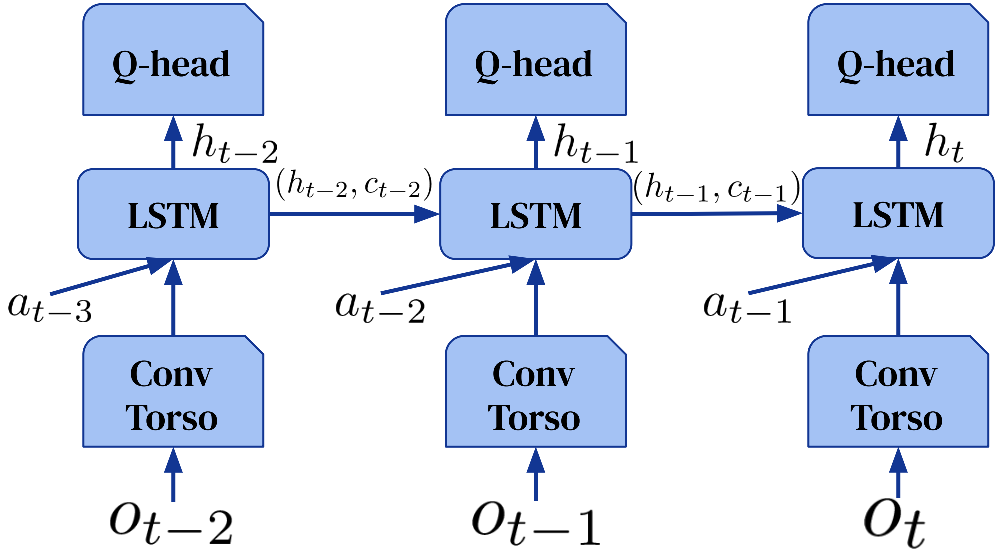
Material:

- R2D2 paper

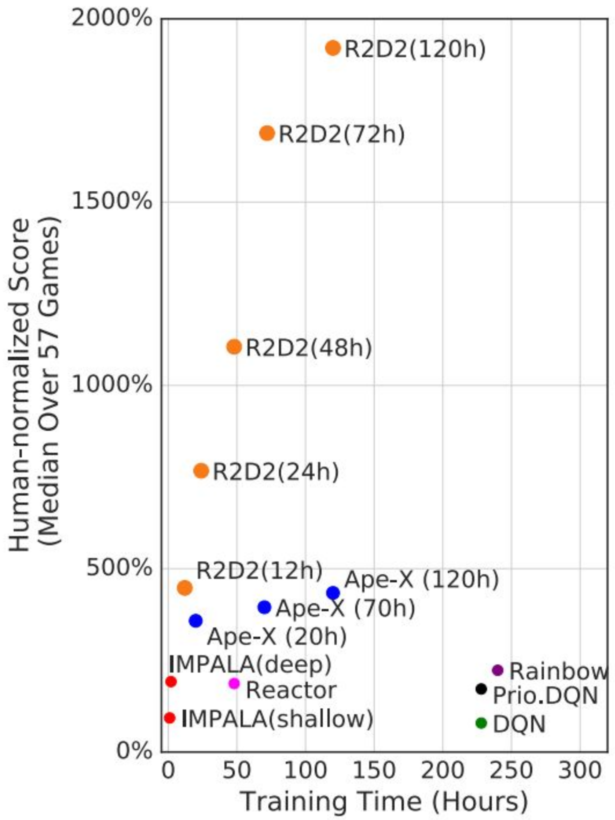
The Idea:

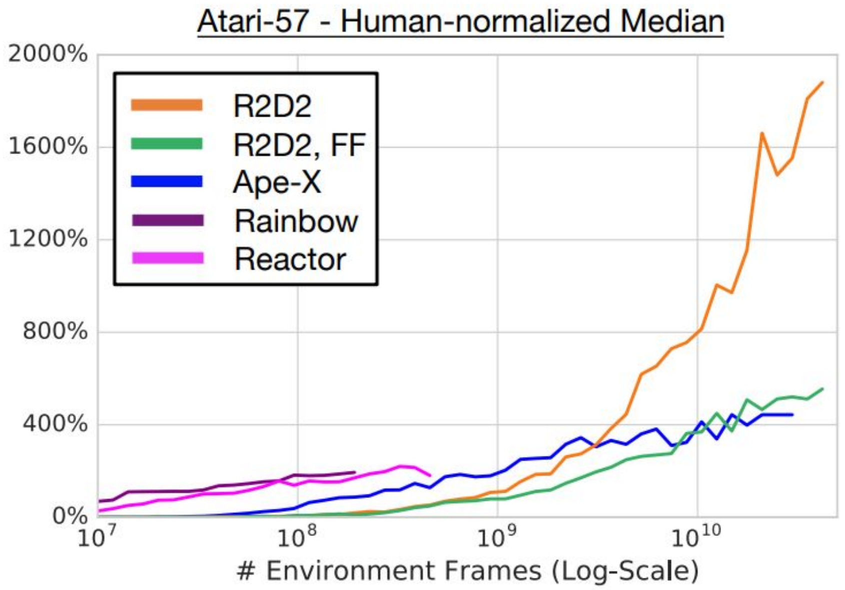
- Use a recurrent neural network to tackle the partial observability problem.





R2D2 results





Exploration

Reinforcement Learning is not only about maximizing the known rewards (exploitation) but also about finding new rewards (exploration).

Exploration mechanism in DQN: epsilon-greedy $\pi_{\theta,\epsilon} = (1-\epsilon)\pi_\theta + \epsilon\pi_U$

There is an entire literature on improving this basic exploration mechanism:

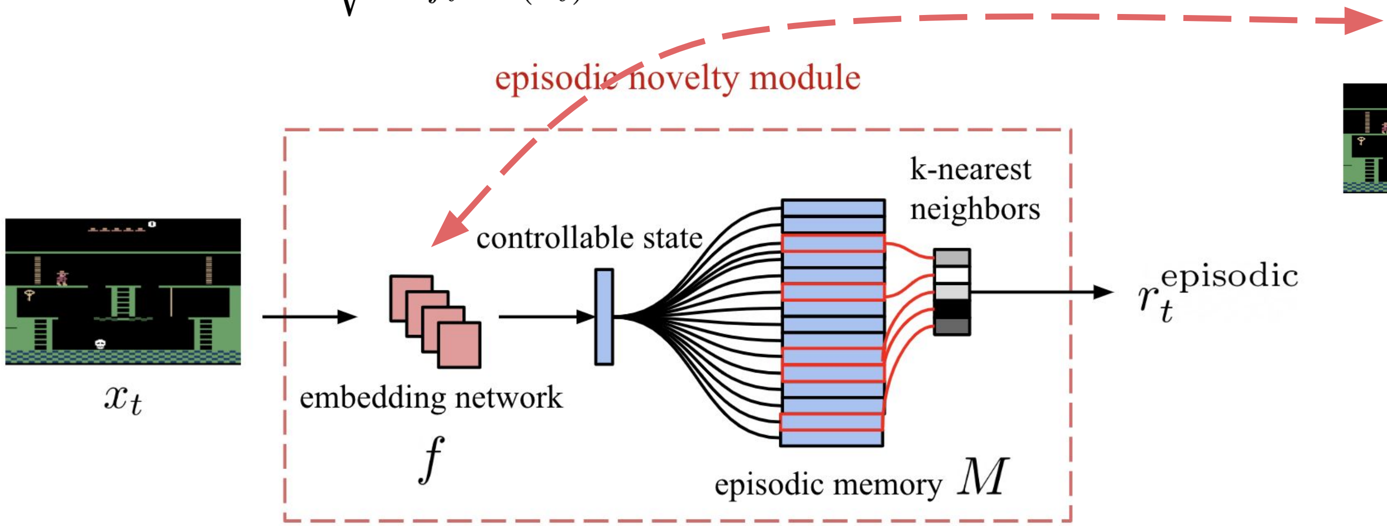
- Uncertainties estimation: Use the uncertainty about the world as an incentive for exploration.
    - State uncertainty: Random Network Distillation
    - Future uncertainty: Prediction error
    - Model uncertainty: Model disagreement
    - Value uncertainty: Uncertainty Bellman Equation
- Entropy maximisation:
    - Episodic entropy maximisation: Never Give Up
    - Global entropy maximisation: Geometric Entropy Maximisation

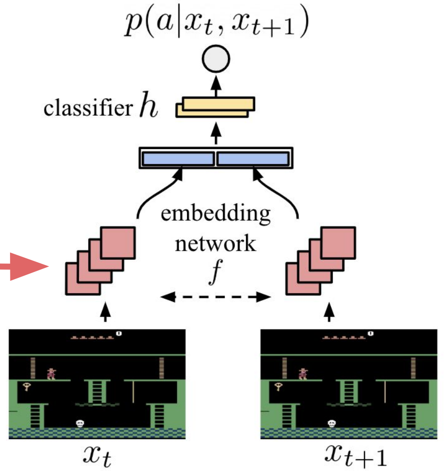
Never Give Up Episodic bonus

Build controllable states: self-supervised inverse dynamics model

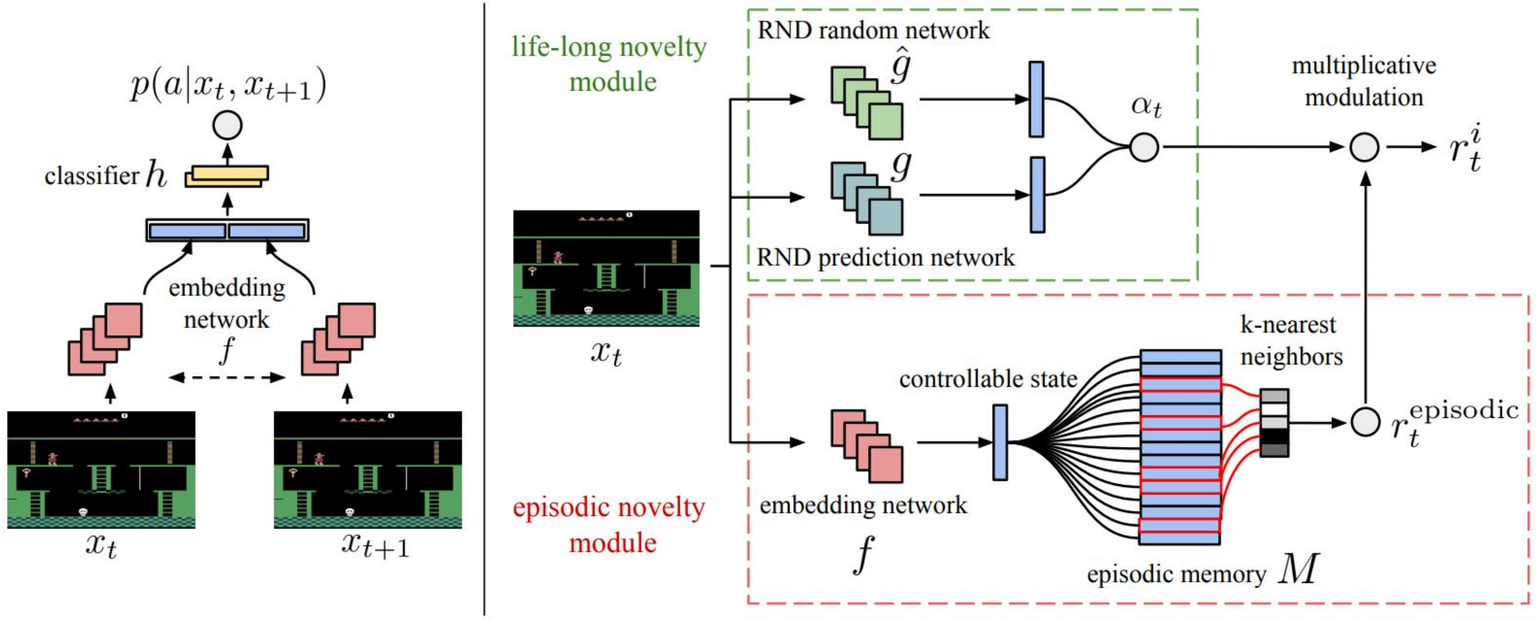
Compute an estimate how familiar is a given state.

$$r_t^{\texttt{episodic}} = \frac{1}{\sqrt{\sum_{f_i \in N(x_t)} K(f(x_t), f_i)}}$$





Never Give Up: complete exploration bonus



Meta-Controller

Arms:

(β_j, γ_j)

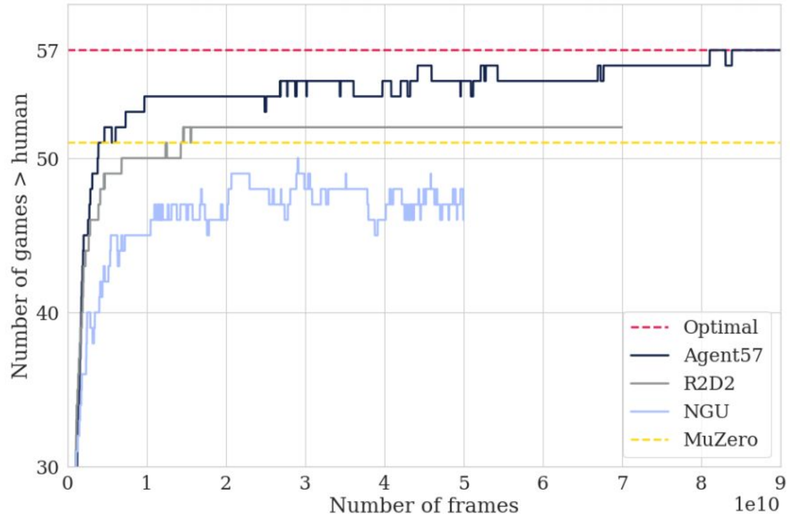
Rewards:
Episode Score

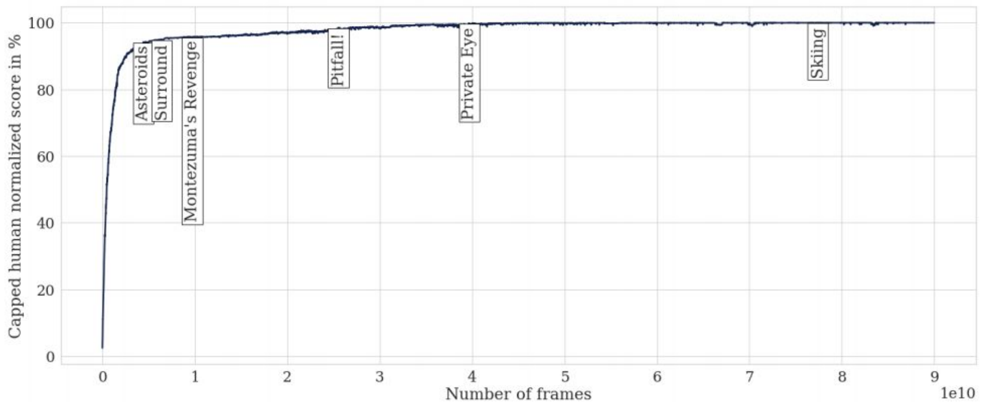
Agent57: Combining most improvements

Agent57 combines:

- Prioritized replay
- Dueling architecture
- Separated Q values, one for exploration and the other for exploitation
- Distributed actors
- Episodic Memory
- Working Memory conditioned on hyperparameters optimized by the Meta-Controller
- Exploration: Never Give Up
- Meta-Controller: Bandit

Agent57: results





Statistics	Agent57	R2D2 (bandit)	NGU	R2D2 (Retrace)	R2D2	MuZero
Capped mean	**100.00**	96.93	95.07	94.20	94.33	89.92
Number of games > human	**57**	54	51	52	52	51
Mean	4766.25	**5461.66**	3421.80	3518.36	4622.09	4998.51
Median	1933.49	**2357.92**	1359.78	1457.63	1935.86	2041.12
40th Percentile	1091.07	**1298.80**	610.44	817.77	1176.05	1172.90
30th Percentile	614.65	**648.17**	267.10	420.67	529.23	503.05
20th Percentile	**324.78**	303.61	226.43	267.25	215.31	171.39
10th Percentile	**184.35**	116.82	107.78	116.03	115.33	75.74
5th Percentile	**116.67**	93.25	64.10	48.32	50.27	0.03