Deep Reinforcement Learning

Matteo Hessel

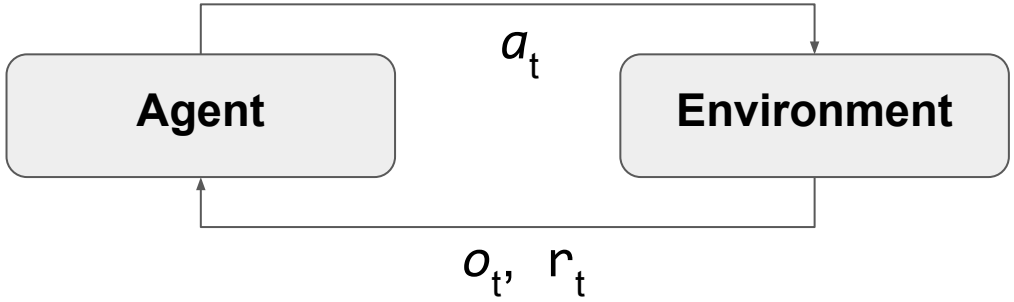
The only stupid question is the one you never asked.

Rich Sutton (allegedly)

1. Introduction

The reinforcement learning problem

A learning system (the **agent**) must learn to act in the universe it's embedded in (the **environment**) to maximize a scalar feedback signal (the **reward**).



Policies

The agent's behaviour is defined by a **policy** $\pi(A_t/S_t)$, i.e. by a probability distribution over actions, conditioned on the current **state**

The agent's objective is to find an **optimal** policy $\pi^*(A_t/S_t)$ that maximises the **return**, i.e. the sum of discounted rewards collected from any state S_t onwards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

the **discount** γ modulates the relative weight of immediate vs distant rewards.

Values

If the environment or the agent is **stochastic**, the return is a random variable, thus the objective is typically formulated as maximizing **expected returns**, or **values**.

$$V(s) = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \quad \rightarrow \quad \textit{state value}$$

$$Q(s, a) = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \quad \rightarrow \quad \textit{state-action value}$$

The expectations are over both the environment and agent stochasticity.

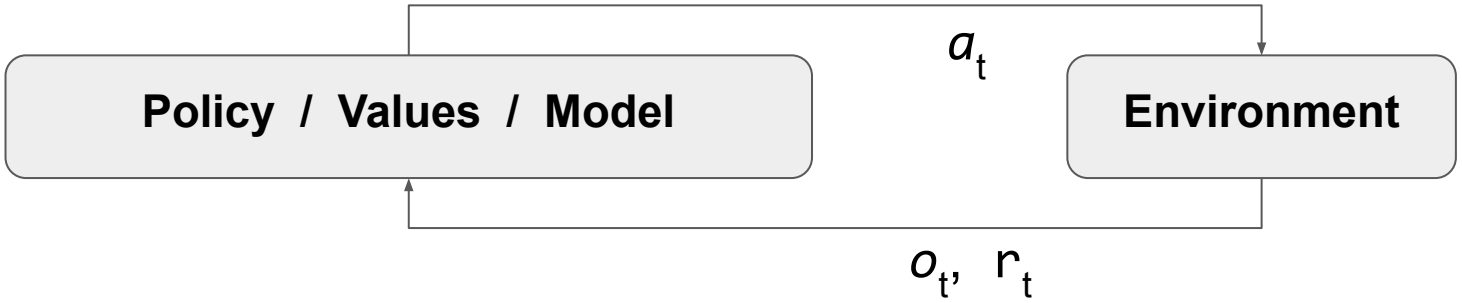
Models

In RL we often assume that we do not know the environment dynamics:

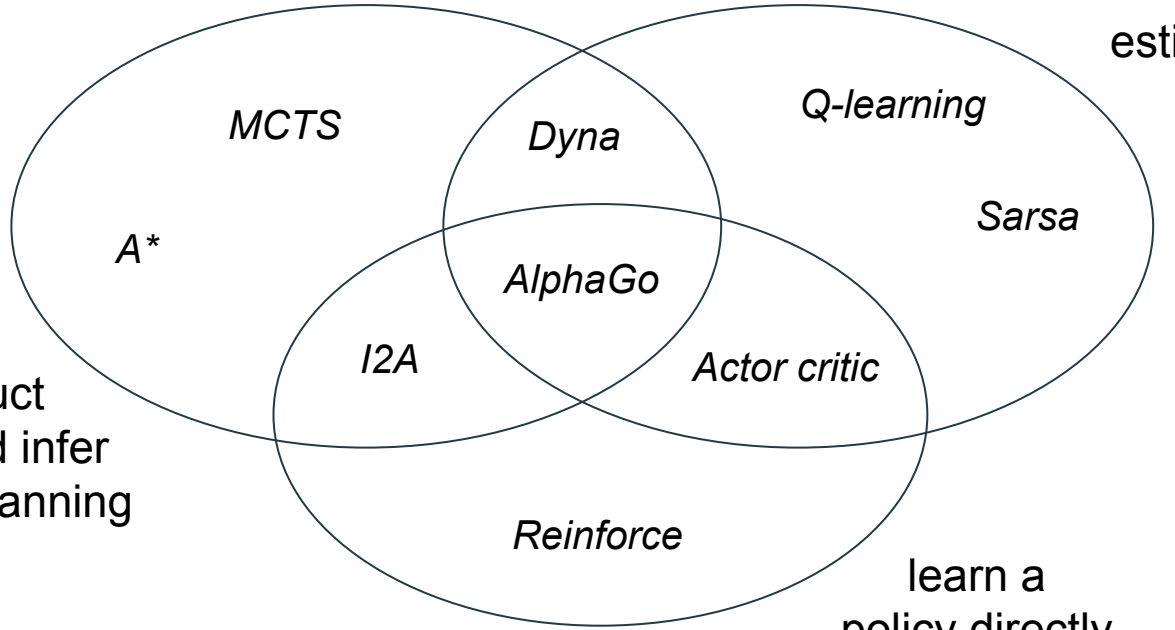
- values and policies may be learned **model-free** from interaction
- we may use the interaction data to learn a **model** of the environment, inferring, from data, how actions affect environment state and rewards.

RL solutions: the Big Picture

- Learn **policy** directly - **execute** it
- Estimate **values** of each action - infer policy by **inspection**
- Construct a **model** - infer policy by **planning**



RL solutions: the Big Picture



estimate values and infer policy by inspection

construct model and infer policy by planning

learn a policy directly

Deep Reinforcement Learning

- *Policy* $\pi : S_t \rightarrow p(A_t)$
- *Value* $v : S_t \rightarrow V(S_t)$
- *Model* $m : S_t, A_t \rightarrow S_{t+1}, R_{t+1}$

Ultimately these are just **functions**,

and we need a flexible way of **representing** and **learning** them

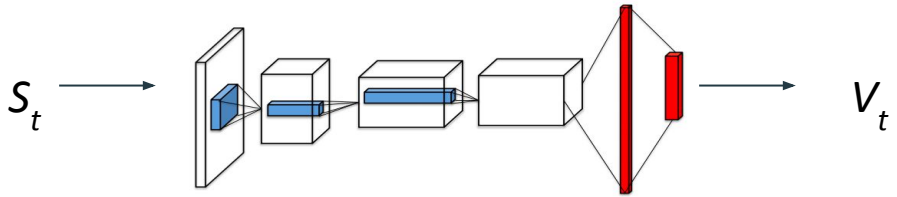
Function approximation

Consider the problem of representing and learning state values V:

A simple choice could be to represent these as a linear function of the state features:

$$v_w(s) = w^T s + b$$

Deep reinforcement learning focuses on the case where the function is represented by a large neural network, whose parameters are trained via **gradient descent** on a suitable loss function



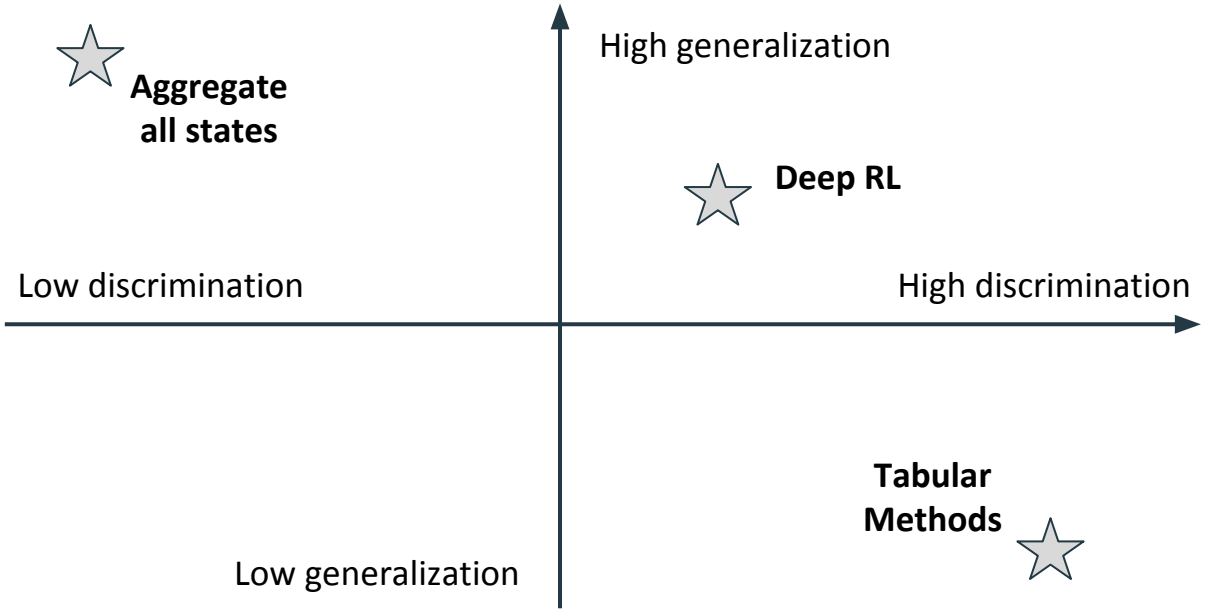
Why Deep Learning?

Deep learning is not the only answer, but it's a pretty convenient one

- **Table look-up**?
  → Learn about each state separately
  → No **generalization**, does not scale to large state spaces

- **Linear**?
  → $y_t = \boldsymbol{w}^T S_t + b$
  → Can work very well if you have a rich state representation S_t
  → Hard-coding a suitable **rich state representation** S_t is difficult!

Why Deep Learning?



Credits: Adam White

2. Introduction to deep RL

MC prediction with deep nets

Consider now the **prediction** problem of estimating state values for a fixed policy π:

Lets approximate the values using a **neural network** $v_\theta(s)$

with parameters θ, state features S as inputs, and a single scalar output.

Deep Monte Carlo Prediction:

$$
\begin{aligned}
L &= (G_t - v_\theta(S_t))^2 \\
\Delta\theta &= -\nabla_\theta L = (G_t - v_\theta(S_t))\nabla_\theta v_\theta(S_t)
\end{aligned}
$$

TD(0) prediction with deep nets - I

In practice Monte-Carlo methods have often high variance

What does a **temporal difference** algorithm for function approximation look like?

Deep TD(0) Prediction:

$$\Delta\theta = (G_t - v_\theta(S_t))\nabla_\theta v_\theta(S_t) \quad \text{[MC]}$$
$$\Delta\theta = (R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t))\nabla_\theta v_\theta(S_t) \quad \text{[TD]}$$

TD(0) prediction with deep nets - II

What objective function is TD(0) optimizing?

→ This update is actually **not the gradient** of any objective function

How do we implement this in practice in the context of common DL frameworks?

$$L = (|R_t + \gamma v_\theta(S_{t+1})| - v_\theta(S_t))^2$$

where we **stop gradients** from flowing into $v_\theta(S_{t+1})$

Q-learning with deep nets

The same approach extends also to **control**.

In this case we strive to estimate the action values $q_\theta(s, a)$ of the optimal policy.

The neural network will now have to output a vector, with a prediction per action

In the tabular setting we may do this with **Q-learning**:

$$\Delta Q(S_t, A_t) \propto R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)$$

Also Q-learning can be extended to function approximation resulting in the update:

$$\Delta\theta = (R_t + \gamma \max_{a'} q_\theta(S_{t+1}, a') - q_\theta(S_t, A_t)) \nabla q_\theta(S_t, A_t)$$

Optimization

Many RL algorithms apply a learning update after each step in the environment using the latest transition $(S_{t-1}, A_{t-1}, R_t, S_t)$ to compute such update:

- Adjacent states, and their values, are **strongly correlated** $\rightarrow$ stochastic gradient descent and the optimizers used in DL assume that the updates are **i.i.d.**
- The transition is then **discarded** $\rightarrow$ in standard DL, it typically requires multiple passes through a dataset to incorporate into a network all useful info

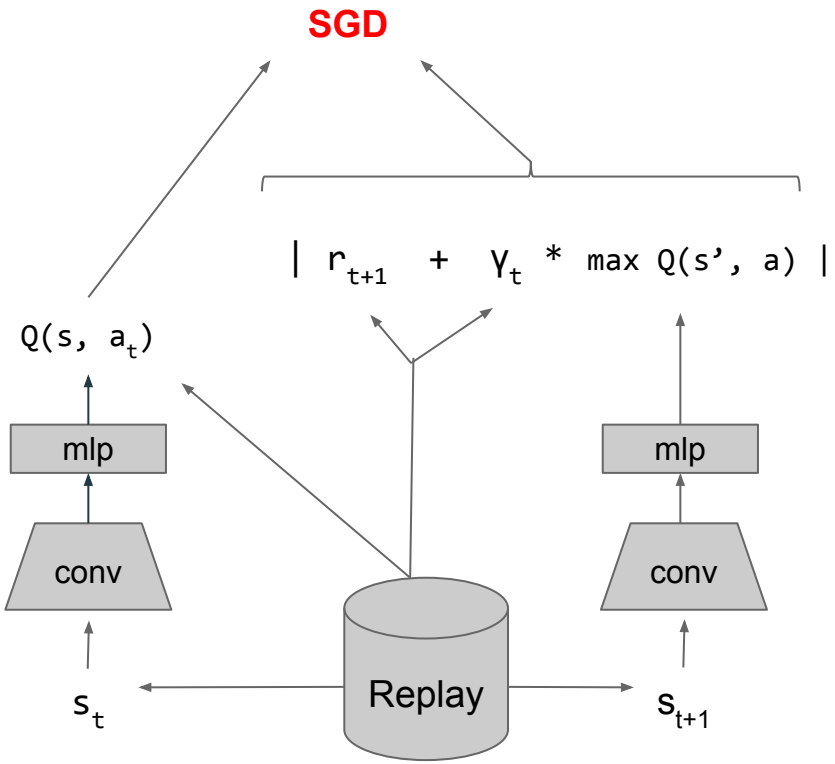
Experience Replay

Compute the Q-learning loss not on the most recent transition in the environment but instead on an old one sampled from a **large memory buffer**

- The larger the buffer, the less correlated consecutive updates
- We can reuse transitions in multiple updates
- We can **prioritize** novel by sampling more often “surprising” data from which there is much to learn
- We can compute updates from batches rather than single samples

Ref: Lin et al, 1992

DQN



Ref: Mnih et al, 2015

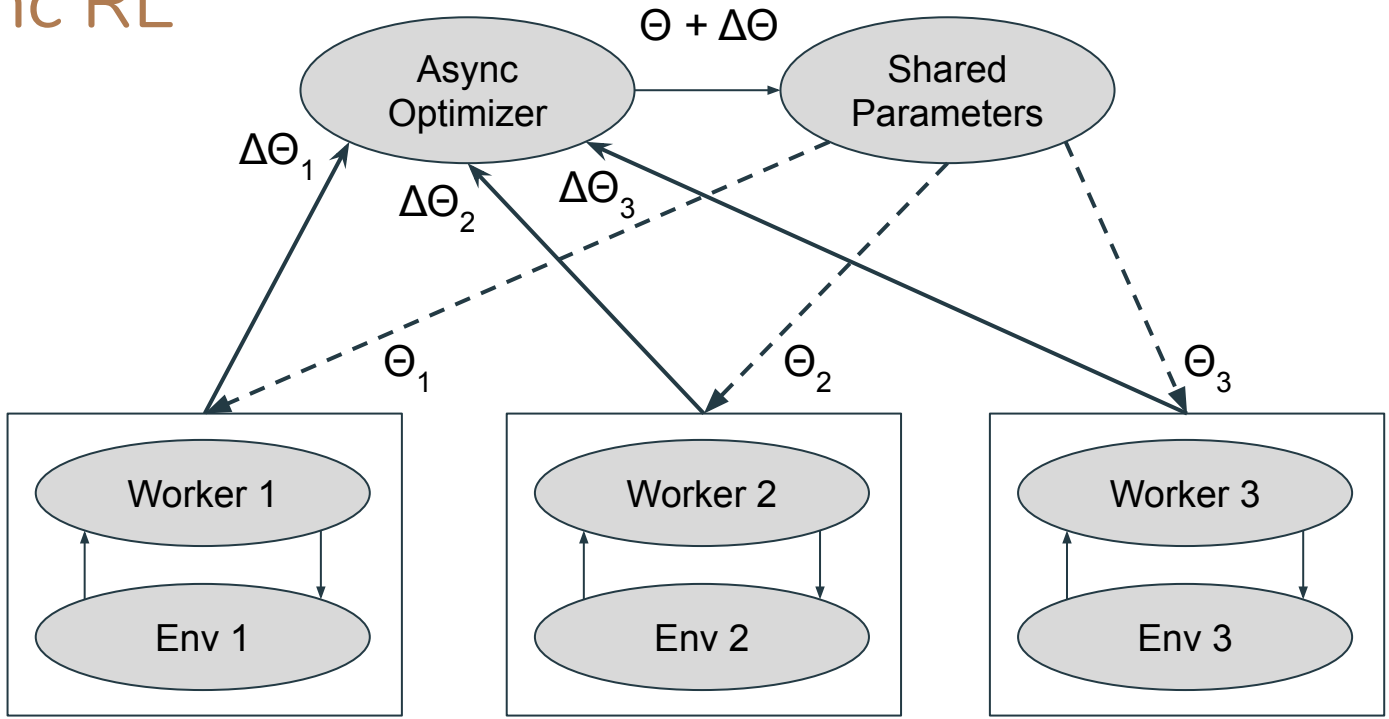
Parallel experience generation

In some cases we may be able to learn from **multiple streams of experience** in parallel:

- Trivial to implement this in simulation
- Internet services where the agent must interact with several users in parallel
- Robot farms, where multiple identical robots attempt the same task

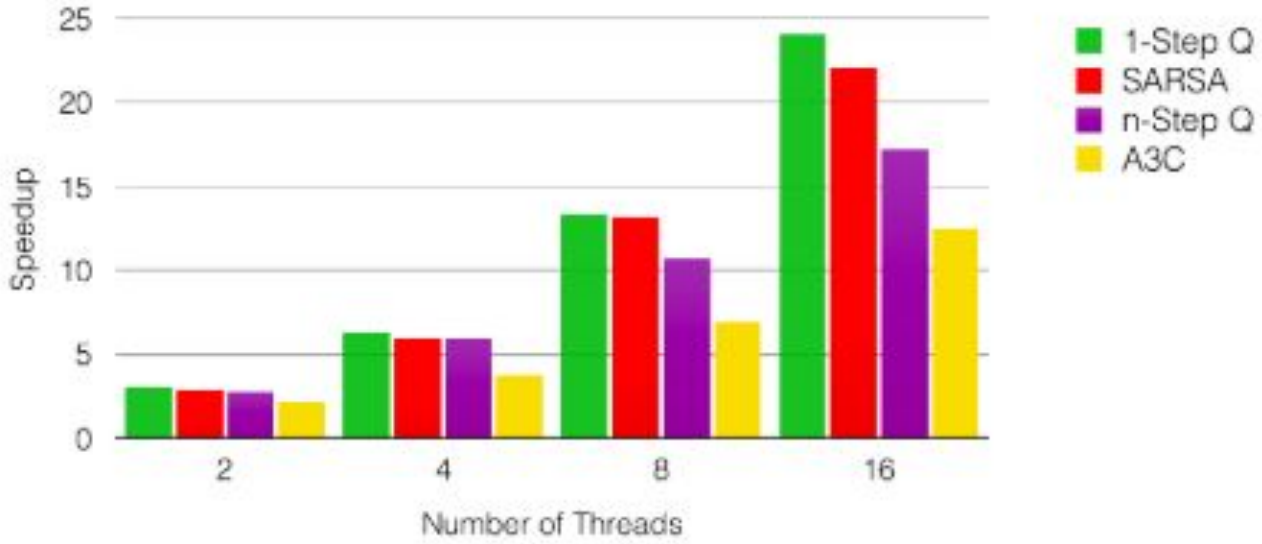
This can also be used to reduce the correlation of updates

Async RL



Ref: Mnih et al, 2016

Async RL


25
20
15
10
5
0
Speedup
2
4
8
16
Number of Threads
1-Step Q
SARSA
n-Step Q
A3C

3. Generalization

Generalization

Generalization is the process of updating your beliefs about one or more states in response to observing novel information about a *different* set of states

The generalization problem is the focus of much of supervised learning, and is framed as

- maximizing performance on a **test set** $D_1 = \{x_i\} \sim p(x)$
- despite only being able to see a different **train set** $D_2 = \{x_i\} \sim p(x)$

In SL generalization performance is optimized for

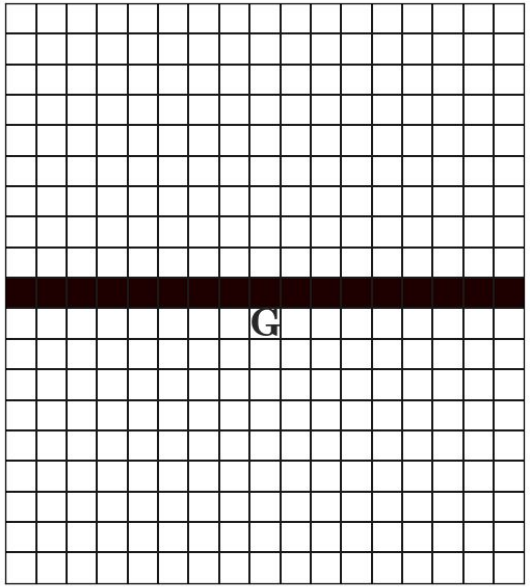
- *explicitly* (through the use of **validation** data and early stopping)
- *implicitly* (through **regularization,** via handcrafted losses or dropout)

Investigating generalization in RL

Consider the grid world on the right and lets

- estimate values for a fixed policy π
- using a neural network $v_\theta(s)$
- trained using SGD on the Monte-Carlo loss

$$L = (G_t - v_\theta(S_t))^2$$

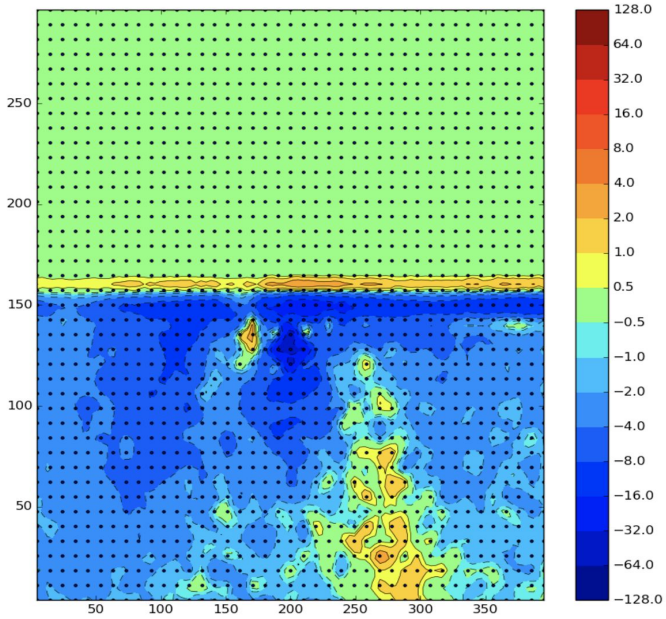


Ref: Penedones et al, 2018

Inappropriate generalization: leakage

Can be difficult to predict how exactly updates to some states will end up affecting other states.

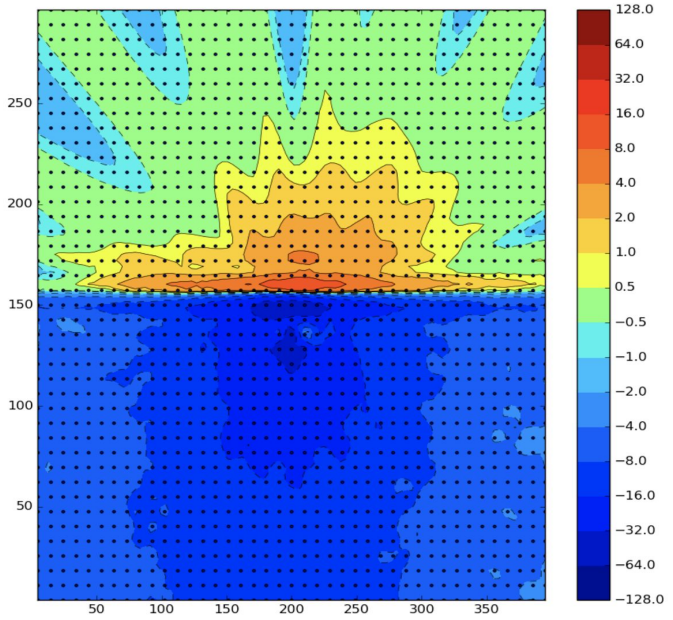
Neural network induce some degree of **smoothness** $\rightarrow$ this inductive bias is reasonable but can lead to **inappropriate generalization** when the dynamics is not smooth



Inappropriate generalization: leakage propagation

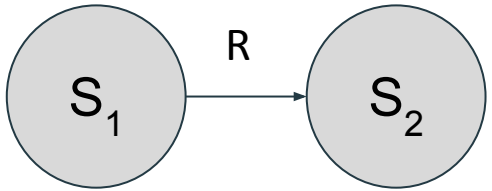
Consider the deep TD(0) algorithm instead:

- It learns much better approximations of the values in the bottom half of the grid
- But causes **leakage propagation**, with high error across the whole of the top half



Inappropriate generalization: feedback loops

Consider the transition on the right:



If we perform deep TD update to a value network $v_\theta(s)$

every time we update the value of state S_1,

we will, through generalization, also perturb the values of S_2

Because we use our own estimates to construct the target $R + \gamma v_\theta(S_2)$ for $v_\theta(S_1)$

This can result in **feedback loops**!

The deadly triad

Function Approximation → Estimate values by training the parameters of some function.

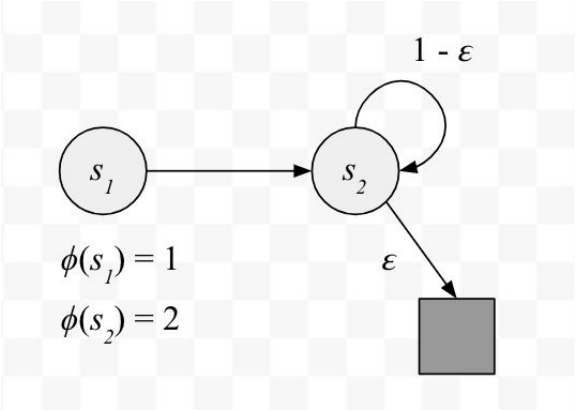
Bootstrapping → Using the current agent's value estimates to construct the targets used to update those same agent's estimates.

Off-Policy → Compute learning updates from a distribution that differs from that arises under the policy whose values are being estimated and the given environment dynamics.

Ref: Sutton and Barto, 2018

A simple example

A very simple example of to understand the issue was provided by Tsitsiklis and Van Roy



Simplest possible parametrization of the value

$$v_w(s) = w\phi(s)$$

Optimal solution is trivially:

$$w^* = 0$$

Ref: Tsitsiklis and Van Roy, 1997

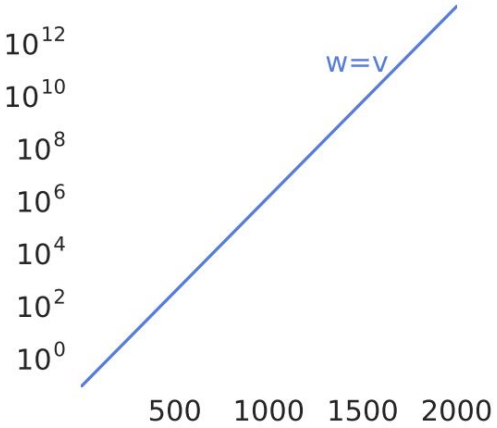
How does DQN work?

Function Approximation $\rightarrow$ DQN uses neural networks to approximate values.

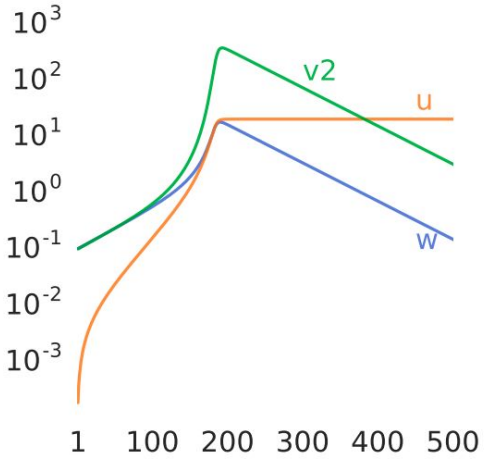
Bootstrapping $\rightarrow$ DQN uses Q-learning with bootstrap targets $R_t + \gamma \max_a Q(S_{t+1}, a)$

Off-Policy $\rightarrow$ DQN uses Q-learning, which is off-policy by construction

The specifics matter!



(b) $v(s) = w\phi(s)$ diverges.

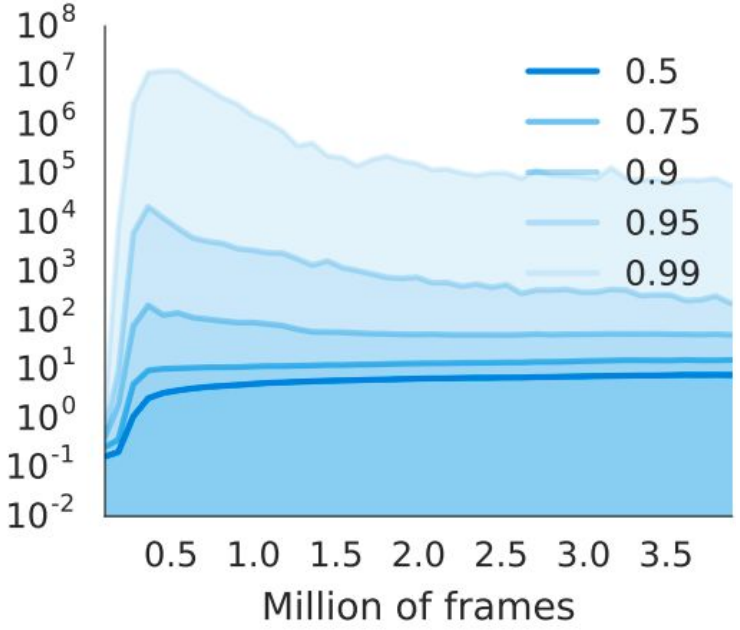


(c) $v(s) = w(\phi(s) + u)$ converges.

Ref: van Hasselt et al, 2018

Soft-divergence

This form of **soft-divergence** where values grow to unreasonably high values but then converge back to sensible estimates, is more common in practice than unbounded divergence.



Target Networks

Target Networks are a common remedy to this type of inappropriate generalization

We can keep **two copies** of the neural network used to estimate Q values:

- The parameters of one are updated at each step,
- The parameters of the other are a slow copy of the first one.

This reduces the opportunity for feedback loops.

Ref: Mnih et al, 2015

Multi-step deep Q-learning

Using our own estimates Q_θ to construct targets $R_t + \gamma \max_a Q_\theta(S_{t+1}, A_{t+1})$ is called **bootstrapping**.

We don't however need to bootstrap necessarily after a single step.

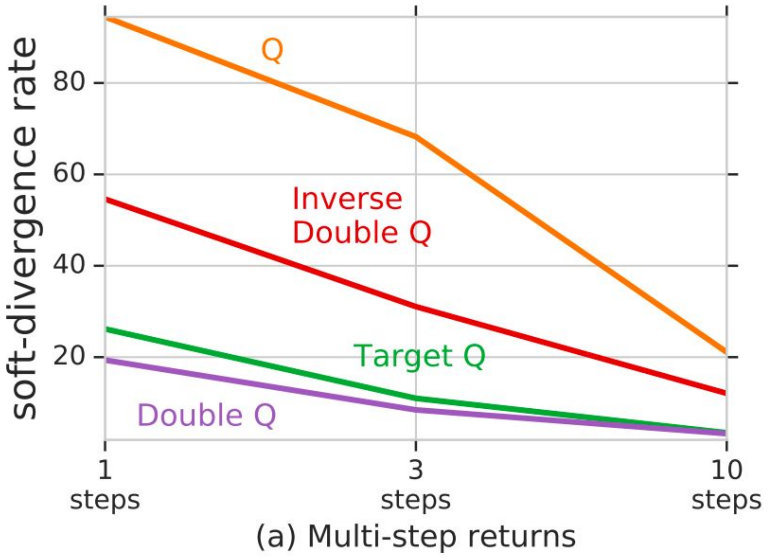
We can construct **multi-step targets**:

$$R_t + \gamma R_{t+1} + \ldots + \gamma^{K-1} R_{t+K-1} + \gamma^K \max_a Q_\theta(S_{t+\mathbf{k}}, A_{t+\mathbf{k}})$$

Ref: Sutton and Barto, 2018

Multi-step targets and soft-divergence

multi-step returns **reduce** the reliance of our updates on bootstrapping, and the likelihood of observing divergence

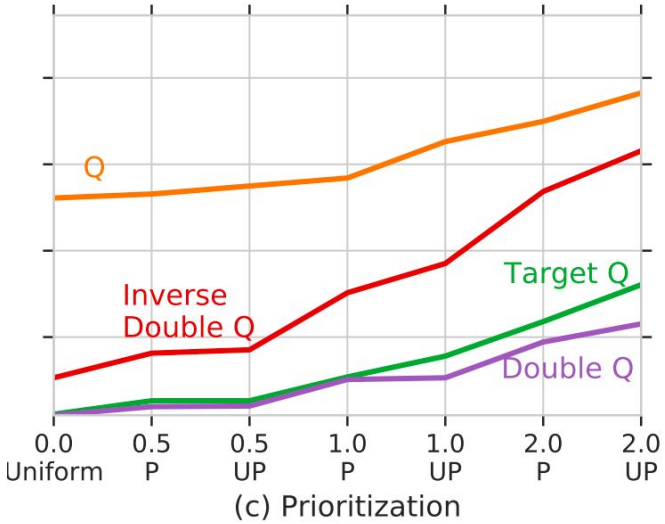


Ref: van Hasselt et al, 2018

Prioritized replay and soft-divergence

Prioritized replay makes the data distribution more off-policy, and therefore increases the likelihood of divergence.

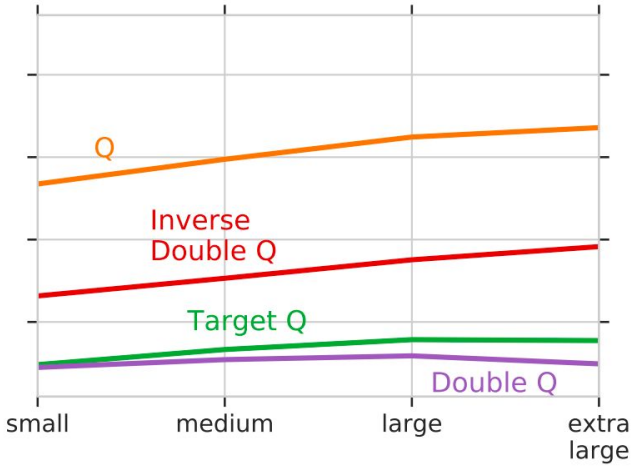
→ importance sampling can help mitigate the effect of prioritization



Ref: van Hasselt et al, 2018

Network size and soft-divergence

The likelihood of observing divergence is often higher for **larger neural networks**



(b) Network size

Ref: van Hasselt et al, 2018

Model-based RL and the deadly triad

Model-based RL can also trigger the deadly triad:

Given a learned environment model, consider the model-based prediction algorithm:

initialize $v_\theta(s)$

while True:

$S_t, A_t \leftarrow$ memory.sample()

$R_t, S_{t+1} \leftarrow$ model.query(S, A)

$\Delta\theta = (R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t))\nabla_\theta v_\theta(S_t)$

This also results in divergence if the state distribution in memory doesn't match the model's state distribution

Ref: van Hasselt et al, 2019

4. RL-aware deep learning

RL-aware deep learning

It’s important to understand how RL algorithm interact with function approximation

1. Experience Replay
2. Target networks

} → **DL-aware reinforcement learning**

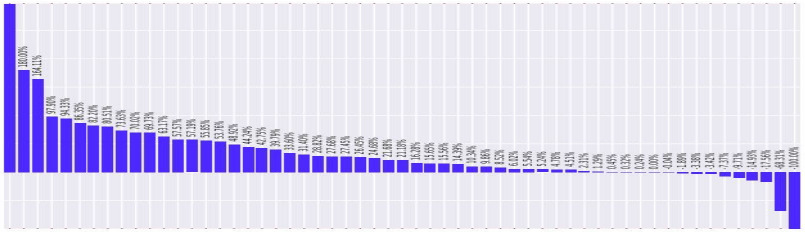
However, we should also design deep learning architecture with the right inductive biases for RL → **RL aware Deep Learning**

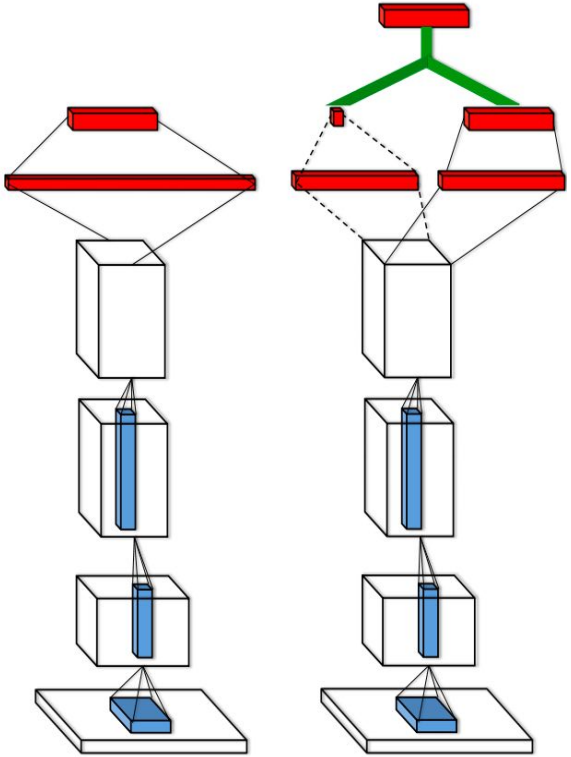
Dueling Networks

Value-Advantage decomposition of Q-values

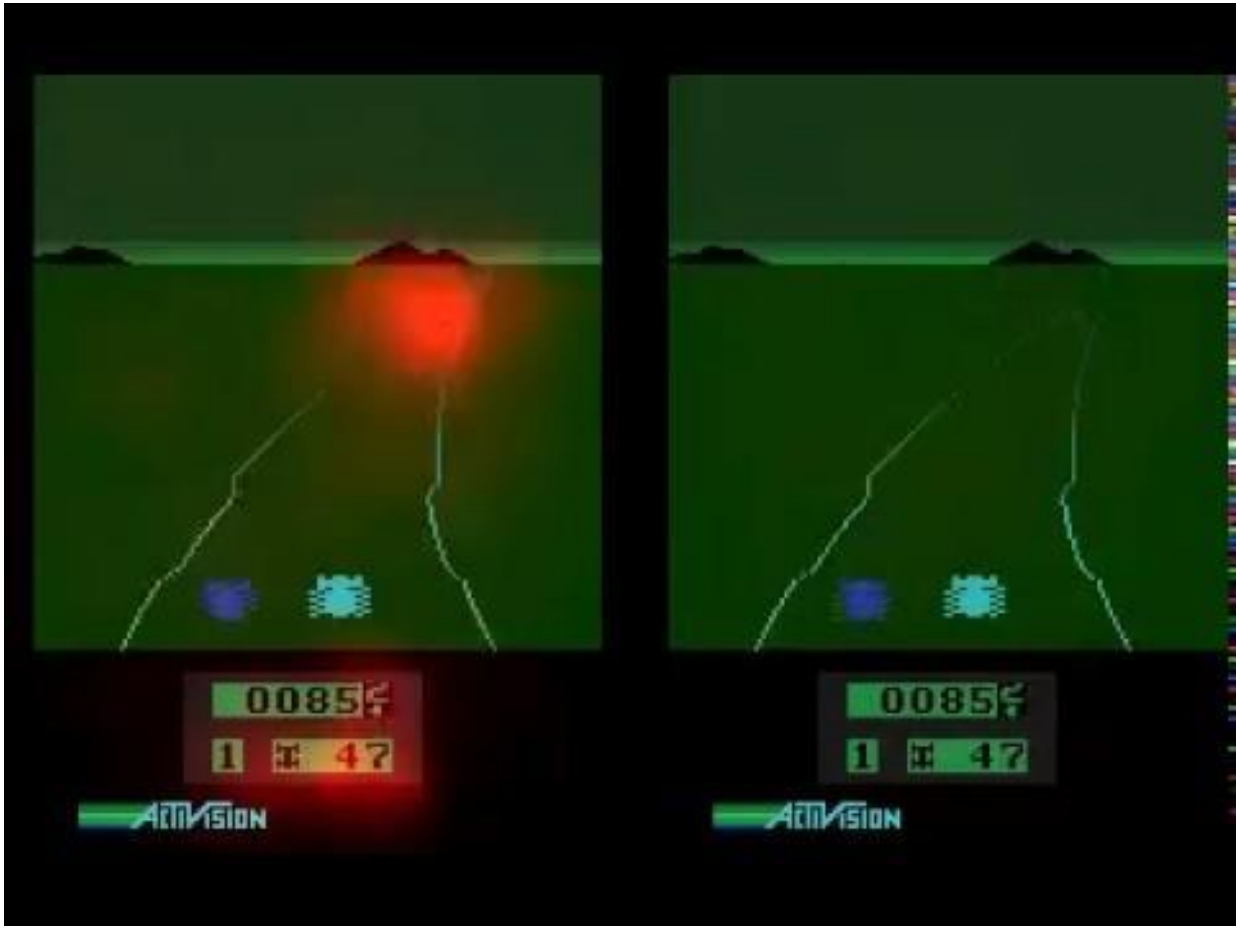
$$Q^{\pi}(s,a) = V^{\pi}(s) + A^{\pi}(s,a)$$

Can we encode this as a suitable deep neural network architecture?





Ref: Wang et al, 2016

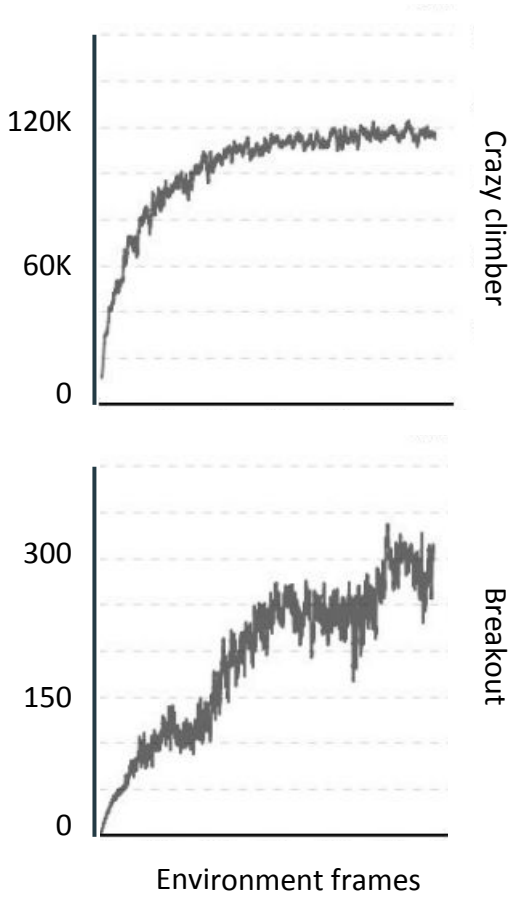

0085
1 47
ACTIVISION
0085
1 47
ACTIVISION

Reward Scaling

An RL task is define by its rewards

- Go: win (1), loss (0)
- Atari: change in score
  (+-1 in Pong, +-1000s in MsPacman)

The magnitude of updates scales directly with the magnitude/frequency of rewards, which is **non-stationary.**



PopArt

PopArt is a principled solution for

- Rescale gradients / updates
- Doing so **adatively**

Can useful beyond RL, whenever applying deep-learning to problems where the scale of the predictions changes over time (e.g. certain time series prediction problems)

Ref: Van Hasselt et al. 2016, Hessel et al. 2019

Art

$V(s) = \mu + \sigma * U^n(s)$

$\mu = (1 - \beta)\,\mu + \beta\, G$

$v = (1 - \beta)\, v + \beta\, G^2$

$\sigma = v - \mu^2$

Adaptively Rescale Targets

$U^n(s) = W^T x + b$

$\sigma \rightarrow \sigma', \quad \mu \rightarrow \mu'$

$W' = \sigma/\sigma' * W$

$b = (\sigma * b + \mu - \mu')\,/\sigma'$

Preserve Output Precisely

Pop

$V(s) = \mu + \sigma * U_n(s)$

$\mu = (1 - \beta)\,\mu + \beta\, G$

$v = (1 - \beta)\, v + \beta\, G^2$

$\sigma = v - \mu^2$

Adaptively Rescale Targets

$U_n(s) = W^T x + b$

$\sigma \rightarrow \sigma', \quad \mu \rightarrow \mu'$

$W' = \sigma/\sigma' * W$

$b = (\sigma * b + \mu - \mu')\,/\sigma'$

Preserve Output Precisely

Normalized value learning

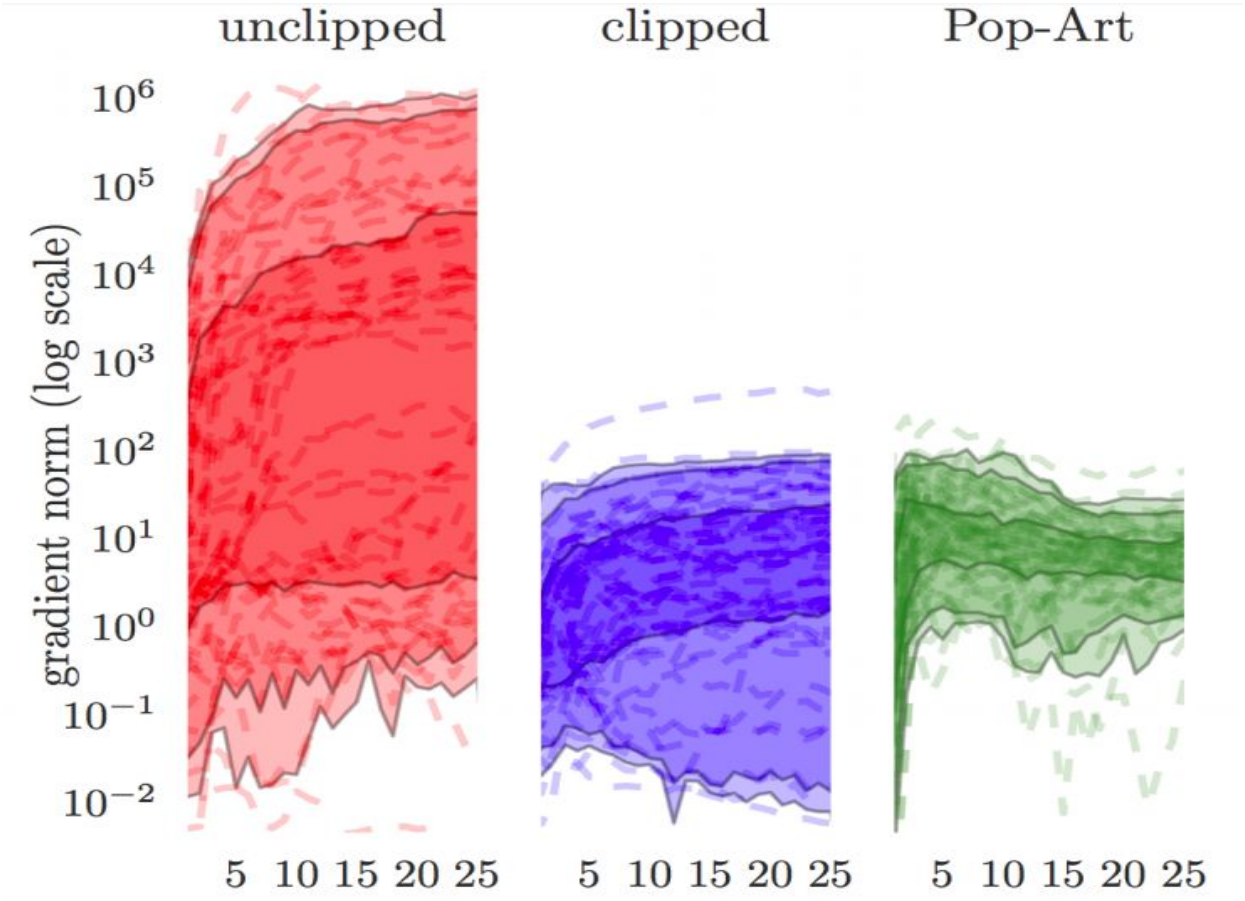
We can rewrite our value learning update

$$(R_t + \gamma v_\theta(S_{t+1}) - v_\theta(S_t))^2$$

in normalized space, by:

- using the unnormalized value V to bootstrap
- then normalizing the target before computing the loss for U

$$([R_t + \gamma V_\theta(S_{t+1}) - \mu]/\sigma - U_\theta(S_t))^2$$


unclipped
clipped
Pop-Art
gradient norm (log scale)
10^6
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
5 10 15 20 25
5 10 15 20 25
5 10 15 20 25


DQN with Pop-Art &
without clipped rewards
20

The End