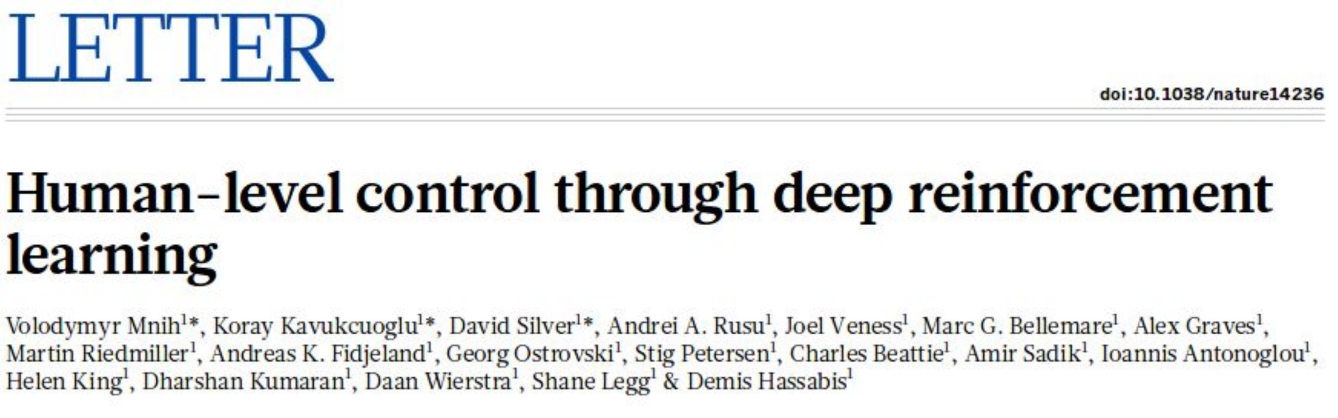
Lesson 5: Deep Q-Network

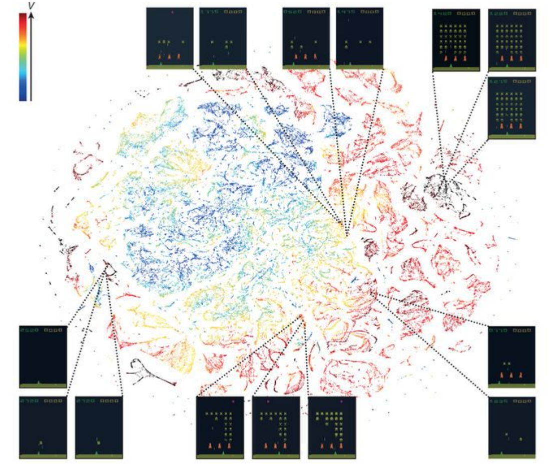
NQL -> DQN: What’s the difference?

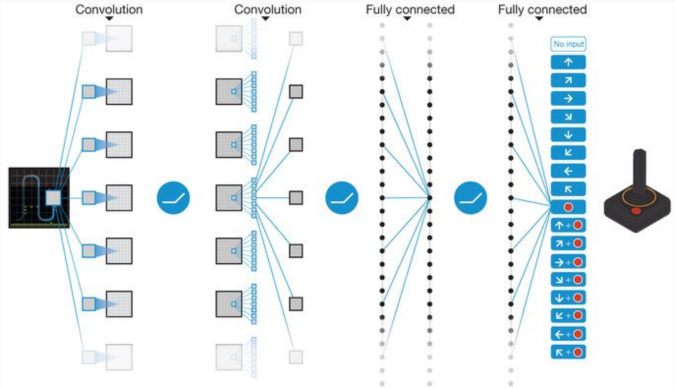

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]





3 Main innovations:

1. ConvNet Value approximator

.

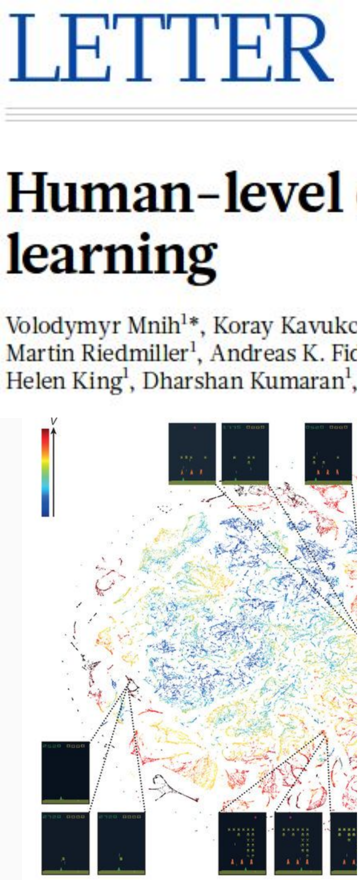
.

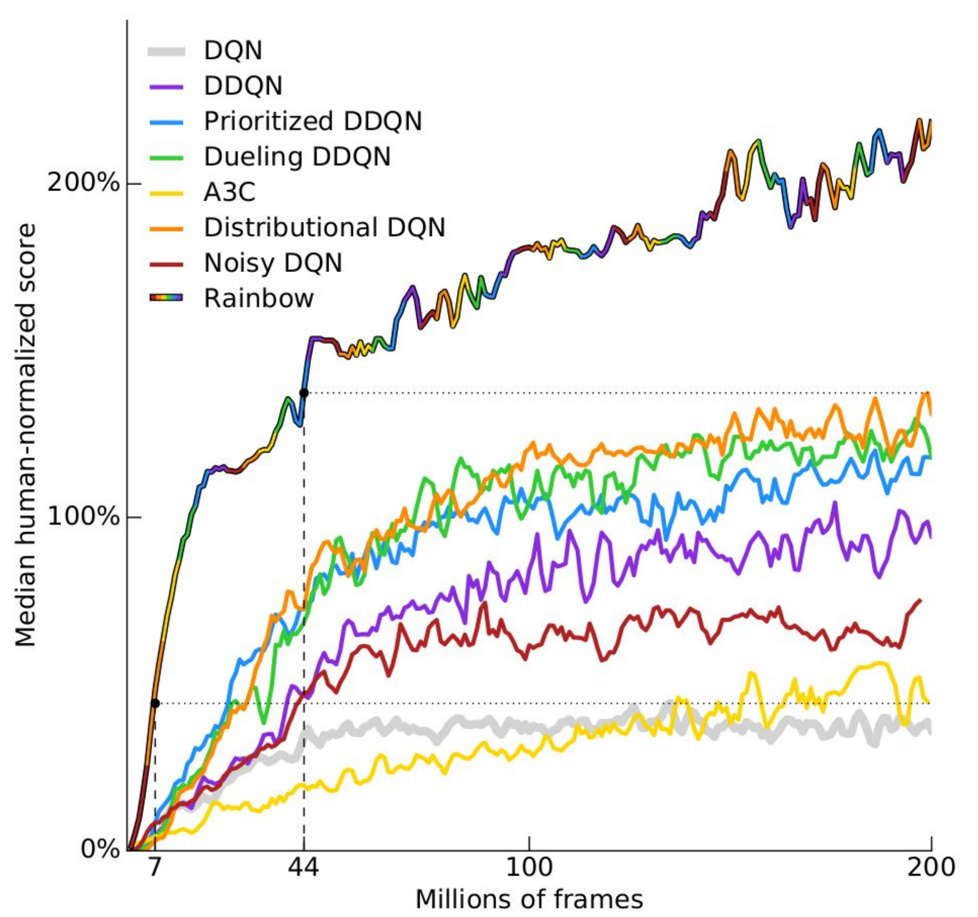
.

CONVNET VALUE APPROXIMATOR!!??

2. Experience Replay
3. Target Networks

NQL -> DQN: What's the difference?





3 Main innovations:

1. ConvNet Value approximator

.

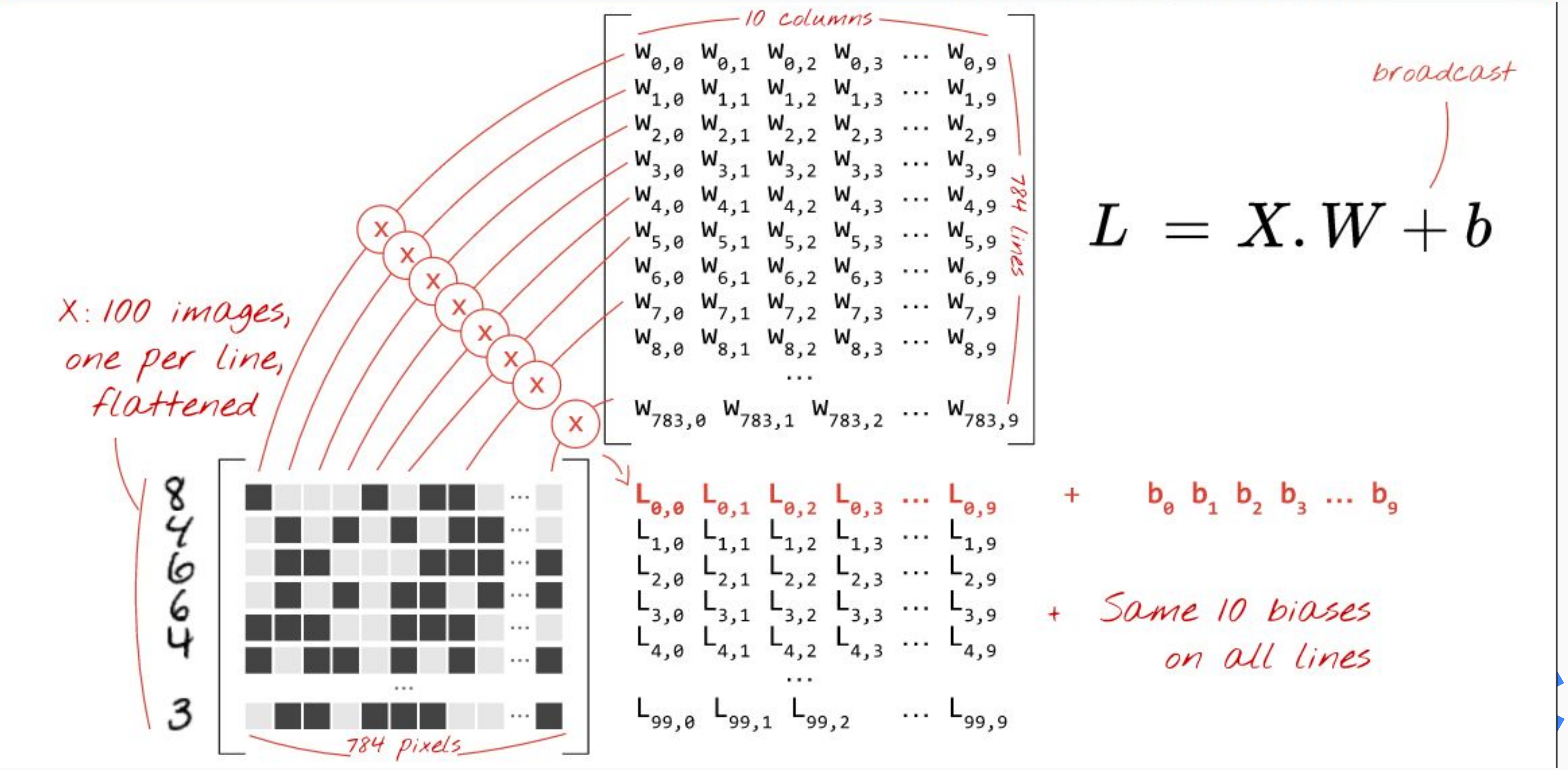
.

.

CONVNET VALUE APPROXIMATOR!!??

2. Experience Replay
3. Target Networks

But first: Batching



Experience Replay

Algorithm 1: Online-Q

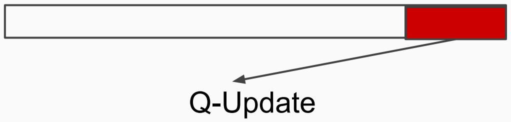
Initialize the value function q
while *not converged* **do**
    Get the initial state s
    while s *is not the terminal state* **do**
        Select an action a according to a ϵ-greedy policy derived from q
        Execute the action a, get the reward r and the next state s'
        Update the value function q with (s, a, r, s') following the Q-learning update rule
        $s = s'$
    end
end

Algorithm 2: Buffer-Q

Initialize the value function q
Initialize the replay buffer $\mathcal{M}$
while *not converged* **do**
    Get the initial state s
    while s *is not the terminal state* **do**
        Select an action a according to a ϵ-greedy policy derived from q
        Execute the action a, get the reward r and the next state s'
        Store the transition (s, a, r, s') into the replay buffer $\mathcal{M}$
        Sample a batch of transitions $\mathcal{E}$ from $\mathcal{M}$
        Update the value function q with $\mathcal{E}$ following the Q-learning update rule
        $s = s'$
    end
end

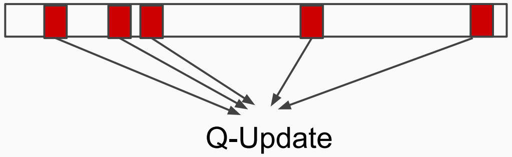
Key Point:
We don't sample from just recent experiences but from all.

Batch:

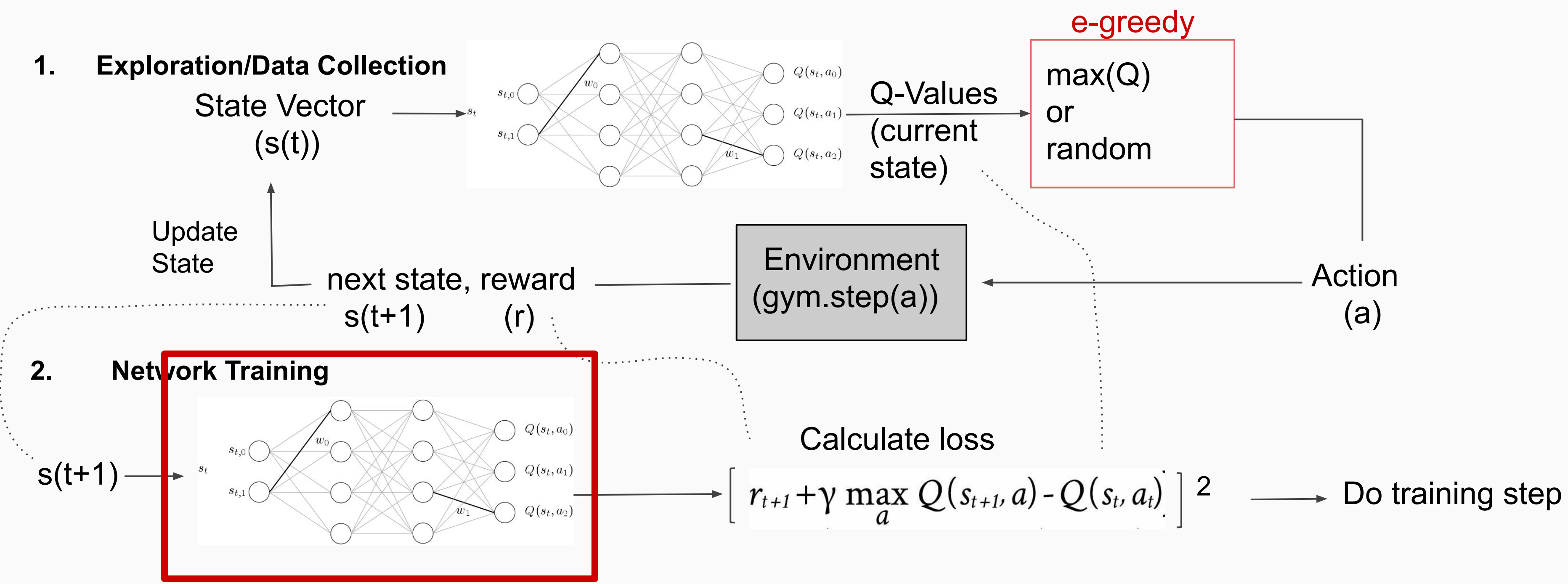


vs.

Replay:



Target Networks



Freeze this network, update every N steps

New Loss: $\left[r_{t+1} + \gamma \max_a \boxed{Q(s_{t+1}, a)} - Q(s_t, a_t) \right]^2$

Target Networks

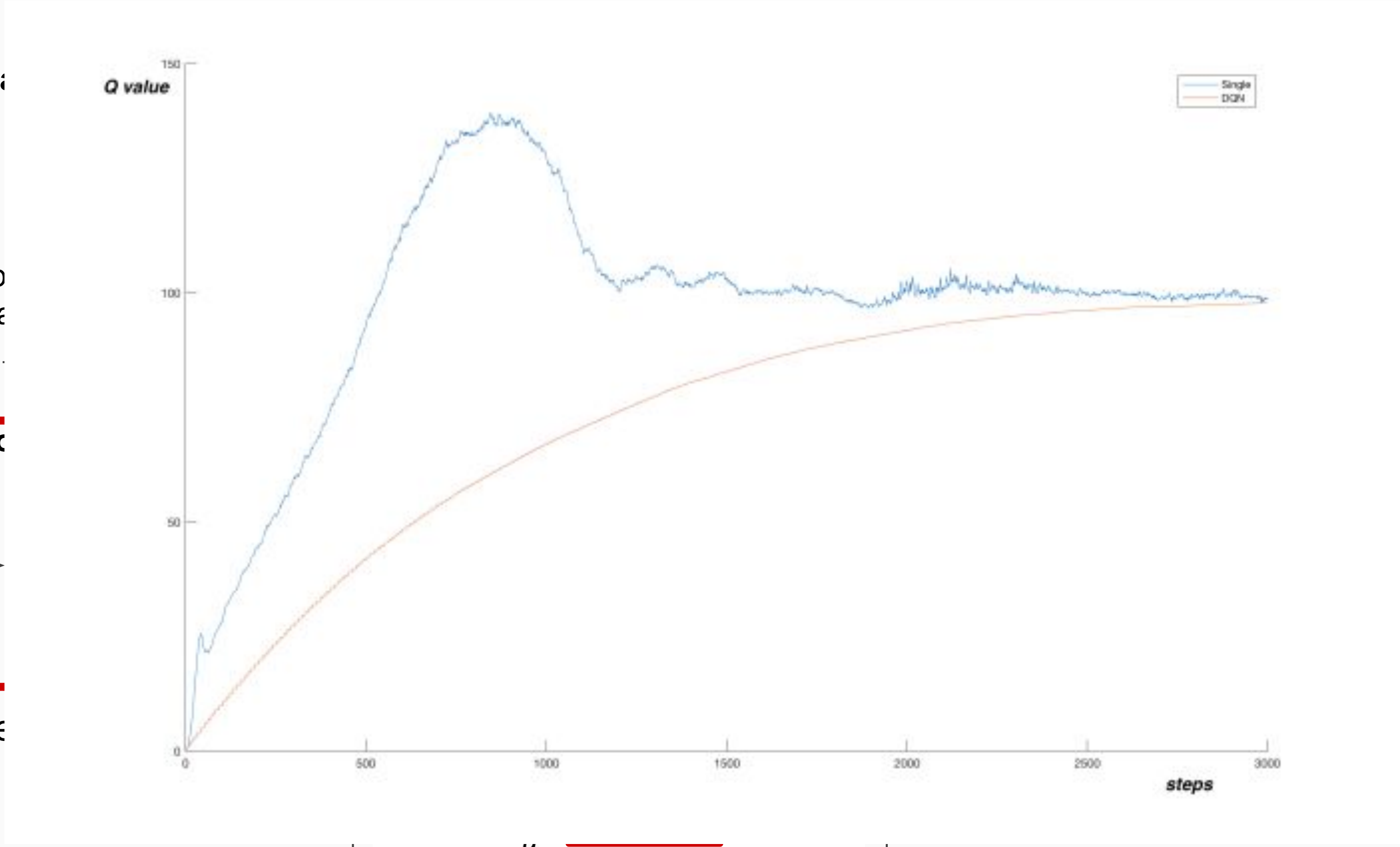
1. **Explora**

Up
Sta

2. **Netwo**

s(t+1)

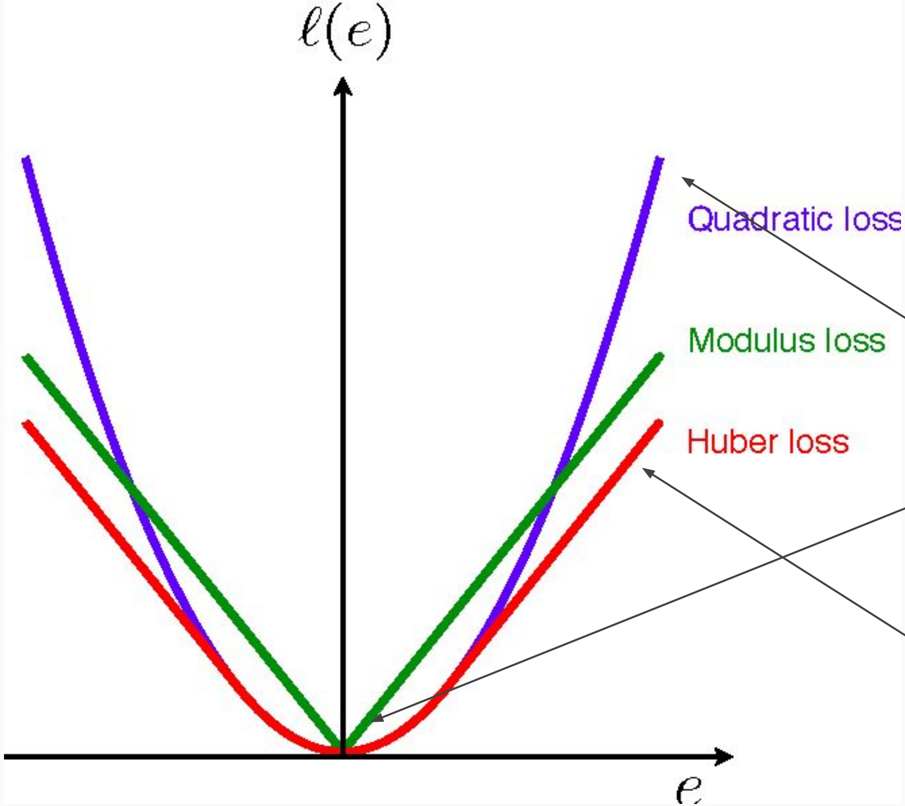
Free



on
a)

o training step

One last thing: Huber Loss



$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

For our purposes:
`tf.losses.huber_loss()`

Outliers have disproportionately strong effect

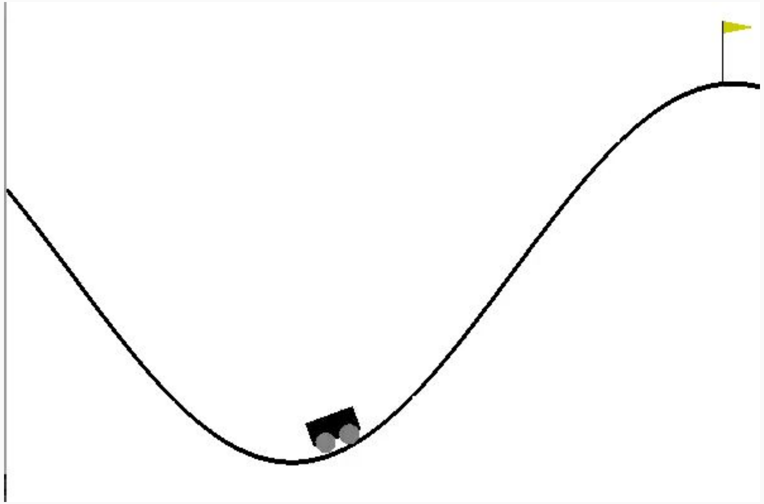
Sensitive to small perturbations

Best of both

Where to from here?

If you want to continue on this code-base:

- Visualize training with tensorboard
- Try to implement some extensions: Double-DQN, Dueling Networks etc.
- How would any of this work in continuous action spaces?
- Try some of the other environments, MountainCar, Inverted pendulum etc.
- Can you get it working with a Convnet?
    - Warning: training times get out of hand quickly.
    - Test on the ATARI Envs



Want to learn about other types of RL?

- Policy Gradients
    - A3C, VPG, TRPO, PPO etc.
- Hybrid/Integrated Methods:
    - UNREAL, NEC, Successor Learning etc.
- Applications/Misc.
    - Inverse RL/Imitation Learning,

Resources:

- To many to list here: UC Berkeley has tons of great stuff, start with their course or RL book by Richard Sutton