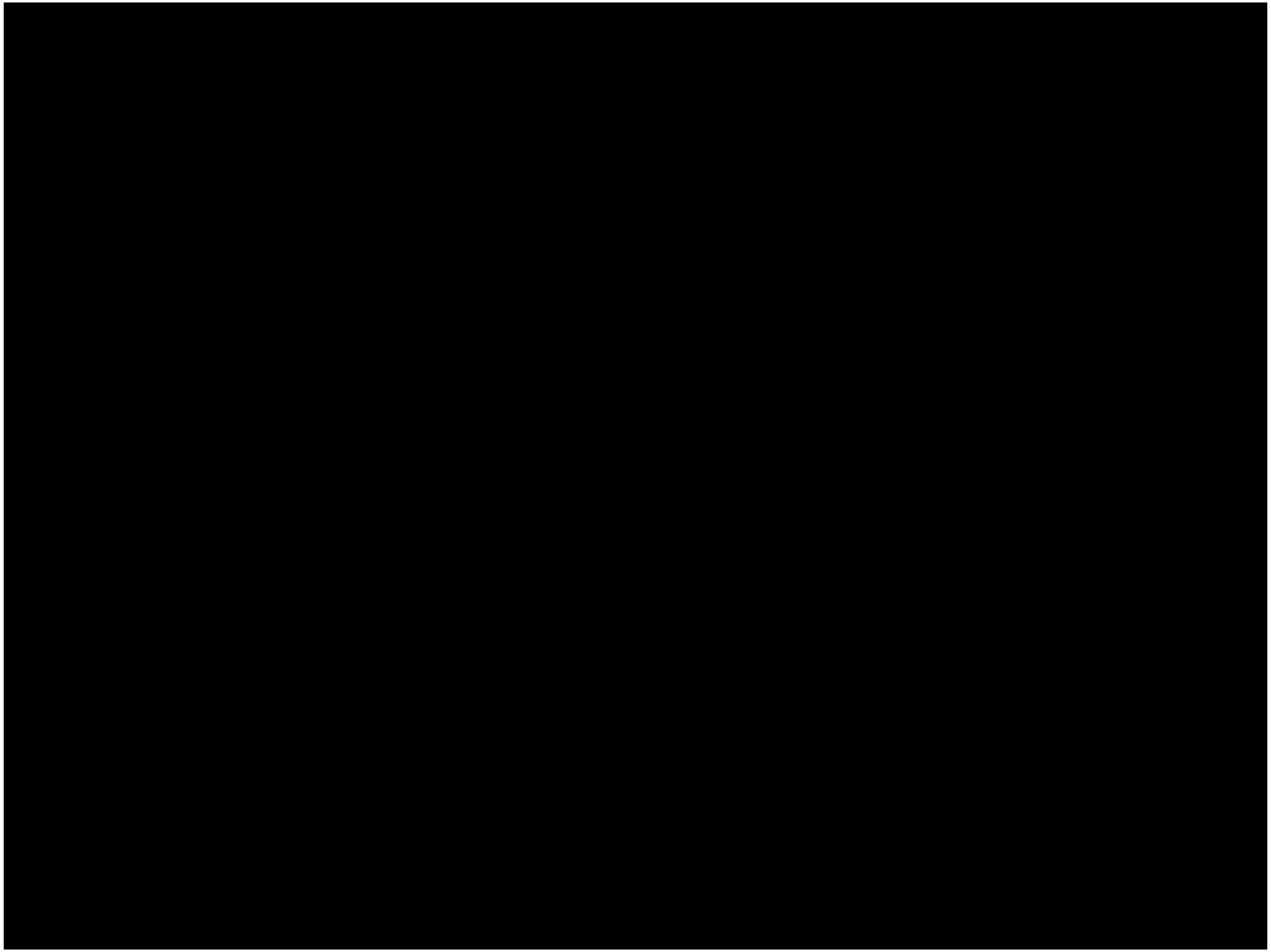
Master
Reinforcement Learning 2022
Lecture 3:
Deep Value Based Methods

Aske Plaat

Different Approaches

- Model-free
    - Value-based [2,**3**]
    - Policy-based [4]
- Model-based
    - Learned [5]
    - Perfect; Two-Agent [6]
- Multi-agent [7]
- Hierarchical Reinforcement Learning (Sub-goals) [8]
- Meta Learning [9]

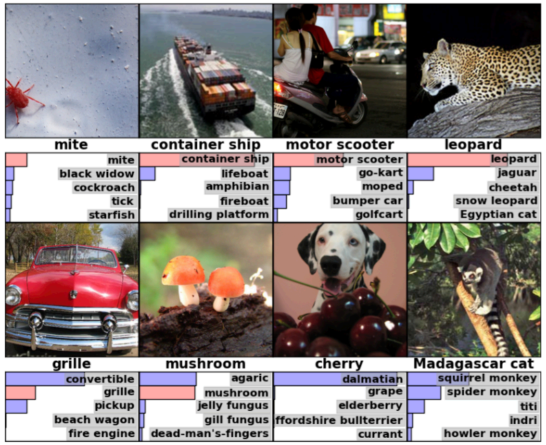
Motivation



The problem of solving Large Problems

Large Problems

- Deep Learning can learn end-to-end features in high-dimensional problems and large state spaces
- Curse of dimensionality
- Dimensionality of 100 x 100 pixels is state space of 256^{10000}
- How to learn? Never enough samples to fill state space
- Approximate, Generalize. Exploit smoothness



Question

- It is 2012

- In SL, End-to-end learning has just had the breakthrough of a lifetime at the ILSVRC, using CNN & GPU

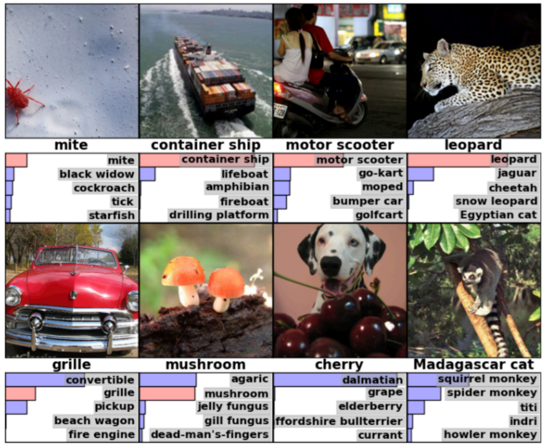
- In RL, state of the art in NN is still TD-Gammon on a small network (40 hidden units), from 1992

- What can we do with Deep CNN in reinforcement learning?

Arcade Learning Environment

Atari Learning Environment

- Deep SL is driven by ImageNet



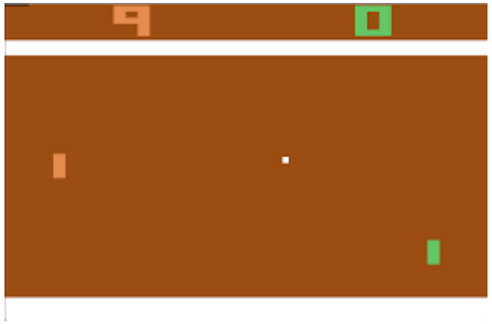
- Deep RL is driven by ALE

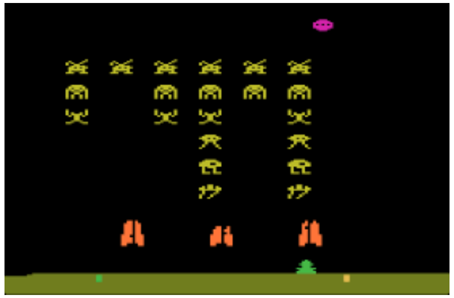
Atari 2600

- console
- ROM game cartridge
- 128 bytes RAM
- inputs: pixels
- output: joystick actions

- *[how to reverse engineer 128 bytes of RAM using Gigabytes of neural networks]*

Atari Games

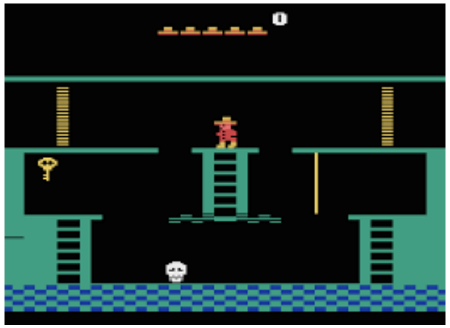


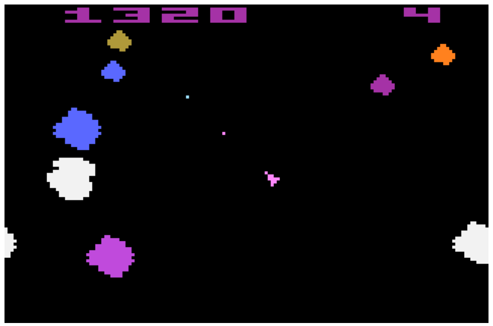


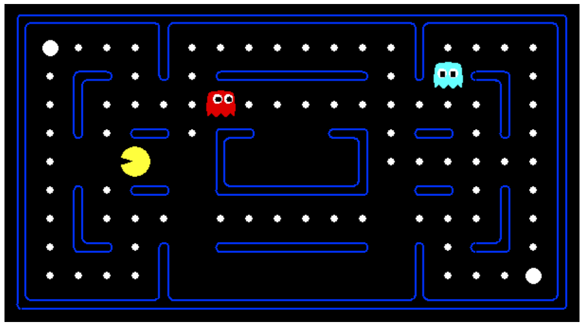

58
SCORE
STAR
21
DATE
3
LEVEL
HI
0
SCORE


OXYGEN
ACTIVISION


000044
SECTOR 01
14




1320



Mnih et al. [2013/2015]

- Atari results
- A computer learns to play 6 Atari 1980's console games
- End-to-end
- From pixels-to-joystick
- Emulator for console

Why is this such a Breakthrough?

Deep RL

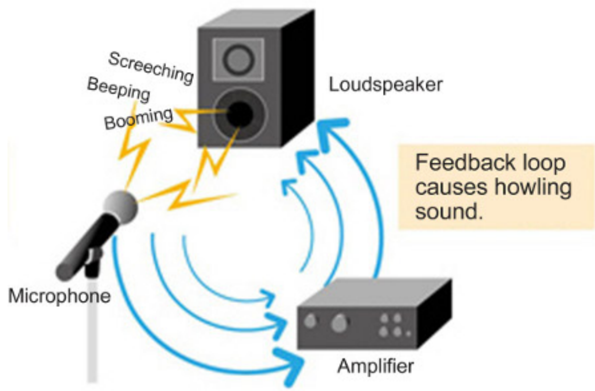
- Supervised: (image, label) -> correct?
  minimizing weights on loss-function

- DRL: (pixels, action) -> reward?
  converging Q-values, minimizing Q-loss

- Should be doable, right?

Deep RL Challenges

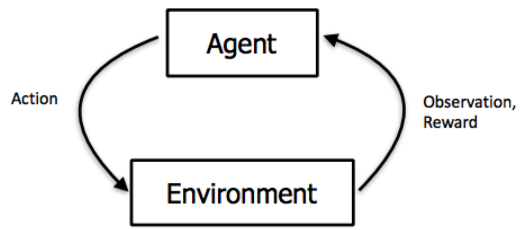
- Computational load
  30 video frames/second



- High-dimensional input
  Does not work with Tabular Q-learning

- **Value Function Approximation has long been known to be Theoretically Inherently Unstable**

- The problem is learning from **feedback.** This causes unstable learning targets



Deadly Triad

- Value Function Approximation is Unstable
  "Deadly Triad" of
  1. function approximation [Deep]
  2. off-policy learning [Q learning]
  3. bootstrapping [TD]

Three Problems

- Coverage
- Correlation
- Convergence

Coverage

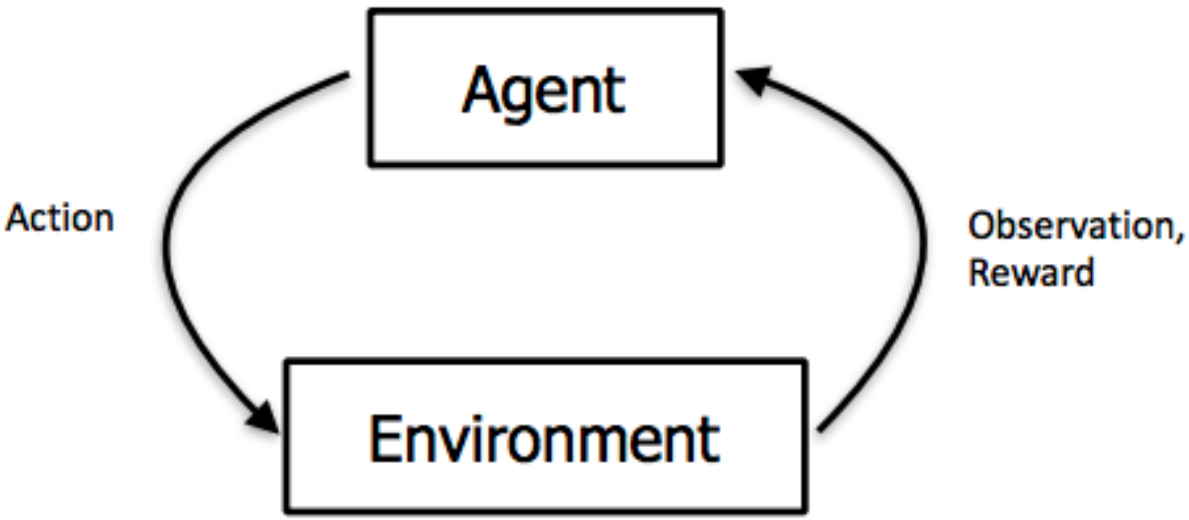
- Convergence proof of Value Iteration, Q-learning and SARSA depend on covering the entire state space, in the end.

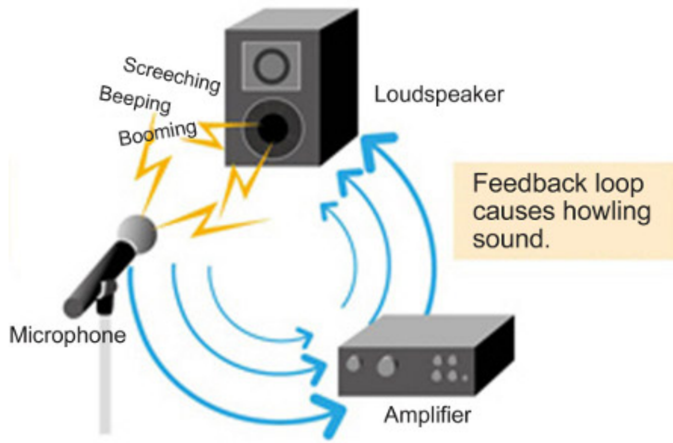
- Not even close, in high dimensional problems

- Human learning also suffers from coverage problems: training against the same sparring partner leads to a narrow skill set.

Correlation

- Supervised: database examples are uncorrelated. Stable learning
- Deep: actions determine next state that will be learned from





Convergence

$$(\hat{f}(x) - y)^2$$

- Supervised: Minimization Loss-target y is fixed
- Reinforcement: Convergence Loss-target Q_{t-1} is moving

$$\gamma \max_{a'} Q_{\theta_{i-1}}(s', a') - Q_{\theta_i}(s, a)$$

- Converging on a moving target is hard

Supervised Minimization

```
def train_sl(data, net, alpha=0.001):      # train classifier
    for epoch in range(max_epochs):        # an epoch is one pass
        sum_sq = 0                         # reset to zero for each pass
        for (image, label) in data:
            output = net.forward_pass(image) # predict
            sum_sq += (output - label)**2 # compute error
        grad = net.gradient(sum_sq)        # derivative of error
        net.backward_pass(grad, alpha)     # adjust weights
    return net
```

RL Convergence

```
def train_qlearn(environment, Qnet, alpha=0.001, gamma=0.0,
    epsilon=0.05
    s = s0                  # initialize start state
    for epoch in range(max_epochs): # an epoch is one pass
        sum_sq = 0  # reset to zero for each pass
        while s not TERMINAL: # perform steps of one full episode
            a = epsilongreedy(Qnet(s,a)) # net: Q[s,a]-values
            (r, sp) = environment(a)
            output = Qnet.forward_pass(s, a)
            target = r + gamma * max(Qnet(sp))
            sum_sq += (target - output)**2
            s = sp
        grad = Qnet.gradient(sum_sq)
        Qnet.backward_pass(grad, alpha)
    return Qnet     # Q-values
```

DQN [Mnih 2013]

- Coverage
- Correlation
- Convergence

- High Exploration
- Replay Buffer
- Low α,
  Infrequent weight updates [2015]

Replay Buffer [2013]

- Store all experience in buffer
- Sample from the history buffer
- Choose action epsilon greedy

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	12	49	-2	26	5	17	-6	84	72	3

Infrequent Weight Updates [2015]

$$(\hat{f}(x) - y)^2$$

- Introduce a separate target network for the convergence targets ("y")

- Every c updates, clone the network to a target network

- Adds delay between updates of the network and the use of the updates in other states

- Reduces oscillations or divergence of the policy

$$\gamma \max_{a'} Q_{\theta_{i-1}}(s', a') - Q_{\theta_i}(s, a)$$

DQN

- Deep Q-Network
- Input scaling to 84x84 pixels; score is clipped to {-1,0,+1}
- layer 1 and 2 convolve with ReLU for spatial generalization
- layer 3 and 4 fully connected for action selection
- 18 output units (joystick actions)
- frame skipping 1/4 to reduce computational burden
- adaptive epsilon-greedy Q-learning

DQN

- Achieve stable reinforcement learning despite correlations between states

- Replay Buffer [2013] (and thus off-policy learning)

- Infrequent Weight Updates [2015] for better convergence

- Works empirically, no proof, little theoretical insight…

Stable Learning

- De-correlation of examples
- Exploration
- Slow learning

DQN performance

- 2013 version of DQN achieves human level play for 6 games

- 2015 version of DQN achieves human level play for 49 games

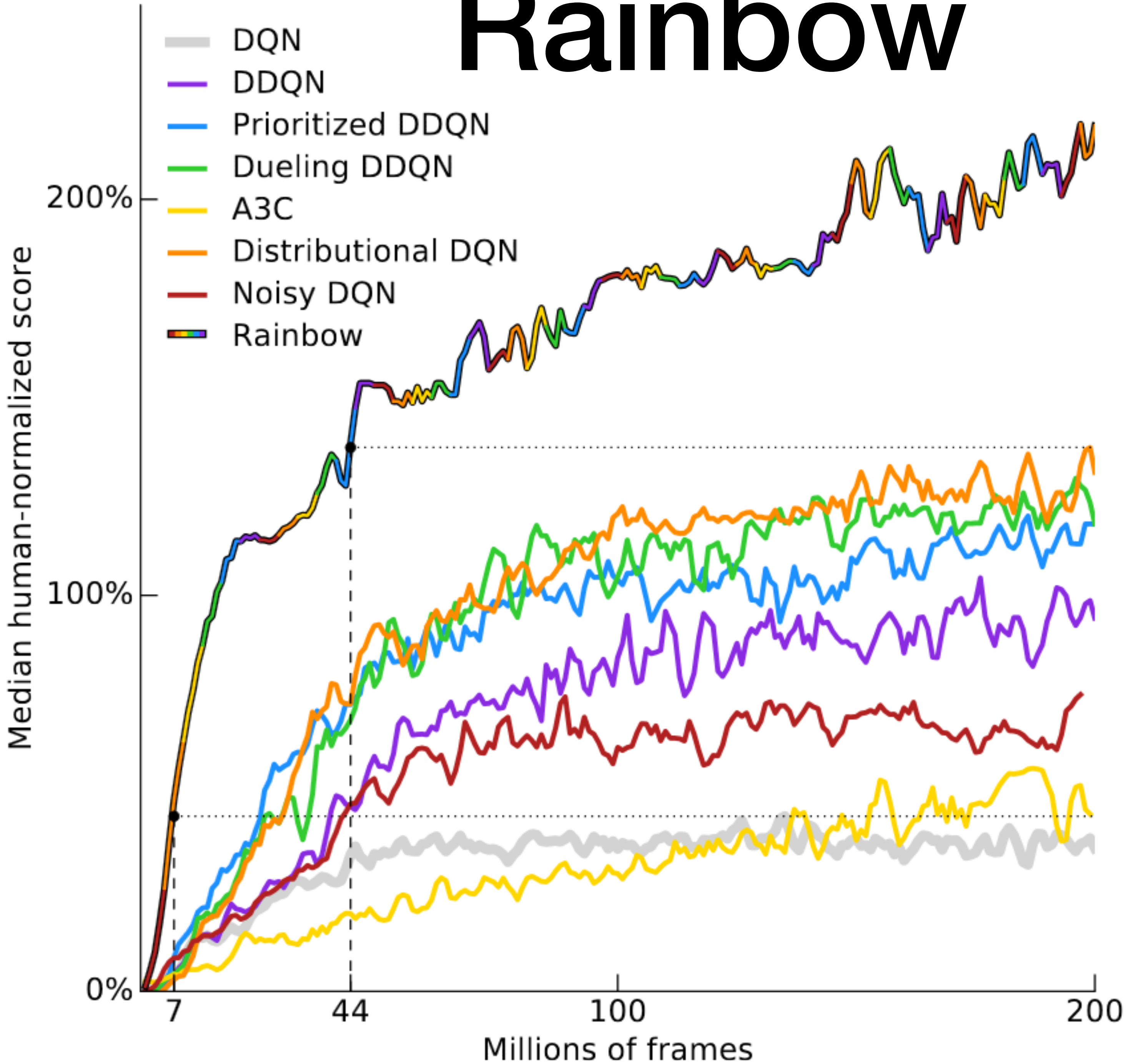
- Some games, such as Montezuma’s Revenge, have very long credit assignment distance, and performance is lacking

What has happened after DQN?

Rainbow

- DQN - baseline
- Double DQN - de-overestimate values
- Prioritized experience - sort replay buffer history
- A3C - parallel actor critic (Ch4)
- Distributional DQN - probability distribution
- Noisy DQN - parametric noise: exploration
- -> **ADDITIVE**

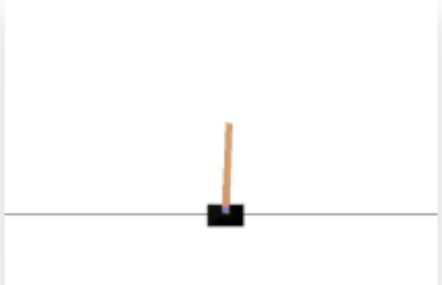
Rainbow



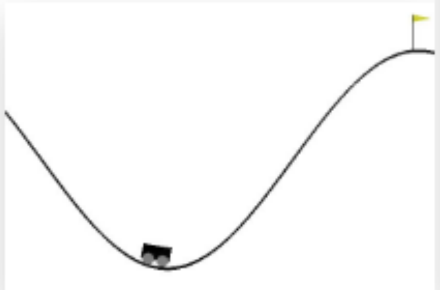
DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
200%
100%
0%
Median human-normalized score
7
44
100
200
Millions of frames

RL Environments

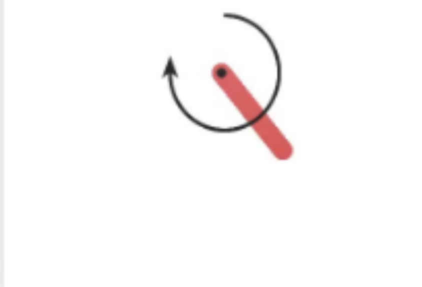
OpenAI Gym



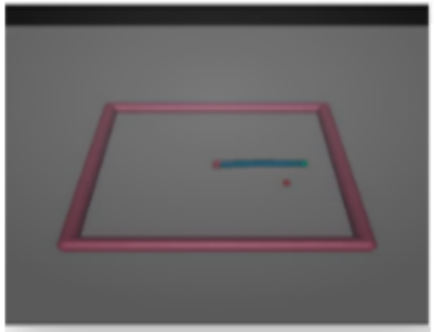
CartPole-v0
Balance a pole on a cart (for a short time).



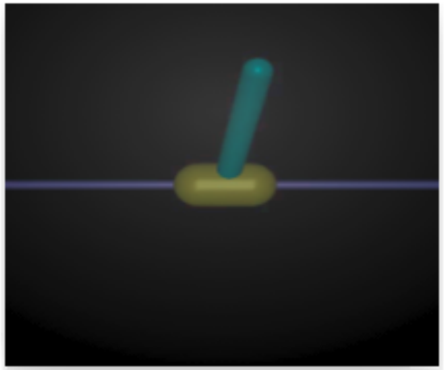
MountainCar-v0
Drive up a big hill.



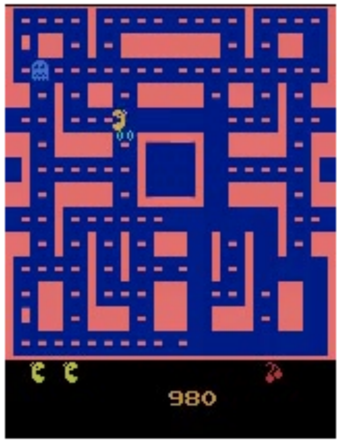
Pendulum-v0
Swing up a pendulum.



Reacher-v2
Make a 2D robot reach to a randomly located target.



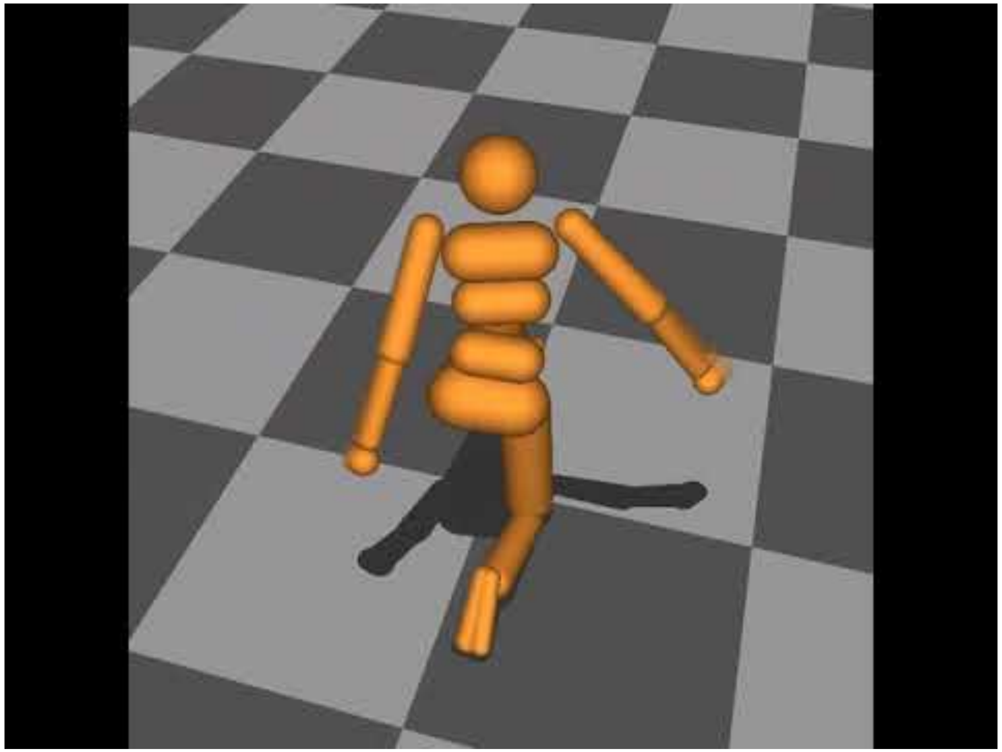
InvertedPendulum-v2
Balance a pole on a cart.



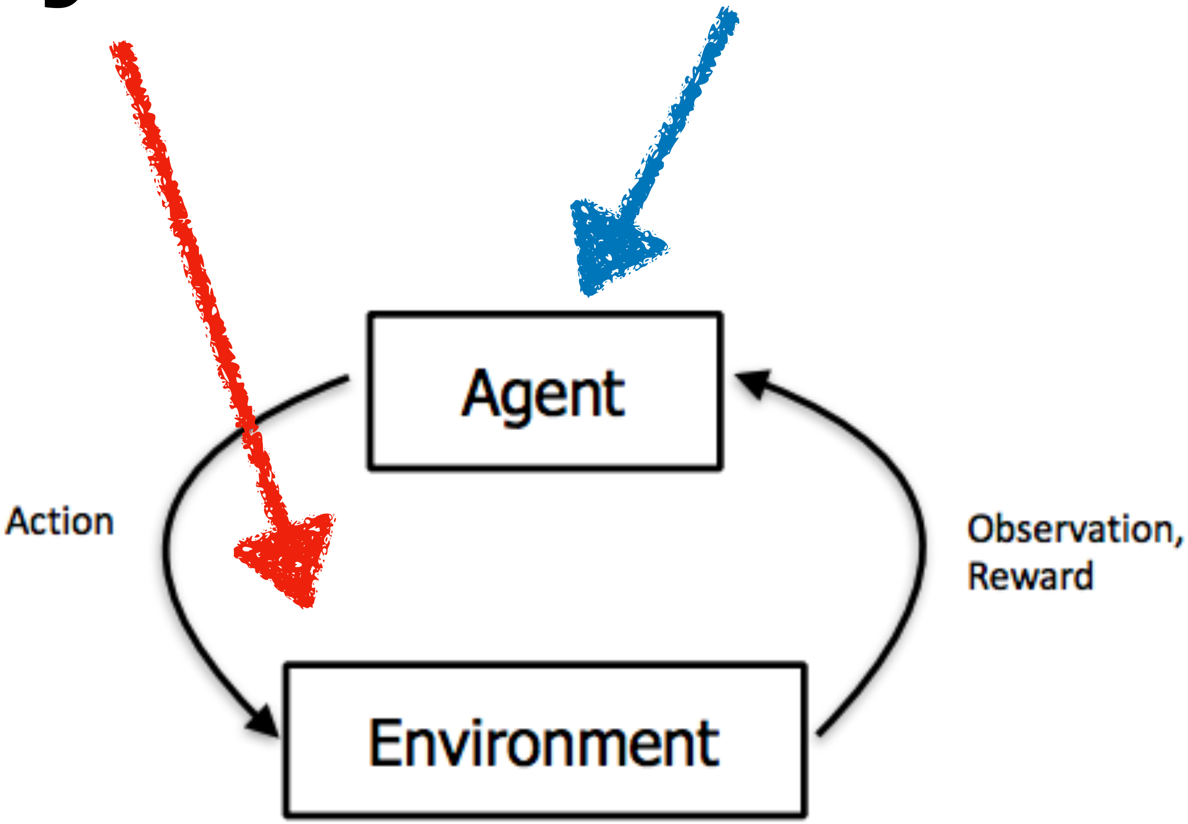
MsPacman-ram-v0

Arcade Learning Environment

Mujoco



Gym & Baselines



Stable Baselines

Stable Baselines

master

Search docs

USER GUIDE

- Installation
- Getting Started
- RL Algorithms
- Examples
- Vectorized Environments
- Using Custom Environments
- Custom Policy Network
- Tensorboard Integration

RL ALGORITHMS

- Base RL Class
- Policy Networks
- A2C

Docs » Welcome to Stable Baselines docs! - RL Baselines Made Easy

Edit on GitHub

Welcome to Stable Baselines docs! - RL Baselines Made Easy

Stable Baselines is a set of improved implementations of Reinforcement Learning (RL) algorithms based on OpenAI Baselines.

Github repository: https://github.com/hill-a/stable-baselines

You can read a detailed presentation of Stable Baselines in the Medium article: link

Main differences with OpenAI Baselines

This toolset is a fork of OpenAI Baselines, with a major structural refactoring, and code cleanups:

- Unified structure for all algorithms
- PEP8 compliant (unified code style)
- Documented functions and classes
- More tests & more code coverage

User Guide

- Installation
  - Prerequisites
  - Stable Release
  - Bleeding-edge version
  - Using Docker Images
- Getting Started

Zoo

RL Baselines Zoo

RL Baselines Zoo. is a collection of pre-trained Reinforcement Learning agents using Stable-Baselines. It also provides basic scripts for training, evaluating agents, tuning hyperparameters and recording videos.

Goals of this repository:

1. Provide a simple interface to train and enjoy RL agents
2. Benchmark the different Reinforcement Learning algorithms
3. Provide tuned hyperparameters for each environment and RL algorithm
4. Have fun with the trained agents!

Installation

1. Install dependencies

```
apt-get install swig cmake libopenmpi-dev zlib1g-dev ffmpeg
pip install stable-baselines box2d box2d-kengz pyyaml pybullet optuna pytablewriter
```

2. Clone the repository:

```
git clone https://github.com/araffin/rl-baselines-zoo
```

Conclusion

- Inspired by Supervised End-to-end Learning in ImageNet
- Highly visual & imaginative Atari results
- Results that were said that could not be done
- A flurry of further activities in RL research
- Impact of AI on society

Questions?