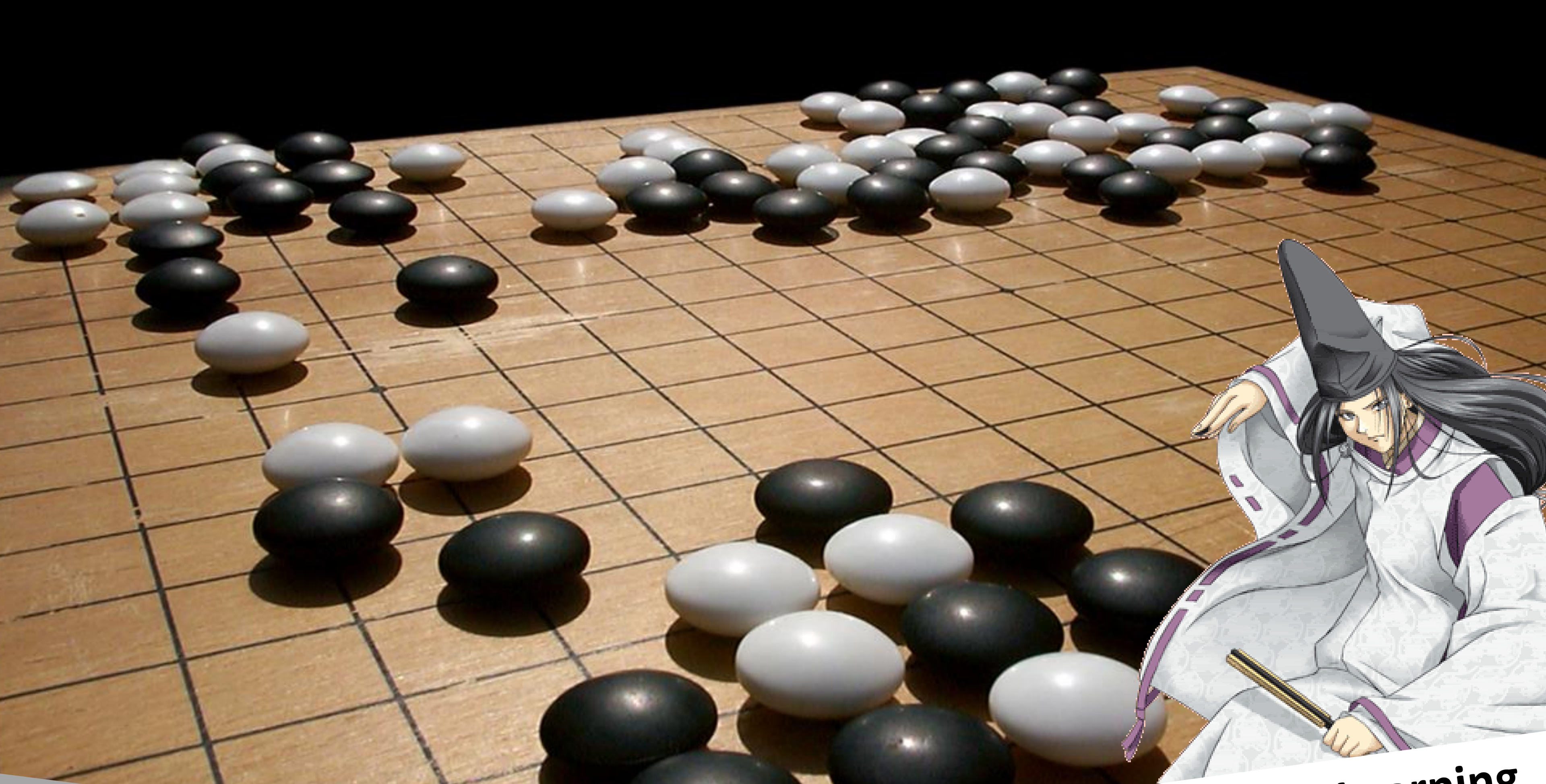


Deep Reinforcement Learning (2)

Dec 8th, 2016

Applied Deep Learning

YUN-NUNG (VIVIAN) CHEN

WWW.CSIE.NTU.EDU.TW/~YVCHEN/F105-ADL

National Taiwan University

Slide credit from David Silver

Review

Reinforcement Learning

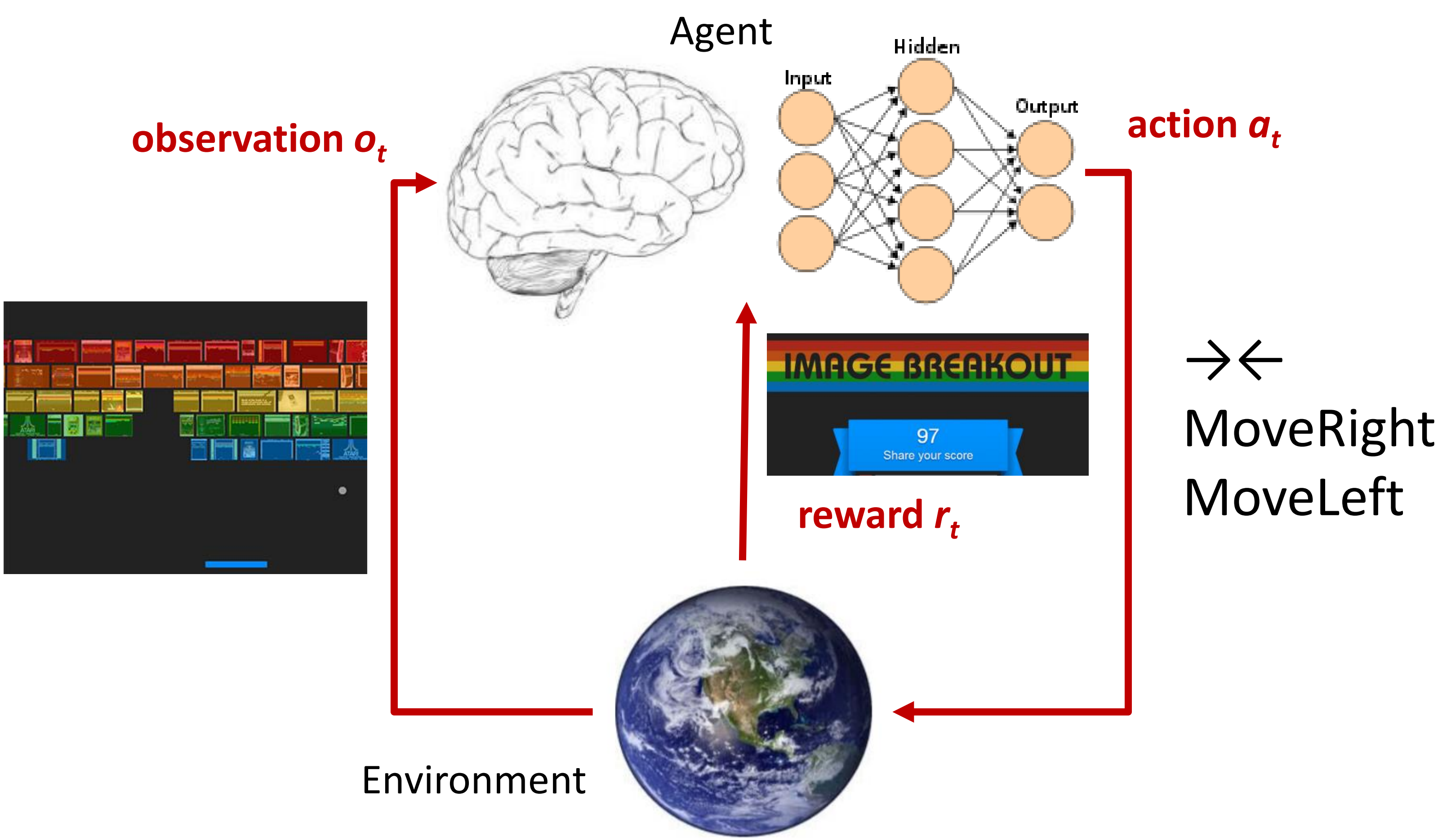
Reinforcement Learning

RL is a general purpose framework for **decision making**

- RL is for an *agent* with the capacity to *act*
- Each *action* influences the agent's future *state*
- Success is measured by a scalar *reward* signal

Big three: action, state, reward

Agent and Environment



Major Components in an RL Agent

An RL agent may include one or more of these components

- **Policy**: agent's behavior function
- **Value function**: how good is each state and/or action
- **Model**: agent's representation of the environment

Reinforcement Learning Approach

Policy-based RL

- Search directly for optimal policy π^*

π^* is the policy achieving maximum future reward

Value-based RL

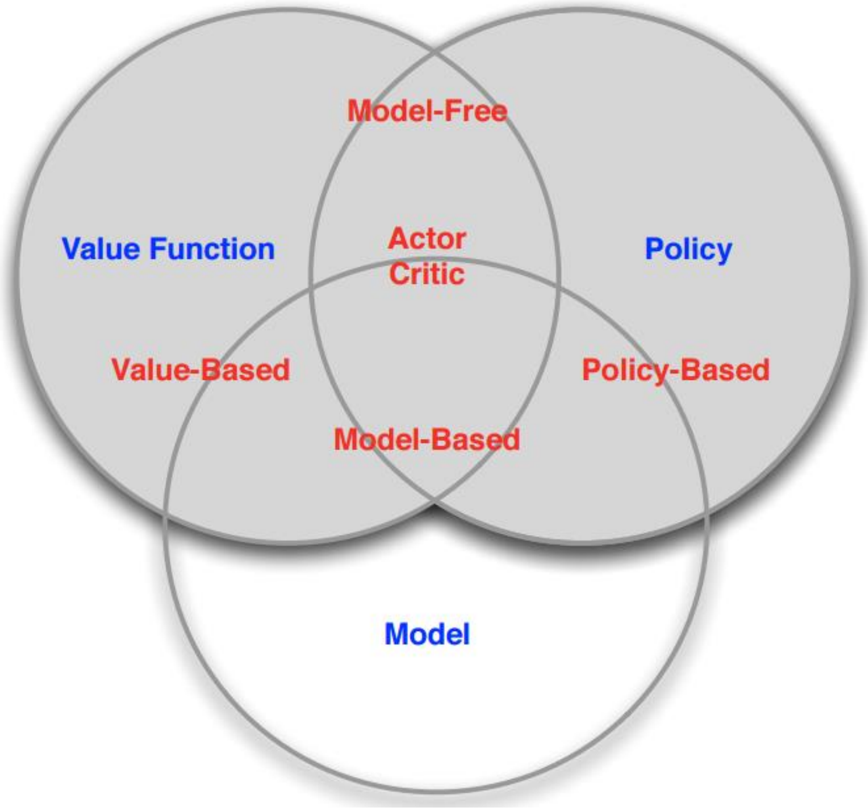
- Estimate the optimal value function $Q^*(s, a)$

$Q^*(s, a)$ is maximum value achievable under any policy

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

RL Agent Taxonomy



Deep Reinforcement Learning

Idea: deep learning for reinforcement learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by SGD

Value-Based Deep RL

Estimate How Good Each State and/or Action is

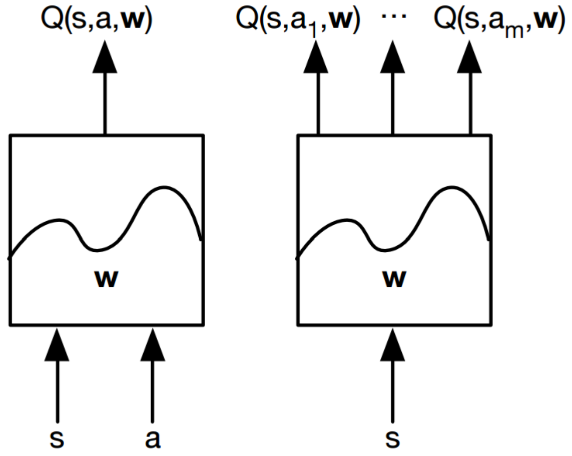
Value Function Approximation

Value functions are represented by a *lookup table*

$$Q(s, a) \;\; \forall s, a$$

- too many states and/or actions to store
- too slow to learn the value of each entry individually

Values can be estimated with *function approximation*

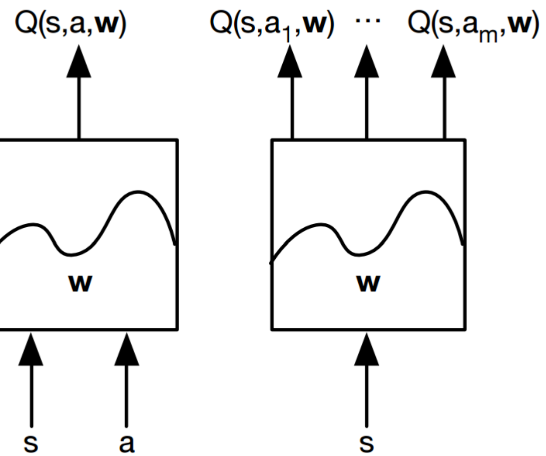


Q-Networks

Q-networks represent value functions with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

- generalize from seen states to unseen states
- update parameter w for function approximation



Q-Learning

Goal: estimate optimal Q-values

- Optimal Q-values obey a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\boxed{\left[r + \gamma \max_{a'} Q^*(s',a')\right.} \mid s,a]$$

learning target

- *Value iteration* algorithms solve the Bellman equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Deep Q-Networks (DQN)

Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

Objective is to minimize MSE loss by SGD

$$L(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

Leading to the following Q-learning gradient

$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

Issue: naïve Q-learning oscillates or diverges using NN due to:
1) correlations between samples 2) non-stationary targets

Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid (independent and identically distributed)
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be unstable when backpropagated

Stable Solutions for DQN

DQN provides a stable solutions to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
2. Freeze target Q-network
   - Avoid oscillation
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

Stable Solution 1: Experience Replay

To remove correlations, build a dataset from agent's experience

- Take action at according to ϵ-greedy policy

small prob for exploration

- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ s, a, r, s'

$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$

- Optimize MSE between Q-network and Q-learning targets

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

Stable Solution 2: Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters w^-

$$r + \gamma \max_{a'} Q(s', a', w^-)$$

- Optimize MSE between Q-network and Q-learning targets

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

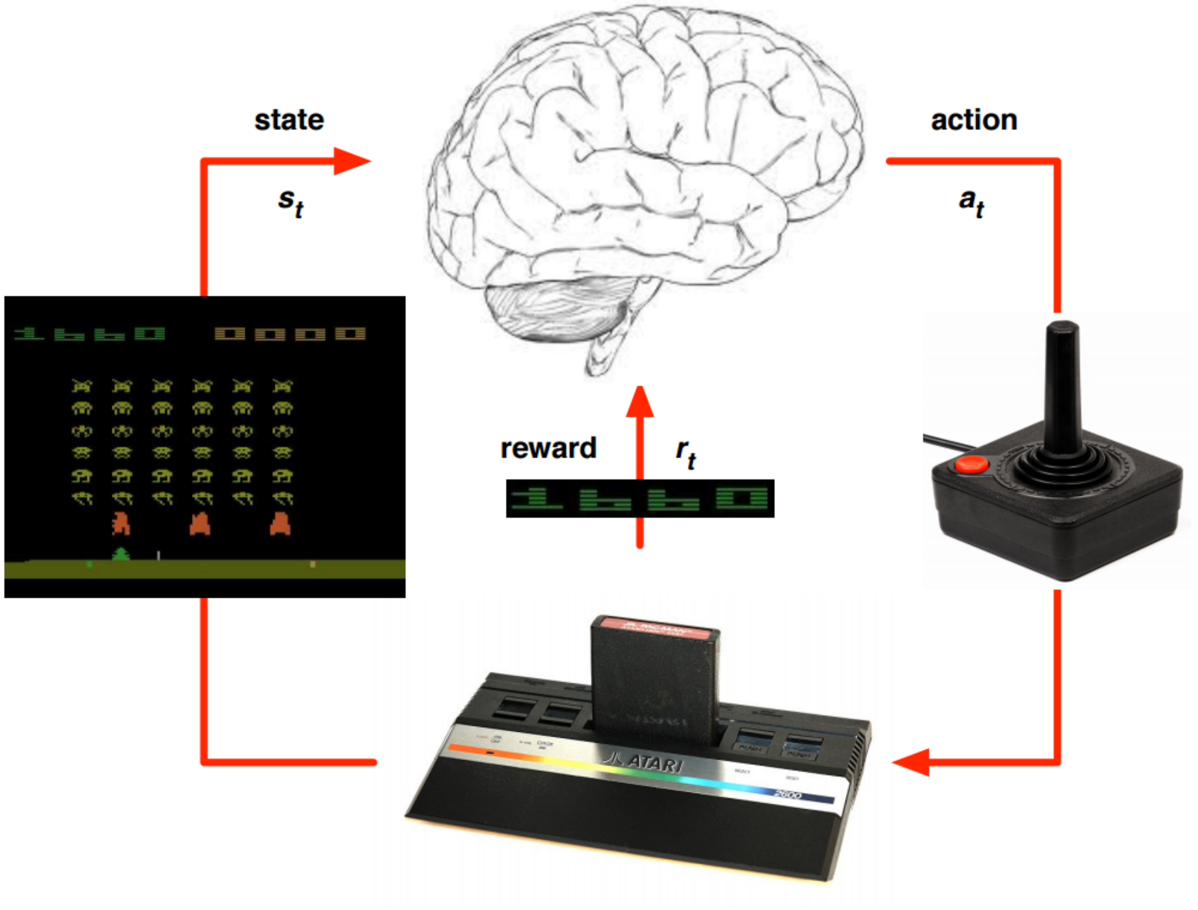
- Periodically update fixed parameters $w^- \leftarrow w$

Stable Solution 3: Reward / Value Range

To avoid oscillations, control the reward / value range

- DQN clips the rewards to $[-1, +1]$
  - Prevents too large Q-values
  - Ensures gradients are well-conditioned

Deep RL in Atari Games

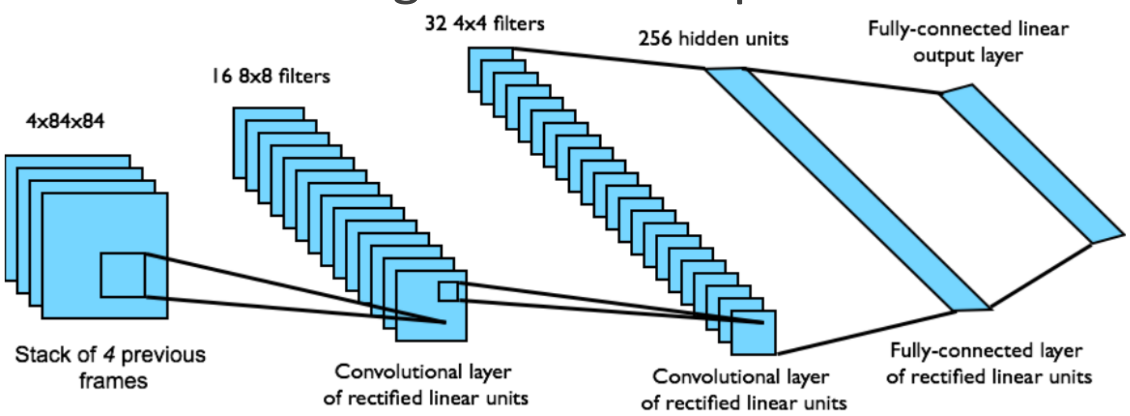


DQN in Atari

Goal: end-to-end learning of values $Q(s, a)$ from pixels

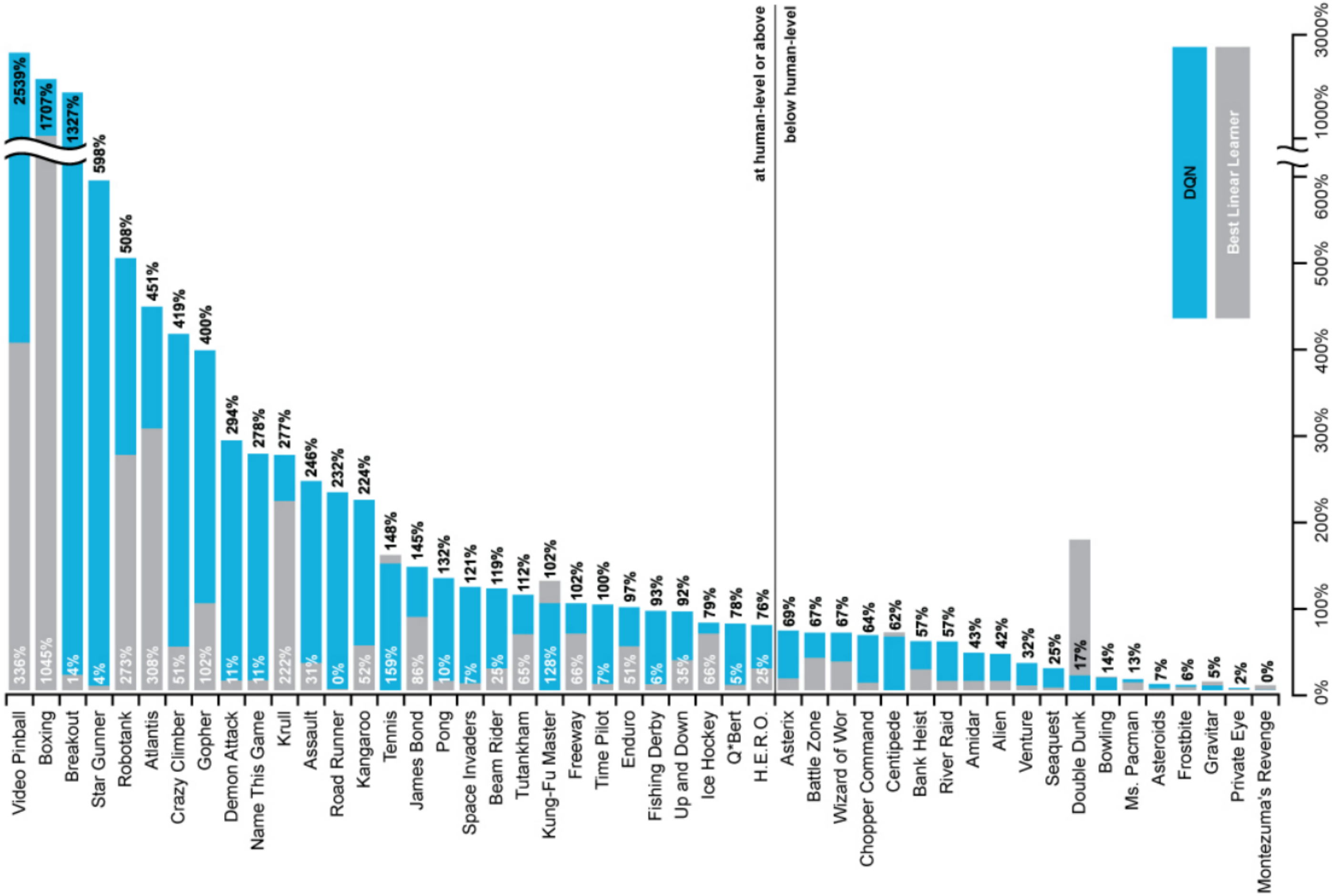
$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

- Input: state is stack of raw pixels from last 4 frames
- Output: $Q(s, a)$ for all joystick/button positions a
- Reward is the score change for that step



DQN in Atari

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO



at human-level or above
below human-level
DQN
Best Linear Learner
0%
100%
200%
300%
400%
500%
600%
1000%
3000%
Video Pinball 2539% 336%
Boxing 1707% 1045%
Breakout 1327% 14%
Star Gunner 598% 4%
Robotank 508% 273%
Atlantis 451% 308%
Crazy Climber 419% 51%
Gopher 400% 102%
Demon Attack 294% 11%
Name This Game 278% 11%
Krull 277% 222%
Assault 246% 31%
Road Runner 232% 0%
Kangaroo 224% 52%
Tennis 148% 159%
James Bond 145% 86%
Pong 132% 10%
Space Invaders 121% 7%
Beam Rider 119% 25%
Tutankham 112% 65%
Kung-Fu Master 102% 128%
Freeway 102% 66%
Time Pilot 100% 7%
Enduro 97% 51%
Fishing Derby 93% 6%
Up and Down 92% 35%
Ice Hockey 79% 66%
Q*Bert 78% 5%
H.E.R.O. 76% 25%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Other Improvements: Double DQN

Nature DQN

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

Double DQN: remove upward bias caused by $\max_{a} Q(s, a, w)$

- Current Q-network w is used to select actions
- Older Q-network w^- is used to evaluate actions

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a', w), w^-) - Q(s, a, w)\right)^2\right]$$

Other Improvements: Prioritized Replay

Prioritized Replay: weight experience based on surprise

- Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right|$$

Other Improvements: Dueling Network

Dueling Network: split Q-network into two channels

$$Q(s,a) = V(s,v) + A(s,a,w)$$

- Action-independent value function $V(s,v)$
  - Value function estimates how good the state is
- Action-dependent advantage function $A(s,a,w)$
  - Advantage function estimates the additional benefit

Policy-Based Deep RL

Estimate How Good An Agent’s Behavior is

Deep Policy Networks

Represent policy by deep network with weights u

$$a = \pi(a \mid s, u) \qquad a = \pi(s, u)$$

stochastic policy — deterministic policy

Objective is to maximize total discounted reward by SGD

$$L(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(\cdot, u)\right]$$

Policy Gradient

The gradient of a stochastic policy $\pi(a \mid s, u)$ is given by

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial \log \pi(a \mid s, u)}{\partial u} Q^\pi(s, a) \right]$$

The gradient of a deterministic policy $\pi(s, u)$ is given by

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial a}{\partial u} \right] \qquad a = \pi(s, u)$$

How to deal with continuous actions

Actor-Critic (Value-Based + Policy-Based)

Estimate value function $Q(s,a,w) \approx Q^{\pi}(s,a)$

Update policy parameters u by SGD

- Stochastic policy

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial \log \pi(a \mid s,u)}{\partial u} Q(s,a,w) \right]$$

- Deterministic policy

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q(s,a,w)}{\partial a} \frac{\partial a}{\partial u} \right]$$

Deterministic Deep Actor-Critic

Deep deterministic policy gradient (DDPG) is the continuous analogue of DQN

- Experience replay: build dataset from agent's experience
- Critic estimates value of current policy by DQN

$$L(w) = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s, a, w)\right)^2\right]$$

$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Actor updates policy in direction that improves Q

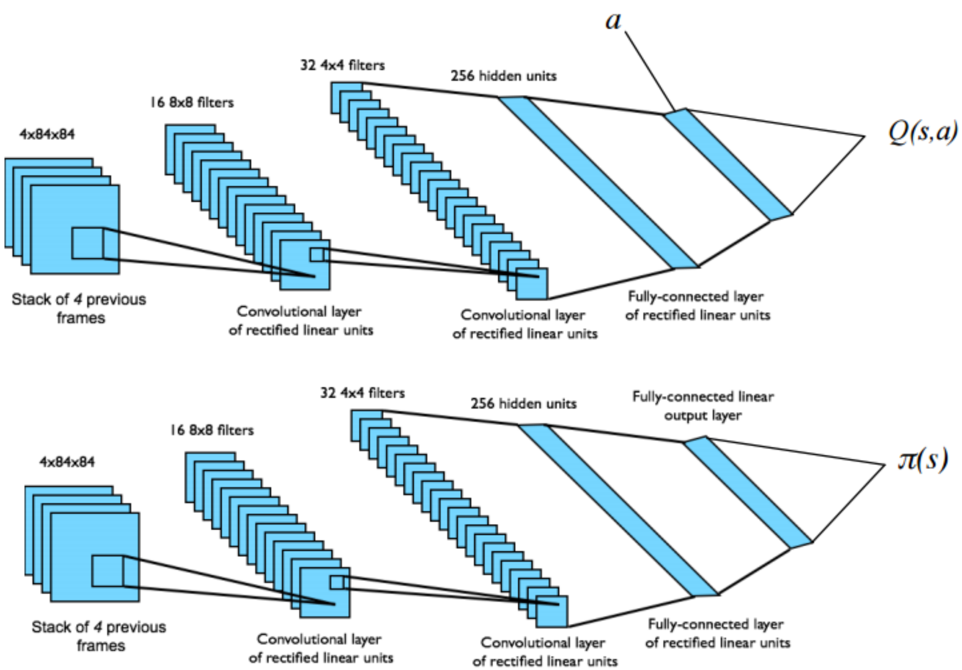
$$\frac{\partial L(u)}{\partial u} = \mathbb{E}\left[\frac{\partial Q(s, a, w)}{\partial a} \frac{\partial a}{\partial u}\right]$$

Critic provides loss function for actor

DDPG in Simulated Physics

Goal: end-to-end learning of control policy from pixels

- Input: state is stack of raw pixels from last 4 frames
- Output: two separate CNNs for Q and π



Model-Based Deep RL

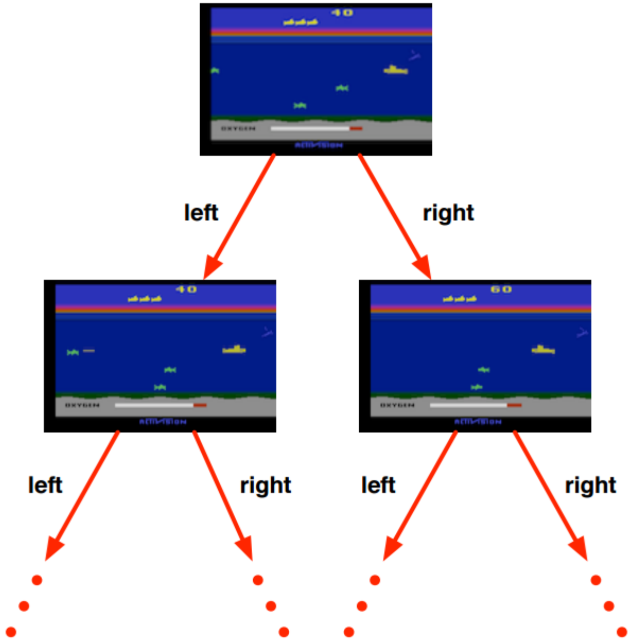
Agent’s Representation of the Environment

Model-Based Deep RL

Goal: learn a transition model of the environment and plan based on the transition model

$$p(r, s' \mid s, a)$$

Objective is to maximize the measured goodness of model



Model-based deep RL is challenging, and so far has failed in Atari

Issues for Model-Based Deep RL

Compounding errors

- Errors in the transition model compound over the trajectory
- A long trajectory may result in totally wrong rewards

Deep networks of value/policy can “plan” implicitly

- Each layer of network performs arbitrary computational step
- n-layer network can “lookahead” n steps

Model-Based Deep RL in Go

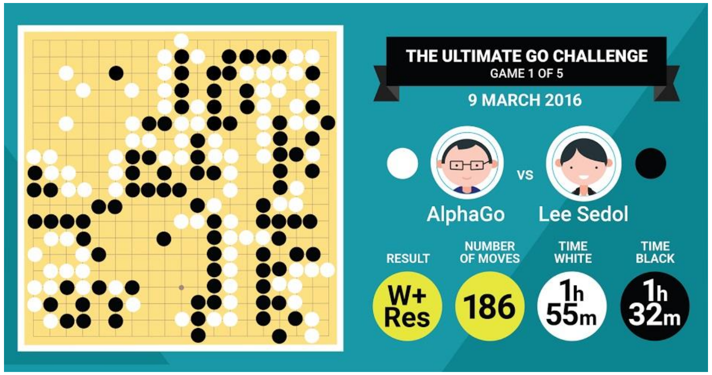
Monte-Carlo tree search (MCTS)

- MCTS simulates future trajectories
- Builds large lookahead search tree with millions of positions
- State-of-the-art Go programs use MCTS

Convolutional Networks

- 12-layer CNN trained to predict expert moves
  - Raw CNN (looking at 1 position, no search at all) equals performance of MoGo with 105 position search tree

  1st strong Go program



Silver, et al., "Mastering the game of Go with deep neural networks and tree search," Nature, 2016.

https://deepmind.com/alphago/

OpenAI Universe

Software platform for measuring and training an AI's general intelligence via the OpenAI gym environment

Concluding Remarks

RL is a general purpose framework for **decision making** under interactions between agent and environment

An RL agent may include one or more of these components

- Policy: agent's behavior function
- Value function: how good is each state and/or action
- Model: agent's representation of the environment

RL problems can be solved by end-to-end deep learning

Reinforcement Learning + Deep Learning = AI

References

Course materials by David Silver: http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html

ICLR 2015 Tutorial: http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

ICML 2016 Tutorial: http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf