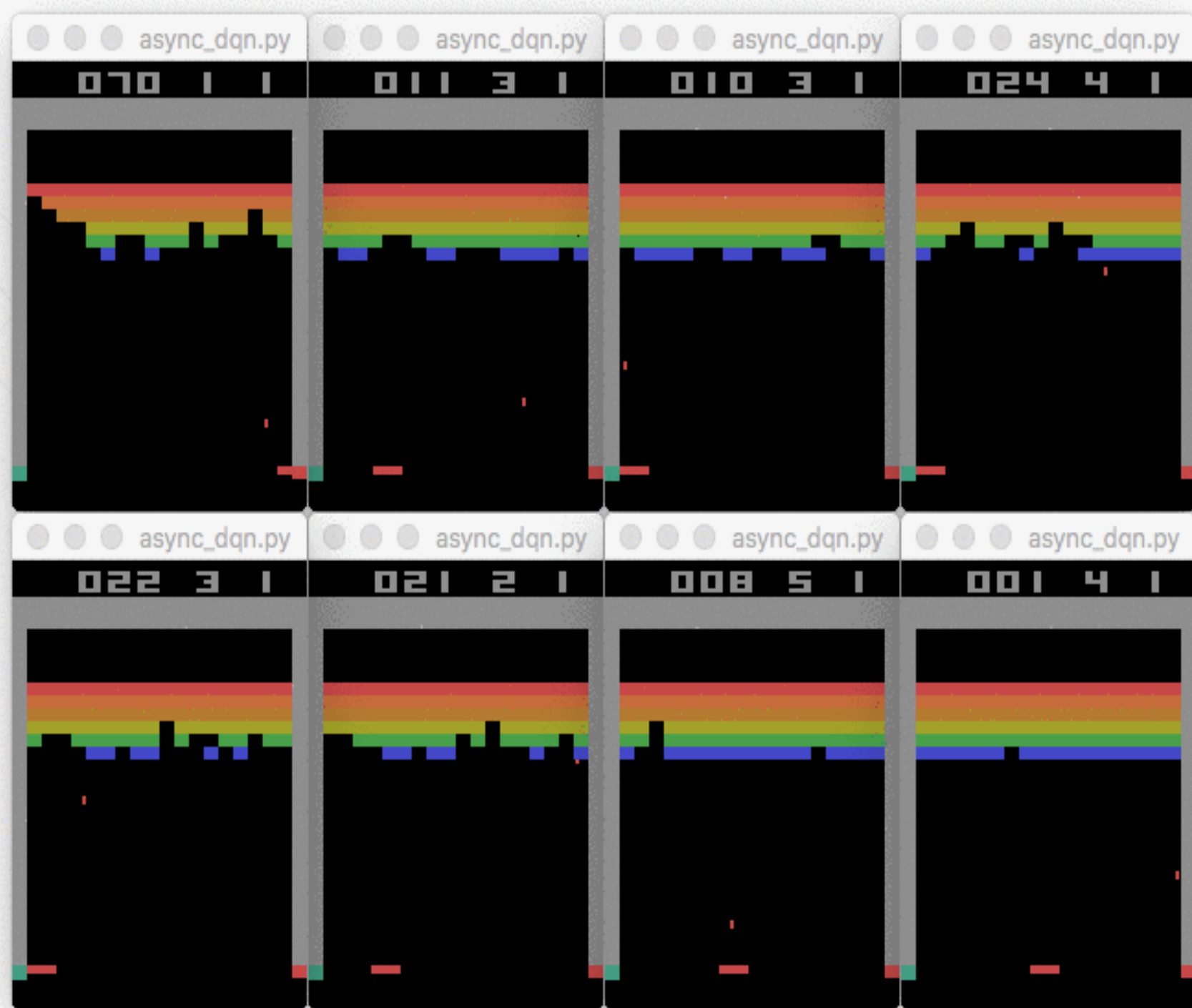# Lecture 13: Deep Reinforcement Learning

Deep Learning @ UvA

Reinforcement Learning

# What is Reinforcement Learning?

- General purpose framework for learning Artificial Intelligence models

- RL assumes that *the agent* (our model) can take *actions*

- These actions affect *the environment* where *the agent* operates, more specifically *the state* of the environment and *the state* of the agent

- Given the state of the environment and the agent, an action taken from the agent causes a reward (can be positive or negative)

- Goal: the goal of an RL agent is to learn how to take actions that maximize future rewards

Some examples of RL

Some examples of RL

- Controlling physical systems
  - Robot walking, jumping, driving

- Logistics
  - Scheduling, bandwidth allocation

- Games
  - Atari, Go, Chess, Pacman

- Learning sequential algorithms
  - Attention, memory

Reinforcement Learning: An abstraction
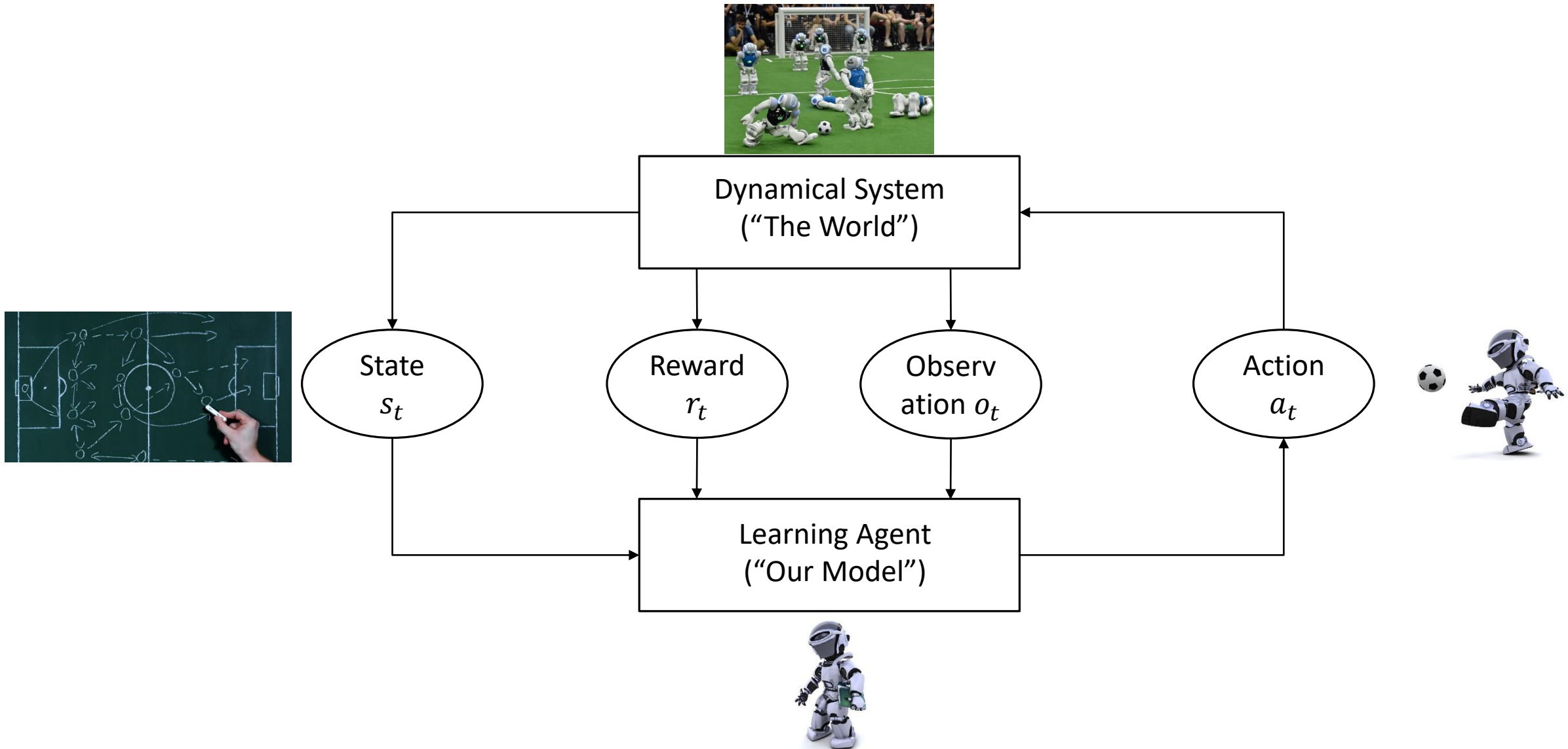
State

o Experience is a series of observations, actions and rewards
$$o_1, r_1, a_1, o_2, r_2, a_2, \ldots, o_t, r_t$$

o The state is the summary of experience so far
$$s_t = f(o_1, r_1, a_1, o_2, r_2, a_2, \ldots, o_t, r_t)$$

o If we have fully observable environments, then
$$s_t = f(o_t)$$

Policy

- Policy is the agent's behavior function

- The policy function maps the state input s_t to an action output a_t

- Deterministic policy: $a_t = f(s_t)$

- Stochastic policy: $\pi(a_t|s_t) = \mathbb{P}(a_t|s_t)$

Value function

o A value function is the prediction of the future reward
   ◦ Given the state s_t what will my reward be if I do action a_t

o The Q-value function gives the expected future reward

o Given state s_t, action a_t, a policy π the Q-value function is $Q^\pi(s_t, a_t)$

How do we decide about actions, states, rewards?

o We model the policy and the value function as machine learning functions that can be optimized by the data

o The **policy function** $a_t = \pi(s_t)$ selects an action given the current state

o The **value function** $Q^\pi(s_t, a_t)$ is the expected total reward that we will receive if we take action a_t given state s_t

o What should our goal then be?

Goal: Maximize future rewards!

○ Learn the policy and value functions such that the action taken at the t-th time step a_t maximizes the expected sum of future rewards

$$Q^\pi(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t, a_t)$$

○ γ is a discount factor. Why do we need it?

Goal: Maximize future rewards!

- Learn the policy and value functions such that the action taken at the t-th time step a_t maximizes the expected sum of future rewards

$$Q^\pi(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t, a_t)$$

- γ is a discount factor. Why do we need it?
  - The further into the future we look $t + 1, \dots, t + 1$, the less certain we can be about our expected rewards $r_{t+1}, \dots, r_{t+T}$

Bellman equation

○ How can we rewrite the value function in more compact form

$$Q^\pi(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t, a_t) = ?$$

Bellman equation

o How can we rewrite the value function in more compact form
$$Q^\pi(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t, a_t)$$
$$= \mathbb{E}_{s',a'}(r + \gamma Q^\pi(s', a') | s_t, a_t)$$

o This is the *Bellman equation*

o How can we rewrite the optimal value function $Q^*(s_t, a_t)$?

Bellman equation

o How can we rewrite the value function in more compact form

$$Q^\pi(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t, a_t)$$
$$= \mathbb{E}_{s'}(r + \gamma Q^\pi(s', a') | s_t, a_t)$$

o This is the *Bellman equation*

Optimal value function

o Optimal value function $Q^*(s, a)$ is attained with the optimal policy π^*

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = Q^{\pi^*}(s, a)$$

o After we have found the optimal policy π^* we do the optimal action

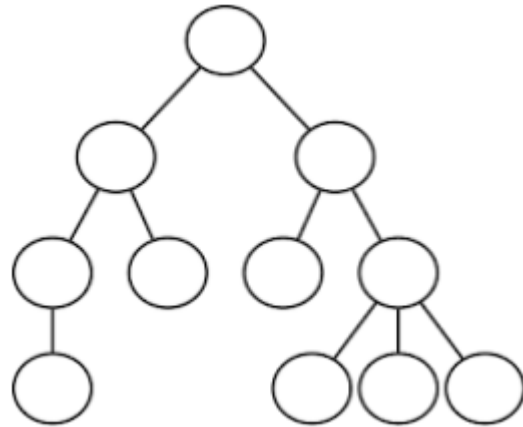$$\pi^* = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

o By expanding the optimal value function

$$Q^*(s, a) = r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

$$Q^*(s, a) = \mathbb{E}_{s'} \left(r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right)$$

Model

o The model is learnt from experience

o The model acts as a replacement for the environment

o When planning, the agent can interact with the model

o For instance look ahead search to estimate the future states given actions
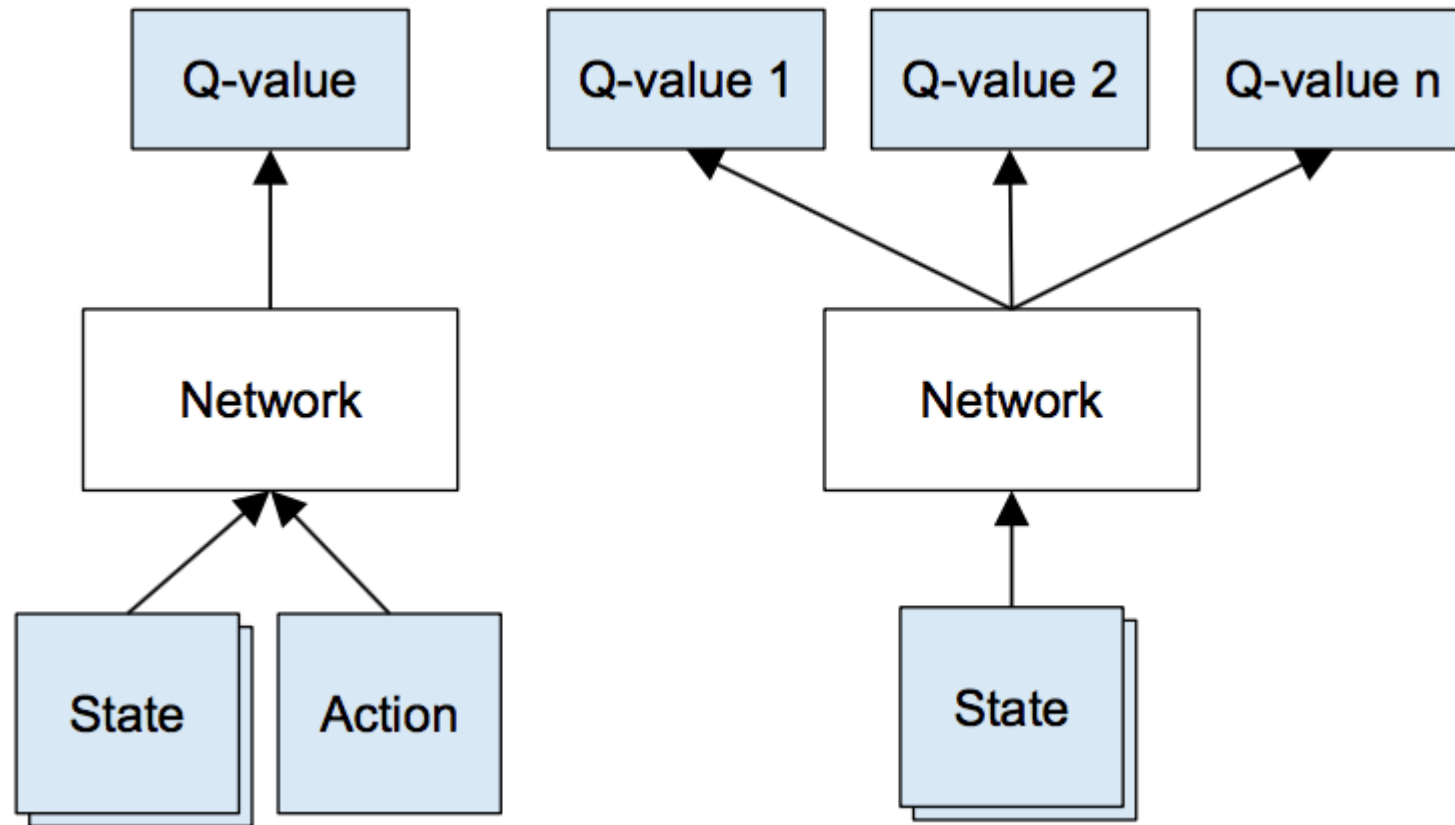
Approaches to Reinforcement Learning

- Policy-based
  - Learn directly the optimal policy π^*
  - The policy π^* obtains the maximum future reward

- Value-based
  - Learn the optimal value function $Q^*(s, a)$
  - This value function applies for any policy

- Model-based
  - Build a model for the environment
  - Plan and decide using that model

How to make RL deep?

How to make RL deep?

o Use Deep Networks for the
- Value function
- Policy
- Model

o Optimize final loss with SGD

How to make RL deep?

Deep Reinforcement Learning

o Non-linear function approximator: Deep Networks

o Input is as raw as possible, e.g. image frame
  ◦ Or perhaps several frames (When needed?)

o Output is the best possible action out of a set of actions for maximizing future reward

o **Important:** no need anymore to compute the actual value of the action-value function and take the maximum: $\arg\max_{\alpha} Q_{\theta}(s, a)$
  ◦ The network returns directly the optimal action

Value-based Deep RL
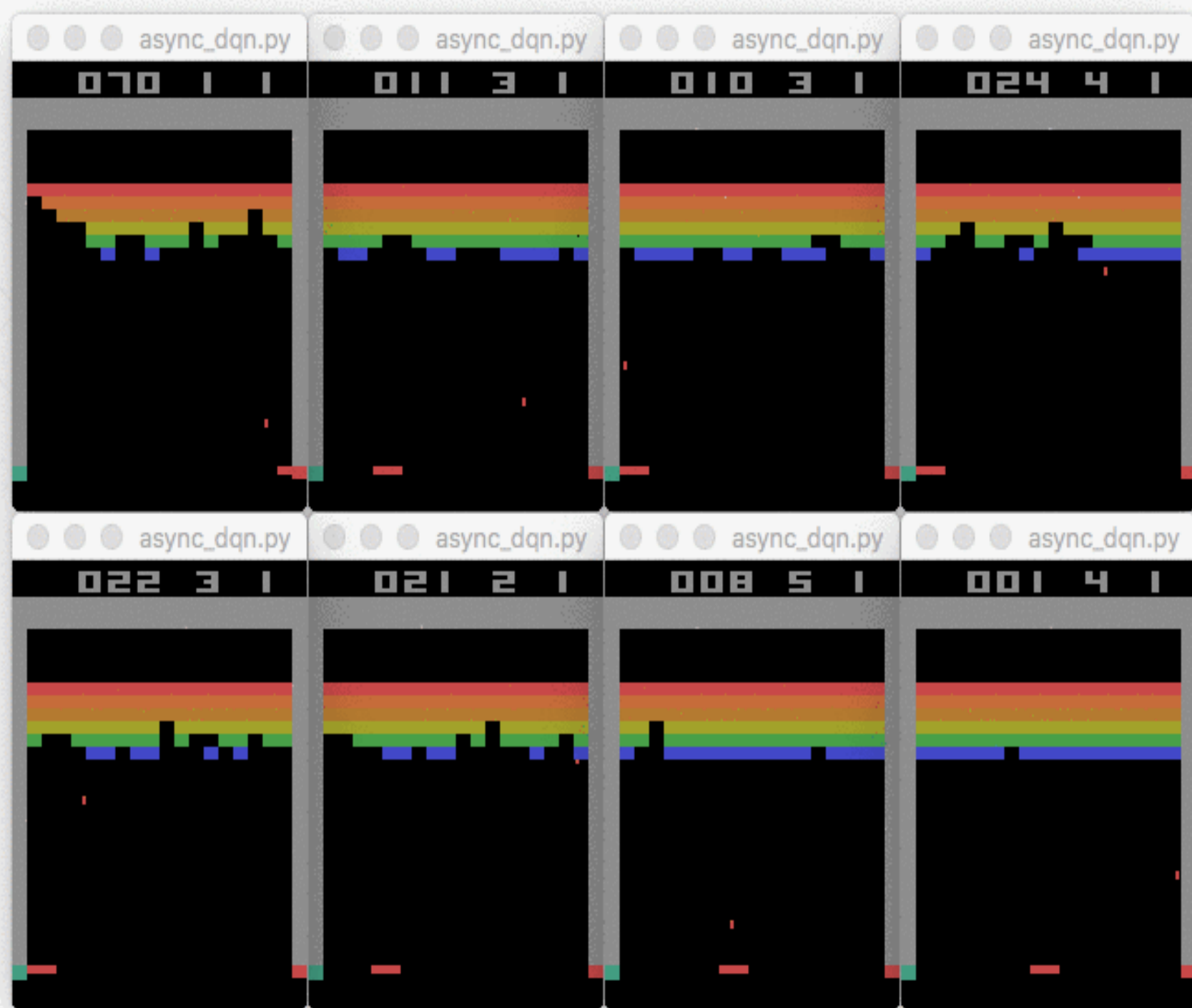
Q-Learning

o Optimize for Q value function
$$Q^\pi(s_t, a_t) = \mathbb{E}_{s'}(r + \gamma Q^\pi(s', a')|s_t, a_t)$$

o In the beginning of learning the function $Q(s, a)$ is incorrect

o We set $r + \gamma \max_{a'} Q_t(s', a')$ to be the learning target

o Then we minimize the loss
$$\min \left(r + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a) \right)^2$$

Q-Learning

○ Value iteration algorithms solve the Bellman equation

$$Q_{t+1}(s, a) = \mathbb{E}_{s'}\left(r + \gamma \max_{a'} Q_t(s', a') \middle| s, a\right)$$

○ In the simplest case Q_t is a table
  ◦ To the limit iterative algorithms converge to Q^*

○ However, a table representation for Q_t is not always enough

How to optimize?

o The objective is the mean squared-error in Q-values

$$\mathcal{L}(\theta) = \mathbb{E}[(\underbrace{r + \gamma \max_{a'} Q(s', a', \theta)}_{\text{target}} - Q(s, a, \theta))^2]$$

o The Q-Learning gradient then becomes

$$\frac{\partial \mathcal{L}}{\partial \theta} = \mathbb{E}[(\underbrace{r + \gamma \max_{a'} Q(s', a', \theta)}_{\text{Scalar target value} \rightarrow \text{Gradient 0}} - Q(s, a, \theta)) \frac{\partial Q(s, a, \theta)}{\partial \theta}]$$

o Optimize end-to-end with SGD

In practice

1. Do a feedforward pass for the current state *s* to get predicted Q-values for all actions

2. Do a feedforward pass for the next state *s'* and calculate maximum overall network outputs $\max_{a'} Q(s', a', \theta)$

3. Set Q-value target to $r + \gamma \max_{a'} Q(s', a', \theta)$
   ◦ use the max calculated in step 2
   ◦ For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs

4. Update the weights using backpropagation.

Deep Q Networks on Atari

o End-to-end learning from raw pixels

o Input: last 4 frames

o Output: 18 joystick positions

o Reward: change of score
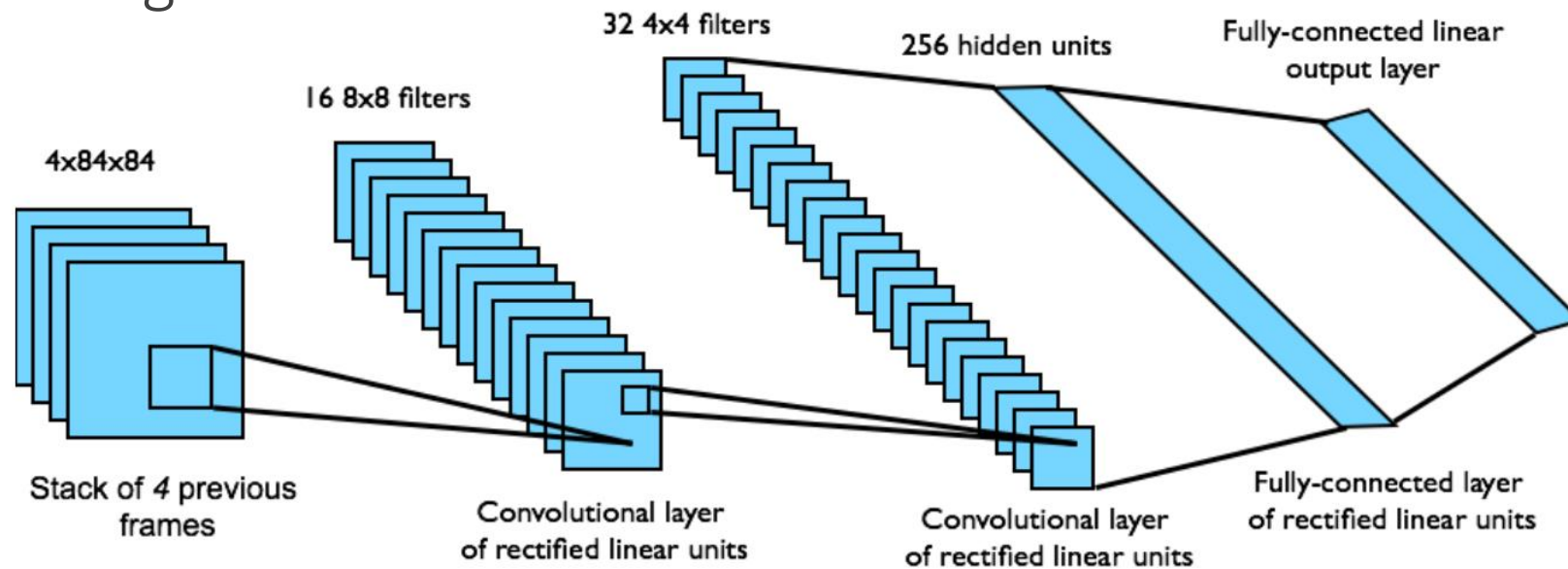
Deep Q Networks on Atari
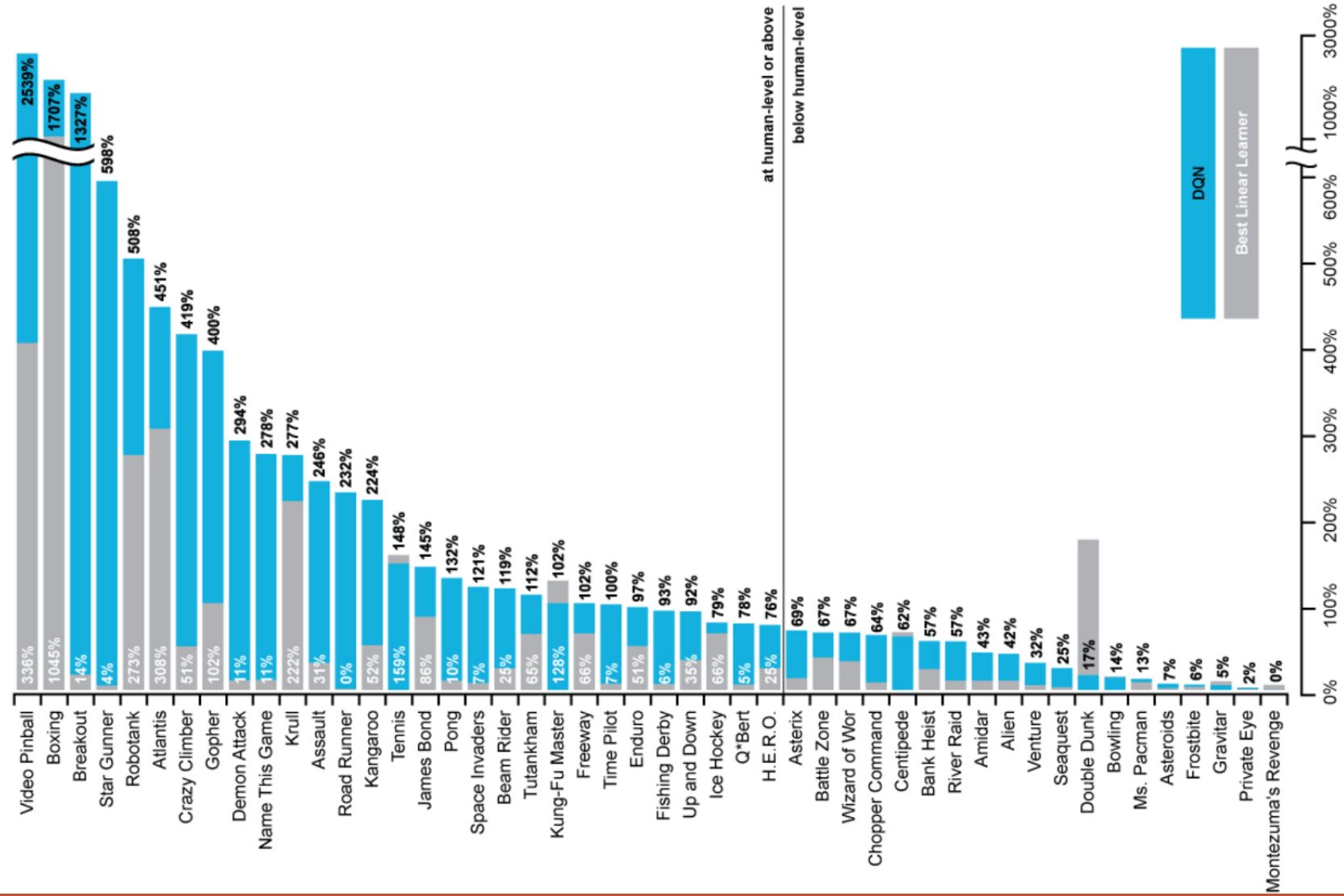
Stability in Deep Reinforcement Learning

Stability problems

○ Naively, Q-Learning oscillates or diverges with neural networks

○ Why?

Stability problems

o Naively, Q-Learning oscillates or diverges with neural networks

o Why?

o Sequential data breaks IID assumption
  ◦ Highly correlated samples break SGD

o However, this is not specific to RL, as we have seen earlier

Stability problems

o Naively, Q-Learning oscillates or diverges with neural networks

o Why?

Stability problems

o The learning objective is

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

o The target depends on the Q function also. This means that if we update the current Q function with backprop, the target will also change

o Plus, we know neural networks are highly non-convex

o Policy changes will change fast even with slight changes in the Q function
  ◦ Policy might oscillate
  ◦ Distribution of data might move from one extreme to another

Stability problems

- Naively, Q-Learning oscillates or diverges with neural networks

- Why?

Stability problems

o Not easy to control the scale of the Q values$\rightarrow$ gradients are unstable Q

o Remember, the Q function is the output of a neural network

o There is no guarantee that the outputs will lie in a certain range
  ◦ Unless care is taken

o Naïve Q gradients can be too large, or too small $\rightarrow$ generally unstable and unreliable

o Where else did we observe a similar behavior?

Improving stability: Experience replay

o Replay memory/Experience replay

o Store memories $< s, a, r, s' >$

o Train using random stored memories instead of the latest memory transition

o Breaks the temporal dependencies – SGD works well if samples are roughly independent

o Learn from all past policies

Experience replay

o Take action a_t according to ε-greedy policy

o Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D

o Sample random mini-batch of transitions (s, a, r, s') from D

o Optimize mean squared error using the mini-batch

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

o Effectively, update your network using random past inputs (experience), not the ones the agent currently sees

Improving stability: Freeze target Q network

- Instead of having "moving" targets, have two networks
  - One Q-Learning and one Q-Target networks

- Copy the Q network parameters to the target network every K iterations
  - Otherwise, keep the old parameters between iterations
  - The targets come from another (Q-Target) network with slightly older parameters

- Optimize the mean squared error as before, only now the targets are defined by the "older" Q function

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

- Avoids oscillations

Improving stability: Take care of rewards

o Clip rewards to be in the range $[-1, +1]$

o Or normalize them to lie in a certain, stable range

o Can't tell the difference between large and small rewards

Results

	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

Some extra tricks

o Skipping frames
  - Saves time and computation
  - Anyways, from one frame to the other there is often very little difference

o ε-greedy behavioral policy with annealed temperature during training
  - Select random action (instead of optimal) with probability ε
  - In the beginning of training our model is bad, no reason to trust the "optimal" action

o Alternatively: Exploration vs exploitation
  - early stages $\rightarrow$ strong exploration
 - late stages $\rightarrow$ strong exploitation

Policy-based Deep RL
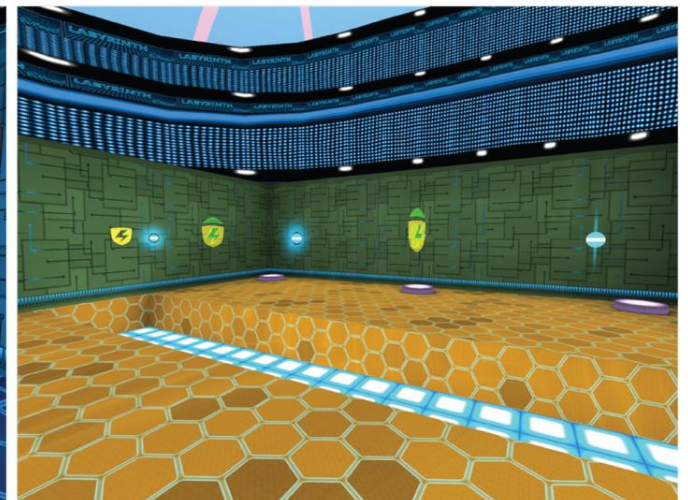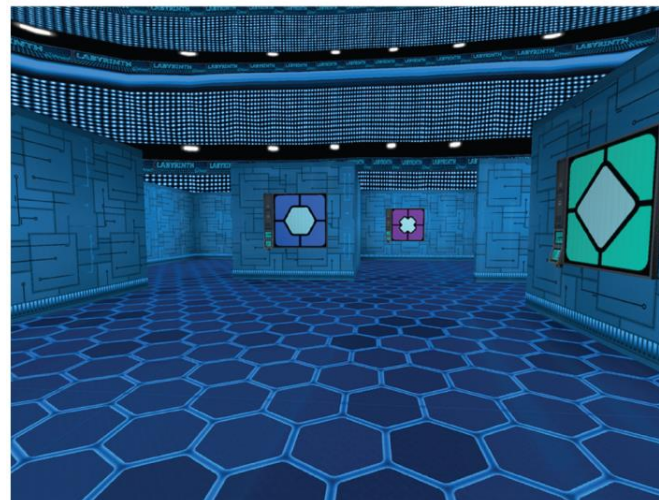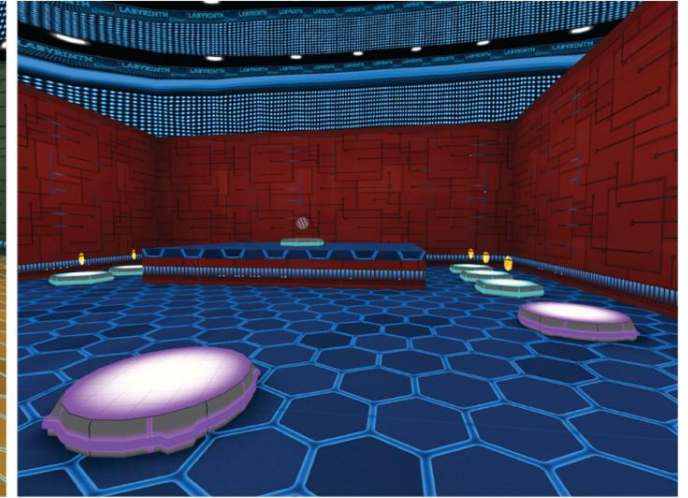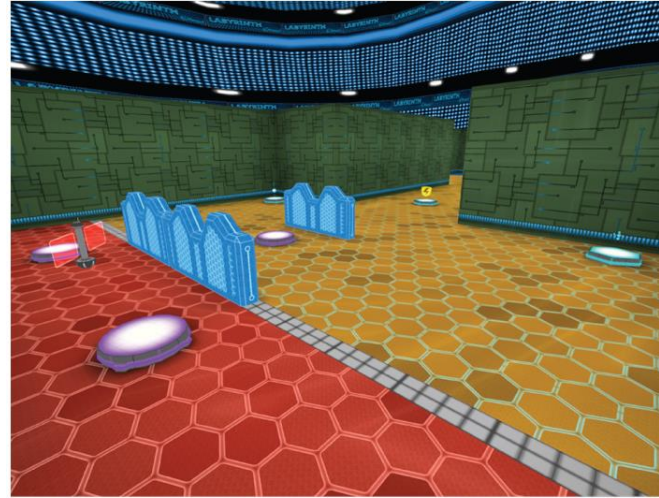
Policy Optimization

o Problems with modelling the Q-value function
  - Often too expensive → must take into account all possible states, actions →Imagine when having continuous or high-dimensional action spaces
  - Not always good convergence ← Oscillations

o Often learning directly a policy $\pi_\theta(a|s)$ that gives the best action without knowing what its expected future reward is easier

o Also, allows for stochastic policies ← no exploration/exploitation dilemma

o Model optimal *action value* with a non-linear function approximator
$$Q^*(s,a) \approx Q(s,a;w)$$

Policy Optimization
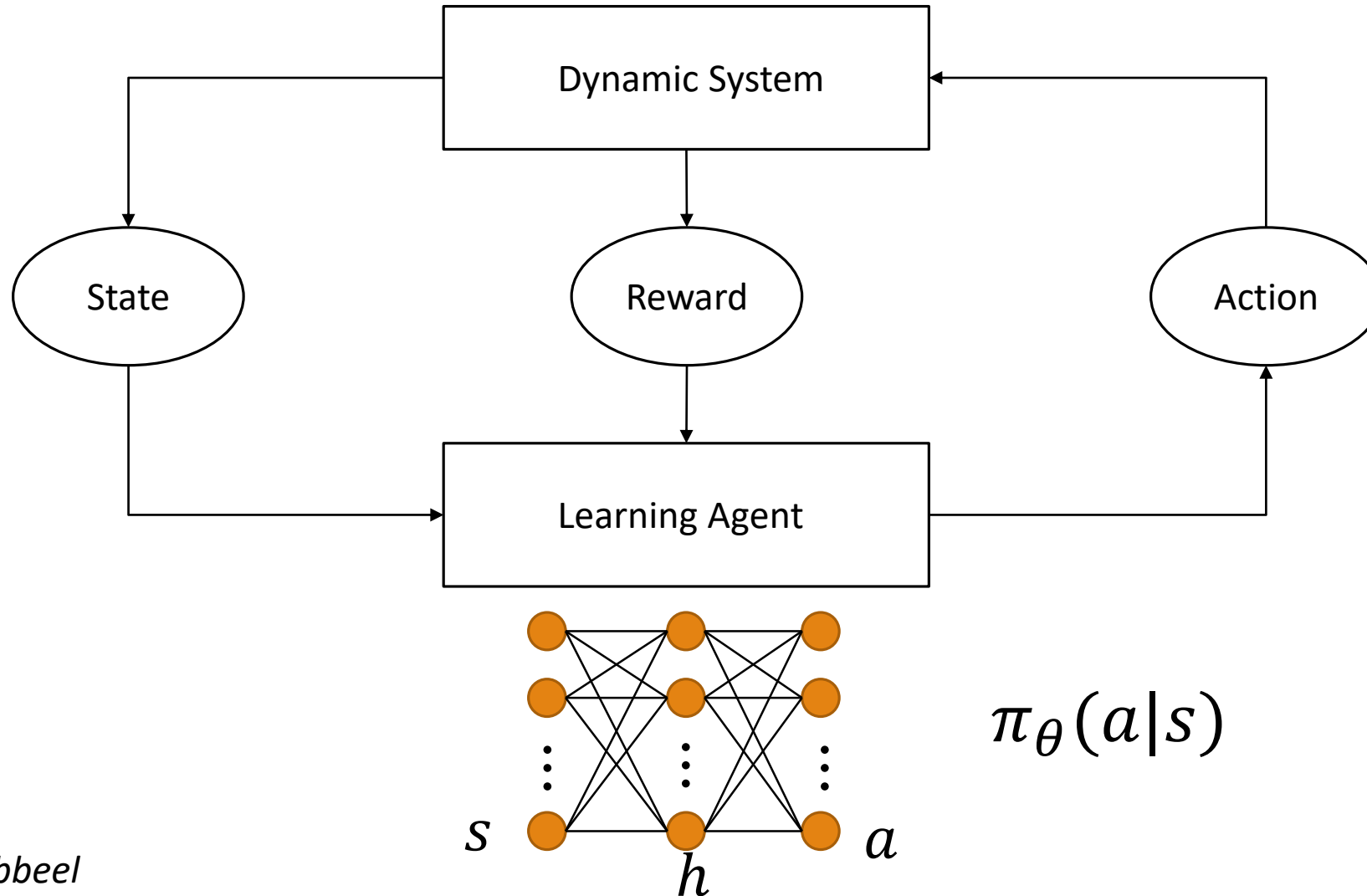
Slide inspired by P. Abbeel
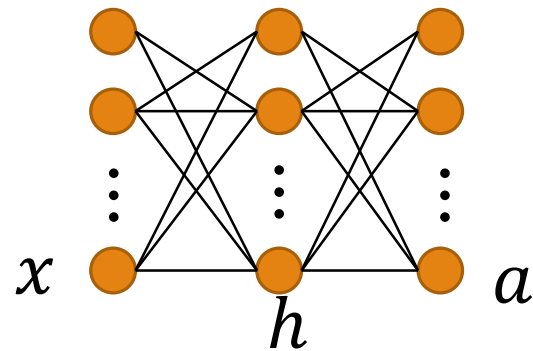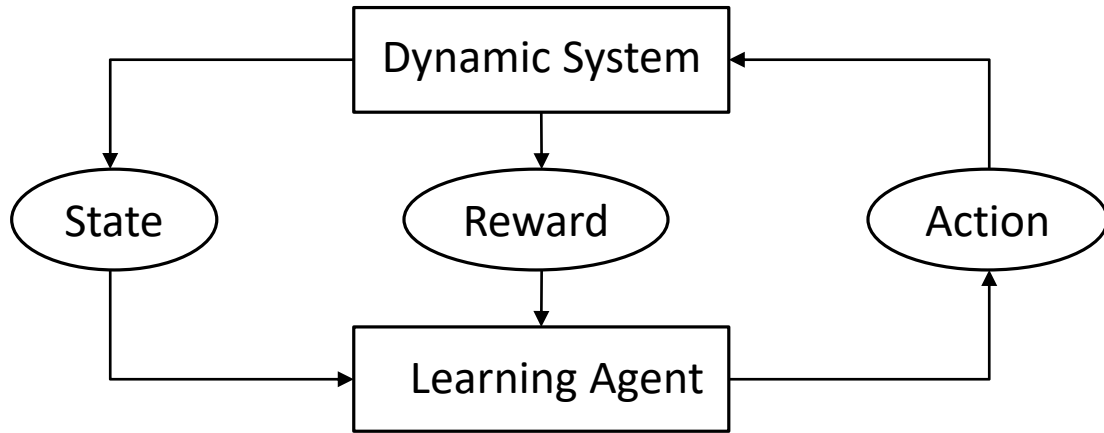
Policy Optimization

o Train learning agent for the optimal policy $\pi_w(a|s)$ given states s and possible actions a

o The policy class can be either deterministic or stochastic

$$\pi_w(a|s)$$

x h a

Slides inspired by P. Abbeel

Policy Optimization

o Use a deep networks as non-linear approximator that finds optimal policy by maximizing $Q(s, a; \theta)$

$$\mathcal{L}(w) = Q(s, a; w)$$
$$= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | \pi_w(s_t, a_t)]$$

o If policy is deterministic

$$\frac{\partial \mathcal{L}}{\partial w} = \mathbb{E}\left[\frac{\partial \log \pi(a|s, w)}{\partial w} Q^\pi(s, a)\right]$$

o If policy is stochastic $a = \pi(s)$

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

o To compute gradients use the log-derivative trick (REINFORCE algorithm (Williams, 1992))

$$\nabla_\theta \log p(x; \theta) = \frac{\nabla_\theta p(x; \theta)}{p(x; \theta)}$$

Asynchronous Advantage Actor-Critic (A3C)

o Estimate Value function
$$V(s, v) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots | s]$$

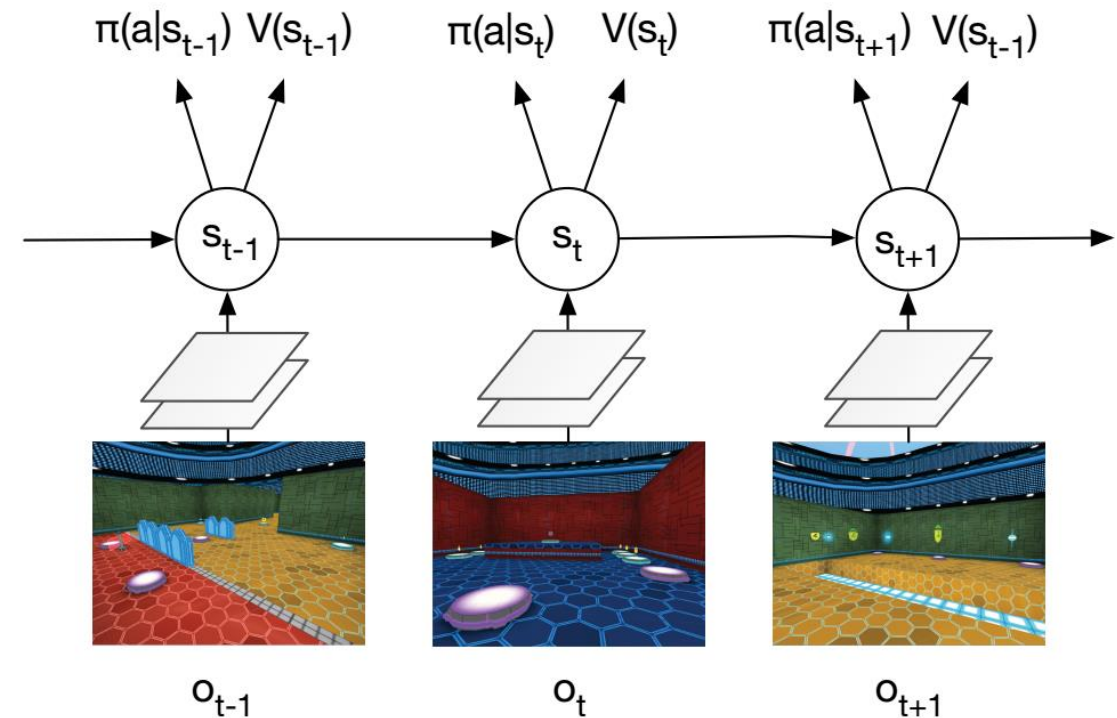o Estimate the Q value after n steps
$$q_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, v)$$

o Update actor by
$$\frac{\partial \mathcal{L}_{actor}}{\partial w} = \frac{\partial \log \pi(a_t | s_t, w)}{\partial w} (q_t - V(s_t, v))$$

A3C in labyrinth

o End-to-end learning of softmax policy from pixels

o Observations are the raw pixels

o The state is implemented as an LSTM

o Outputs value $V(s)$ and softmax over actions $\pi(a|s)$

o Task
  ◦ Collect apples (+1)
  ◦ escape (+10)

o Demo

Model-based Deep RL

Learning models of the environment

o Often quite challenging because of cumulative errors

o Errors in transition models accumulate over trajectory

o Planning trajectories are different from executed trajectories

o At the end of a long trajectory final rewards are wrong
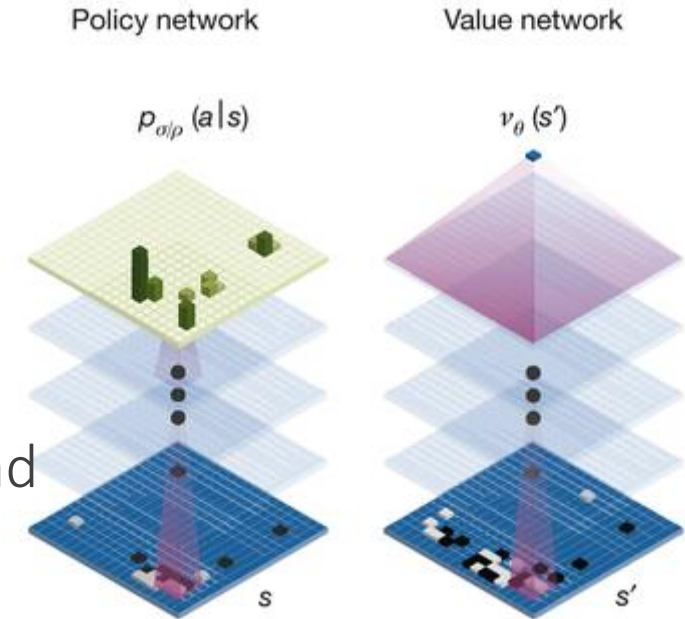
o Can be better if we know the rules

AlphaGo

- At least $10^{10^{48}}$ possible game states
  - Chess has 10^{120}

- Monte Carlo Tree Search used mostly
  - Start with random moves and evaluate how often they lead to victory
  - Learn the value function to predict the quality of a move
  - Exploration-exploitation trade-off
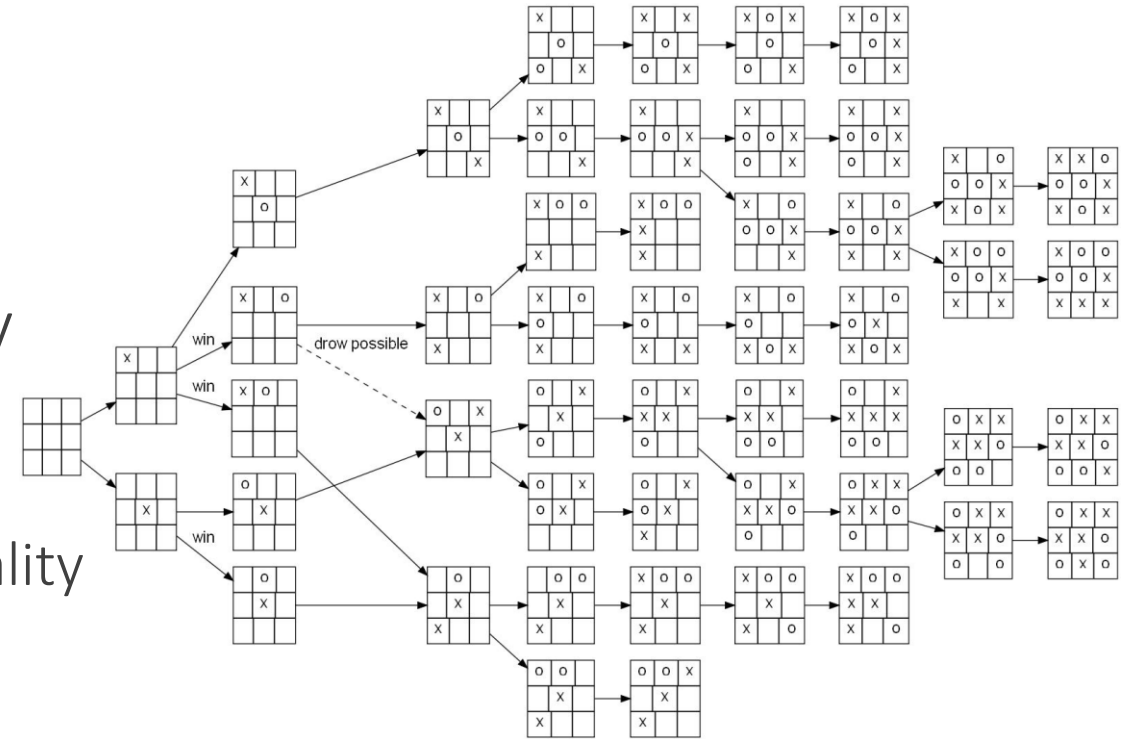

Tic-Tac-Toe possible game states

AlphaGo

o AlphaGo relies on a tree procedure for search

o AlphaGo relies on ConvNets to guide the tree search

o A ConvNet trained to predict human moves achieved 57% accuracy
  ◦ Humans make intuitive moves instead of thinking too far ahead

o For Deep RL we don't want to predict human moves
  ◦ Instead, we want the agent to learn the optimal moves

o Two policy networks (one per side) + One value network

o Value network trained on 30 million positions while policy networks play

Policy network $p_{\sigma/\rho}(a|s)$

Value network $v_\theta(s')$

AlphaGo

Policy network $p_{\sigma/\rho}(a|s)$

Value network $v_\theta(s')$

o Both humans and Deep RL agents play better end games
   ◦ Maybe a fundamental cause?

o In the end the value of a state is computed equally from Monte Carlo simulation and the value network output
   ◦ Combining intuitive play and thinking ahead

o Where is the catch?

AlphaGo

Policy network

$p_{\sigma/\rho}(a|s)$

Value network

$v_\theta(s')$

o Both humans and Deep RL agents play better end games
  ◦ Maybe a fundamental cause?

o In the end the value of a state is computed equally from Monte Carlo simulation and the value network output
  ◦ Combining intuitive play and thinking ahead

o Where is the catch?
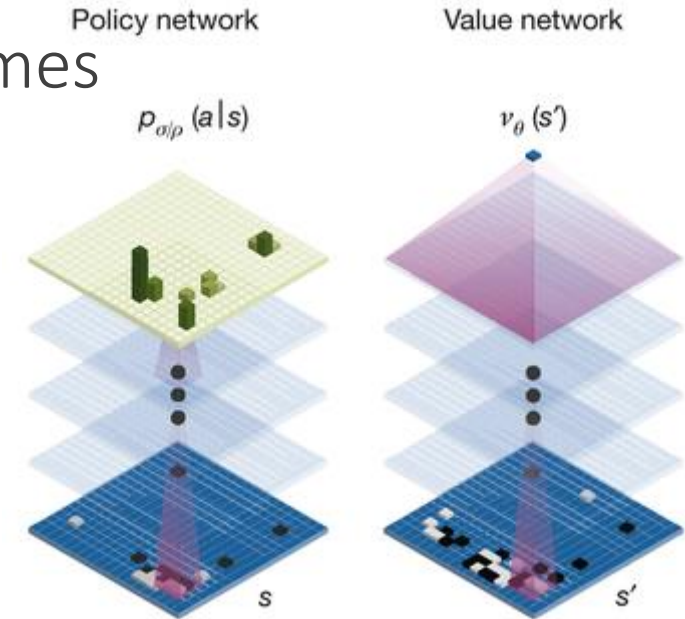
o State is not the pixels but positions

o Also, the game states and actions are highly discrete
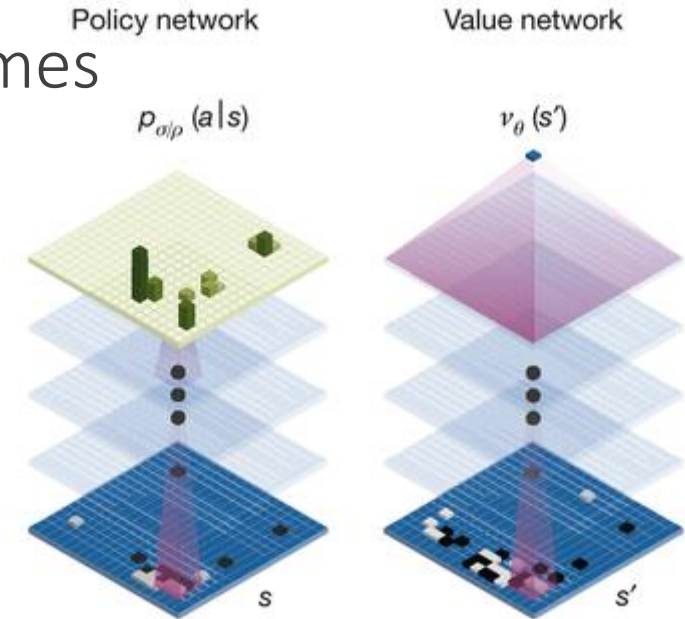
Exam

o What is allowed
  - 1 A4 page with whatever you want on it
  - Printed slides from the course
  - The Deep Learning book

o What is not allowed
  - Internet
  - Phones
  - And all sorts of personal communication

Summary

o Reinforcement Learning

o Q-Learning

o Deep Q-Learning

o Policy-based Deep RL

o Model-based Deep RL

o Making Deep RL stable