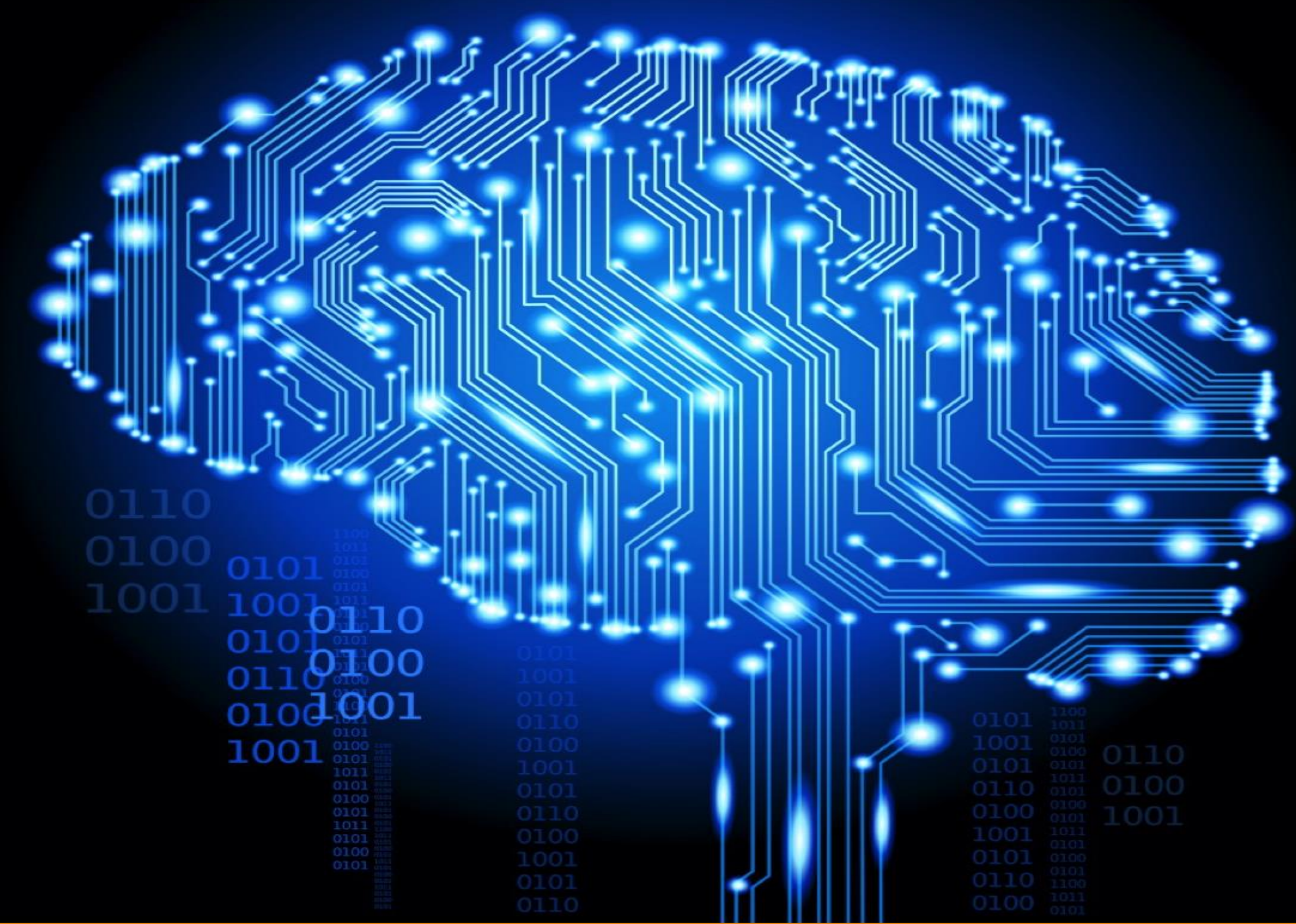
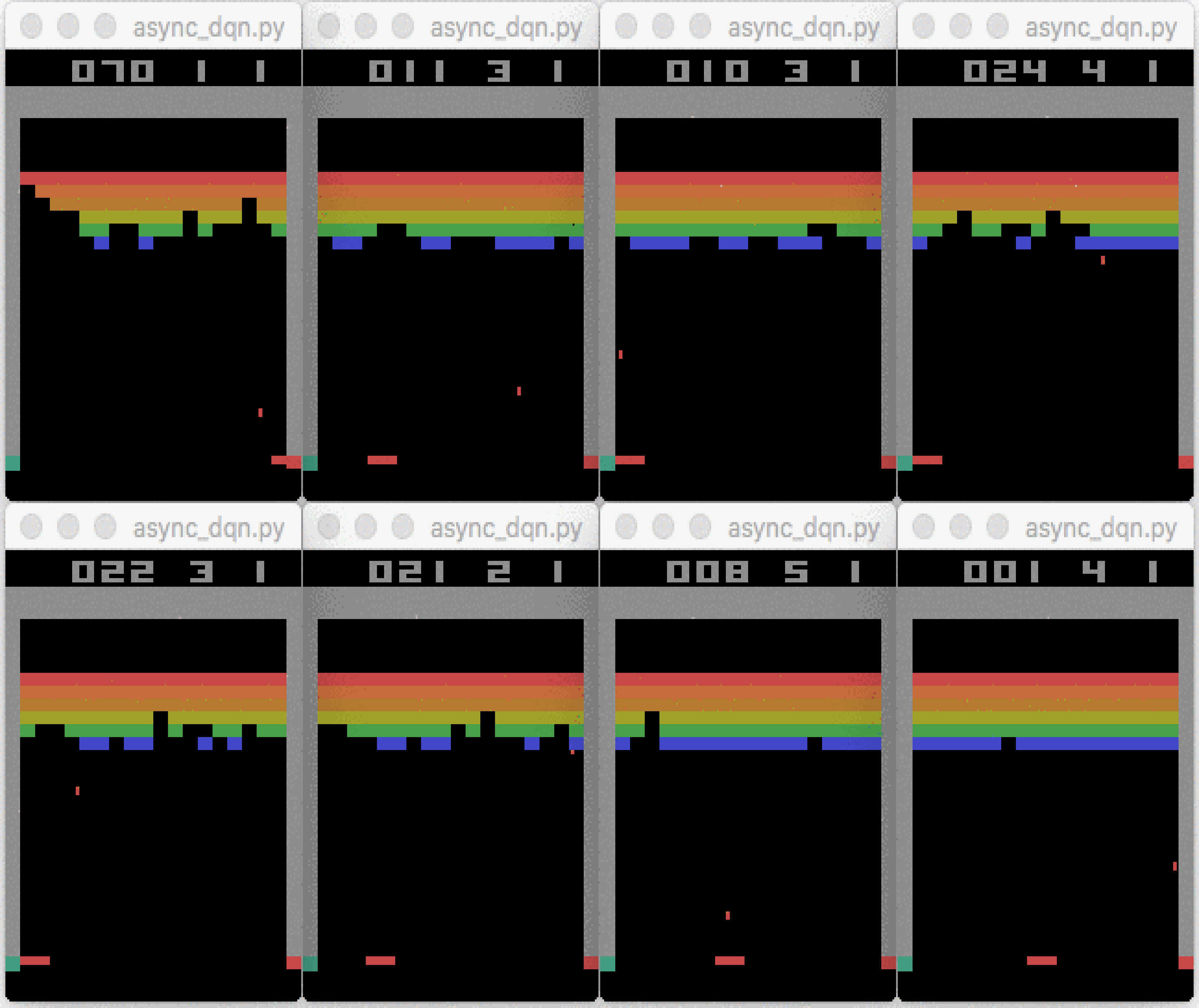


Lecture 13: Deep Reinforcement Learning

Deep Learning @ UvA

Reinforcement Learning



What is Reinforcement Learning?

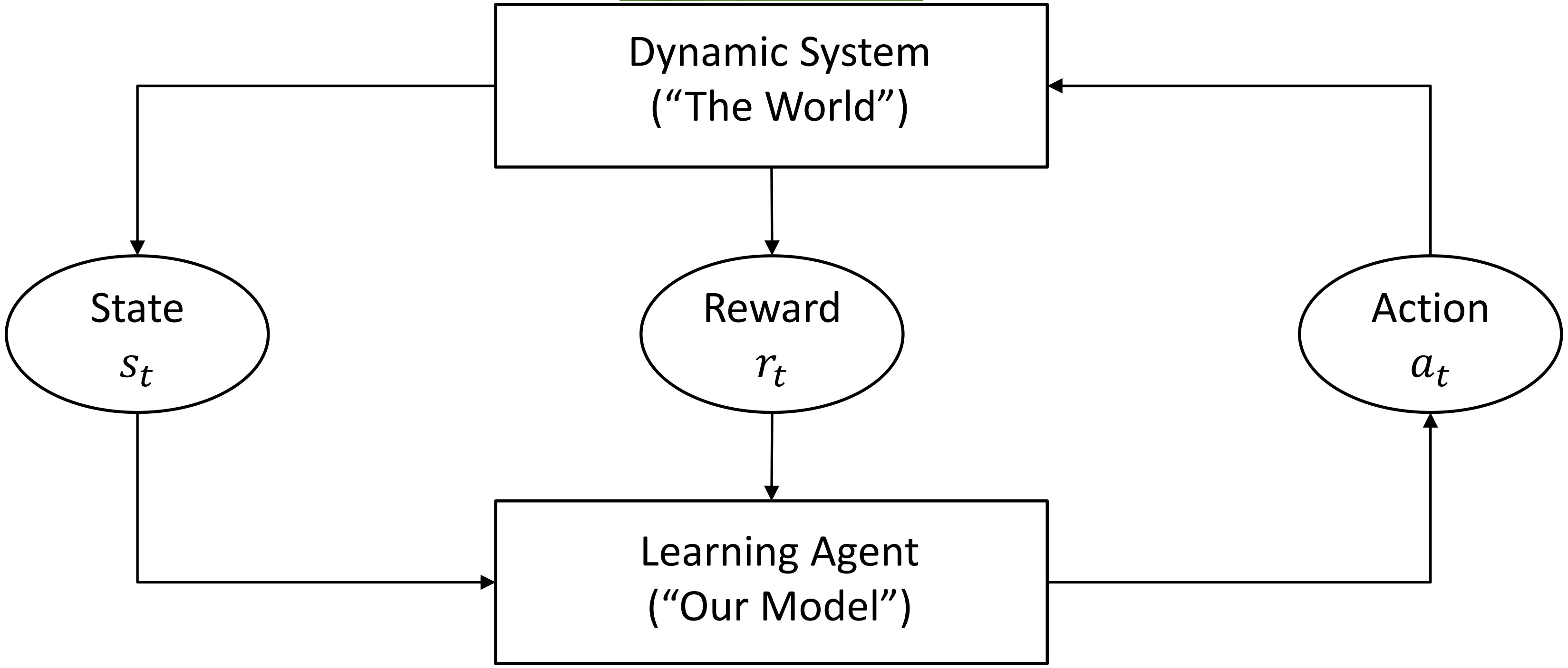
- General purpose framework for learning Artificial Intelligence models
- RL assumes that ***the agent*** (our model) can take ***actions***
- These actions affect ***the environment*** where ***the agent*** operates, more specifically ***the state*** of the environment and ***the state*** of the agent
- 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



Slides inspired by P. Abbeel

How do we decide about actions, states, rewards?

- We model them as functions
- The ***policy function*** $\mathrm{a_t} = \pi(s_t)$ selects an action given the current state
- 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
- What should our goal then be?

How do we decide about actions, states, rewards?

- We model them as functions
- The *policy function* $\mathrm{a}_t = \pi(s_t)$ selects an action given the current state
- 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
- What should our goal then be?

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

- Learn to take actions a_t that maximize the value function for different states

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
- Pros and cons?

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

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

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

- This is the ***Bellman equation***
- How can we rewrite the optimal value function $Q^*(s_t, a_t)$?

Bellman equation

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

- This is the ***Bellman equation***

- How can we rewrite the optimal value function $Q^*(s, a)$?
$$Q^*(s, a) = \mathbb{E}_{s'}\left(r + \gamma \max_{a'} Q^*(s', a') \middle| s, a\right)$$

Q-Learning

- In the simplest case the value function $Q(s, a)$ is a table
- In the beginning of the learning the function $Q(s, a)$ is incorrect
- Still, to the limit value iteration algorithms solve the Bellman equation

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

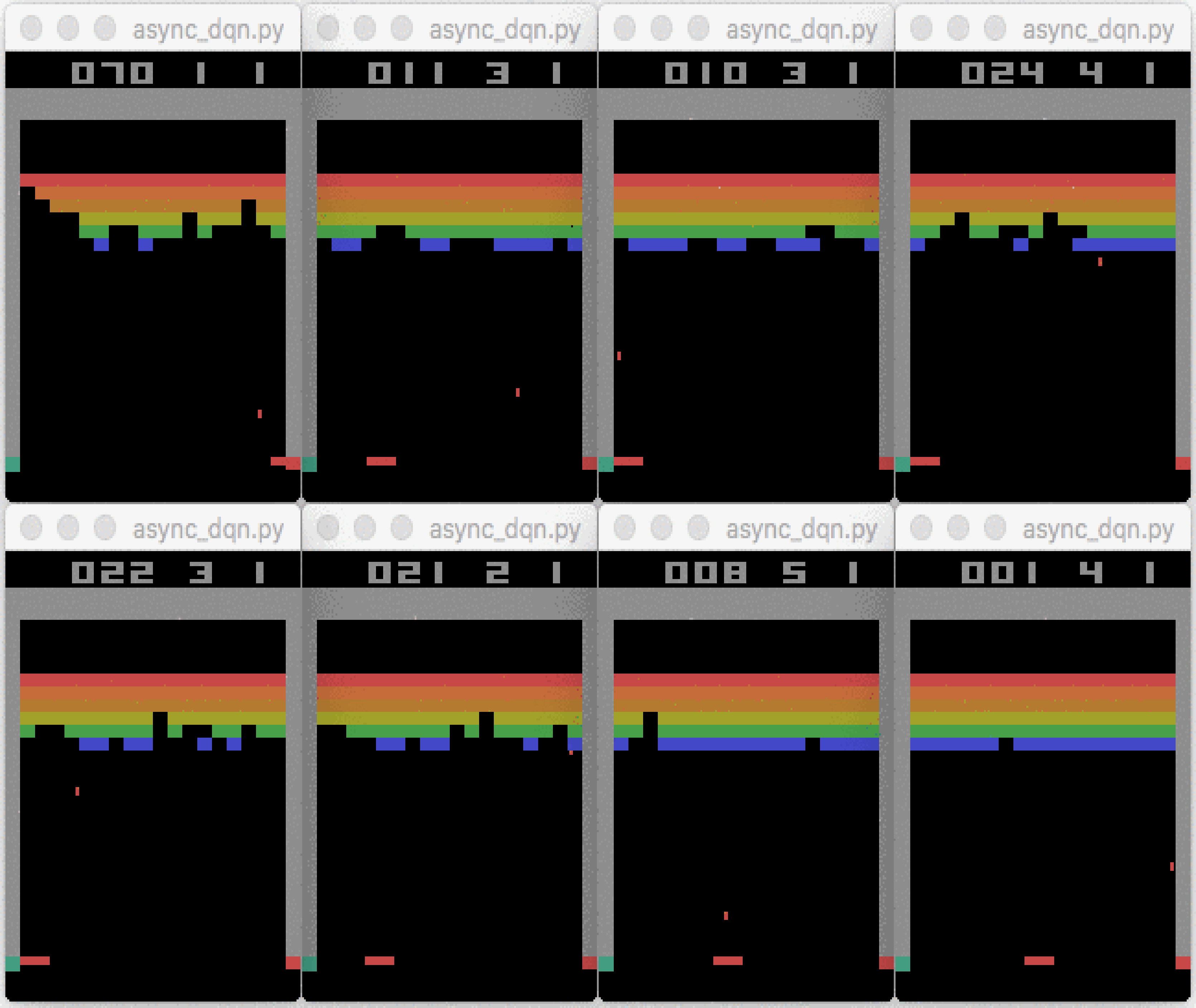
Policy Optimization

- Computing the Q-value is often too expensive
  - Hard to solve $\arg\max_{\alpha} Q_\theta(s, a)$
  - Especially when having continuous or high-dimensional action spaces
- Often defining the policy $\pi_\theta(u|s)$ is easier than defining the Q-function
- Use a non-linear function approximator to model the action value function

$$Q^*(s, a) \approx Q(s, a; \theta)$$

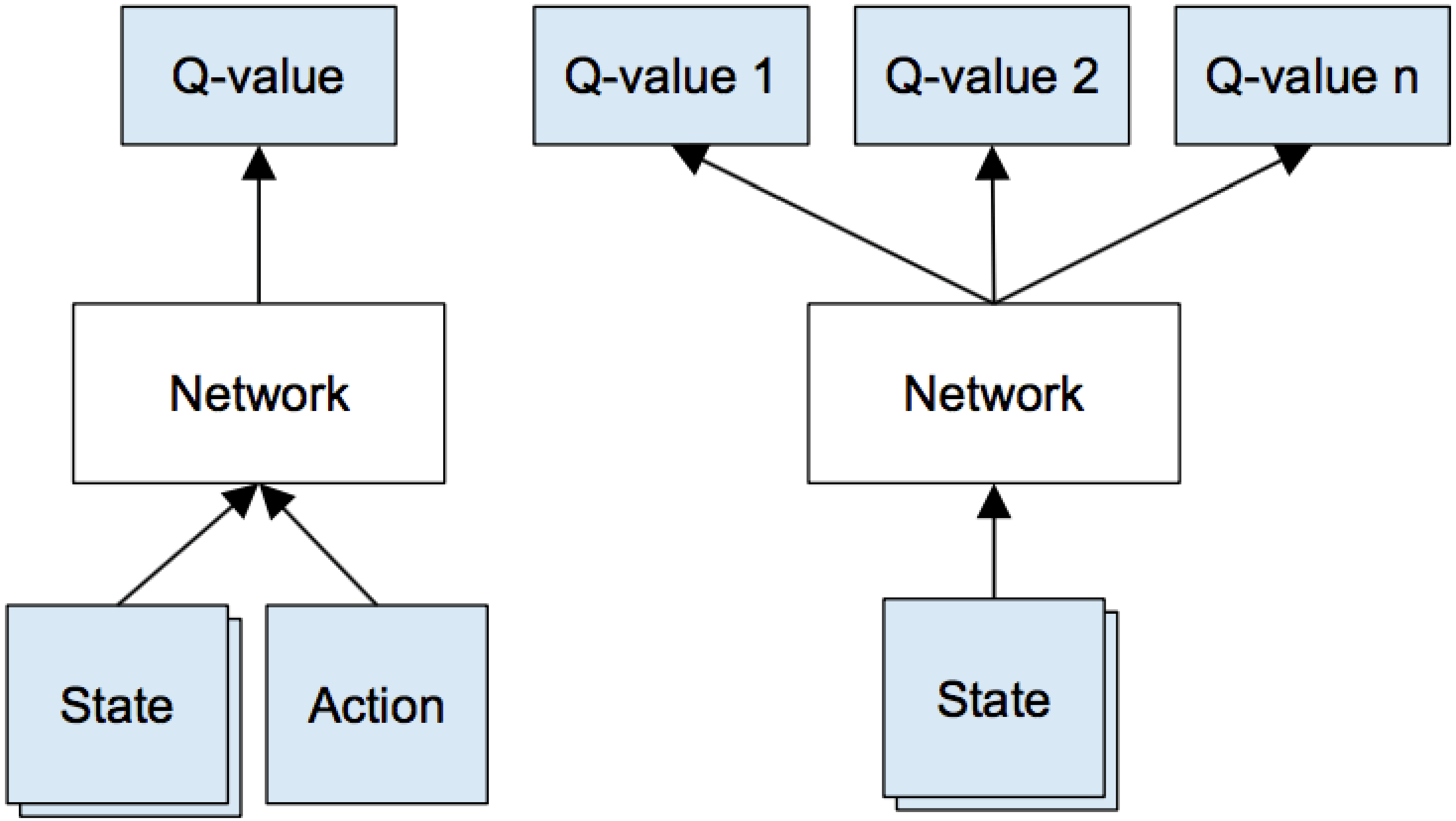
- Our deep network can be such a non-linear function approximator optimize the $\pi_\theta(u|s)$

Deep Reinforcement Learning



How to make RL deep?

How to make RL deep?



Deep Reinforcement Learning

- Rewards r_t at time t
- Actions π taken according to a policy $\pi = \mathrm{P}(a|s)$
- Again, the action-value function

$$Q(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi]$$

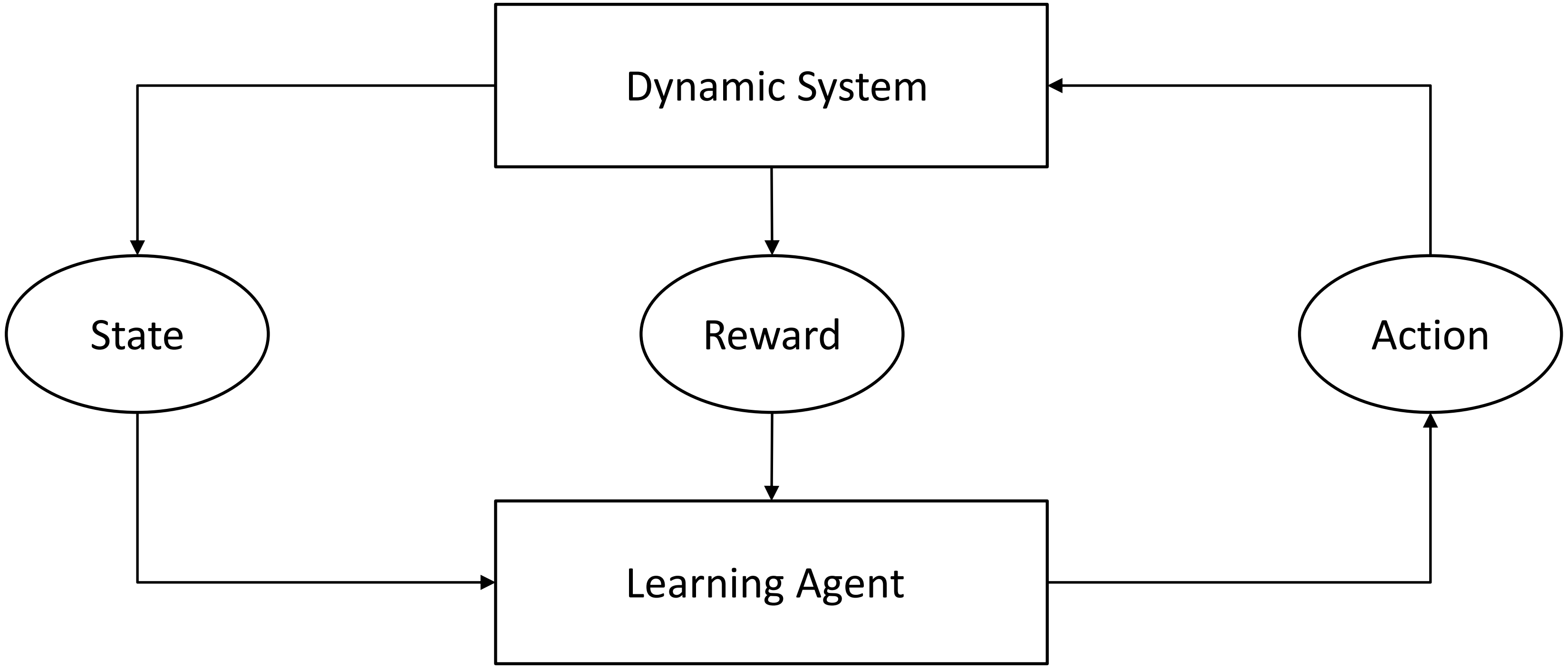
where γ is a discount factor of the future rewards

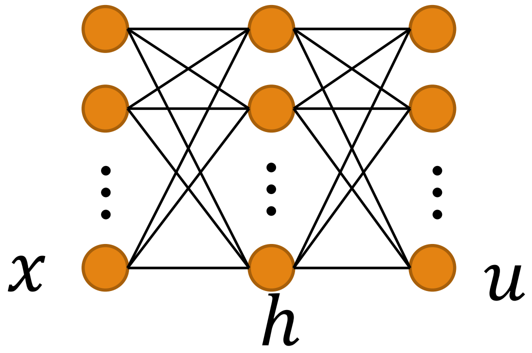
  - Future rewards should not be as important, because we do not know the future

- Use a non-linear function approximator to model the action value function

$$Q^*(s,a) \approx Q(s,a;\theta)$$

Policy optimization

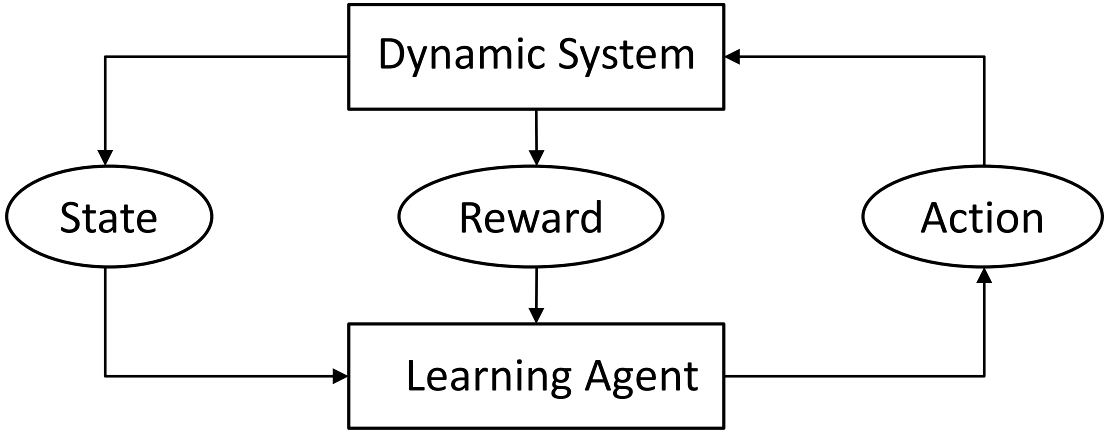


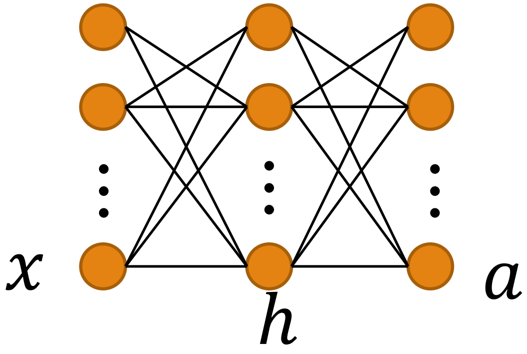


$$\pi_\theta(u|s)$$

Slides inspired by P. Abbeel

Policy optimization





$$\pi_\theta(a|s)$$

- Train learning agent for the optimal policy $\pi_\theta(a|s)$ given states s and possible actions a
- The policy class can be either deterministic or stochastic

Slides inspired by P. Abbeel

Deep Reinforcement Learning

- Non-linear function approximator: Deep Networks
- Input is as raw as possible, e.g. image frame
  - Or perhaps several frames (When needed?)
- Output is the best possible action out of a set of actions for maximizing future reward
- **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

How to optimize?

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

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

- The Q-Learning gradient then becomes

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

- 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 for action 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.

Stability in Deep Reinforcement Learning

Stability problems

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

Stability problems

- Naively, Q-Learning oscillates or diverges with neural networks
- Why?
- Sequential data breaks i.i.d. assumption
  - Highly correlated samples break SGD
- However, this is not specific to RL, as we have seen earlier

Stability problems

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

Stability problems

- The learning objective is

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

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

- Not easy to control the scale of the Q values→ gradients are unstable Q
- Remember, the Q function is the output of a neural network
- There is no guarantee that the outputs will lie in a certain range
  - Unless care is taken
- Naïve Q gradients can be too large, or too small → generally unstable and unreliable
- Where else did we observe a similar behavior?

Improving stability: Experience replay

- Replay memory/Experience replay
- Store memories $< s, a, r, s' >$
- Train using random stored memories instead of the latest memory transition
- Breaks the temporal dependencies – SGD works well if samples are roughly independent
- Learn from all past policies

Experience replay

- Take action a_t according to ε-greedy policy
- 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
- Optimize mean squared error using the mini-batch

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

- 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

- Clip rewards to be in the range $[-1, +1]$
- Or normalize them to lie in a certain, stable range
- Can’t tell the difference between large and small rewards

Results

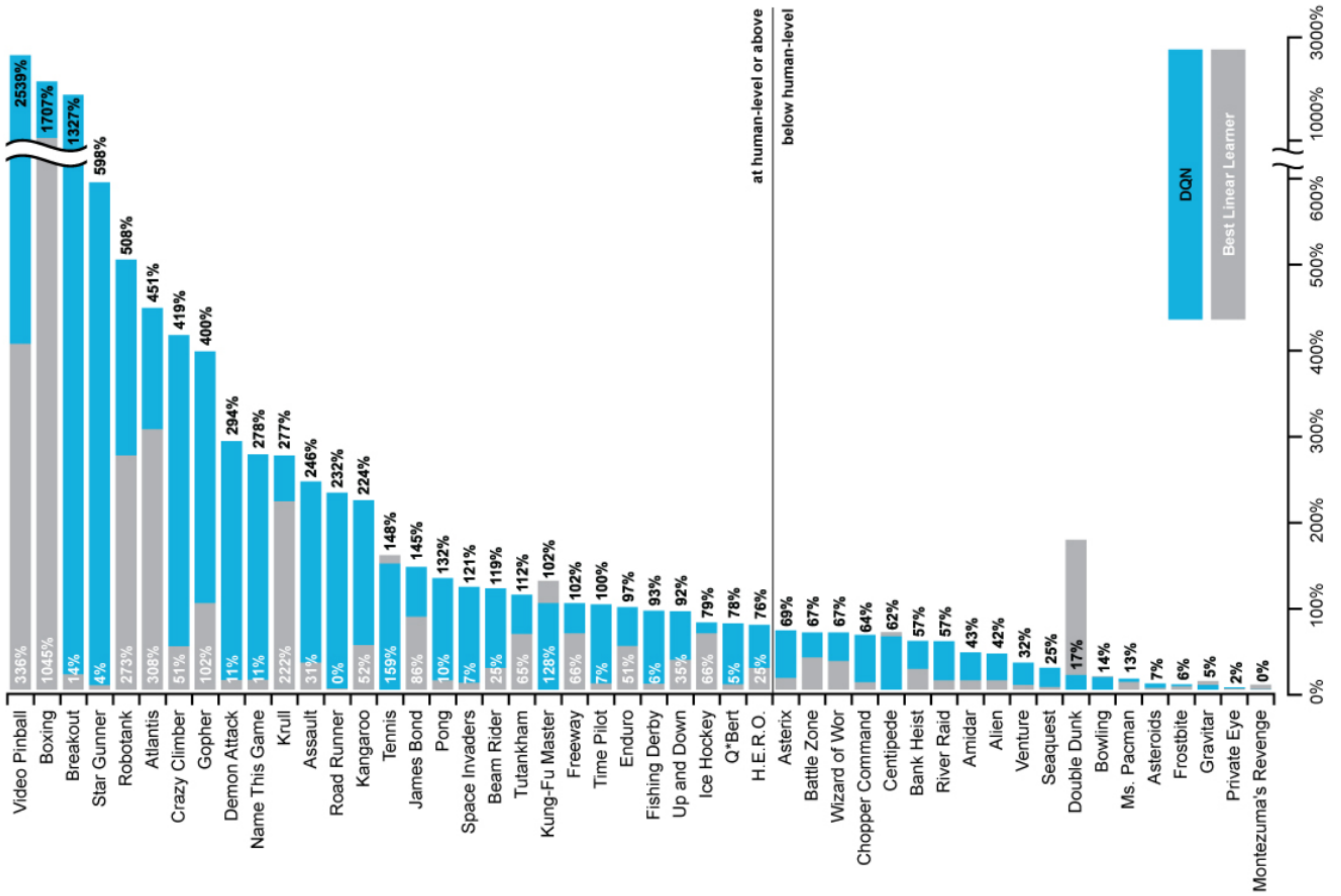
	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

Some extra tricks

- Skipping frames
  - Saves time and computation
  - Anyways, from one frame to the other there is often very little difference
- 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
- Alternatively: Exploration vs exploitation
  - early stages $\equiv$ strong exploration
  - late stages $\equiv$ strong exploitation

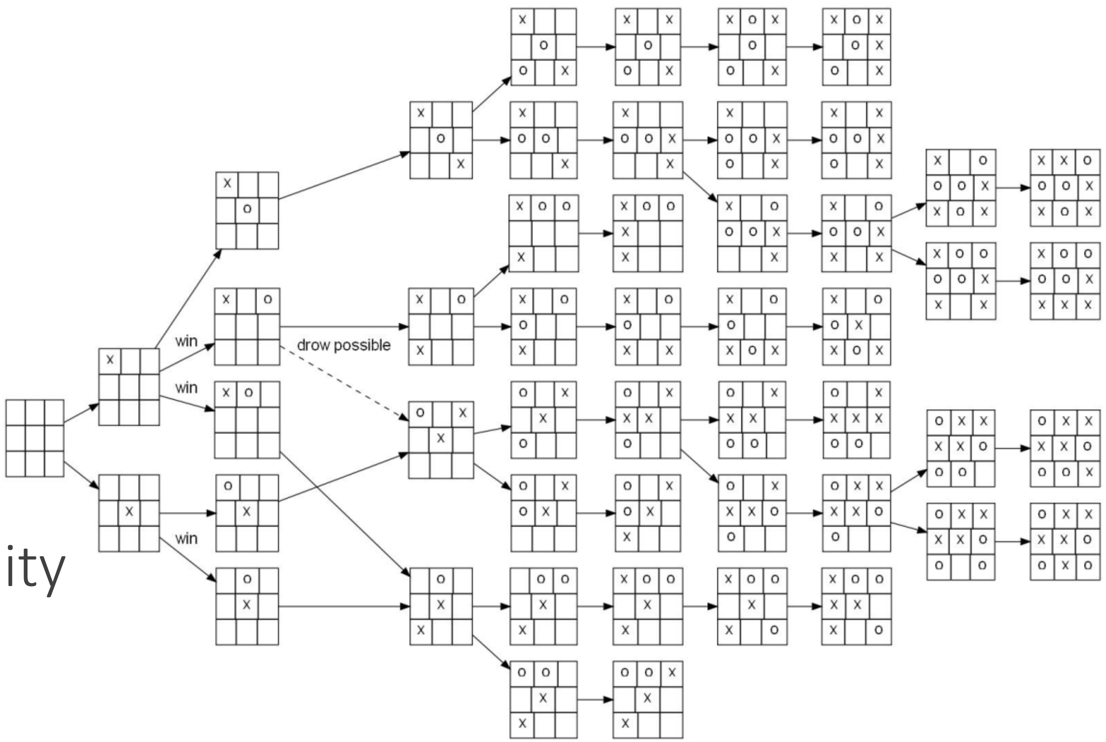
Examples of Deep Reinforcement Learning

Deep Reinforcement Learning in Atari



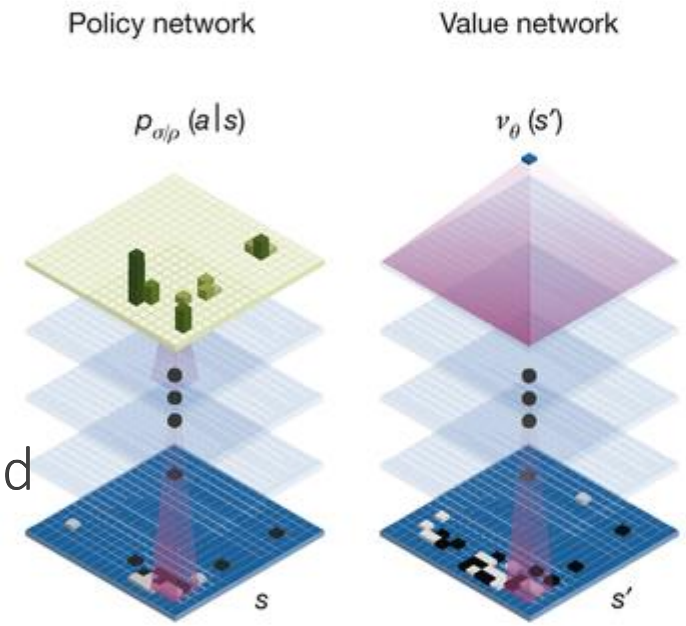
AlphaGo

- 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

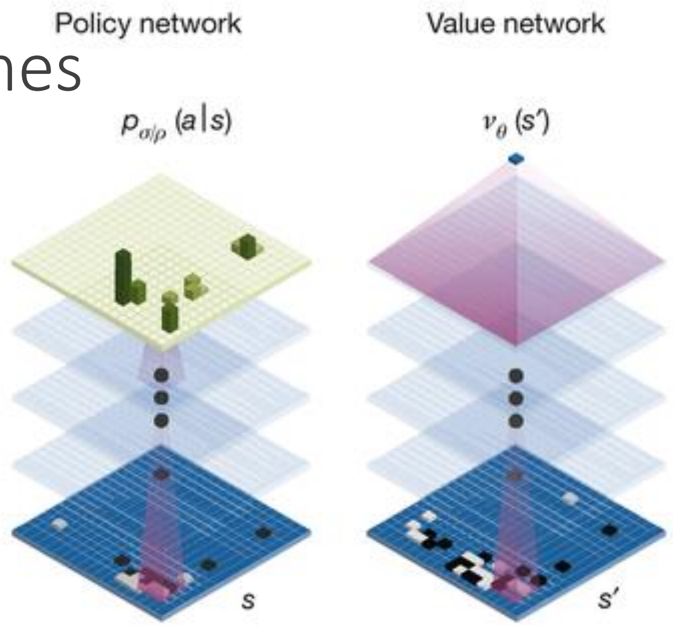
AlphaGo



- AlphaGo relies on a tree procedure for search
- AlphaGo relies on ConvNets to guide the tree search
- A ConvNet trained to predict human moves achieved 57% accuracy
  - Humans make intuitive moves instead of thinking too far ahead
- For Deep RL we don't want to predict human moves
  - Instead, we want the agent to learn the optimal moves
- Two policy networks (one per side) + One value network
- Value network trained on 30 million positions while policy networks play

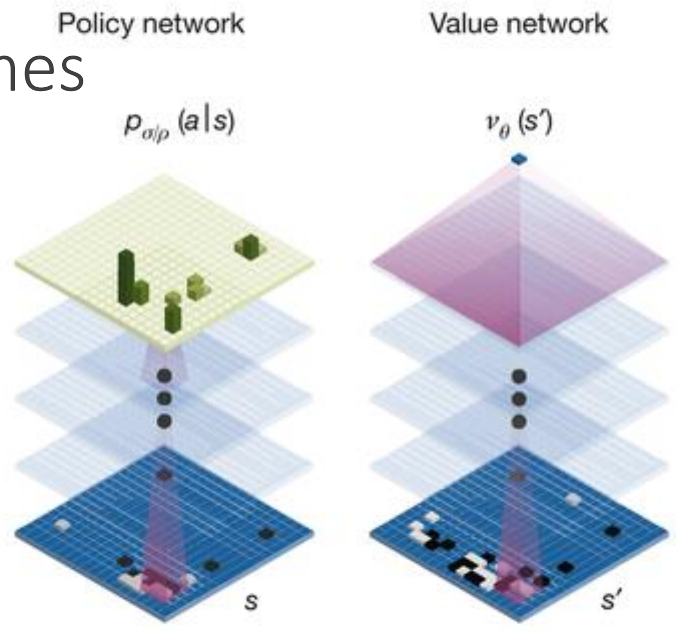
AlphaGo

- Both humans and Deep RL agents play better end games
  - Maybe a fundamental cause?
- 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
- Where is the catch?



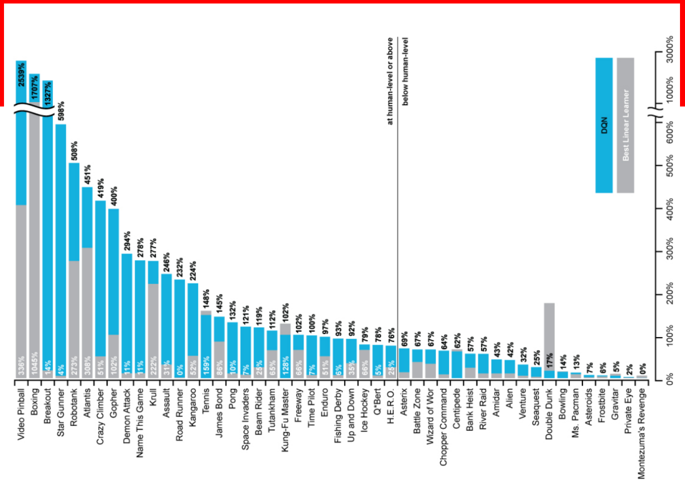
AlphaGo

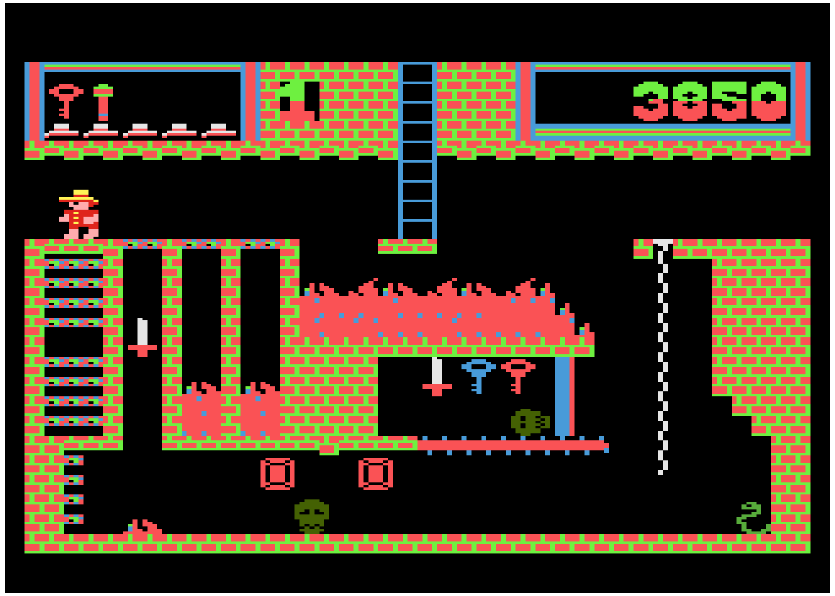
- Both humans and Deep RL agents play better end games
  - Maybe a fundamental cause?
- In the end the value of a state is computed equally from Monte Carlo simulation and the value network output
  - Combining intuitive play and thinking ahead
- Where is the catch?
- State is not the pixels but positions
- Also, the game states and actions are highly discrete



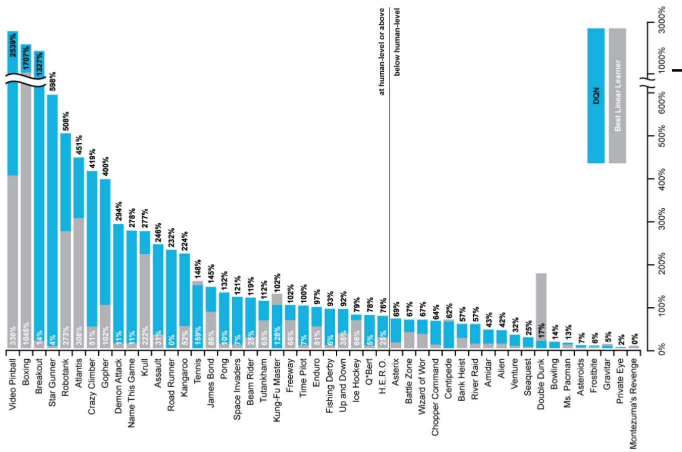
Montezuma's Revenge

- Hardest of the Atari games
- Why is it so hard?

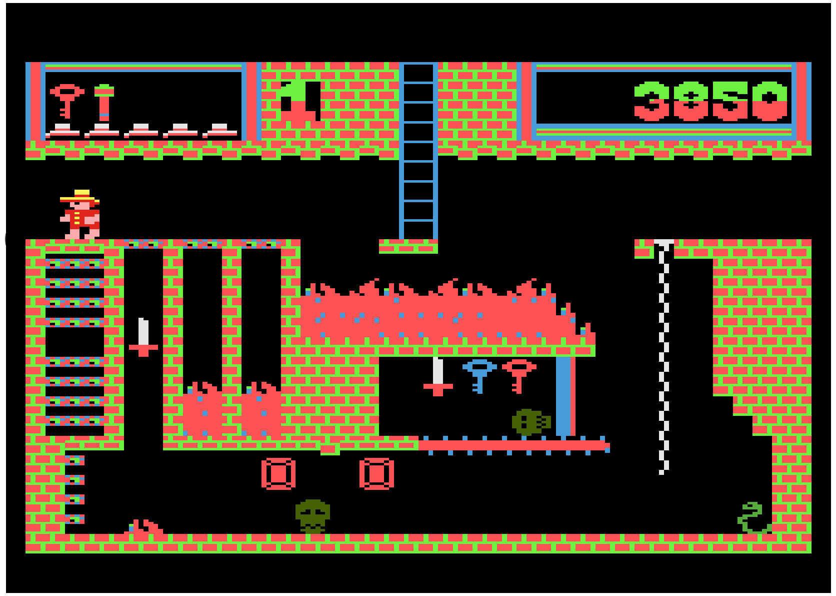




Montezuma’s Revenge



- Hardest of the Atari games
- Why is it so hard?
- Very long-term dependencies
  - Must search in multiple rooms to find the “secret”
- Future rewards are too delayed
  - It takes a while to evaluate an action was good or n
  - Hard to optimize
- FeUdal Networks for Hierarchical Reinforcement Learning, A. Vezhnevets



Starcraft II

- Dr. O. Vinyals in DeepMind tries to bring Starcraft II and Deep RL together
  - There is a Machine Learning API from Blizzard
  - There is a dataset of anonymized game replays
  - An open source python toolkit from DeepMind
  - And there is a paper
- More info
  - https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/
  - https://www.youtube.com/watch?v=-fKUyT14G-8
- What are the possible difficulties?

Summary

- Reinforcement Learning
- Q-Learning
- Deep Q-Learning
- Making Deep Q-Learning stable
- Examples of Deep Q-Learning