CS 7643: Deep Learning

Topics:

- Review of Classical Reinforcement Learning
- Value-based Deep RL
- Policy-based Deep RL

Dhruv Batra

Georgia Tech

Types of Learning

- Supervised learning
  - Learning from a “teacher”
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Reinforcement learning
  - Learning to act under evaluative feedback (rewards)

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

x y

→ Cat

Classification

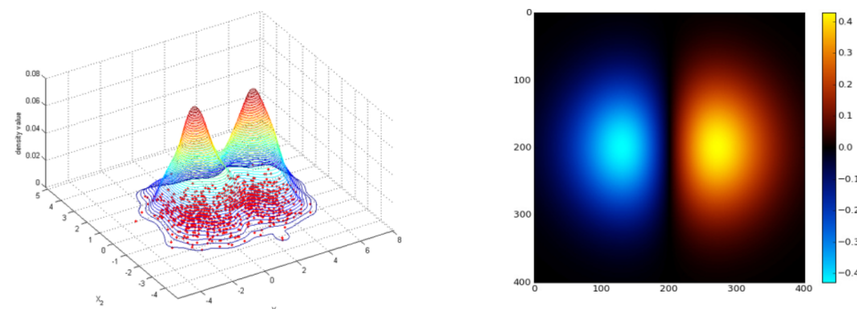
Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

1-d density estimation

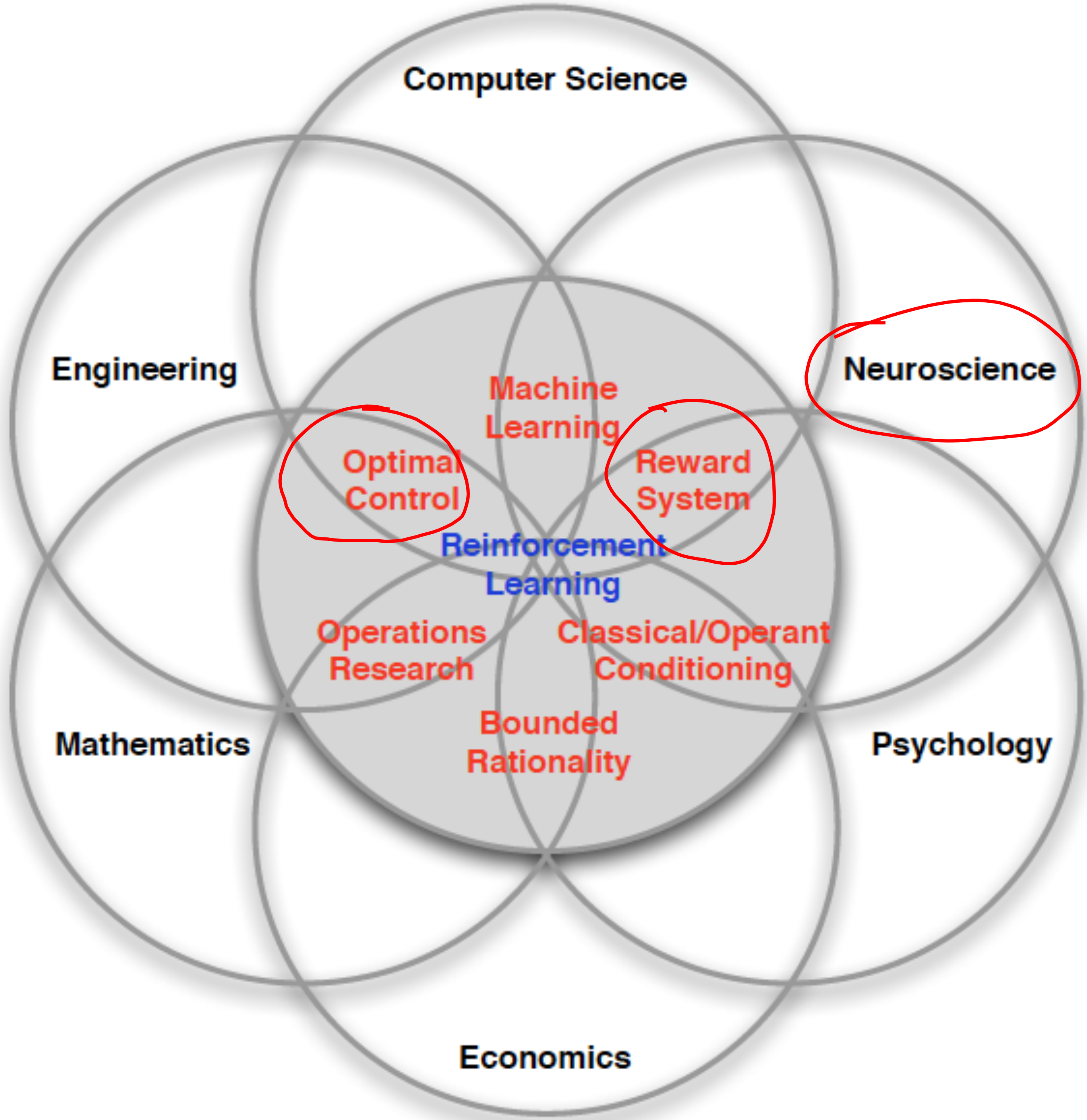


2-d density estimation

What is Reinforcement Learning?

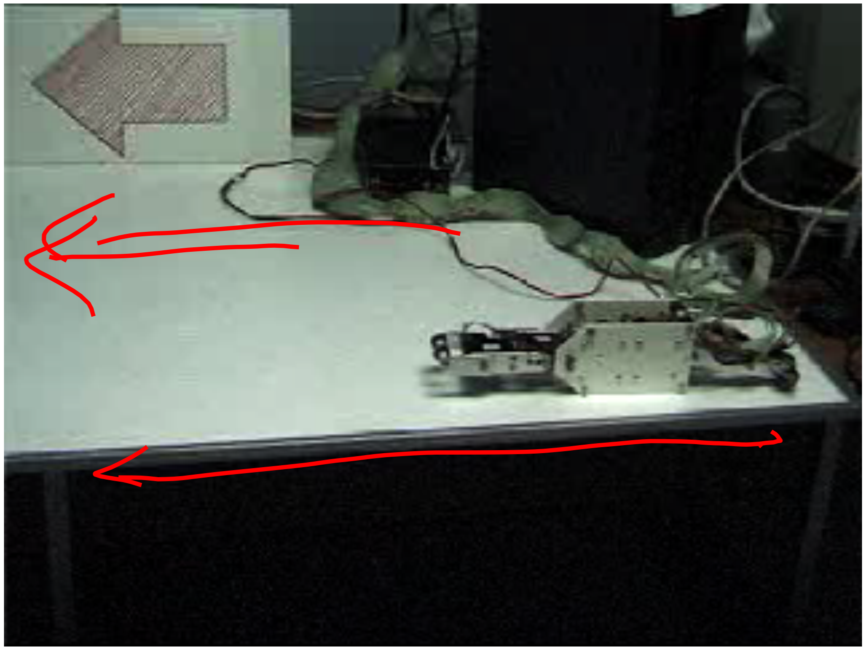
- Agent-oriented learning—learning by interacting with an environment to achieve a goal
  - more realistic and ambitious than other kinds of machine learning
- Learning by trial and error, with only delayed evaluative feedback (reward)
  - the kind of machine learning most like natural learning
  - learning that can tell for itself when it is right or wrong

Slide Credit: Rich Sutton

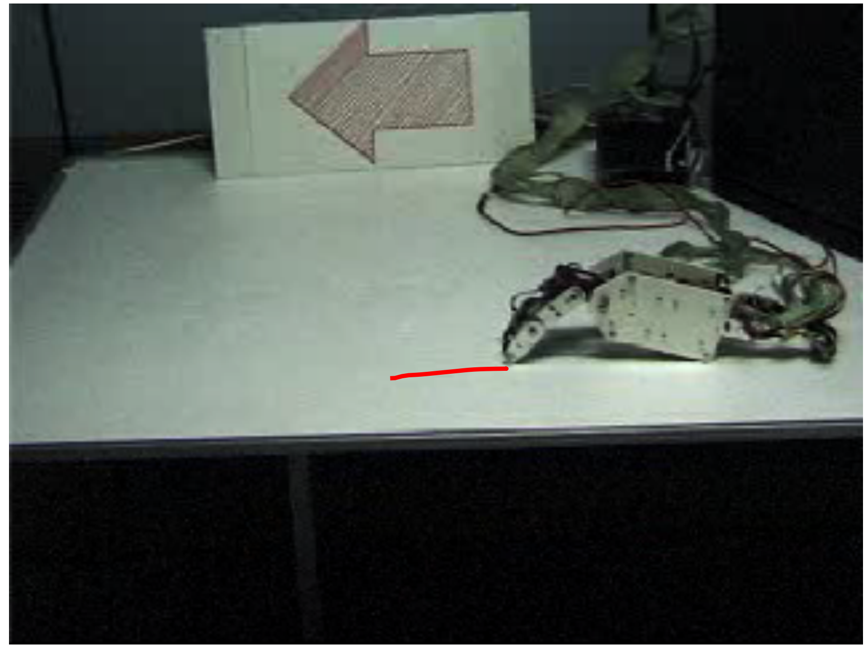

Computer Science
Engineering
Neuroscience
Machine Learning
Optimal Control
Reward System
Reinforcement Learning
Operations Research
Classical/Operant Conditioning
Bounded Rationality
Mathematics
Psychology
Economics

Silver 2015

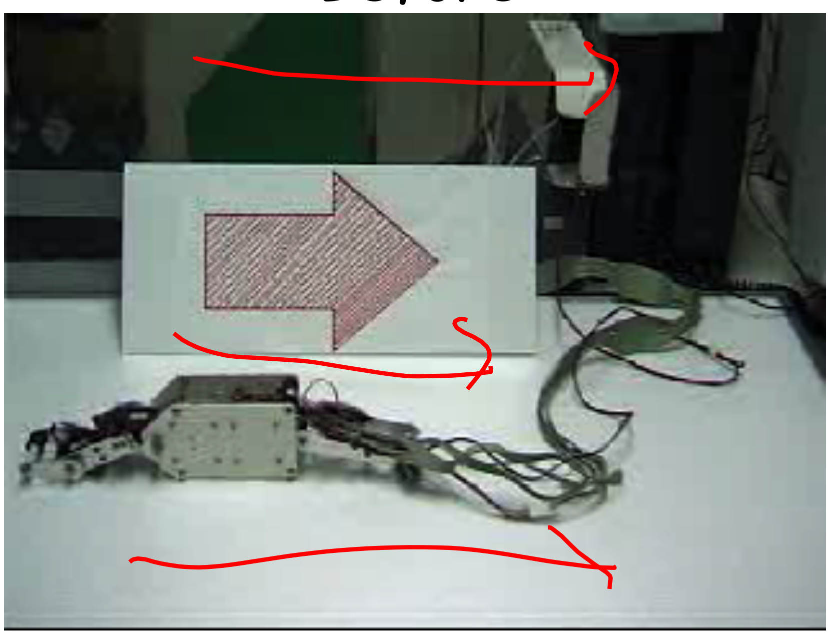
Example: Hajime Kimura's RL Robots



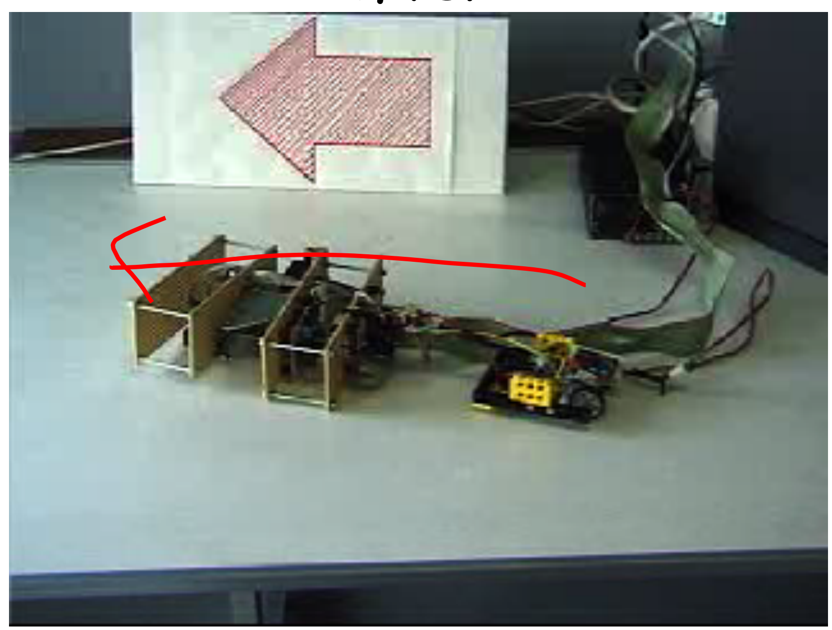
Before



After



Backward



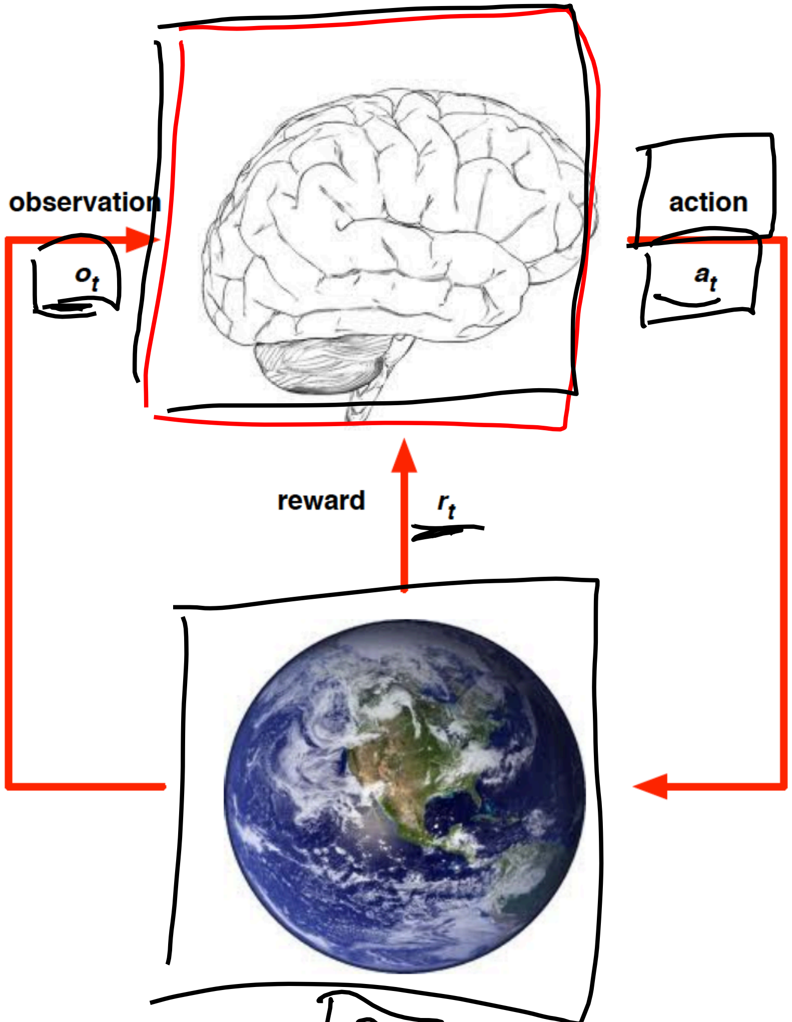
New Robot, Same algorithm

Signature challenges of RL

- Evaluative feedback (reward)
- Sequentiality, delayed consequences
- Need for trial and error, to explore as well as exploit
- Non-stationarity
- The fleeting nature of time and online data

Slide Credit: Rich Sutton

RL API



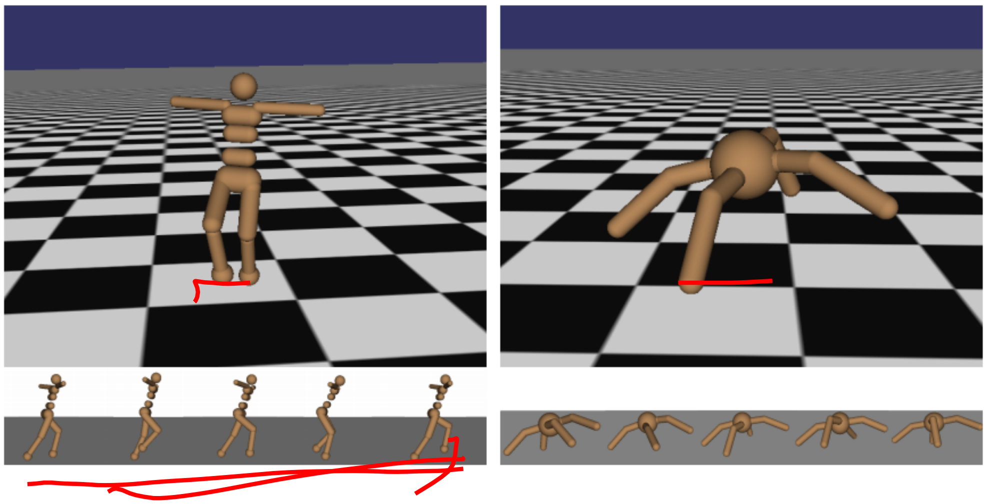
$o_t = s_t$

$o_t = f(s_t)$

- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

State

Robot Locomotion



Objective: Make the robot move forward

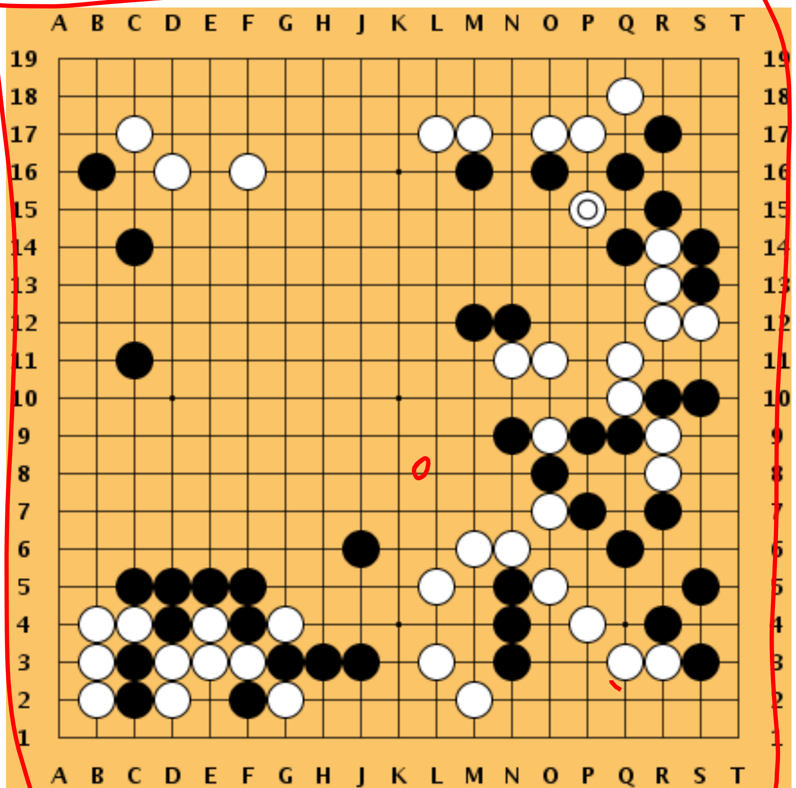
State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Demo

Markov Decision Process

- Mathematical formulation of the RL problem

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

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Process

- Mathematical formulation of the RL problem

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- Life is trajectory: $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$

Markov Decision Process

- Mathematical formulation of the RL problem

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- Life is trajectory: $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$

- **Markov property**: Current state completely characterizes the state of the world

$$p(r, s'|s, a) = Prob\left[R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

Components of an RL Agent

(Deep)

- Policy -based RL
  - How does an agent behave?
- Value function RL
  - How good is each state and/or state-action pair?
- Model RL
  - Agent's representation of the environment

Policy

- A policy is how the agent acts
- Formally, map from states to actions

e.g.

State	Action
A	→ 2
B	→ 1

Deterministic policy: $a = \pi(s)$

Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

The optimal policy π^*

What's a good policy?

The optimal policy **π***

What's a good policy?

Maximizes current reward? Sum of all future reward?

The optimal policy **π***

What's a good policy?

Maximizes current reward? Sum of all future reward?

Discounted future rewards!

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$(0, 1)$

The optimal policy **π***

What's a good policy?

Maximizes current reward? Sum of all future reward?

Discounted future rewards!

Formally: $$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$$

with $$s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

Value Function

- A value function is a prediction of future reward

- "State Value Function" or simply "Value Function"
  - How good is a state?
  - Am I screwed? Am I winning this game?

- "Action Value Function" or Q-function
  - How good is a state action-pair?
  - Should I do this now?

$Q(s, a)$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state?
The **value function** at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …

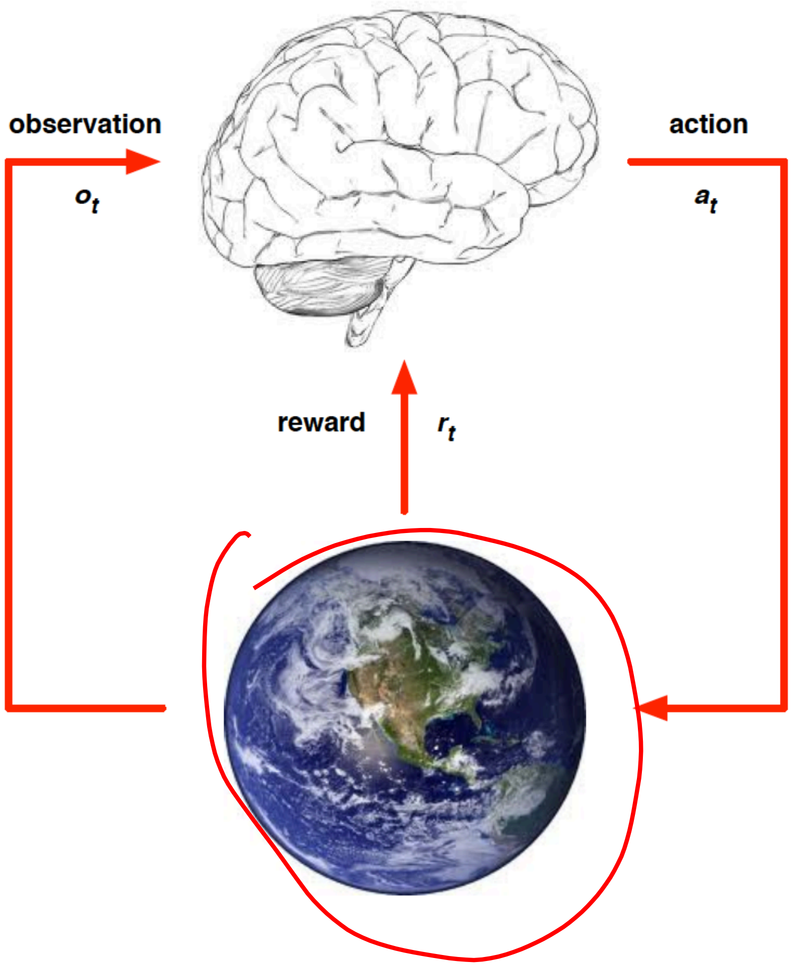
How good is a state?
The **value function** at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

How good is a state-action pair?
The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s (and following the policy thereafter):

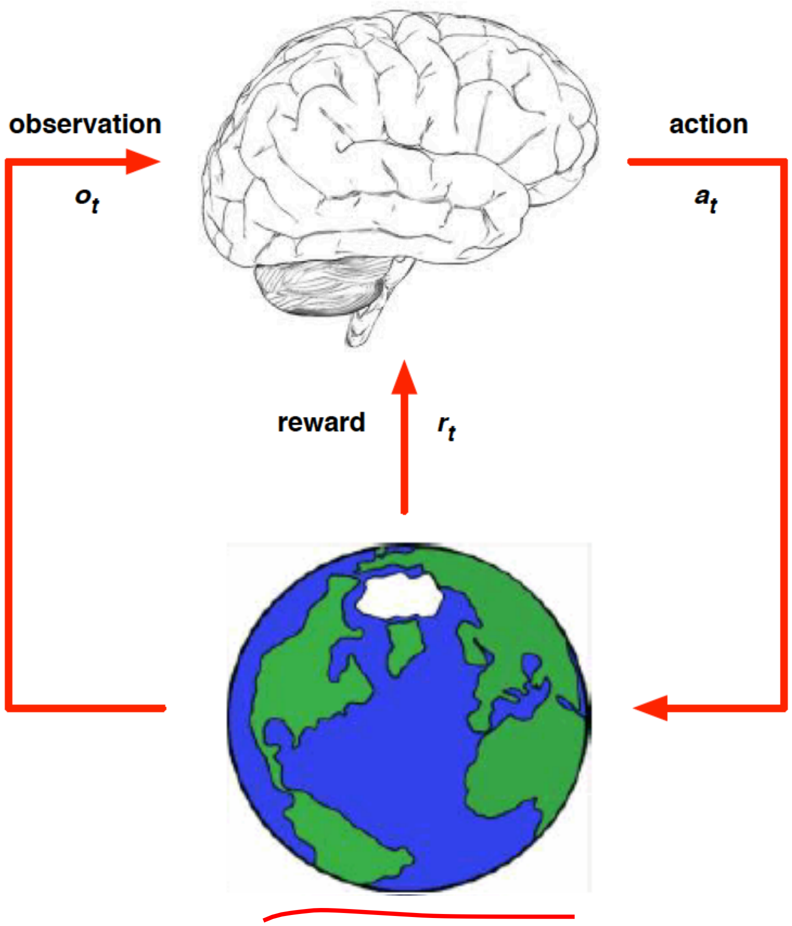
$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Model



Model

- Model predicts what the world will do next

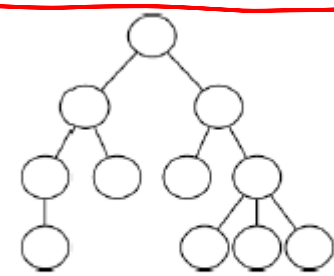


Model is learnt from experience

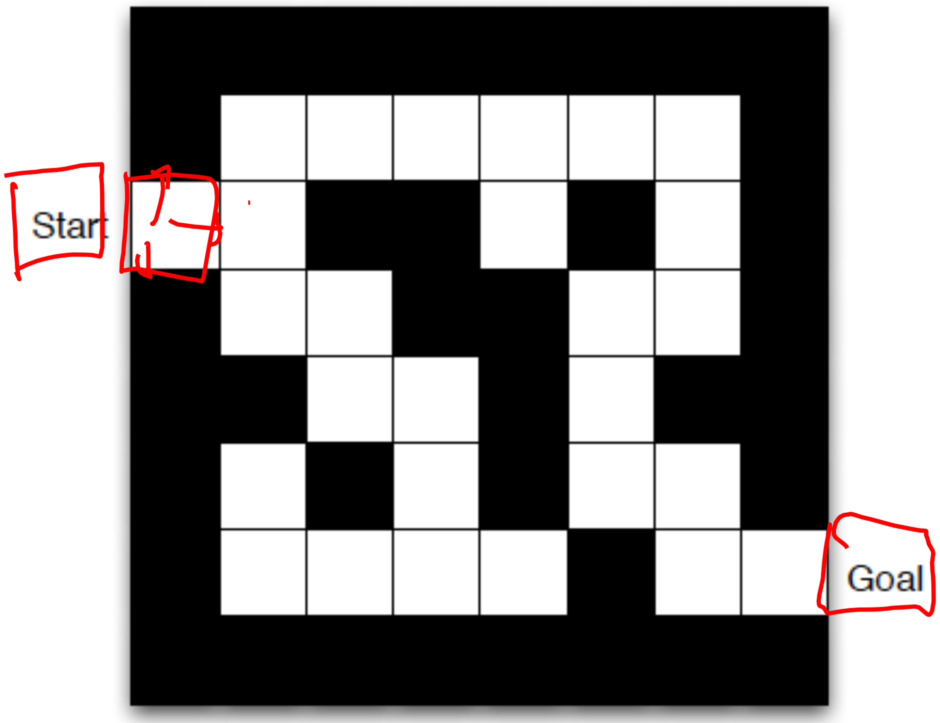
Acts as proxy for environment

Planner interacts with model

e.g. using lookahead search

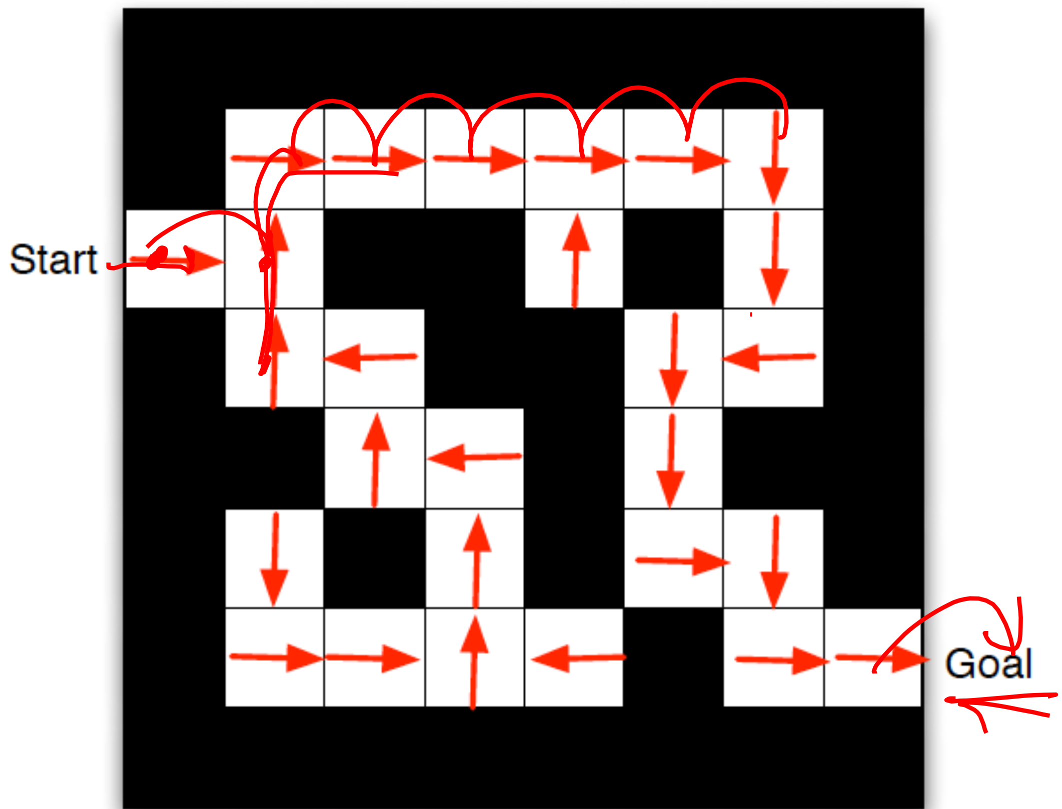


Maze Example



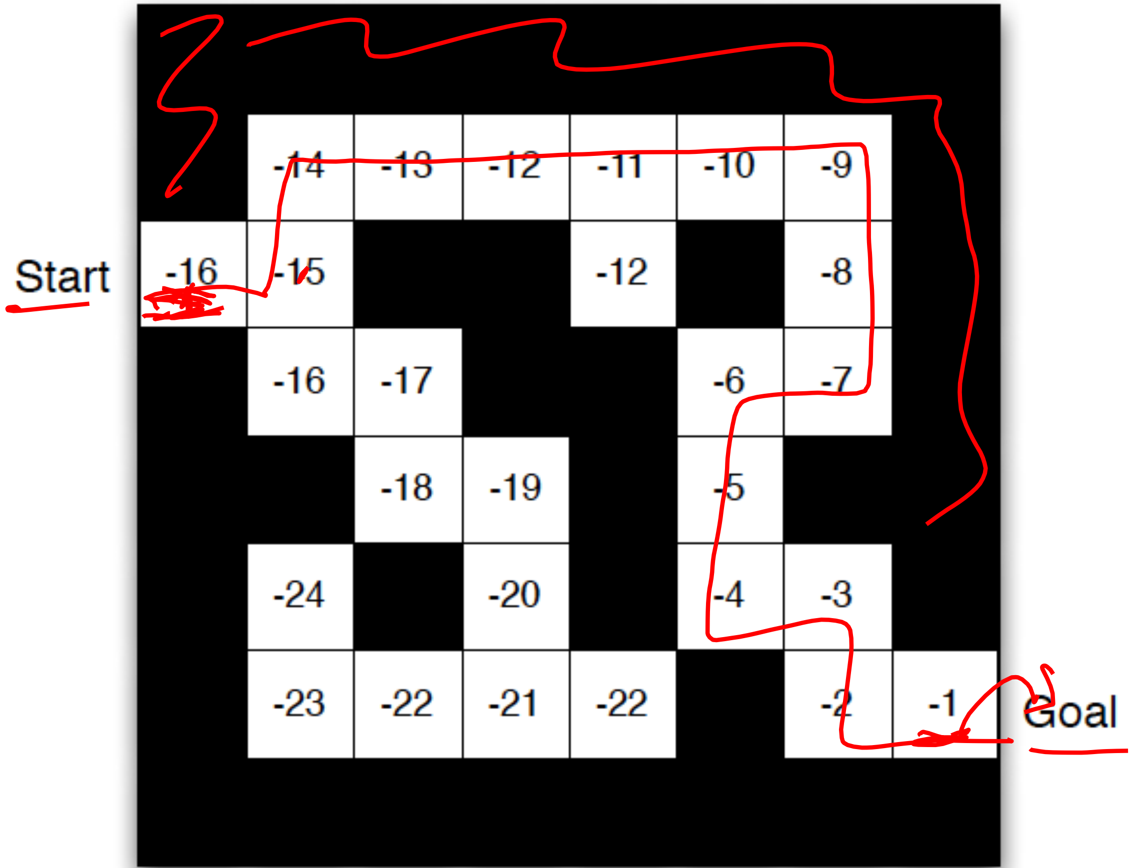
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



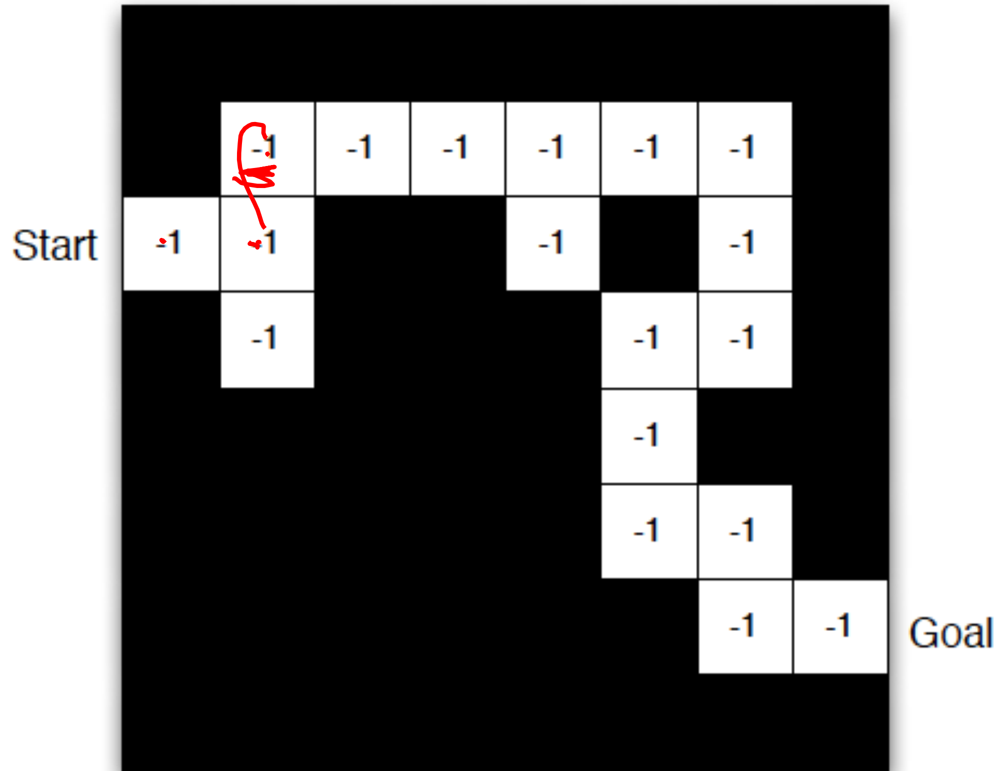
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value



- Numbers represent value $v_\pi(s)$ of each state s

Maze Example: Model



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $\mathcal{P}^a_{ss'}$
- Numbers represent immediate reward $\mathcal{R}^a_s$ from each state s (same for all a)

Components of an RL Agent

- Value function
  - How good is each state and/or state-action pair?
- Policy
  - How does an agent behave?
- Model
  - Agent's representation of the environment

Approaches to RL

- Value-based RL
  - Estimate the optimal action-value function $Q^*(s, a)$
- Policy-based RL
  - Search directly for the optimal policy π^*
- Model
  - Build a model of the world
    - State transition, reward probabilities
  - Plan (e.g. by look-ahead) using model

Deep RL

- Value-based RL
  - Use neural nets to represent Q function $Q(s, a; \theta)$

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

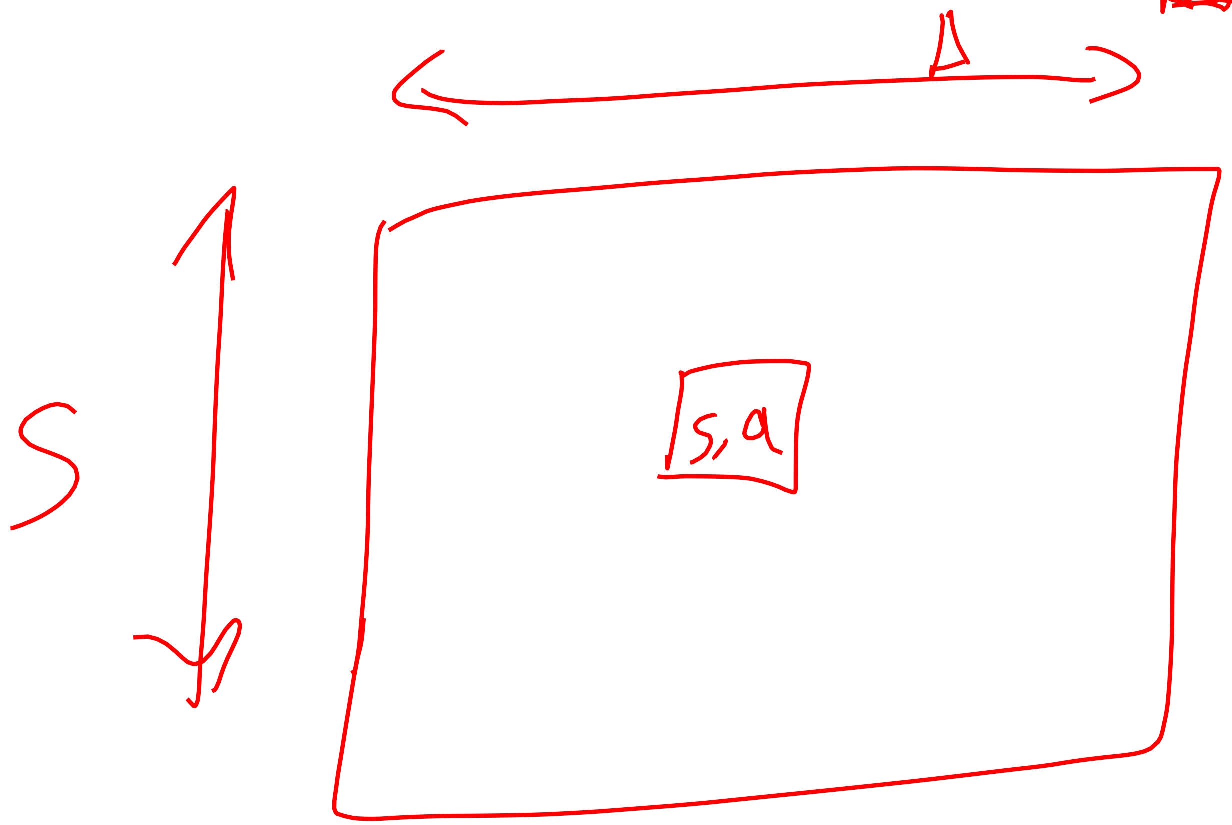
- Policy-based RL
  - Use neural nets to represent policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

- Model
  - Use neural nets to represent and learn the model

Approaches to RL

- Value-based RL
  - Estimate the optimal action-value function $Q^*(s, a)$



Optimal Value Function

- Optimal Q-function is the maximum achievable value

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

Optimal Value Function

- Optimal Q-function is the maximum achievable value

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

- Once we have it, we can act optimally

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

Optimal Value Function

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

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

- Optimal value maximizes over all future decisions

$$Q^*(s,a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots$$

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

Optimal Value Function

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

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

- Optimal value maximizes over all future decisions

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

$Q_{i+1}(s,a) \leftarrow Q_i(s,a)$

- Formally, Q* satisfies Bellman Equations

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

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q^* as i -> infinity

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q^* as i -> infinity

What's the problem with this?

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q^* as i -> infinity

What's the problem with this?
Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q^* as i -> infinity

What's the problem with this?
Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate Q(s,a). E.g. a neural network!

Demo

- http://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Deep RL

- Value-based RL
  - Use neural nets to represent Q function $Q(s, a; \theta)$

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

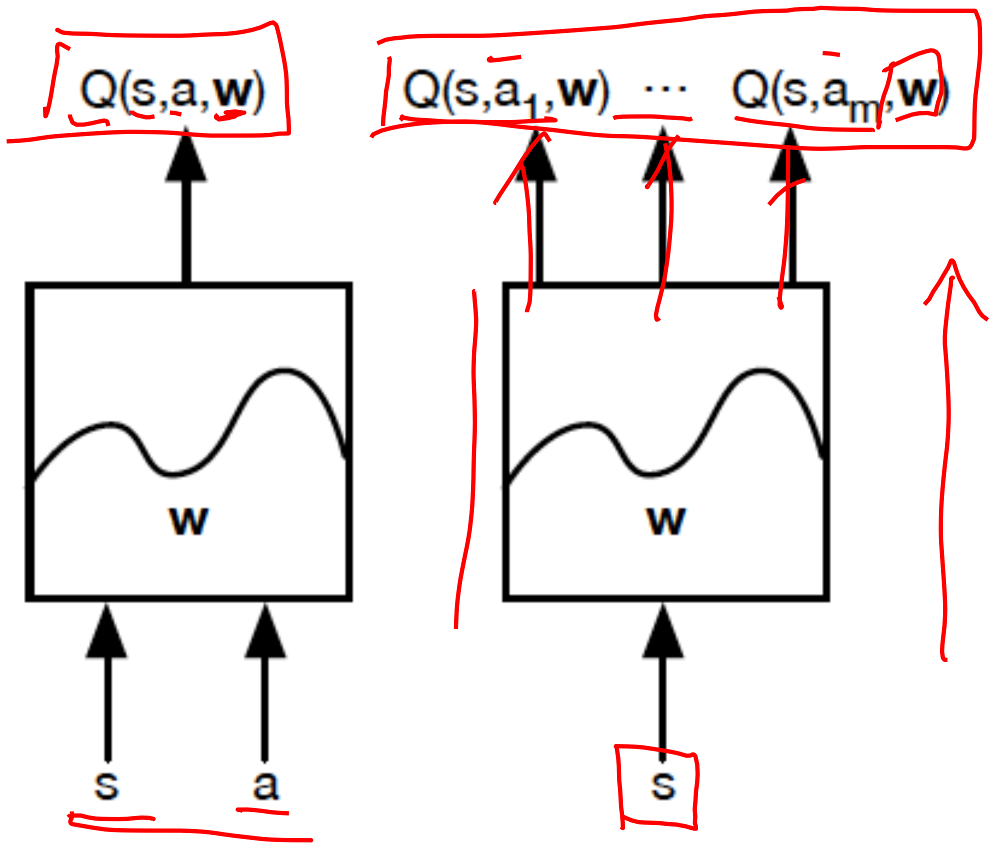
- Policy-based RL
  - Use neural nets to represent policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

- Model
  - Use neural nets to represent and learn the model

Q-Networks

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



Slide Credit: David Silver

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games

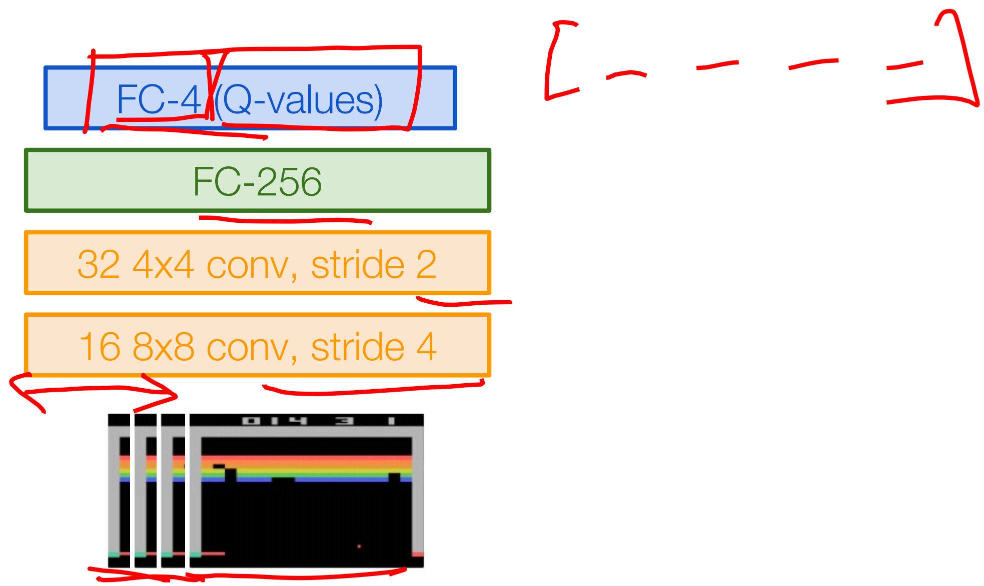
Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

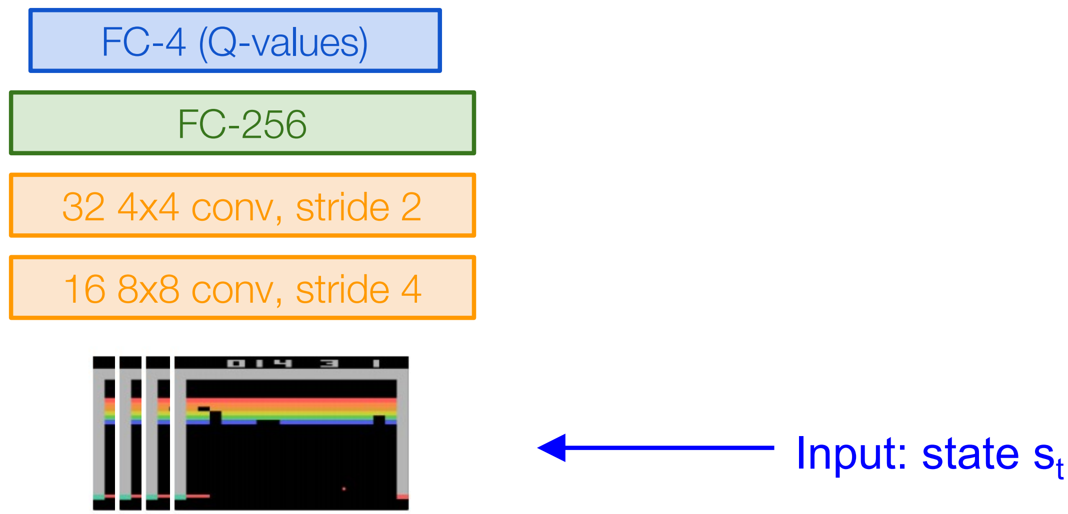


Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

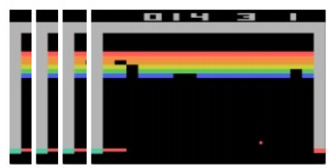
FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

← Familiar conv layers, FC layer

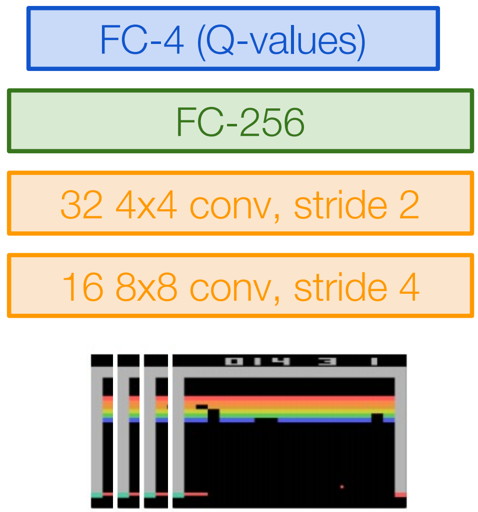


Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



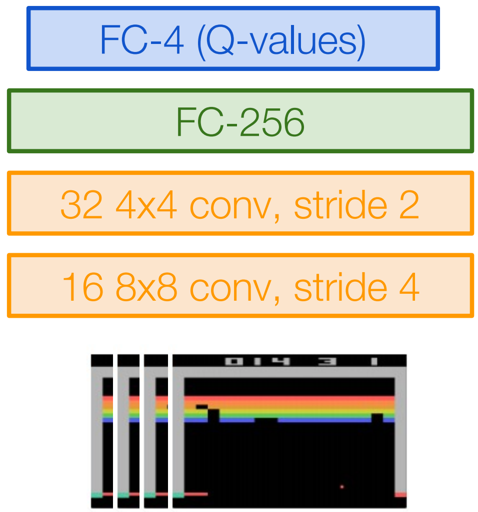
Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$: neural network with weights θ



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

Number of actions between 4-18 depending on Atari game

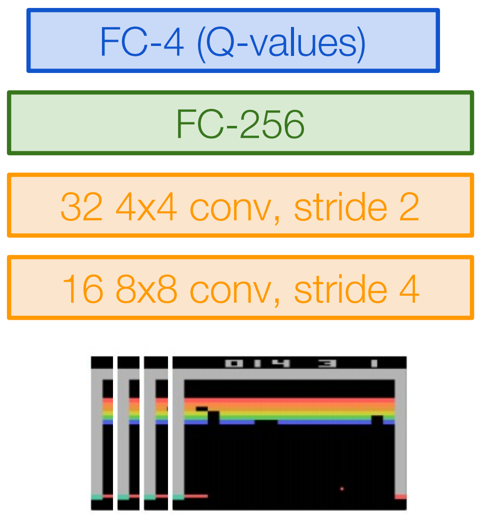
Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$: neural network with weights θ

A single feedforward pass to compute Q-values for all actions from the current state => efficient!



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i))^2\right]$

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i)^2\right]$

where $y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i))^2\right]$

where $y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Deep Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i)^2\right]$

where $y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$

Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy **π***)

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:
- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**
- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

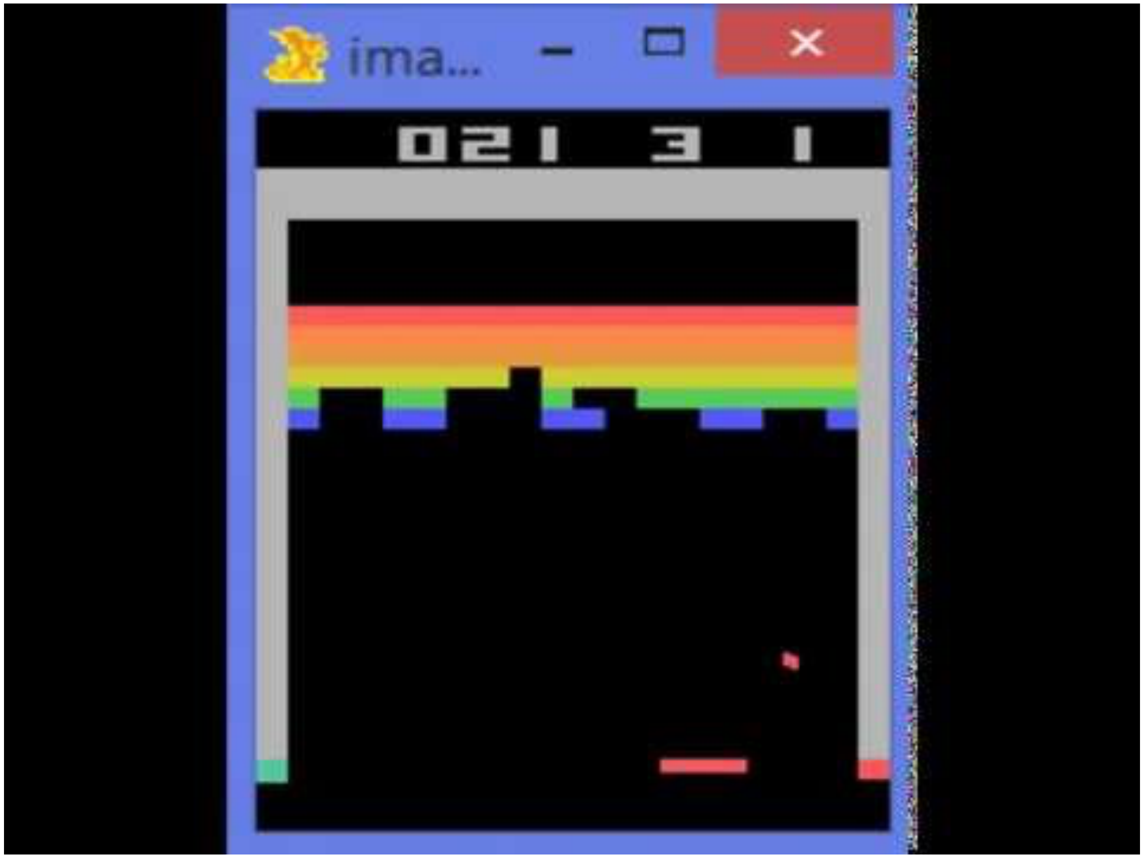
Experience Replay

To remove correlations, build data-set from agent's own experience

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

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$ (added to data-set)

(sampled from data-set) $\rightarrow s, a, r, s'$



https://www.youtube.com/watch?v=V1eYniJ0Rnk

Video by Károly Zsolnai-Fehér. Reproduced with permission.

Deep RL

- Value-based RL
  - Use neural nets to represent Q function $Q(s, a; \theta)$

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

- Policy-based RL
  - Use neural nets to represent policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

- Model
  - Use neural nets to represent and learn the model

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

$$\frac{\partial J}{\partial \theta}$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Gradient ascent on policy parameters!

REINFORCE algorithm

Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

REINFORCE algorithm

Expected reward:
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau;\theta) \mathrm{d}\tau$$

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Now let's differentiate this: $$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

Intractable! Expectation of gradient is problematic when p depends on θ

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Now let's differentiate this: $$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

Intractable! Expectation of gradient is problematic when p depends on θ

However, we can use a nice trick: $$\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$$
$$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

Intractable! Expectation of gradient is problematic when p depends on θ

However, we can use a nice trick: $$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta) \right) p(\tau;\theta) \mathrm{d}\tau$$
$$= \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau) \nabla_\theta \log p(\tau;\theta)]$$

Can estimate with Monte Carlo sampling

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

1 sample

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Intuition

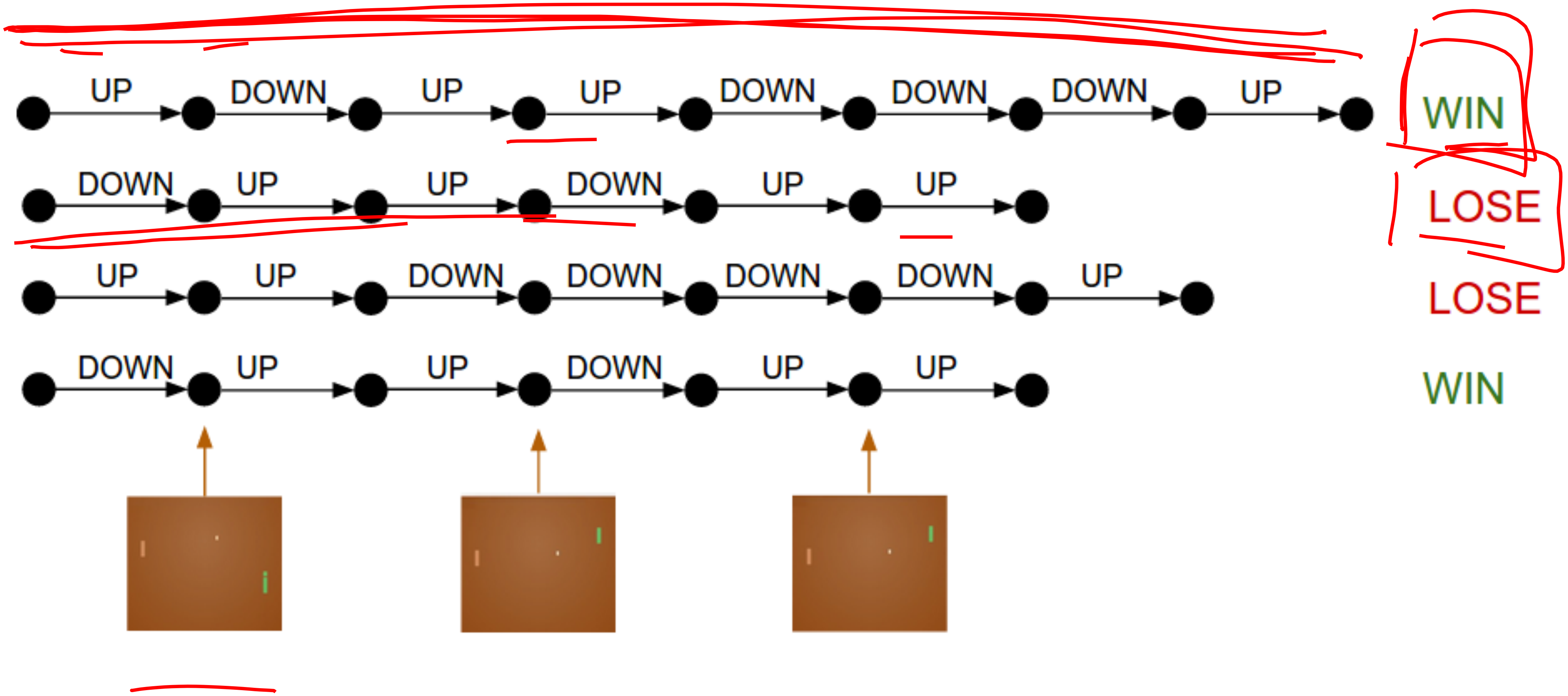
Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

Intuition


UP
DOWN
UP
UP
DOWN
DOWN
DOWN
UP
WIN
DOWN
UP
UP
DOWN
UP
UP
LOSE
UP
UP
DOWN
DOWN
DOWN
DOWN
UP
LOSE
DOWN
UP
UP
DOWN
UP
UP
WIN

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

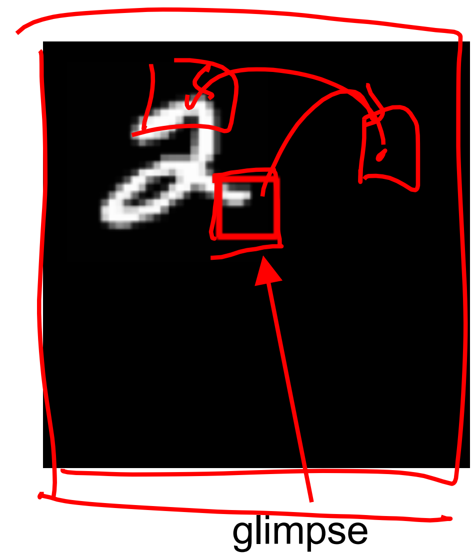
REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise

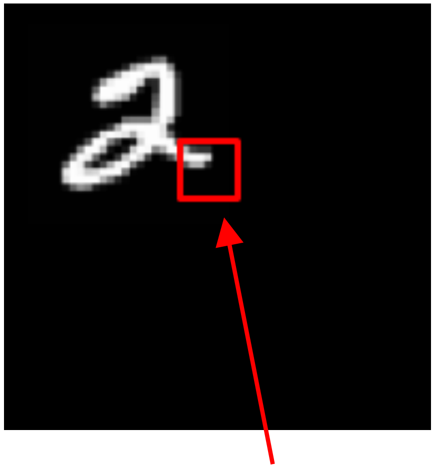


[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class
- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

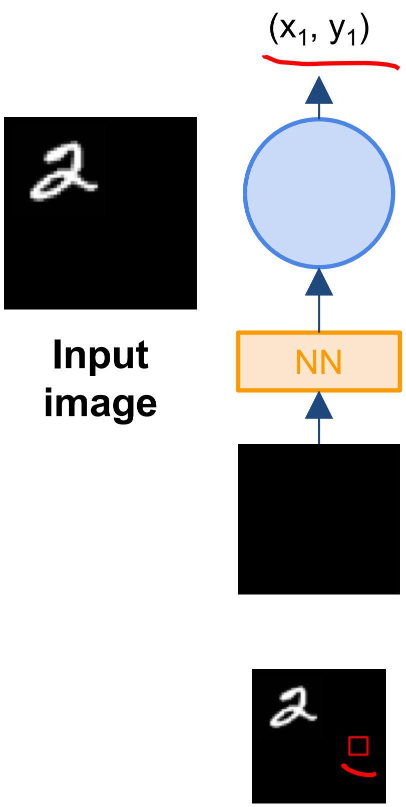


State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise

Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

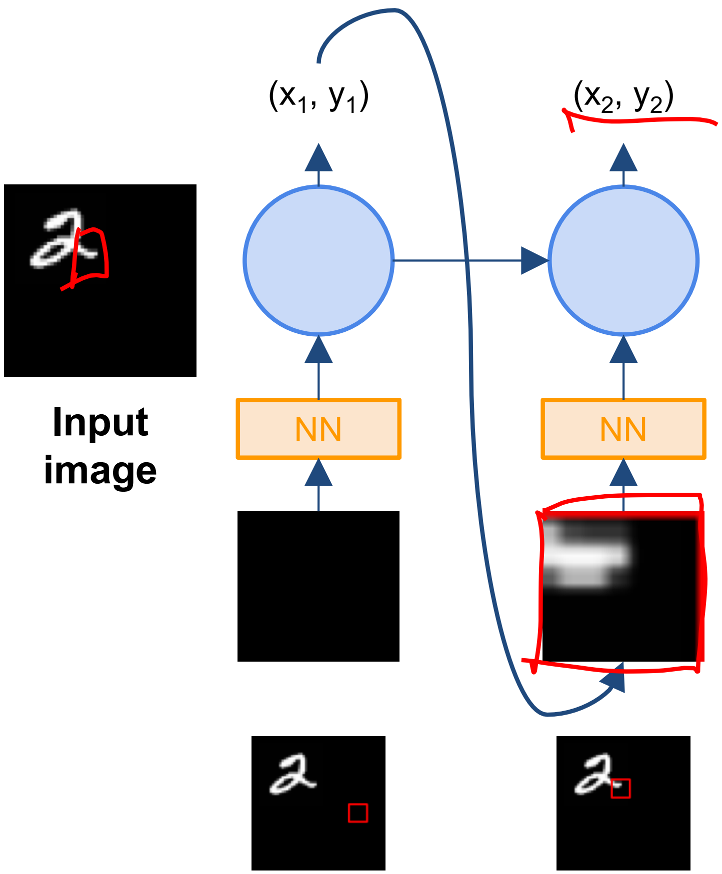
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



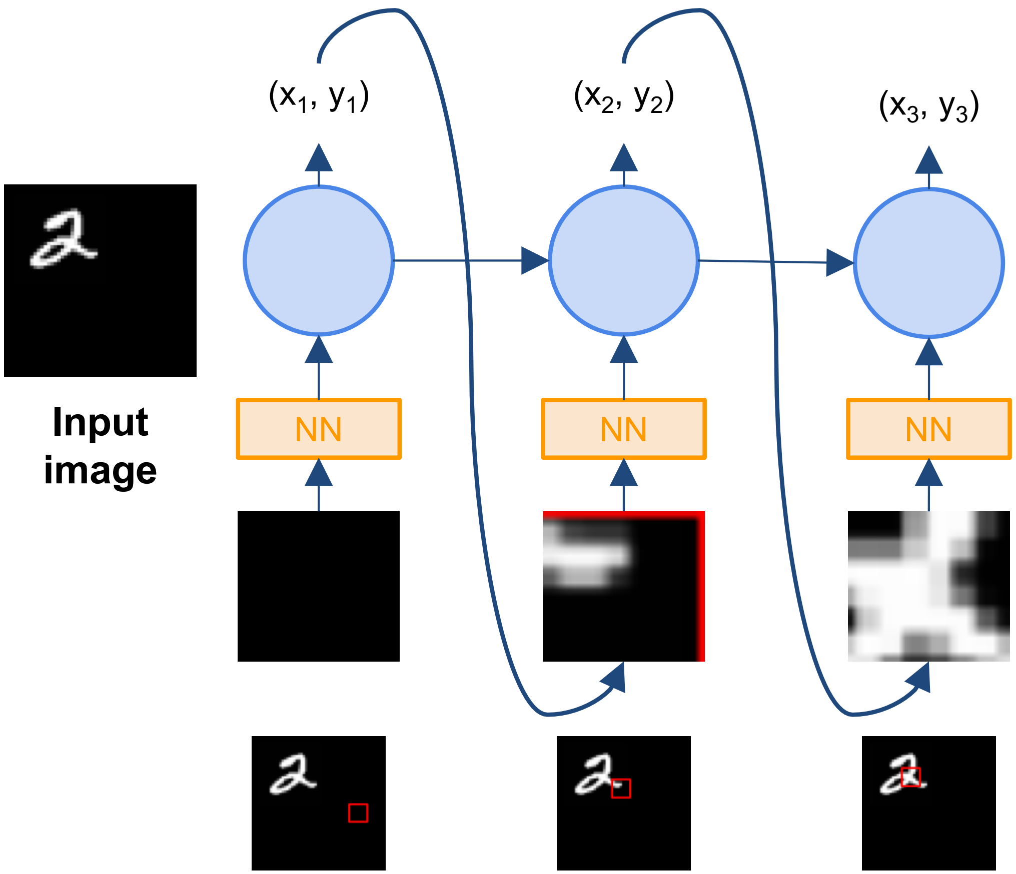
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



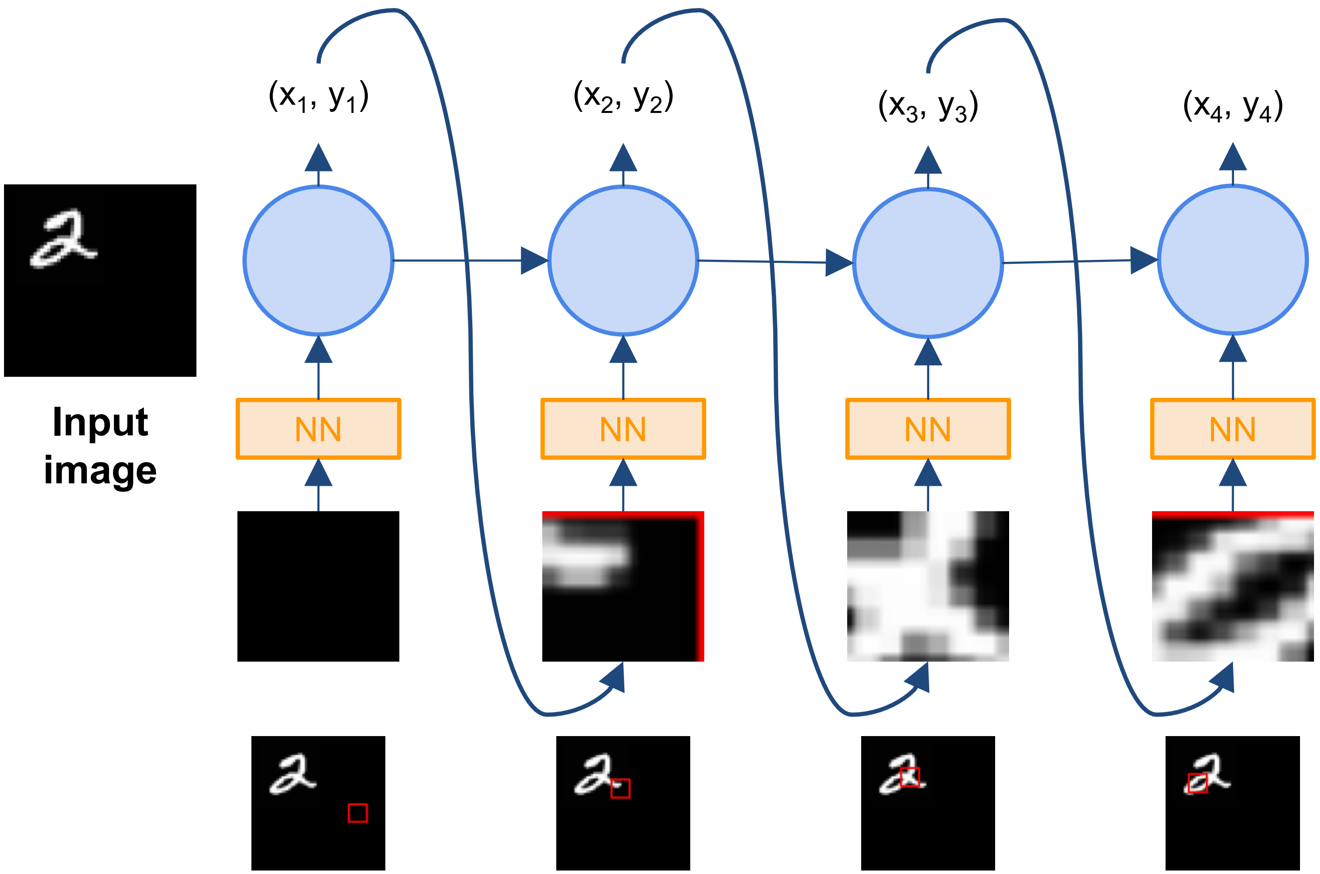
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



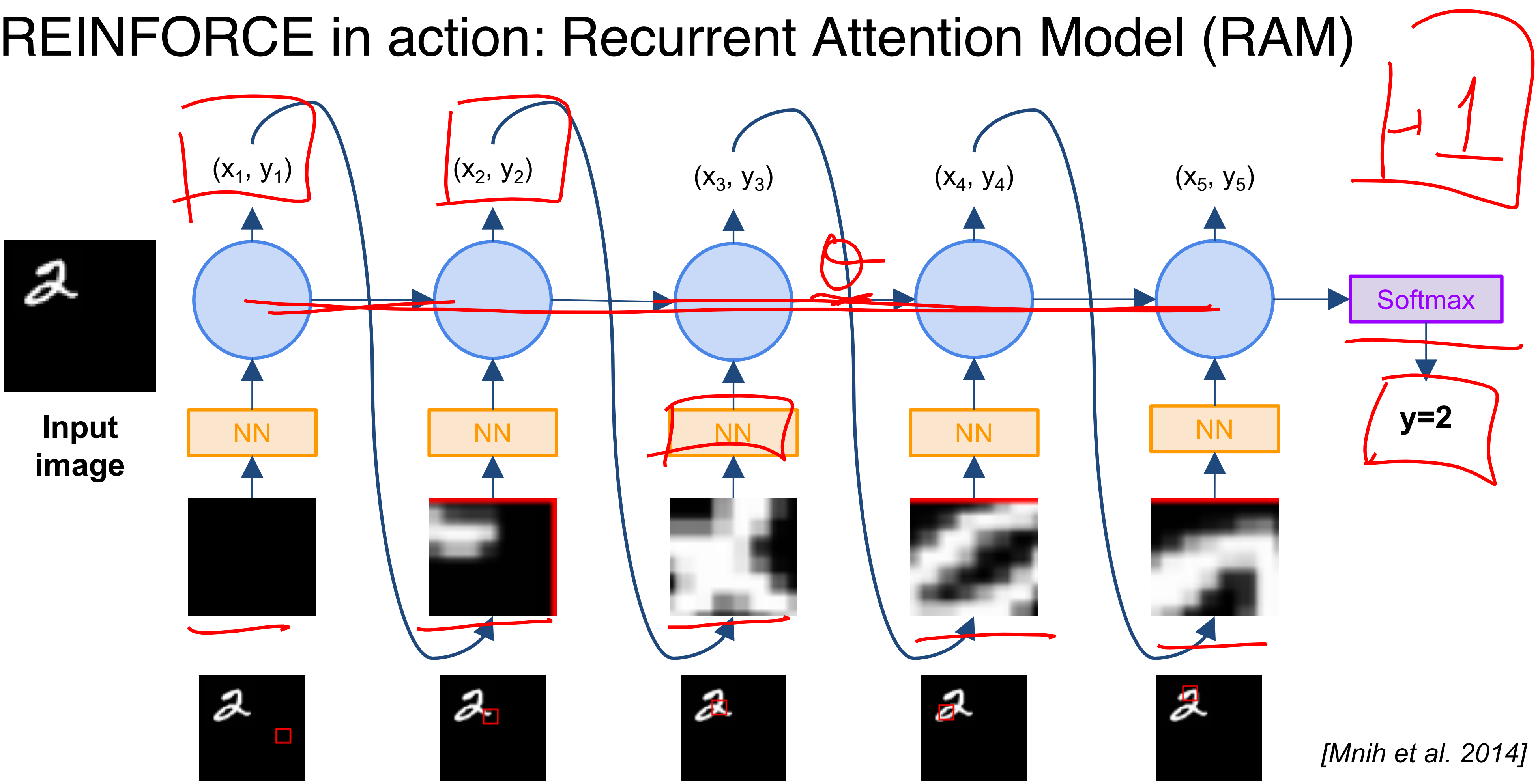
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



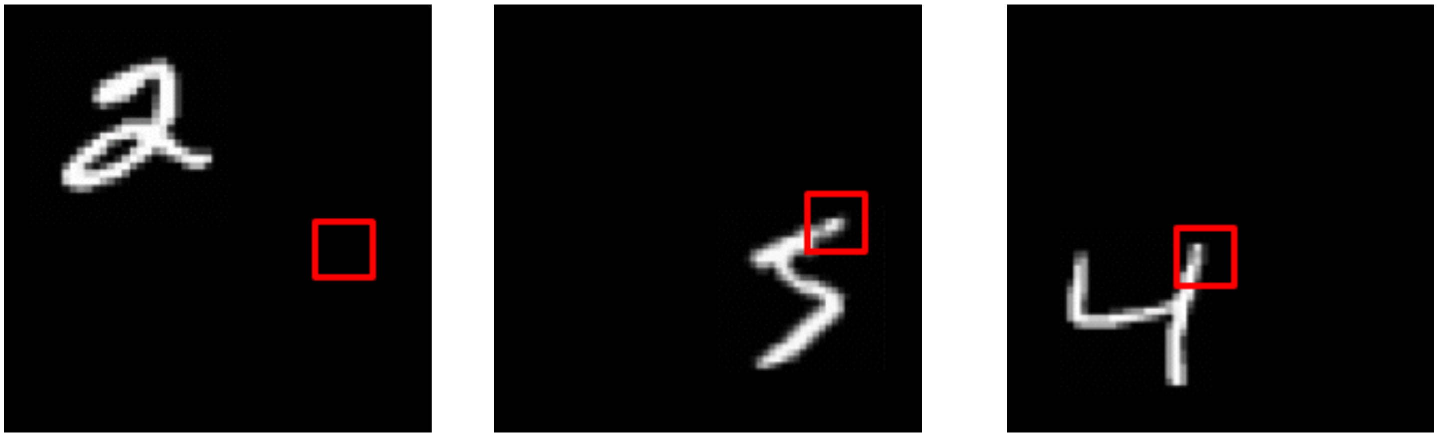
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih et al. 2014]

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in "Vanilla REINFORCE"

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

End for

Summary

- **Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency
- **Q-learning**: does not always work but when it works, usually more sample-efficient. **Challenge**: exploration

- Guarantees:
  - **Policy Gradients**: Converges to a local minima of $J(\theta)$, often good enough!
  - **Q-learning**: Zero guarantees since you are approximating Bellman equation with a complicated function approximator