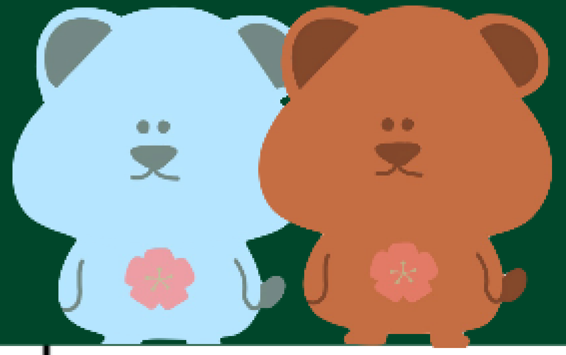


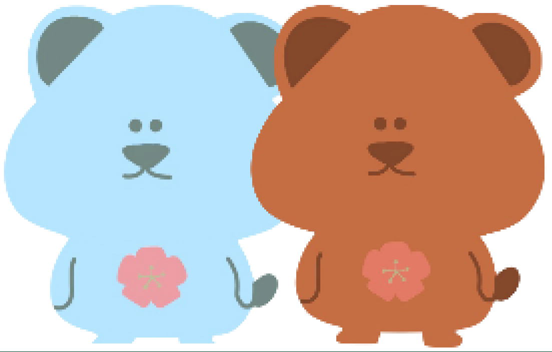
Deep Reinforcement Learning

Week16 구미진, 민소연

Index

EWHA EURON

Reinforcement Learning

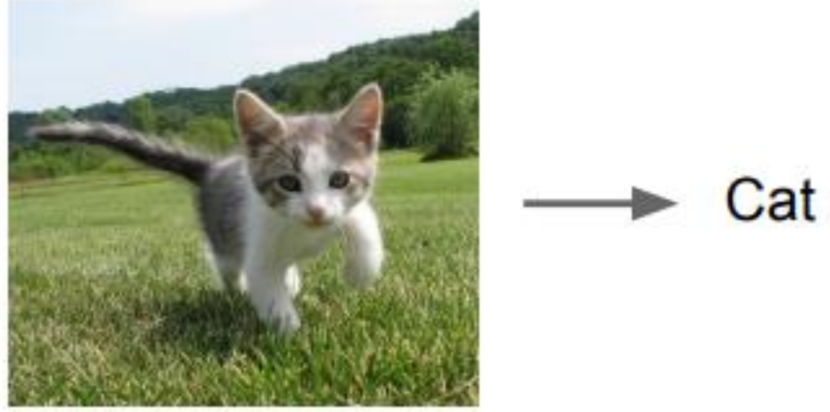


01 Supervised vs Unsupervised Learning

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.



Classification

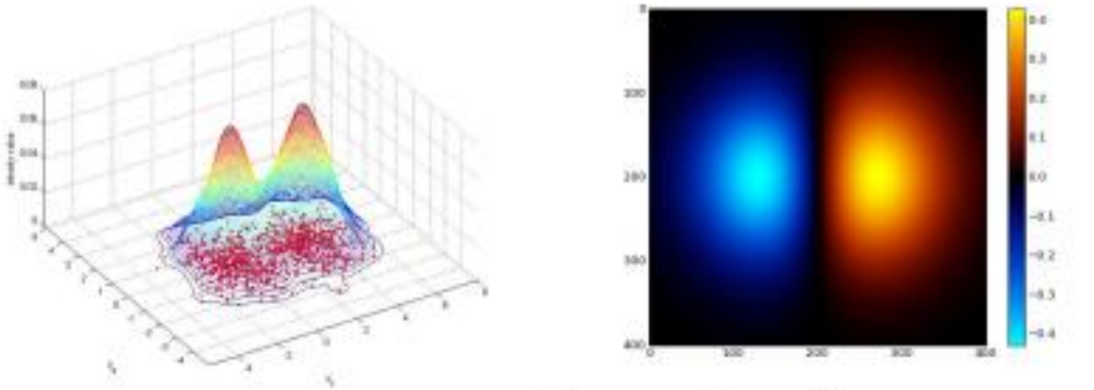
Data: x
Just data, no labels!

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

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

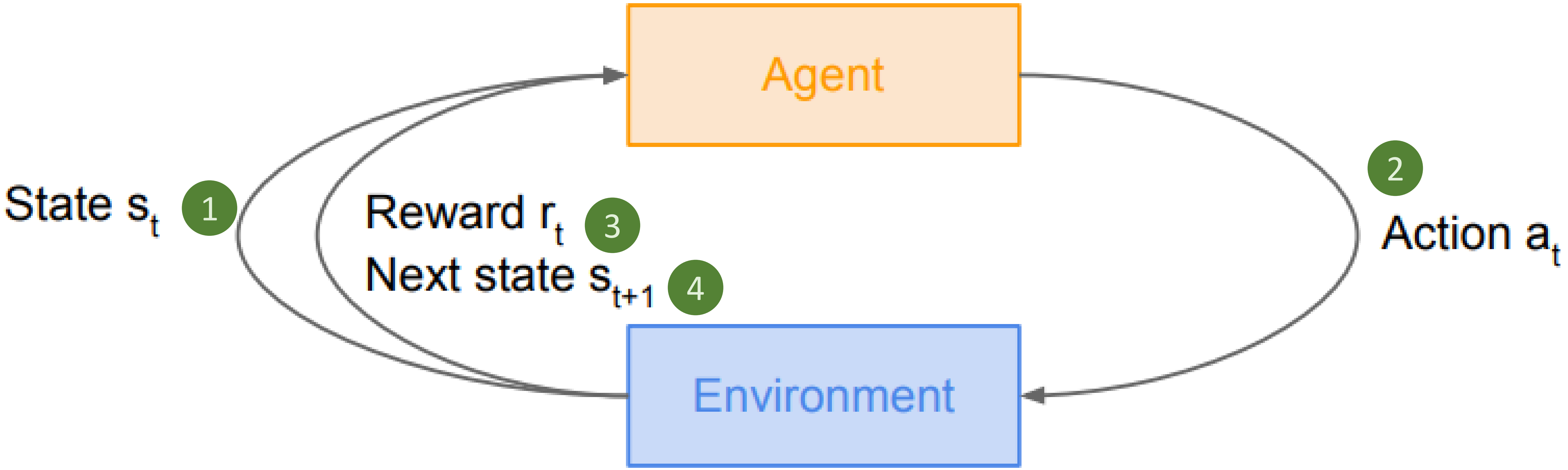
Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation

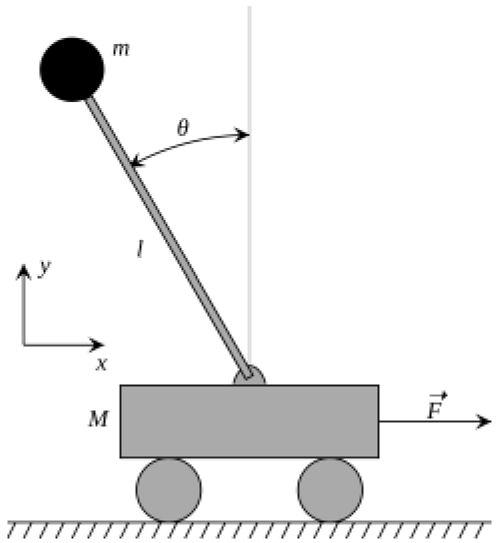


2-d density estimation

02 Reinforcement Learning



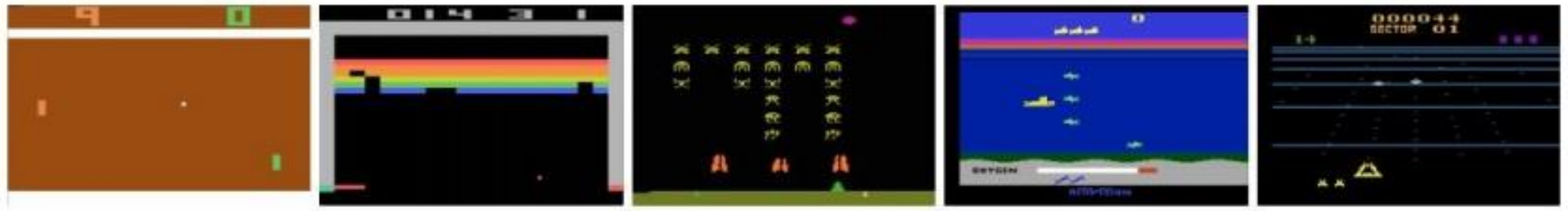
03 Examples



Objective: Balance a pole on top of a movable cart

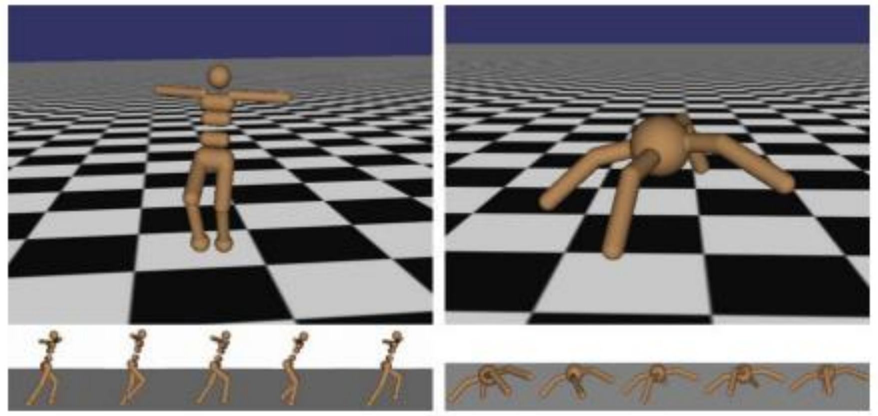
State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Atari Games



Objective: Complete the game with the highest score

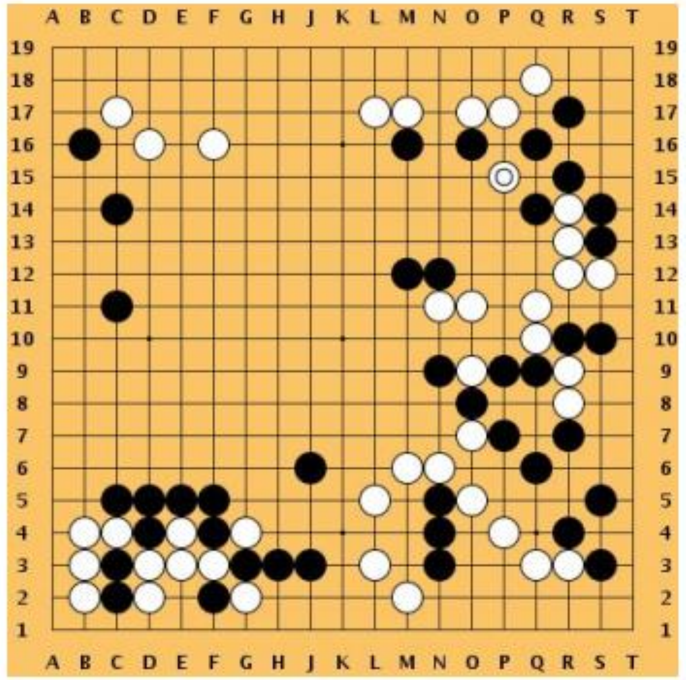
State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step



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

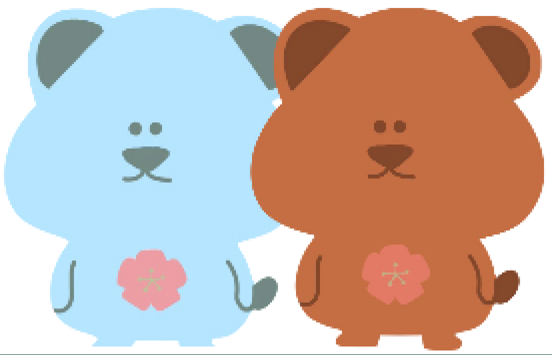
Go



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

Markov Decision Process



EWHA EURON

Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

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

- A policy **π** is a function from S to A that specifies what action to take in each state
- **Objective**: find policy **π*** that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

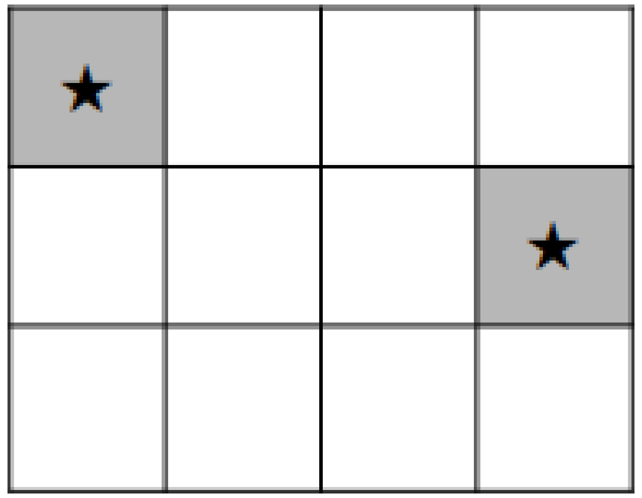
02 A simple MDP

actions = {

1. right
2. left
3. up
4. down

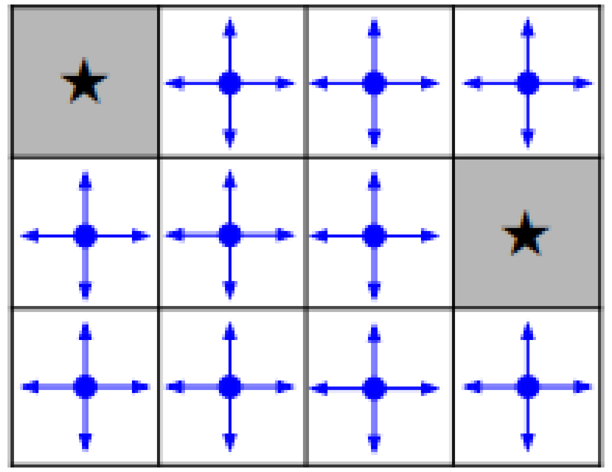
}

states

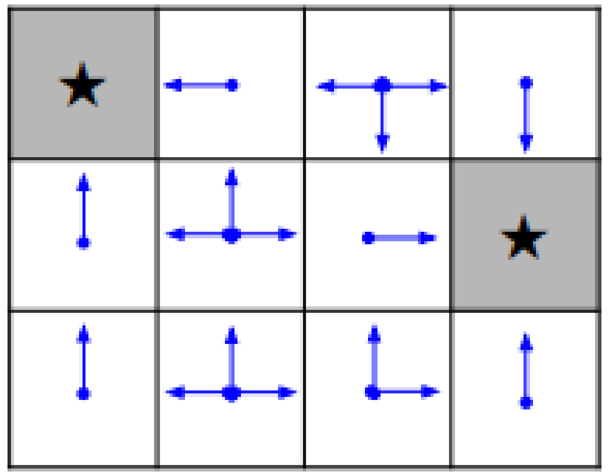


Set a negative "reward"
for each transition
(e.g. $r = -1$)

Objective: reach one of terminal states (greyed out) in least number of actions



Random Policy



Optimal Policy

03 How to Find Optimal Policy

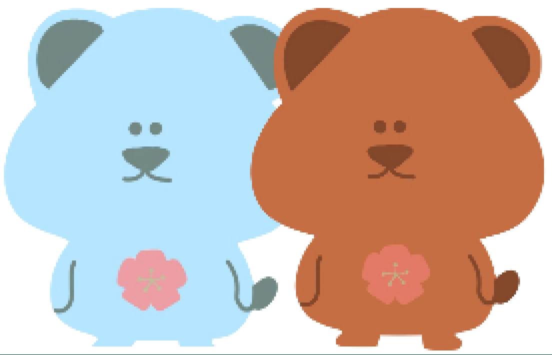
The optimal policy **π***

We want to find optimal policy **π*** that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?
Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Q-Learning



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 following the policy from state s:

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

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 then following the policy:

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

EWHA EURON

Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

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

Q* satisfies the following **Bellman equation**:

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

Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

The optimal policy **π*** corresponds to taking the best action in any state as specified by Q*

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!

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

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

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

04 Q-Learning

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

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

Forward Pass

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

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$

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):

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

Q-network Architecture

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

FC-4 (Q-values)

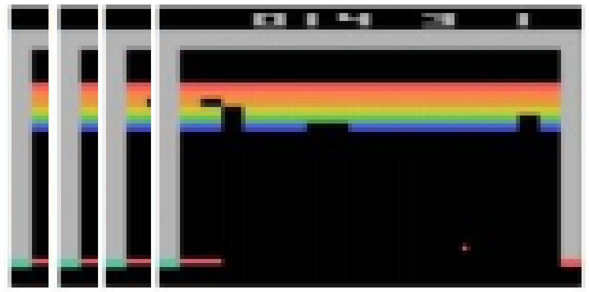
FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

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

Familiar conv layers, FC layer



Input: State St

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

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

Each transition can also contribute to multiple weight updates
=> greater data efficiency

EWHA EURON

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Initialize replay memory, Q-network

Play M episodes (full games)

Initialize state (starting game screen pixels) at the beginning of each episode

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

For each timestep t of the game

With small probability, select a random action(explore), otherwise select greedy action from current policy

Take the action (a_t), and observe the reward r_t and next state s_{t+1}

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Store transition in replay memory

Experience Replay:
Sample a random minibatch of transitions from replay memory and perform a gradient descent step

Policy Gradients

01 Policy Gradients

Q-learning의 문제점

- Q-function이 아주 복잡함
- 모든 (state, action) 쌍들을 학습해야 함
- 로봇이 어떤 물체를 손으로 잡는 문제를 해결 한다고 했을 때는?

-> 로봇의 모든 관절의 위치와 각도가 이룰 수 있는 모든 경우의 수에 대해
모든 (state, action)을 학습시켜야 함

01 Policy Gradients

Q-learning의 문제점

- Q-function이 아주 복잡함
- 모든 (state, action) 쌍들을 학습해야 함
- 로봇이 어떤 물체를 손으로 잡는 문제를 해결 한다고 했을 때는?

-> 로봇의 모든 관절의 위치와 각도가 이룰 수 있는 모든 경우의 수에 대해
모든 (state, action)을 학습시켜야 함
-> 모든 (state, action)을 학습시키는 대신, **정책** 자체를 학습시키는 방법!

"Policy Gradients"

01 Policy Gradients

매개변수화된 정책들의 집합
(가중치 θ)

$$\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

미래에 받을 보상들의 누적 합의 기댓값

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

J(θ)를 최대로 만드는 최적의 정책 θ*

$$\theta^* = \arg\max_\theta J(\theta)$$

02 REINFORCE algorithm

경로에 대한 미래 보상의 기댓값

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

미분

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

(계산 불가능)

Monte carlo sampling

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

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

02 REINFORCE algorithm

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

log

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

미분

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

전이 확률을 몰라도 J(θ) 미분 값 계산 가능!

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

02 REINFORCE algorithm

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

어떤 경로로부터 얻은 보상 r(τ)이 크다면, 그 행동들을 할 확률이 높아짐
어떤 경로로부터 얻은 보상 r(τ)이 작다면, 그 행동들을 할 확률이 낮아짐

어떤 경로가 좋다는 것은 그 경로에 포함되는 모든 행동이 좋았다는 것을 의미
-> 기댓값에 의해서 모두 averages out 됨
-> 구체적으로 어떤 행동이 좋은지 알 수 없음

02 REINFORCE algorithm

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

어떤 경로로부터 얻은 보상 $r(\tau)$이 크다면, 그 행동들을 할 확률이 높아짐
어떤 경로로부터 얻은 보상 $r(\tau)$이 작다면, 그 행동들을 할 확률이 낮아짐

어떤 경로가 좋다는 것은 그 경로에 포함되는 모든 행동이 좋았다는 것을 의미
-> 기댓값에 의해서 모두 averages out 됨
-> 구체적으로 어떤 행동이 좋은지 알 수 없음

문제점: 높은 분산(high variance)
-> 분산을 낮추고 충분한 샘플링을 통해 estimator의 성능을 높여야 함

03 Variance reduction

분산을 줄이는 방법

1. 특정 상태로부터 받을 미래 보상만을 고려하여 어떤 행동을 취할 확률을 키우는 방법

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

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

2. 지연된 보상에 대해 할인률을 적용하는 방법

Second idea: Use discount factor γ to ignore delayed effects

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

03 Variance reduction

3. Baseline
중요한 것은 실제로 얻은 보상이 얻을 것이라고 예상했던 것보다 좋은지 아닌지를 판단하는 것
-> Baseline function을 사용, 상태를 이용하는 방법!

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

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

Baseline function

- 해당 상태에서 얼마만큼의 보상을 원하는지 설명해주는 함수

03 Variance reduction

Baseline을 선택하는 방법

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

1. 단순한 baseline

- 지금까지 경험했던 보상들에 대해서 moving average를 계산하는 방법
- 이런 variance reduction 방법이 "vanilla REINFORCE"
- 할인율을 적용, 미래에 받을 보상의 누적합을 계산, 단순한 baseline 추가

03 Variance reduction

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

2. 더 좋은 baseline

- 우리는 어떤 행동이 그 상태에서의 기댓값보다 좋은 경우에 그 행동을 수행할 확률이 커지기를 원함
  -> 여기서 Q-function과 value function을 이용

$Q^\pi(s_t, a_t) - V^\pi(s_t)$ 이 클수록 현재 행동이 좋음을 의미

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

03 Variance reduction

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

2. 더 좋은 baseline

- 우리는 어떤 행동이 그 상태에서의 기댓값보다 좋은 경우에 그 행동을 수행할 확률이 커지기를 원함
  -> 여기서 Q-function과 value function을 이용

$Q^\pi(s_t, a_t) - V^\pi(s_t)$ 이 클수록 현재 행동이 좋음을 의미

이때 구체적인 Q-function과 Value function을 몰라도 됨
Q-learning과 Policy gradient를 이용!

04 Actor-Critic Algorithm

- Q-function과 Value function을 몰라도 Q-learning과 Policy gradient를 조합해서 training 시킬 수 있음
- Actor(Policy) – 우리가 어떤 행동을 할지 결정
- Critic(Q-function) – 그 행동이 얼마나 좋았는지, 또 어떻게 조절해야 하는지 알려줌
- Advantage function
  - 그 행동이 예상했던 것보다 얼마나 더 큰 보상을 주는지 알려주는 함수

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

04 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

1. θ, ϕ를 초기화 시킨다.
2. 현재의 policy를 기반으로 M개의 경로를 샘플링한다.
3. Gradient를 계산한다. 각 경로마다 보상 함수를 계산하고 이용한다.
4. 보상함수를 이용해서 gradient estimator를 계산하고 이를 전부 누적시킨다.
5. ϕ를 학습시키기 위해 가치 함수를 학습시킨다. 이는 보상 함수를 최소화시키는 것과 동일하므로, 가치 함수가 벨만 방정식(Bellman equation)에 근접하도록 학습시킨다.
6. 앞선 단계를 계속 반복한다.

05 RAM

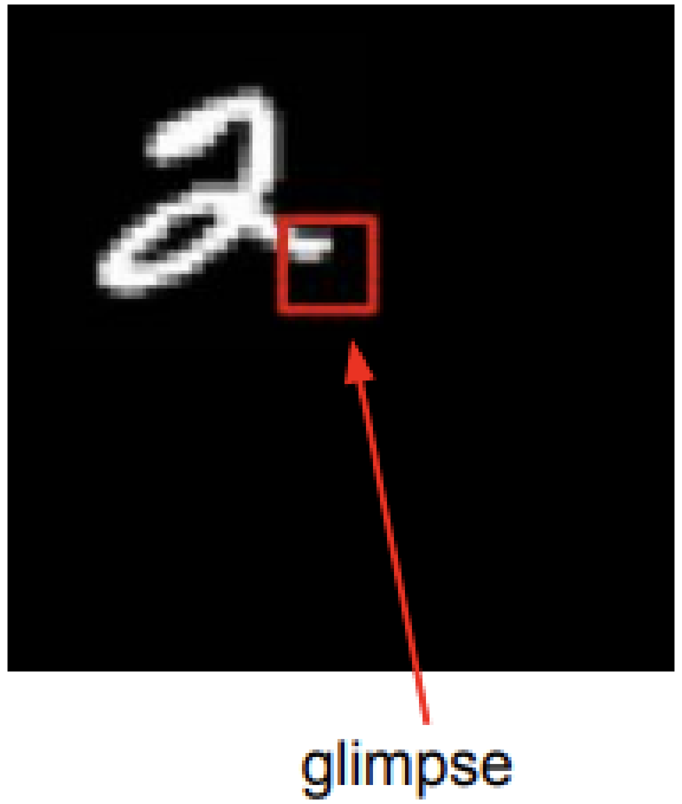
Recurrent Attention Model(RAM)

- Hard attention 기법
- Image classification task에서 이미지의 glimpse만 가지고 예측
- 이미지 전체가 아닌 지역적인 부분만을 봄

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



05 RAM

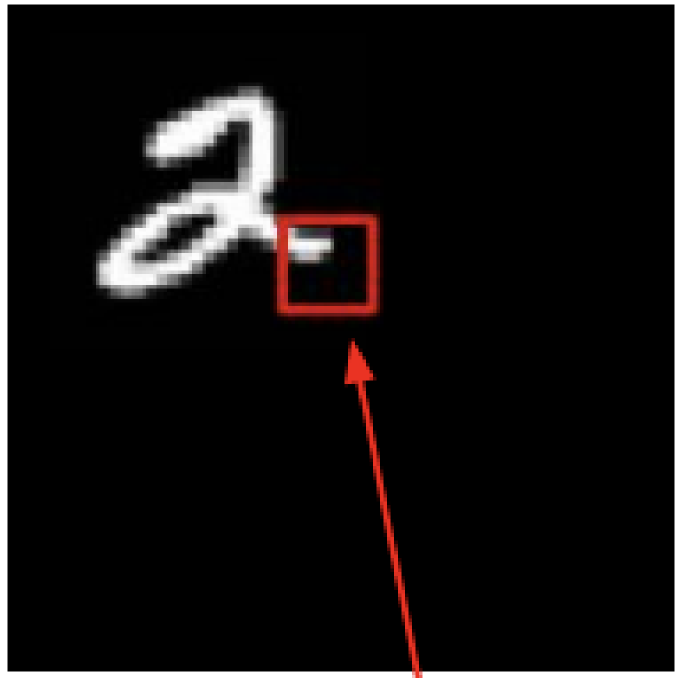
Recurrent Attention Model(RAM)

- Hard attention 기법
- Image classification task에서 이미지의 glimpse만 가지고 예측
- 이미지 전체가 아닌 지역적인 부분만을 봄

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



glimpse

이 문제를 강화학습으로 풀어보자!

05 RAM

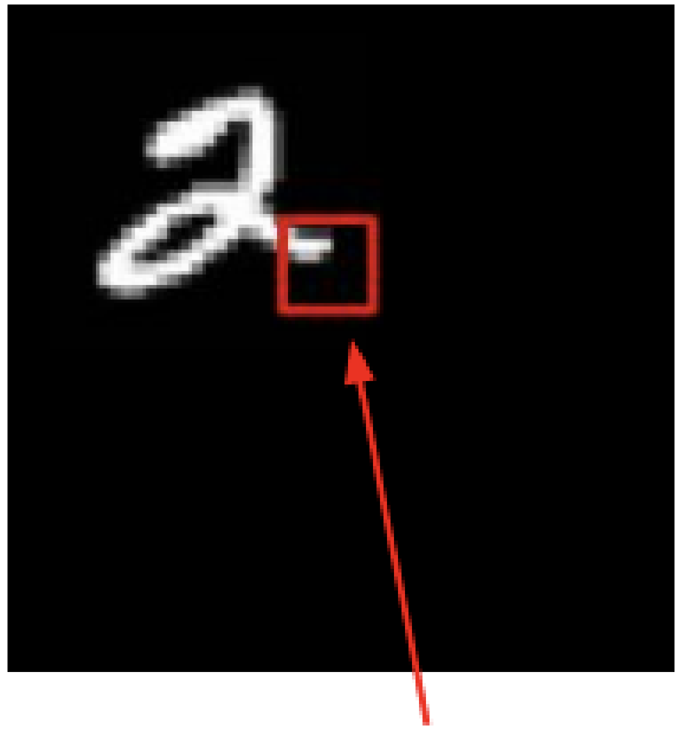
Recurrent Attention Model(RAM)

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



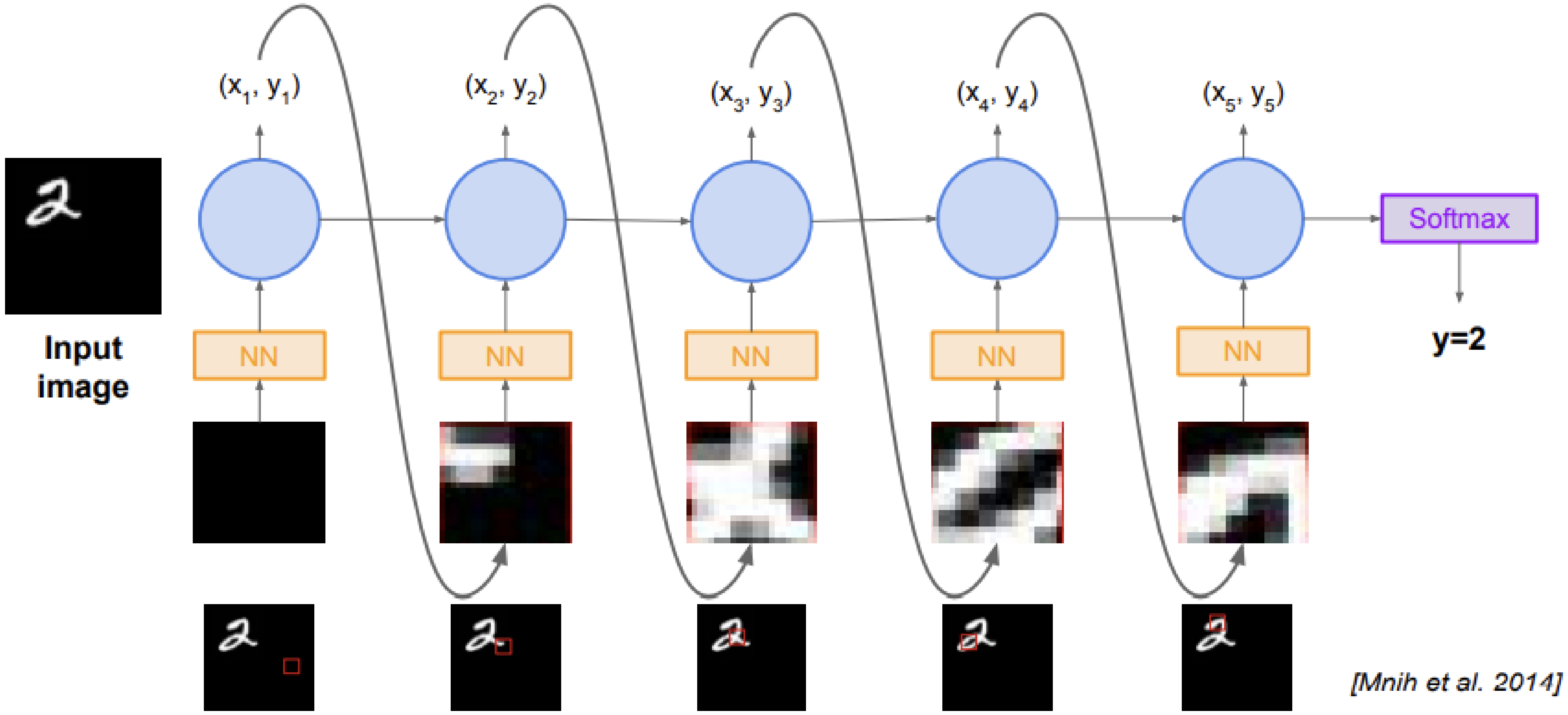
- 상태는 지금까지 관찰한 glimpses
- 행동은 다음에 어떤 부분을 볼 것인지를 결정하는 것
- 보상은 classification의 성공 유무

  -> 어떻게 glimpses를 얻어낼 것인지를 정책을 통해 학습!

05 RAM

상태를 모델링하기 위해 RNN을 이용

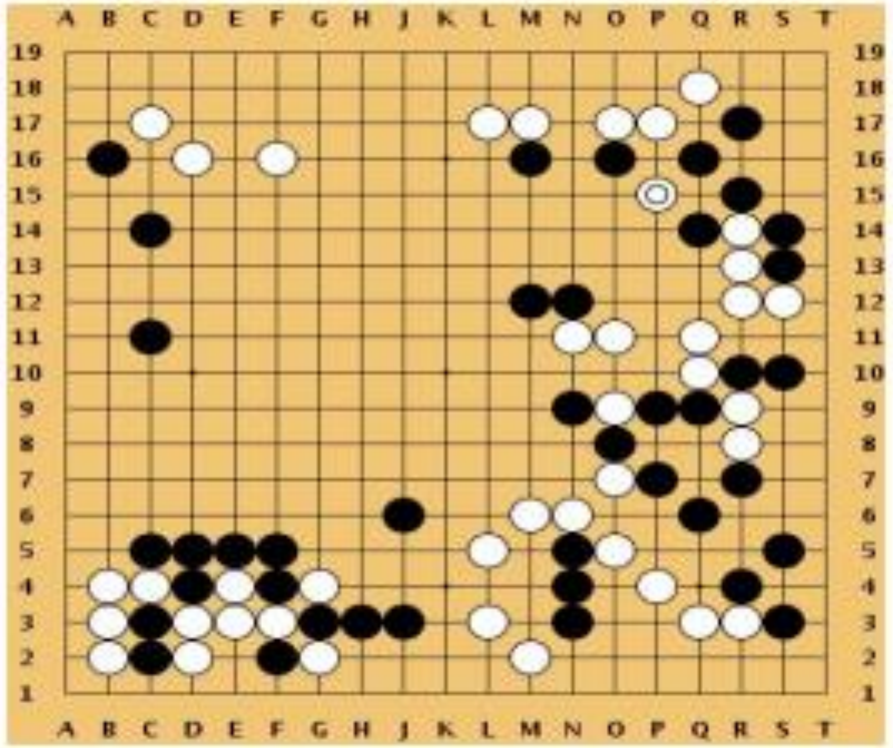
REINFORCE in action: Recurrent Attention Model (RAM)



[Mnih et al. 2014]

06 AlphaGo

More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016]

THANK YOU