CRIPAC 智能感知与计算研究中心 Center for Research on Intelligent Perception and Computing

中国科学院自动化研究所 Institute of Automation Chinese Academy of Sciences

2018

"Deep Learning Lecture"

Lecture 10: Reinforcement Learning

Wei Wang

Center for Research on Intelligent Perception and Computing (CRIPAC)

National Laboratory of Pattern Recognition (NLPR)

Institute of Automation, Chinese Academy of Science (CASIA)

Outline

1. Course Review
2. Background
3. Model-free learning
4. Model-based learning
5. Applications

GAN's Formulation

$$\min_G \max_D V(D,G)$$

- It is formulated as a **minimax game**, where:
  - The Discriminator is trying to maximize its reward $\boldsymbol{V(D,G)}$
  - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

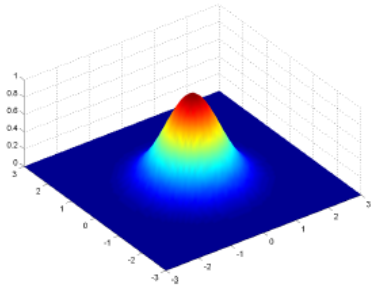
$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}[\log(1 - D(G(z)))]$$

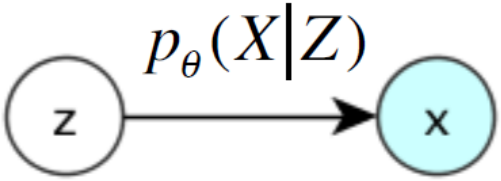
- The Nash equilibrium of this particular game is achieved at:
  - $P_{data}(x) = P_{gen}(x) \;\; \forall x$
  - $\mathrm{D}(x) = \frac{1}{2} \;\; \forall x$

Principle Idea decoder network

- We have a set of N-observations (e.g. images) $\{x^{(1)},x^{(2)},\ldots,x^{(N)}\}$
- Complex model parameterized with θ
- There is a latent space z with

$$z \sim p(z) \quad \text{multivariate Gaussian}$$
$$x|z \sim p_\theta(x|z)$$





One Example

Wish to learn θ from the N training observations $x^{(i)}$ i=1,…,N

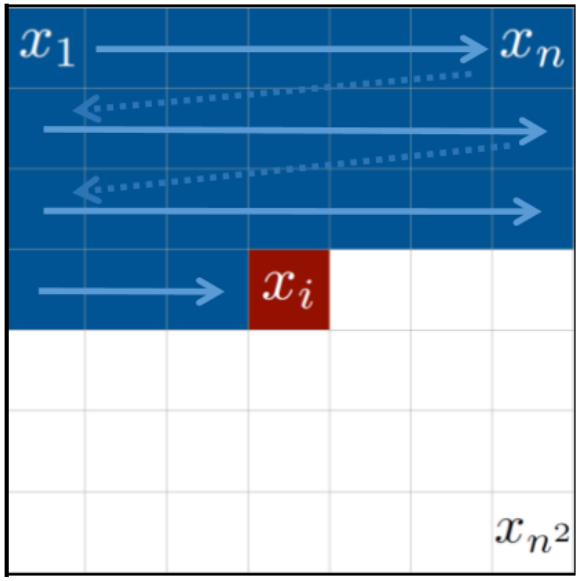
Pixel RNN

$$p(\mathbf{x}) = p(x_1, x_2, ..., x_{n^2})$$

Bayes Theorem:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

A sequential model!



Pixel recurrent neural networks, ICML 2016

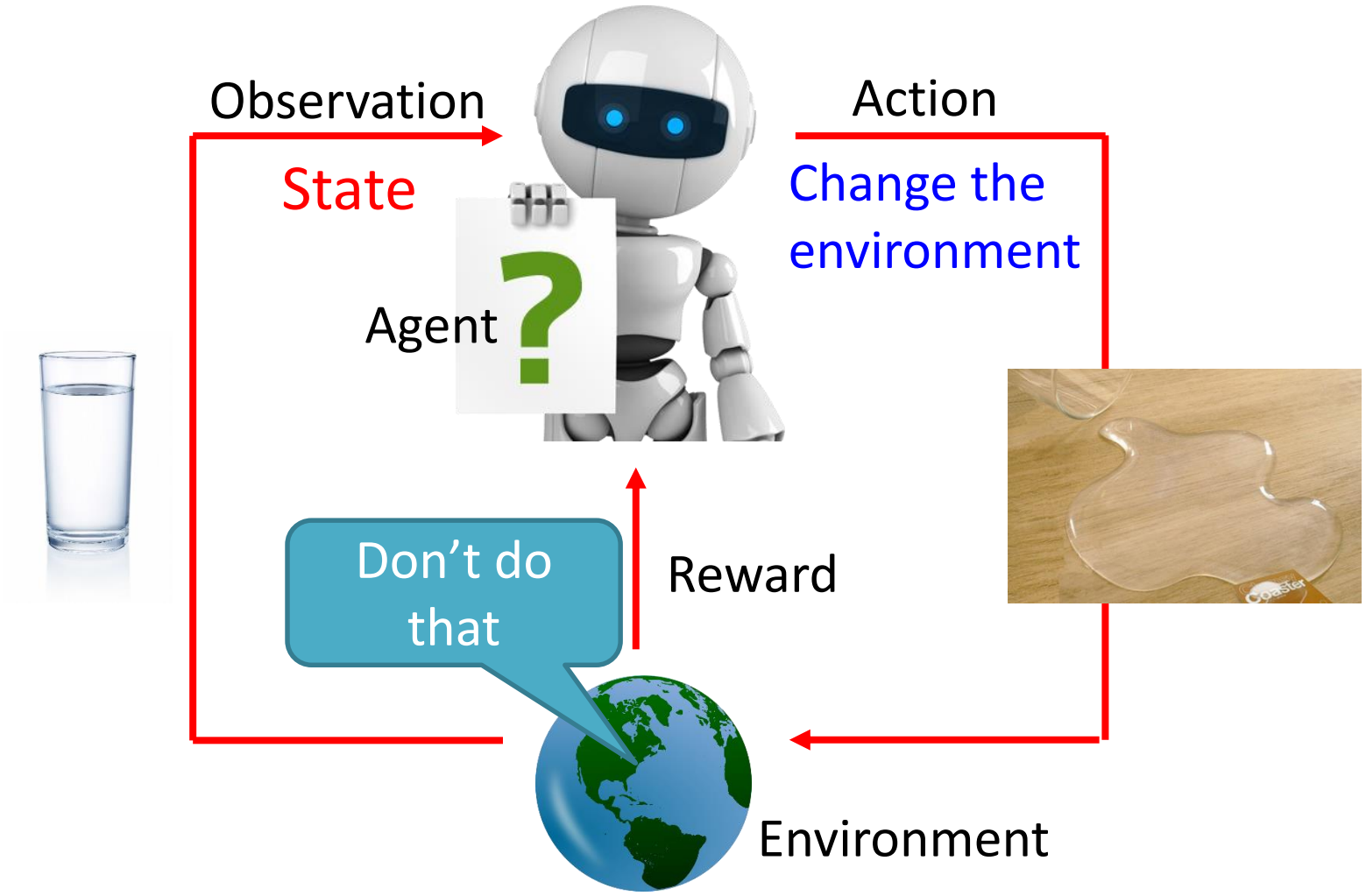
Outline

1. Course Review
2. Background
3. Model-free learning
4. Model-based learning
5. Applications

Background

Deep Reinforcement Learning: AI = RL + DL

Scenario of Reinforcement Learning

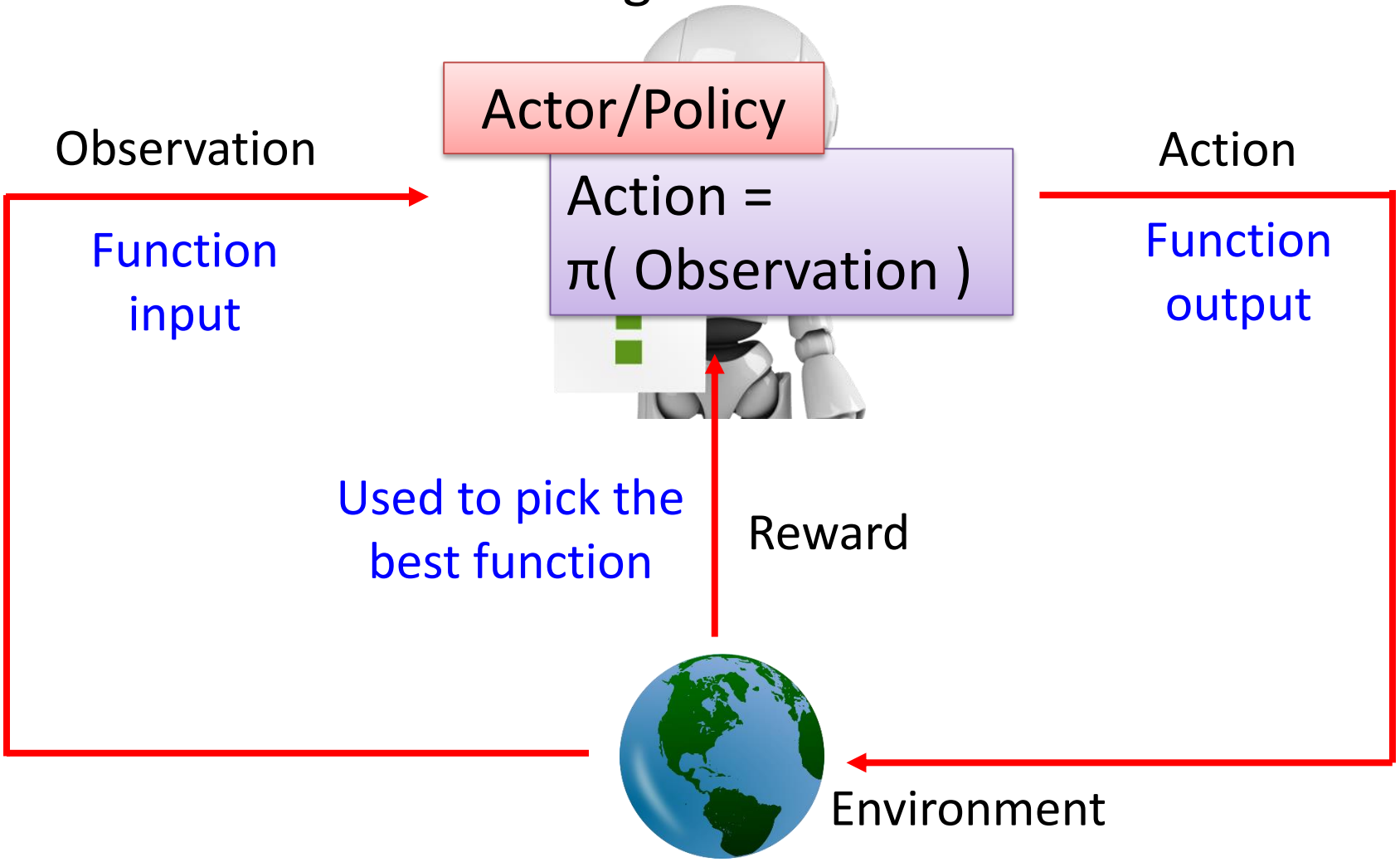


Scenario of Reinforcement Learning

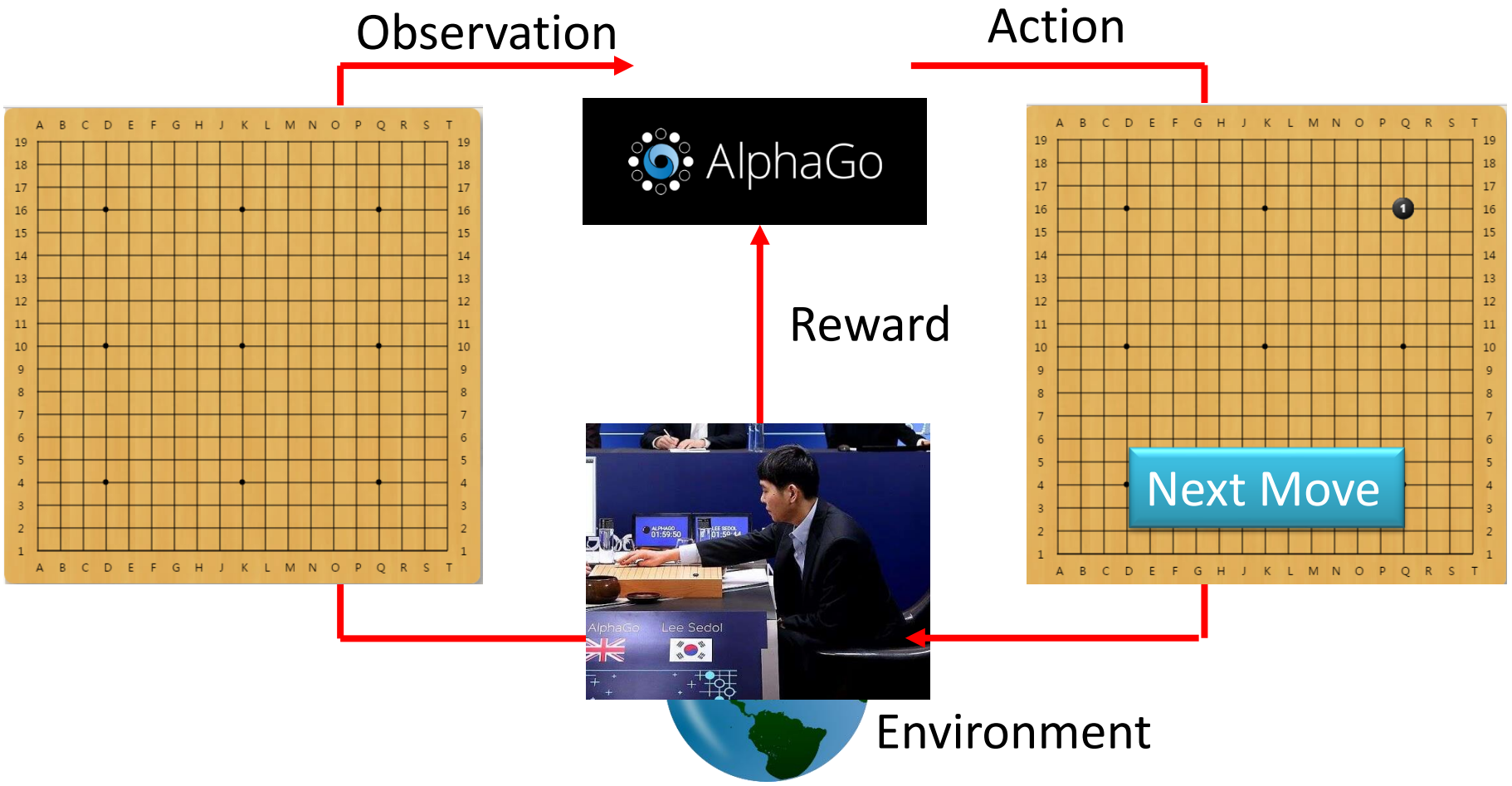
https://yoast.com/how-to-clean-site-structure/

Scenario of Reinforcement Learning

Machine Learning
≈ Looking for a Function

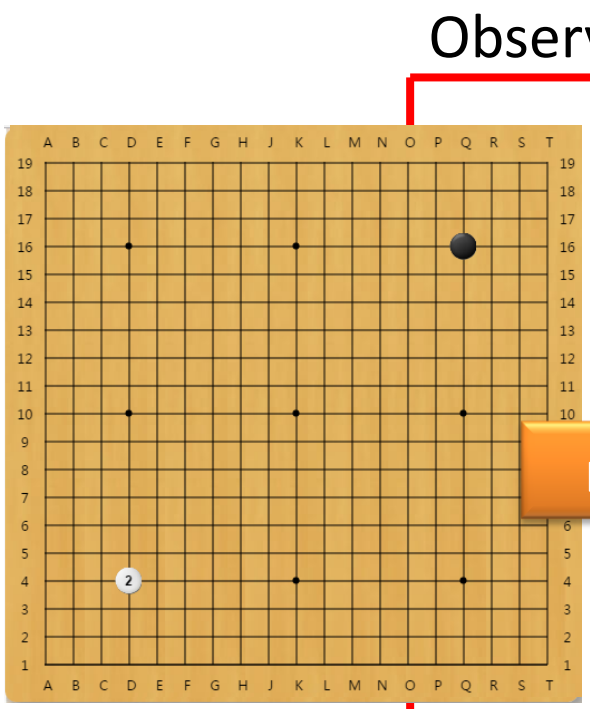


Learning to play Go



Learning to play Go

Agent learns to take actions maximizing expected reward.



Observation

Action

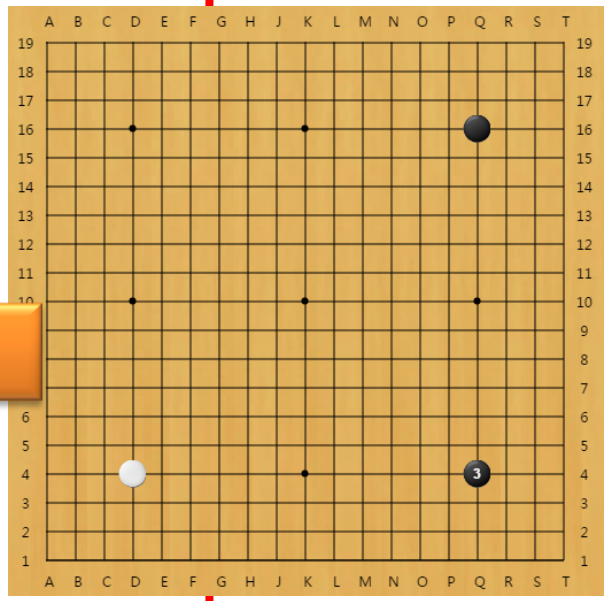
AlphaGo

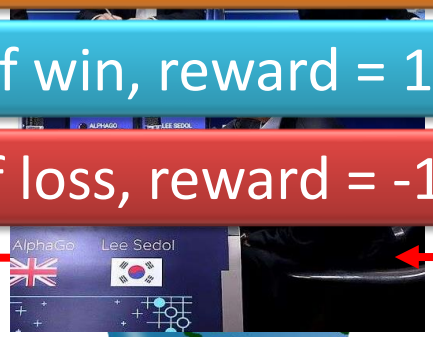
Reward

reward = 0 in most cases

If win, reward = 1

If loss, reward = -1



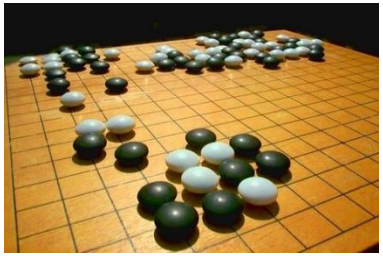


Environment

Learning to play Go

- Supervised: Learning from teacher

Next move: "5-5"



Next move: "3-3"

Learning from experience

- Reinforcement Learning

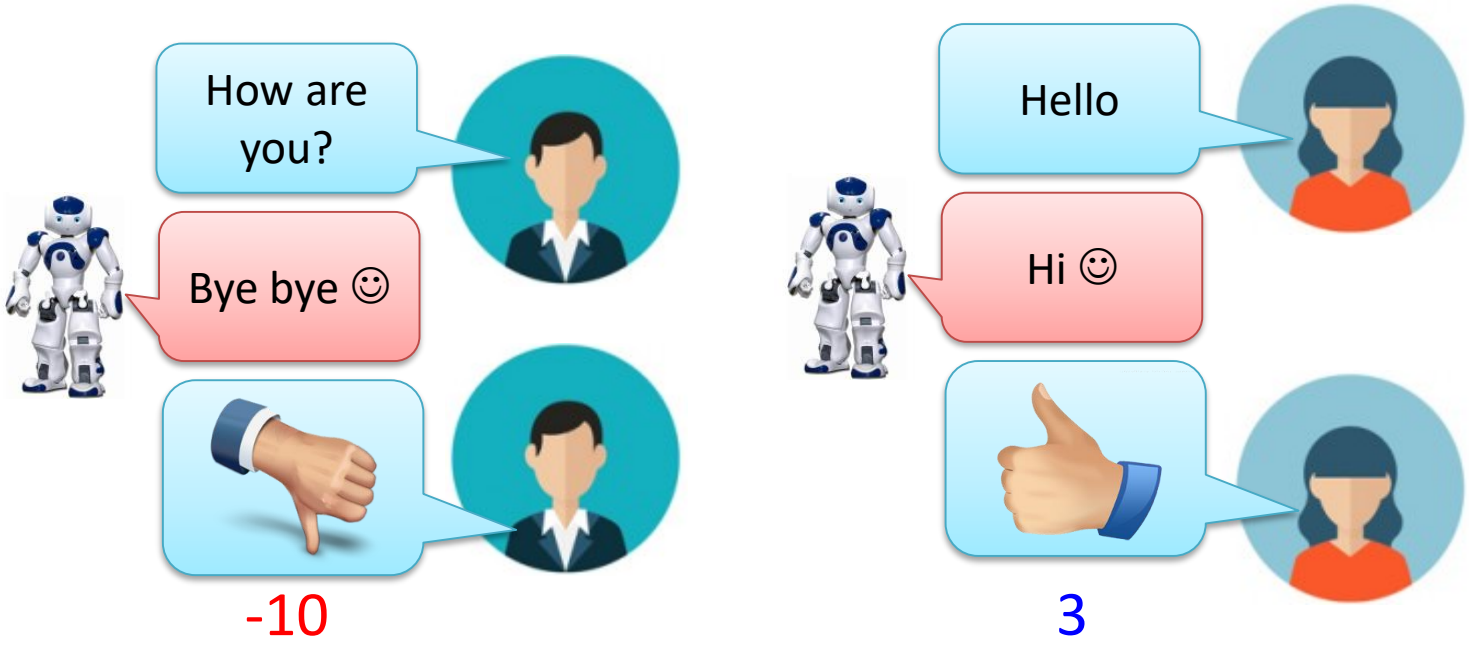
First move → …… many moves …… → Win!

(Two agents play with each other.)

Alpha Go is supervised learning + reinforcement learning.

Learning a chat-bot

- Machine obtains feedback from user



- Chat-bot learns to maximize the ***expected reward***

Learning a chat-bot

- Let two agents talk to each other (sometimes generate good dialogue, sometimes bad)

How old are you?

See you.

See you.

See you.

How old are you?

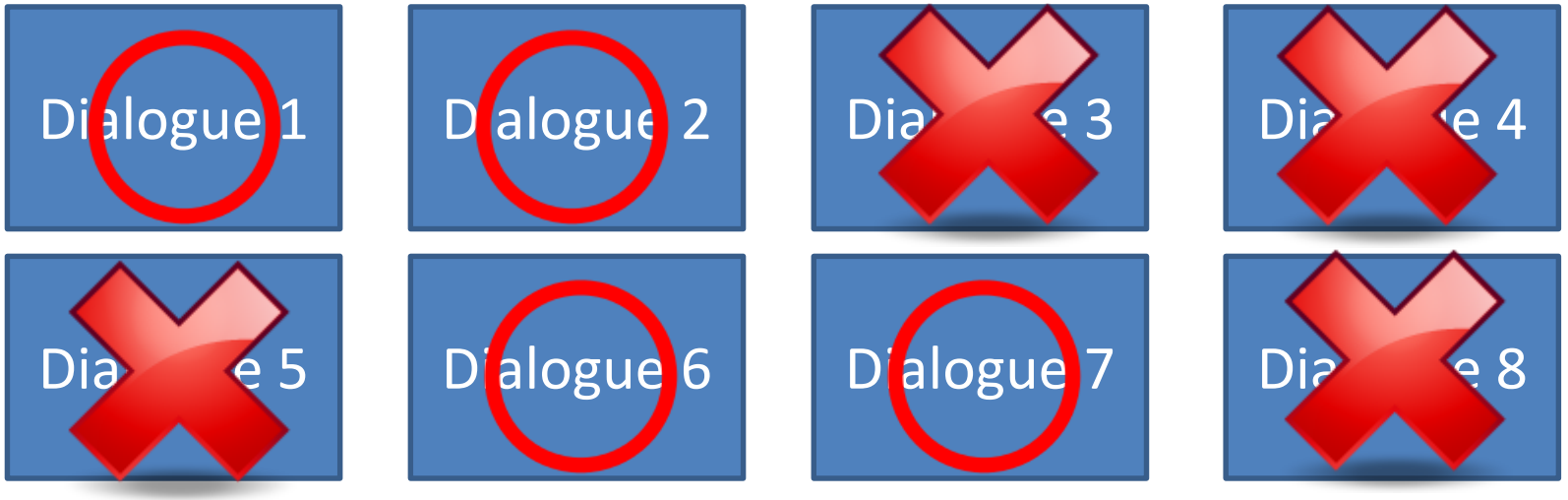
I am 16.

I though you were 12.

What make you think so?

Learning a chat-bot

- By this approach, we can generate a lot of dialogues.
- Use some pre-defined rules to evaluate the goodness of a dialogue

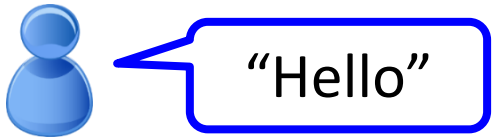


Machine learns from the evaluation

Deep Reinforcement Learning for Dialogue Generation
https://arxiv.org/pdf/1606.01541v3.pdf

Learning a chat-bot

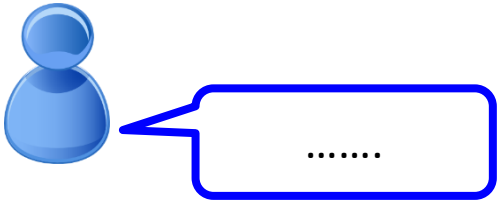
- Supervised

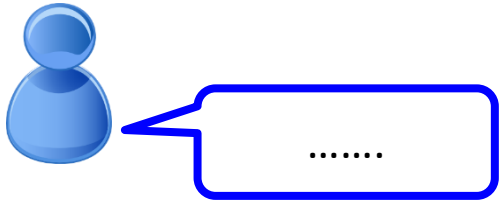


Say “Hi”

Say “Good bye”

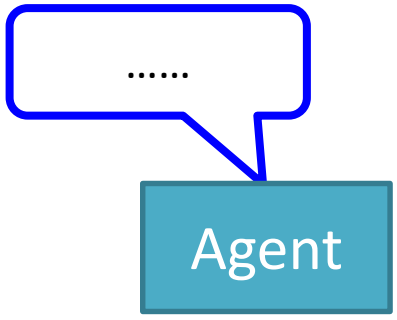
- Reinforcement





.....

Bad



More applications

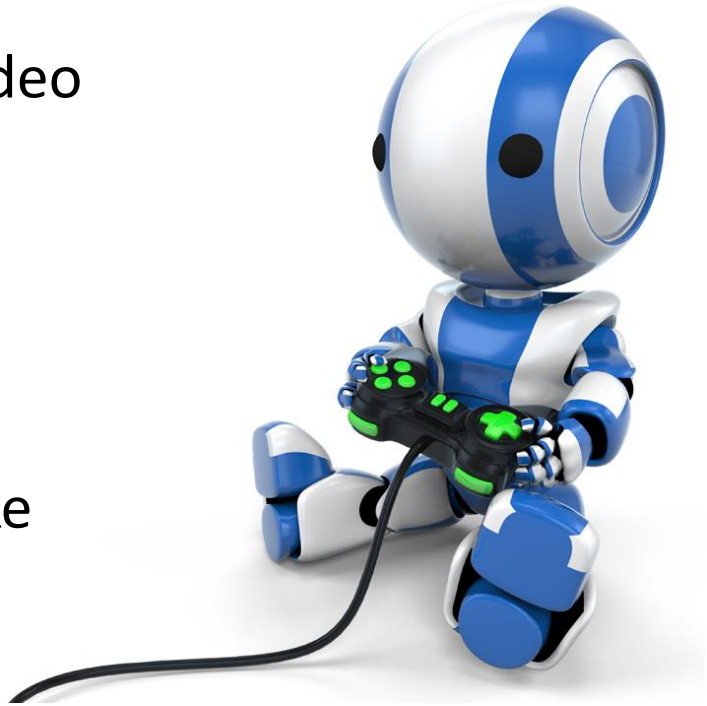
- Flying Helicopter
  - https://www.youtube.com/watch?v=0JL04JJjocc
- Driving
  - https://www.youtube.com/watch?v=0xo1Ldx3L5Q
- Robot
  - https://www.youtube.com/watch?v=370cT-OAzzM
- Google Cuts Its Giant Electricity Bill With DeepMind-Powered AI
  - http://www.bloomberg.com/news/articles/2016-07-19/google-cuts-its-giant-electricity-bill-with-deepmind-powered-ai
- Text generation
  - https://www.youtube.com/watch?v=pbQ4qe8EwLo

Example: Playing Video Game

- Widely studies:
  - Gym: https://gym.openai.com/
  - Universe: https://openai.com/blog/universe/

Machine learns to play video games as human players

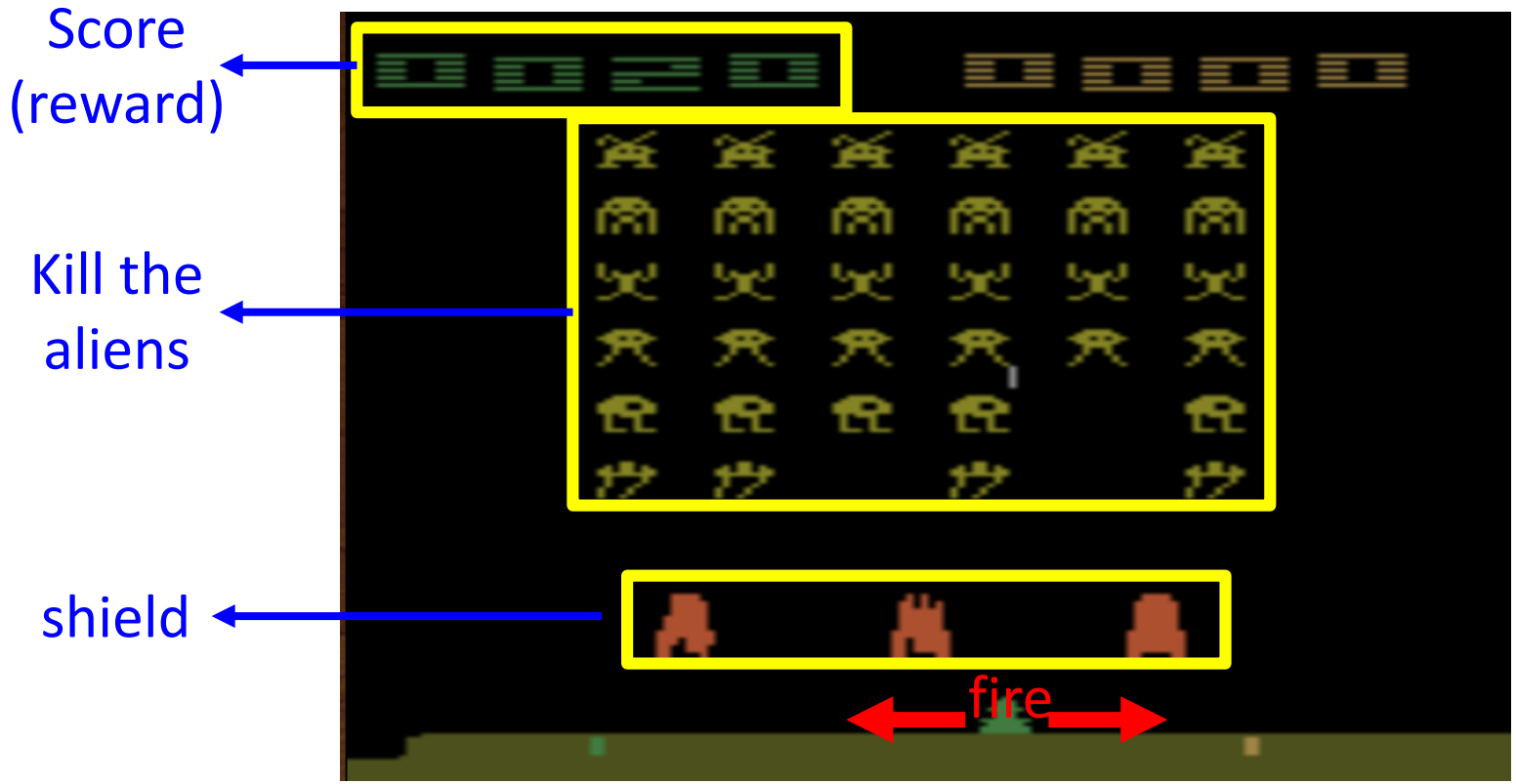
- What machine observes is pixel
- Machine learns to take proper action itself



Example: Playing Video Game

- Space invader

Termination: all the aliens are killed, or your spaceship is destroyed.

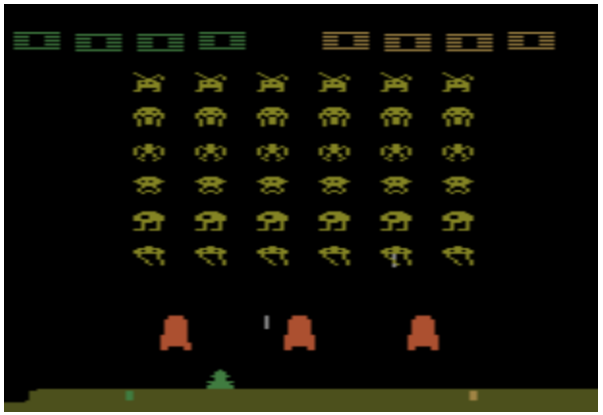


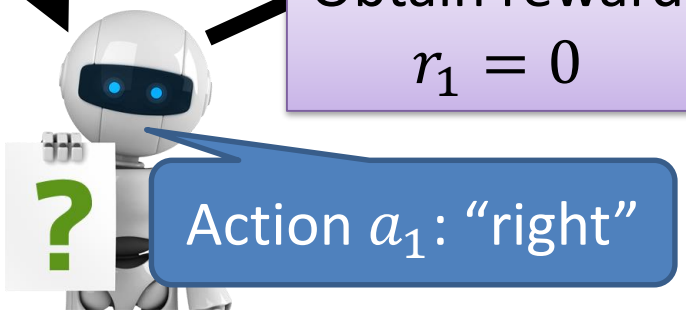
Example: Playing Video Game

Start with observation s_1

Observation s_2

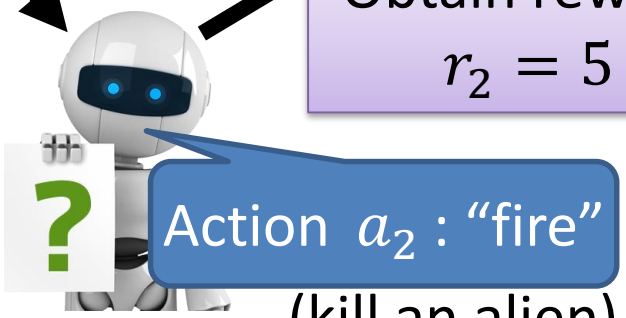
Observation s_3





Obtain reward $r_1 = 0$

Action a_1: "right"



Obtain reward $r_2 = 5$

Action a_2: "fire"

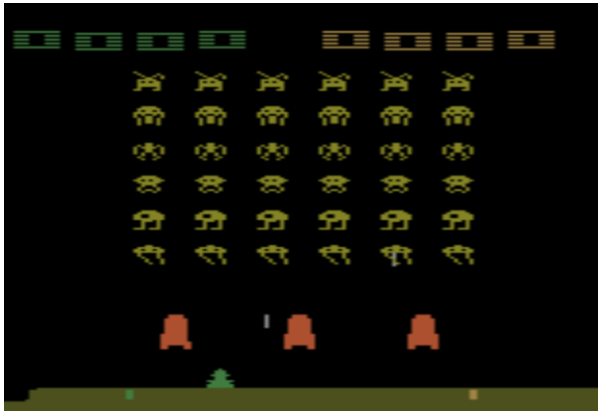
(kill an alien)

Usually there is some randomness in the environment

Example: Playing Video Game

Start with observation s_1

Observation s_2



Observation s_3

After many turns

Game Over (spaceship destroyed)

Obtain reward r_T

Action a_T

This is an ***episode***.

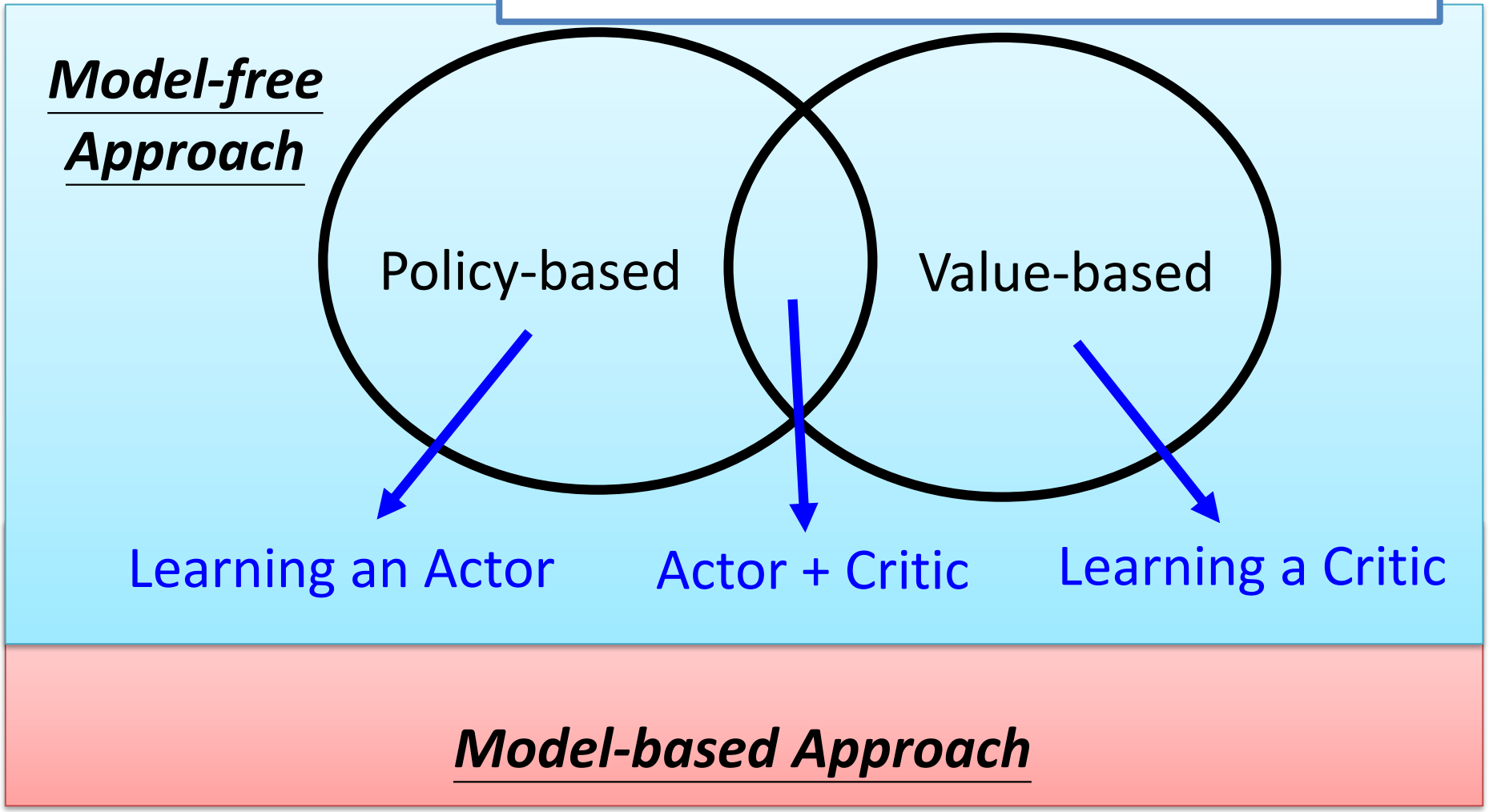
Learn to maximize the expected cumulative reward per episode

Properties of Reinforcement Learning

- **Reward delay**
  - In space invader, only “fire” obtains reward
    - Although the moving before “fire” is important
  - In Go playing, it may be better to sacrifice immediate reward to gain more long-term reward
- Agent’s actions **affect the subsequent data it receives**
  - E.g. Exploration

Methods of Reinforcement Learning

Alpha Go: policy-based + value-based + model-based



Outline

1. Course Review
2. Background
3. Model-free learning
4. Model-based learning
5. Applications

Policy-based Approach

Learning an Actor

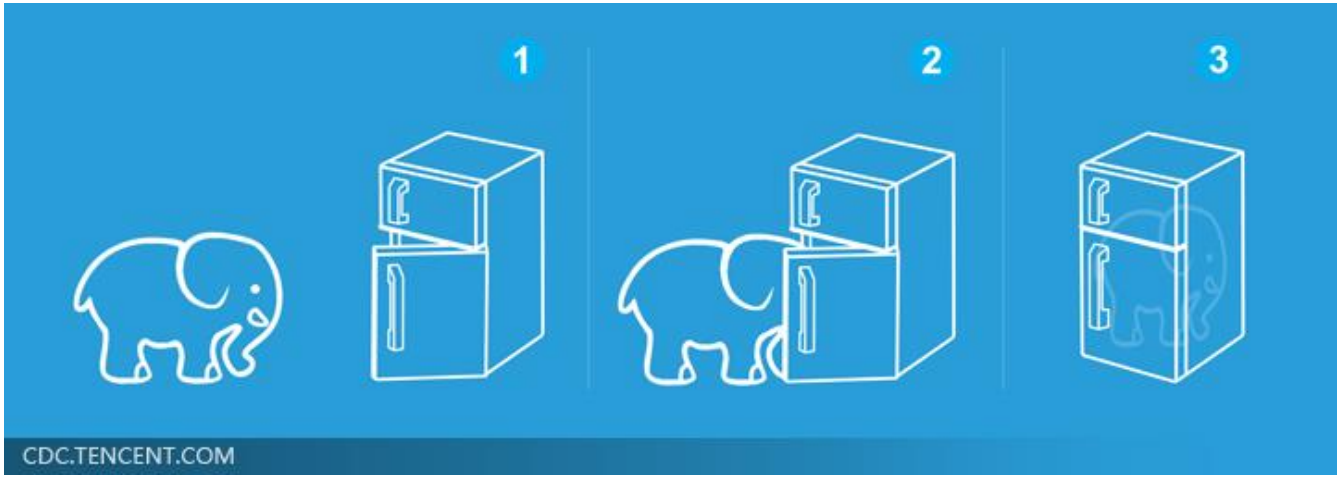
Three Steps for Deep Learning

Step 1: Neural Network as Actor

Step 2: goodness of function

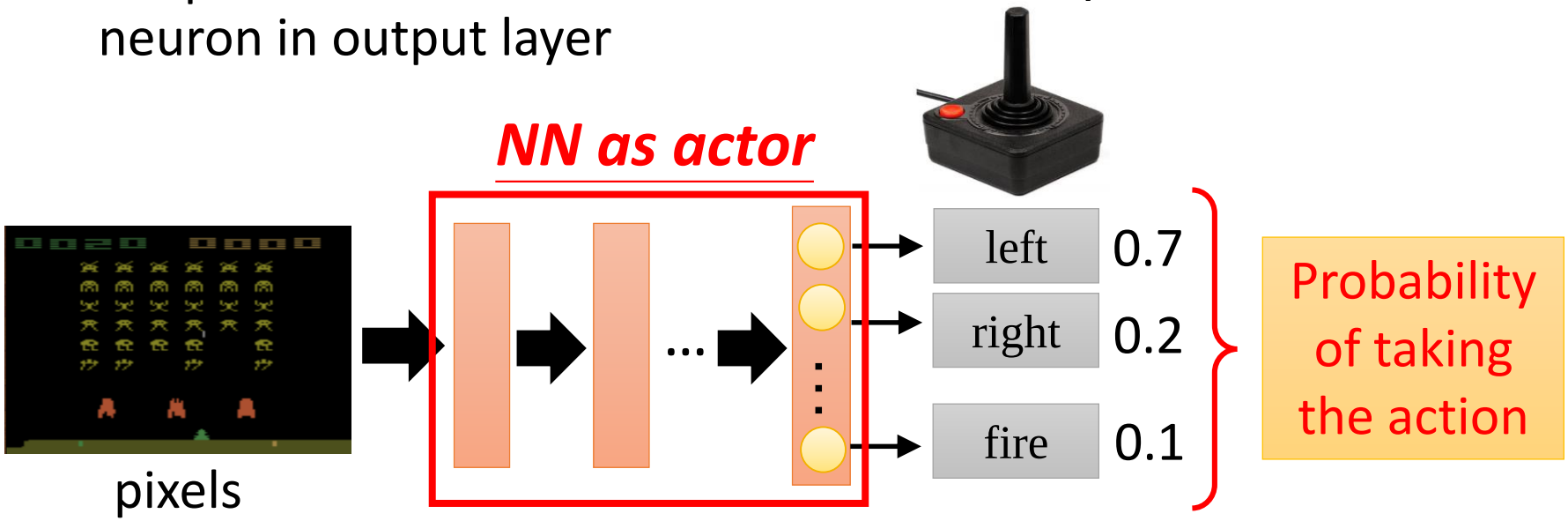
Step 3: pick the best function

Deep Learning is so simple ……



Neural network as Actor

- Input of neural network: the observation of machine represented as a vector or a matrix
- Output neural network : each action corresponds to a neuron in output layer



What is the benefit of using network instead of lookup table?

generalization

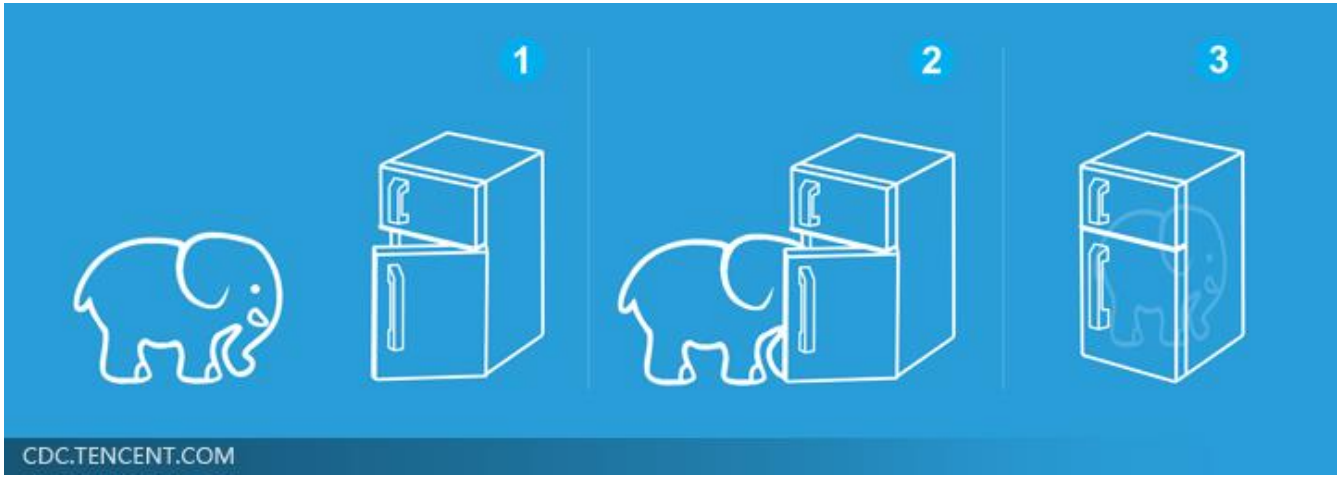
Three Steps for Deep Learning

Step 1: Neural Network as Actor

Step 2: goodness of function

Step 3: pick the best function

Deep Learning is so simple ……

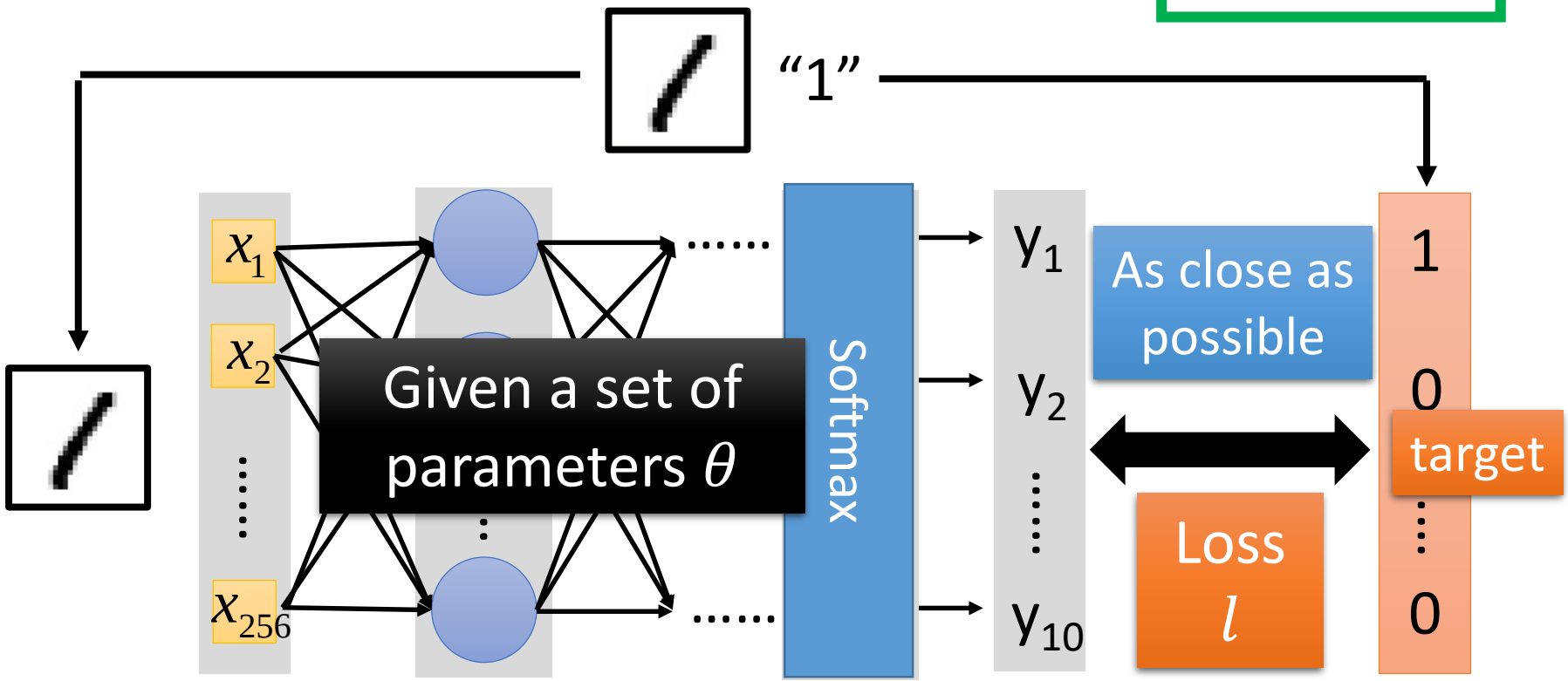


Goodness of Actor

- Review: Supervised learning

Total Loss:

$$L = \sum_{n=1}^{N} l_n$$



Goodness of Actor

- Given an actor $\pi_\theta(s)$ with network parameter θ
- Use the actor $\pi_\theta(s)$ to play the video game
  - Start with observation s_1
  - Machine decides to take a_1
  - Machine obtains reward r_1
  - Machine sees observation s_2
  - Machine decides to take a_2
  - Machine obtains reward r_2
  - Machine sees observation s_3
  - ……
  - Machine decides to take a_T
  - Machine obtains reward r_T

END

Total reward: $R_\theta = \sum_{t=1}^{T} r_t$

Even with the same actor, R_θ is different each time

Randomness in the actor and the game

We define $\bar{R}_\theta$ as the *expected value* of R_θ

$\bar{R}_\theta$ evaluates the goodness of an actor $\pi_\theta(s)$

Goodness of Actor

We define $\bar{R}_\theta$ as the *expected value* of R_θ

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

$$P(\tau|\theta) =$$

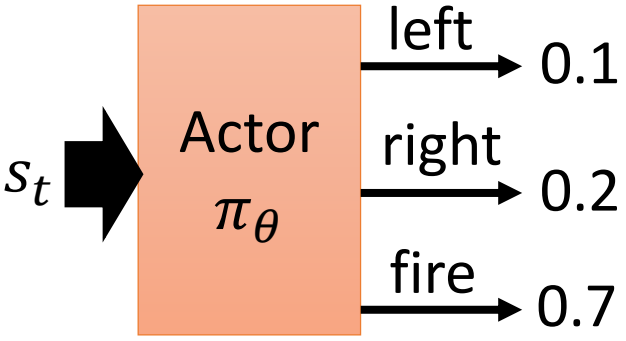
$$p(s_1)p(a_1|s_1,\theta)p(r_1,s_2|s_1,a_1)p(a_2|s_2,\theta)p(r_2,s_3|s_2,a_2)\cdots$$

$$= p(s_1)\prod_{t=1}^{T} p(a_t|s_t,\theta)p(r_t,s_{t+1}|s_t,a_t)$$

not related to your actor

Control by your actor π_θ

$p(a_t = "fire"|s_t,\theta) = 0.7$



Goodness of Actor

- An episode is considered as a trajectory τ
  - $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
  - $R(\tau) = \sum_{t=1}^{T} r_t$
  - If you use an actor to play the game, each τ has a probability to be sampled
    - The probability depends on actor parameter θ: $P(\tau|\theta)$

$$\bar{R}_\theta = \sum_\tau R(\tau) P(\tau|\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n)$$

Sum over all possible trajectory

Use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

Sampling τ from $P(\tau|\theta)$ N times

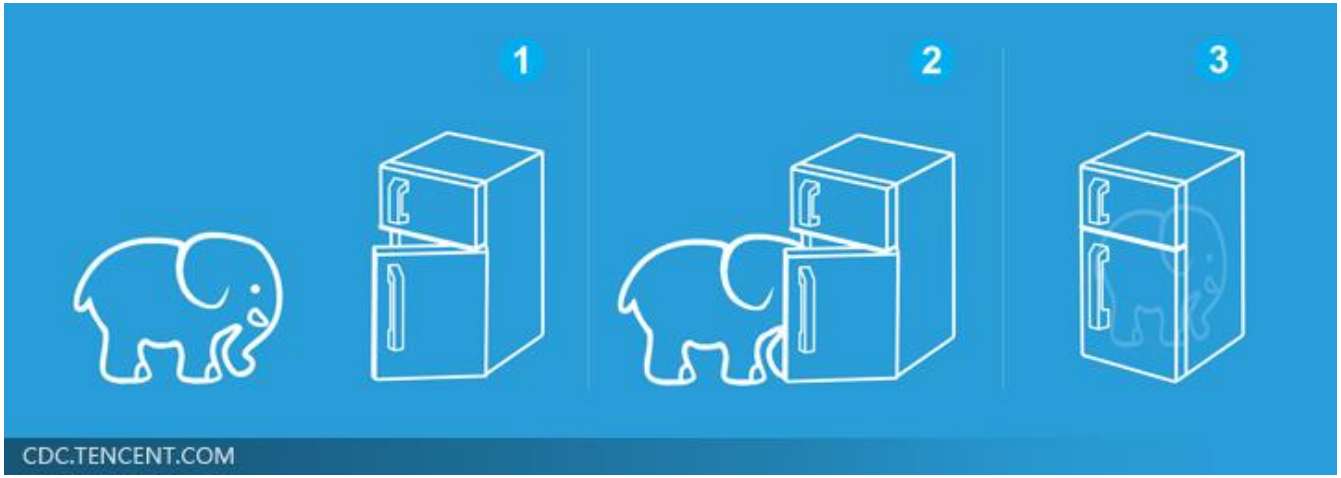
Three Steps for Deep Learning

Step 1:
Neural Network as Actor

Step 2: goodness of function

Step 3: pick the best function

Deep Learning is so simple ……



Gradient Ascent

- Problem statement

$$\theta^* = arg \max_{\theta} \bar{R}_\theta$$

- Gradient ascent
  - Start with θ^0
  - $\theta^1 \leftarrow \theta^0 + \eta \nabla \bar{R}_{\theta^0}$
  - $\theta^2 \leftarrow \theta^1 + \eta \nabla \bar{R}_{\theta^1}$
  - ……

$$\theta = \{w_1, w_2, \cdots, b_1, \cdots\}$$

$$\nabla \bar{R}_\theta = \begin{bmatrix} \partial \bar{R}_\theta / \partial w_1 \\ \partial \bar{R}_\theta / \partial w_2 \\ \vdots \\ \partial \bar{R}_\theta / \partial b_1 \\ \vdots \end{bmatrix}$$

Policy Gradient

$$\bar{R}_\theta = \sum_\tau R(\tau)P(\tau|\theta) \quad \nabla\bar{R}_\theta = ?$$

$$\nabla\bar{R}_\theta = \sum_\tau R(\tau)\nabla P(\tau|\theta) = \sum_\tau R(\tau)P(\tau|\theta)\frac{\nabla P(\tau|\theta)}{P(\tau|\theta)}$$

$R(\tau)$ do not have to be differentiable

It can even be a black box.

$$= \sum_\tau R(\tau)P(\tau|\theta)\nabla logP(\tau|\theta)$$

$$\frac{dlog\big(f(x)\big)}{dx} = \frac{1}{f(x)}\frac{df(x)}{dx}$$

$$\approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla logP(\tau^n|\theta)$$

Use π_θ to play the game N times,

Obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

Policy Gradient

$\nabla logP(\tau|\theta) = ?$

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

$$P(\tau|\theta) = p(s_1)\prod_{t=1}^{T} p(a_t|s_t,\theta)p(r_t, s_{t+1}|s_t, a_t)$$

$$logP(\tau|\theta)$$

$$= logp(s_1) + \sum_{t=1}^{T} logp(a_t|s_t,\theta) + logp(r_t, s_{t+1}|s_t, a_t)$$

$$\nabla logP(\tau|\theta) = \sum_{t=1}^{T} \nabla logp(a_t|s_t,\theta)$$

Ignore the terms not related to θ

Policy Gradient

$$\nabla logP(\tau|\theta) = \sum_{t=1}^{T} \nabla logp(a_t|s_t,\theta)$$

$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla logP(\tau^n|\theta) = \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\sum_{t=1}^{T_n} \nabla logp(a_t^n|s_t^n,\theta)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla logp(a_t^n|s_t^n,\theta)$$

What if we replace $R(\tau^n)$ with r_t^n .....

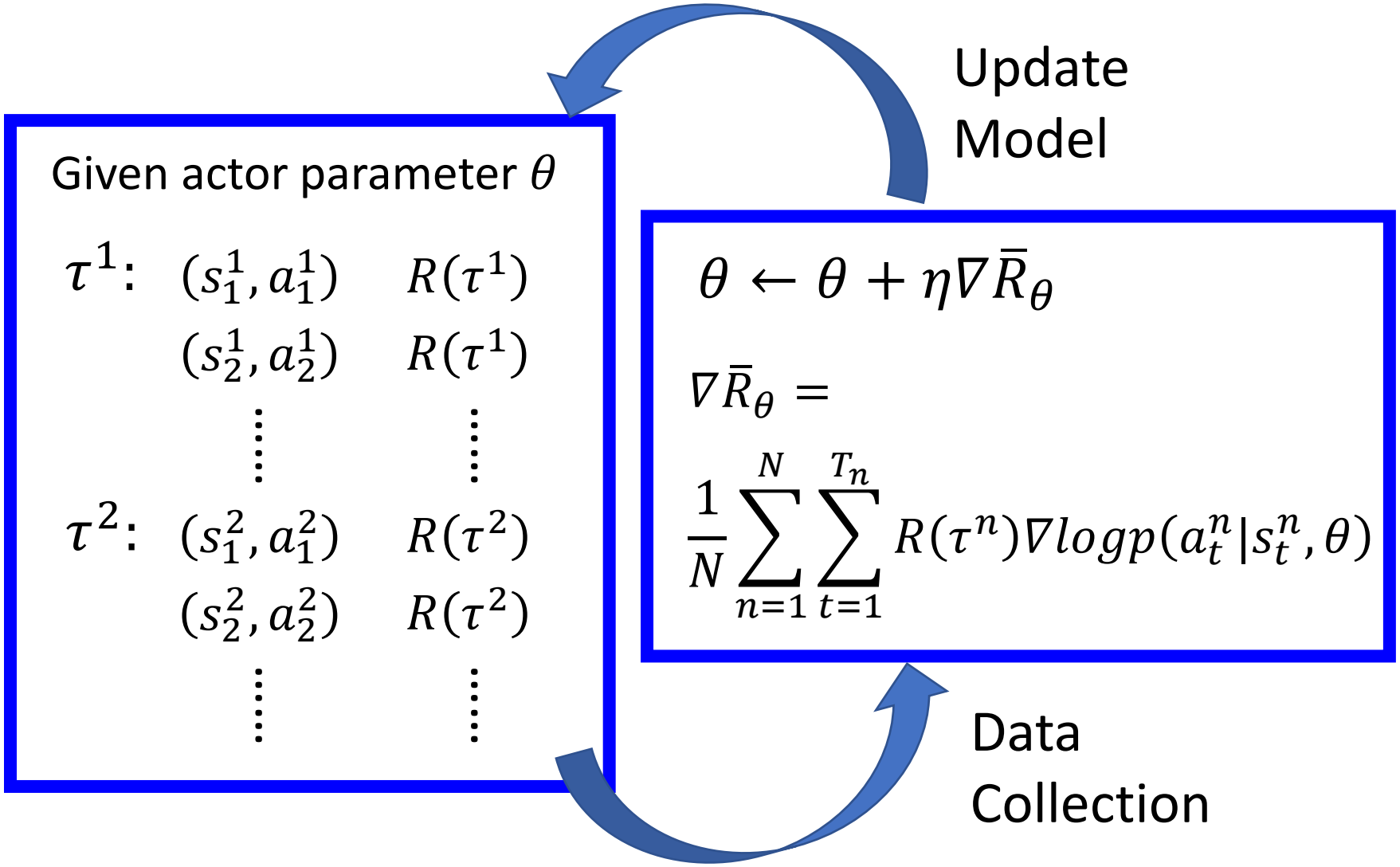
If in τ^n machine takes a_t^n when seeing s_t^n in

$R(\tau^n)$ is positive ⇒ Tuning θ to increase $p(a_t^n|s_t^n)$

$R(\tau^n)$ is negative ⇒ Tuning θ to decrease $p(a_t^n|s_t^n)$

It is very important to consider the cumulative reward $R(\tau^n)$ of the whole trajectory τ^n instead of immediate reward r_t^n

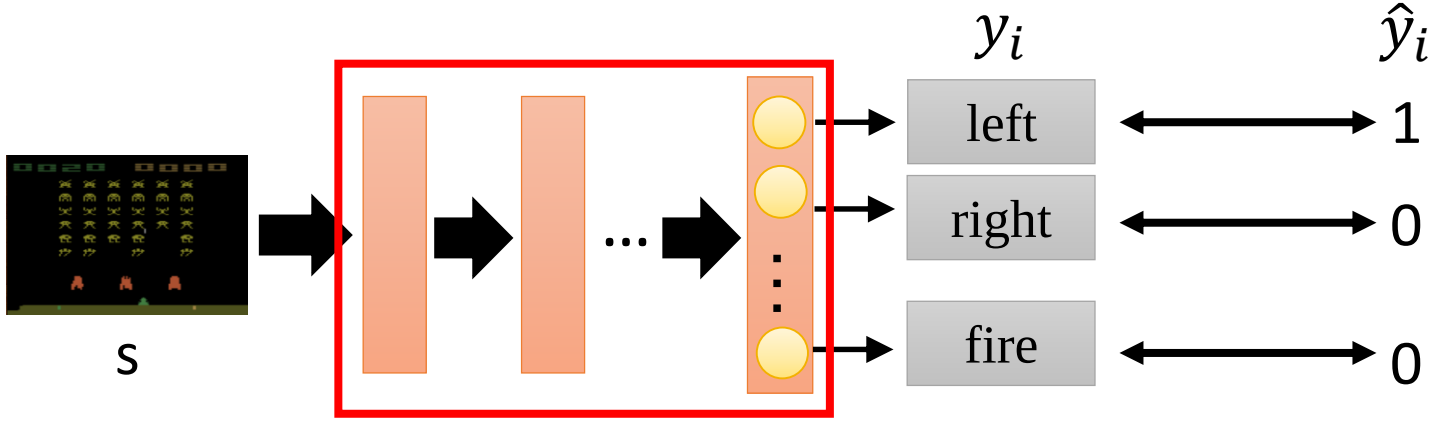
Policy Gradient



Policy Gradient

Considered as Classification Problem

Minimize: $-\sum_{i=1}^{3} \hat{y}_i log y_i$



Maximize: $logy_i = logP("left"|s)$

$$\theta \leftarrow \theta + \eta \nabla logP("left"|s)$$

Policy Gradient

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \blacksquare \nabla log p(a_t^n | s_t^n, \theta)$$

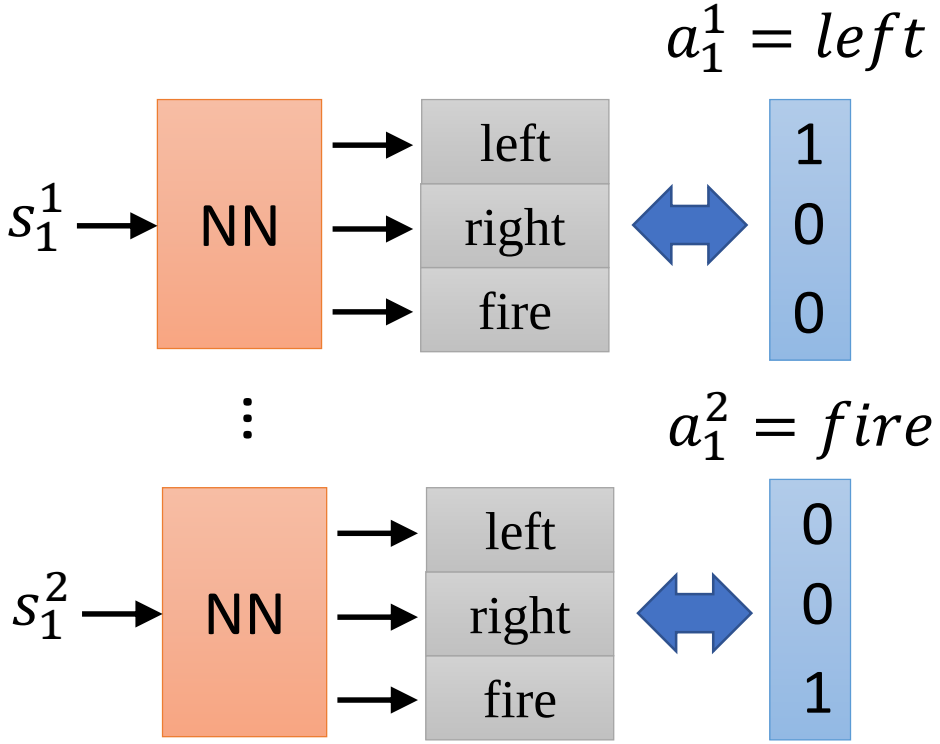
Given actor parameter θ

τ^1: (s_1^1, a_1^1) $R(\tau^1)$

(s_2^1, a_2^1) $R(\tau^1)$

τ^2: (s_1^2, a_1^2) $R(\tau^2)$

(s_2^2, a_2^2) $R(\tau^2)$



Policy Gradient

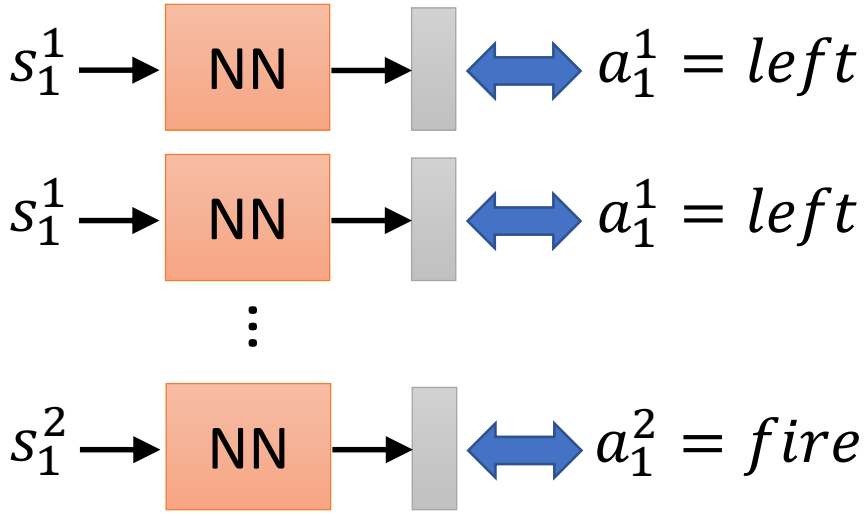
$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla log p(a_t^n | s_t^n, \theta)$$

Given actor parameter θ

τ^1:	(s_1^1, a_1^1)	$R(\tau^1)$	2
	(s_2^1, a_2^1)	$R(\tau^1)$	2
	⋮	⋮	
τ^2:	(s_1^2, a_1^2)	$R(\tau^2)$	1
	(s_2^2, a_2^2)	$R(\tau^2)$	1
	⋮	⋮	

Each training data is weighted by $R(\tau^n)$



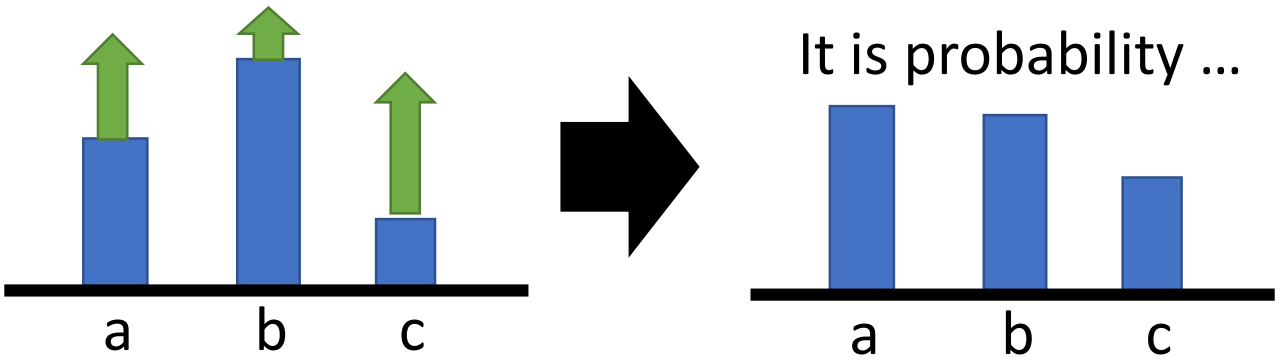
Add a Baseline

It is possible that $R(\tau^n)$ is always positive.

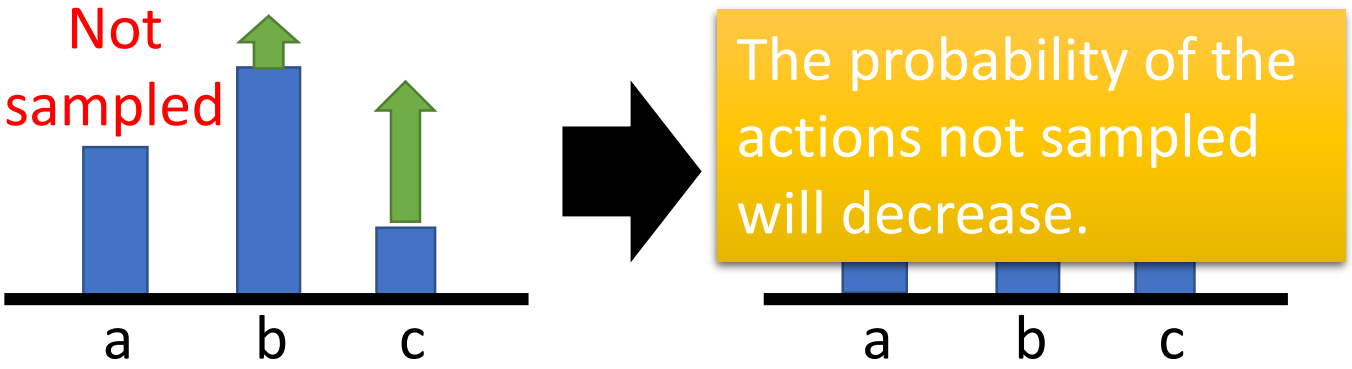
$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla log p(a_t^n | s_t^n, \theta)$$

Ideal case



Sampling ……



Value-based Approach

Learning a Critic

Critic

- A critic does not determine the action.
- Given an actor π, it evaluates the how good the actor is

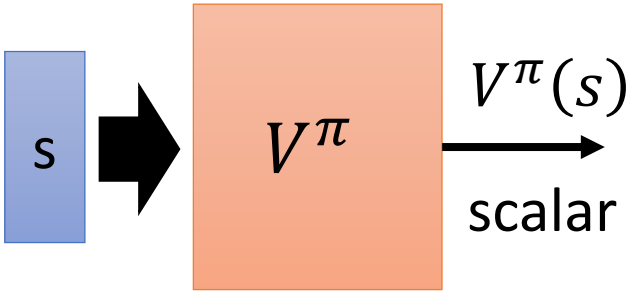
An actor can be found from a critic.

e.g. Q-learning

http://combiboilersleeds.com/picaso/critics/critics-4.html

Critic

- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation (state) s



$V^{\pi}(s)$ is large

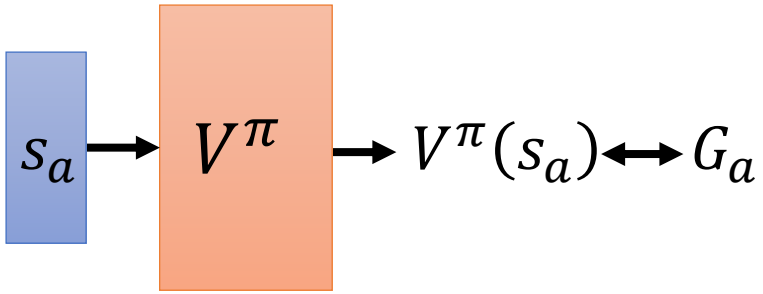
$V^{\pi}(s)$ is smaller

How to estimate $V^{\pi}(s)$

- Monte-Carlo based approach
  - The critic watches π playing the game

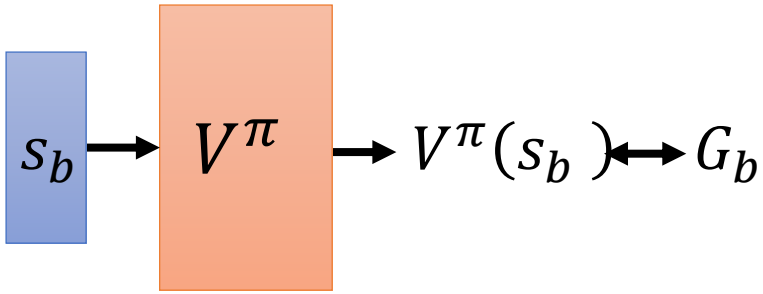
After seeing s_a,

Until the end of the episode, the cumulated reward is G_a



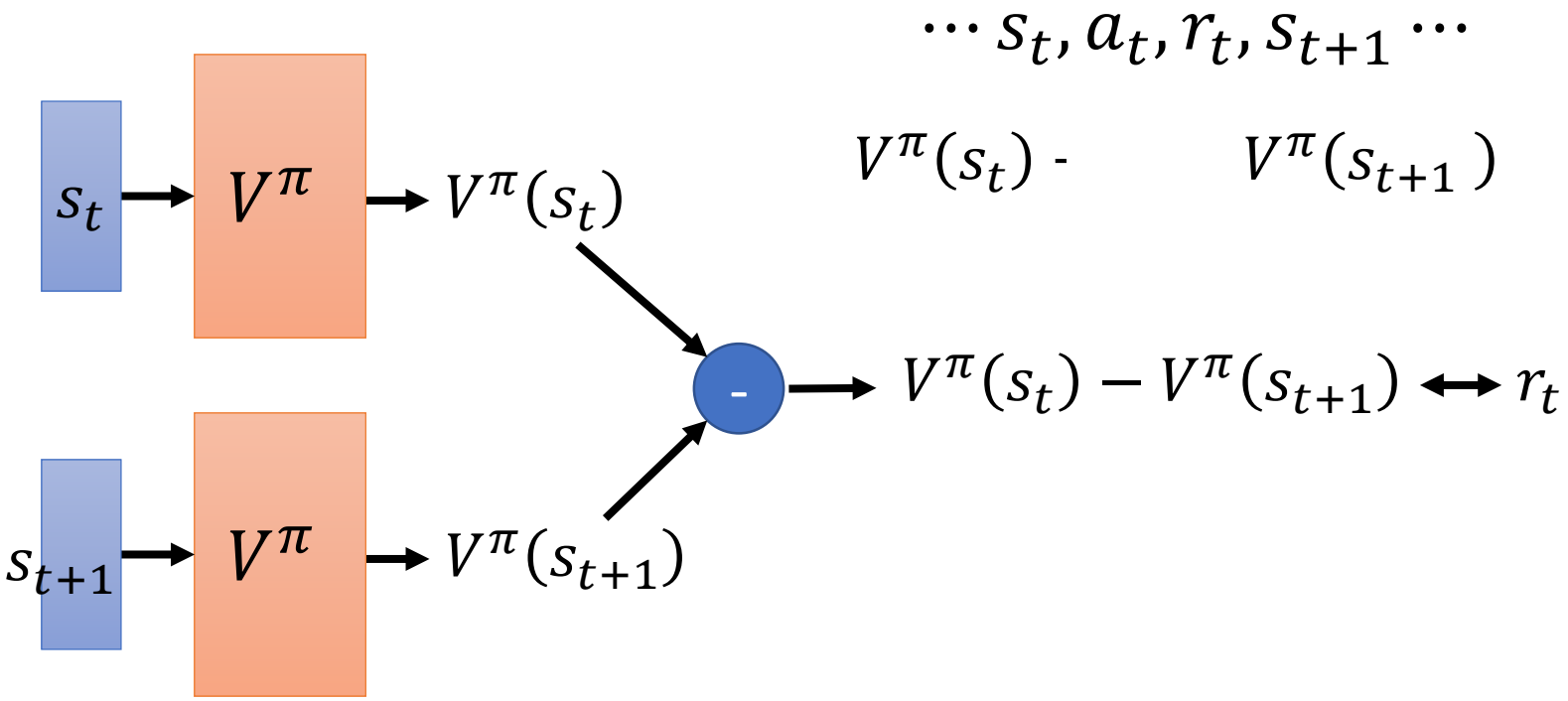
After seeing s_b,

Until the end of the episode, the cumulated reward is G_b



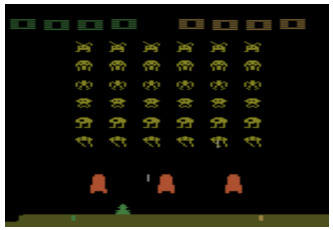
How to estimate $V^{\pi}(s)$

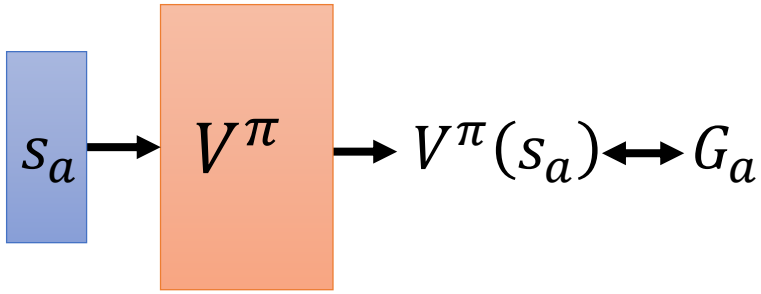
- Temporal-difference approach



Some applications have very long episodes, so that delaying all learning until an episode's end is too slow.

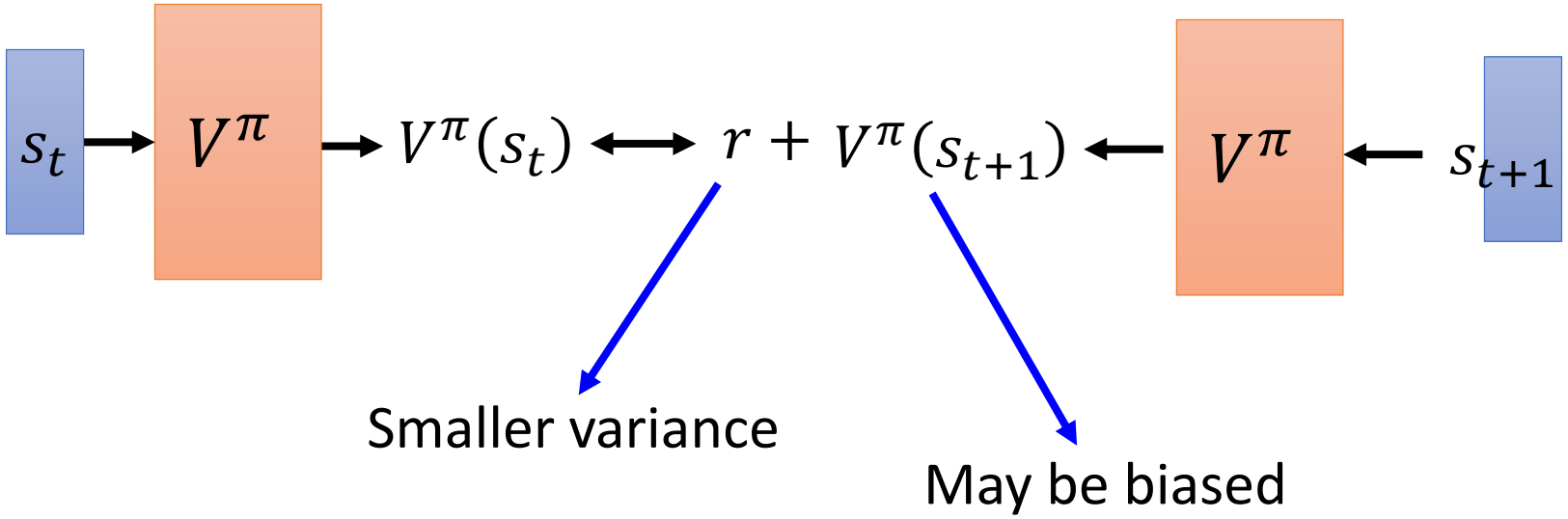
MC v.s. TD



...

Larger variance

unbiased



MC v.s. TD

[Sutton, v2, Example 6.4]

- The critic has the following 8 episodes
  - $s_a, r = 0, s_b, r = 0$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 0$, END

$$V^{\pi}(s_b) = 3/4$$

$$V^{\pi}(s_a) = ? \quad 0? \quad 3/4?$$

Monte-Carlo: $V^{\pi}(s_a) = 0$

Temporal-difference:

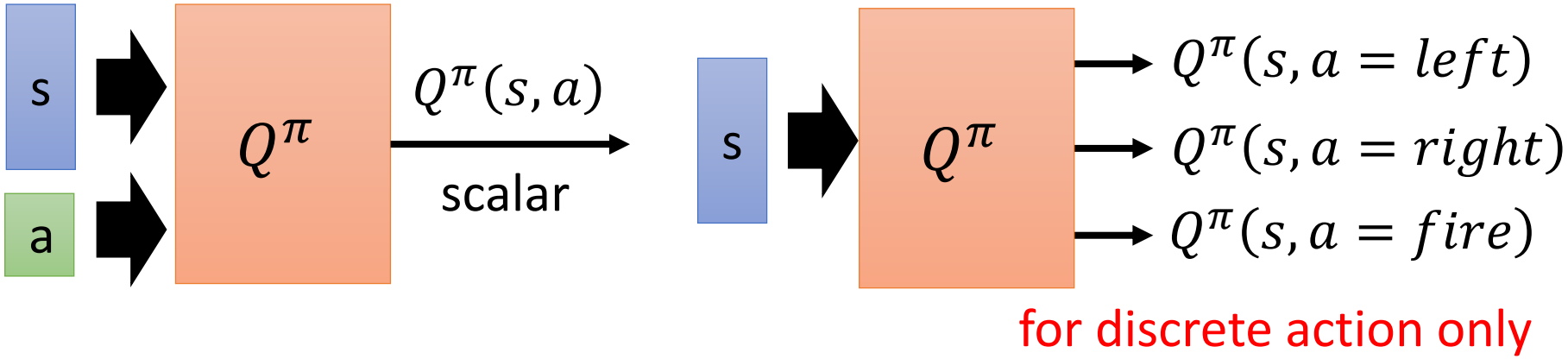
$$V^{\pi}(s_b) + r = V^{\pi}(s_a)$$

3/4 0 3/4

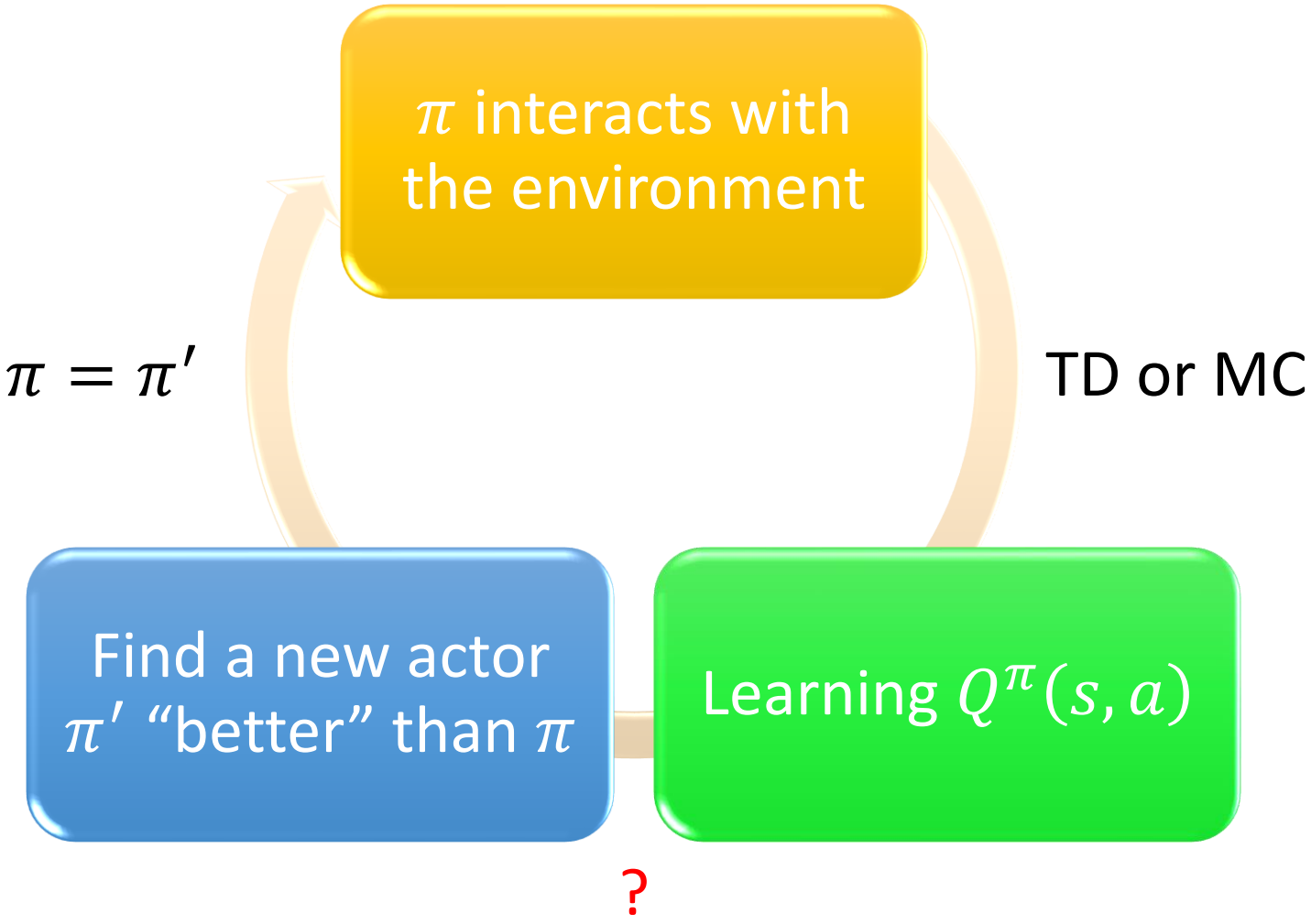
(The actions are ignored here.)

Another Critic

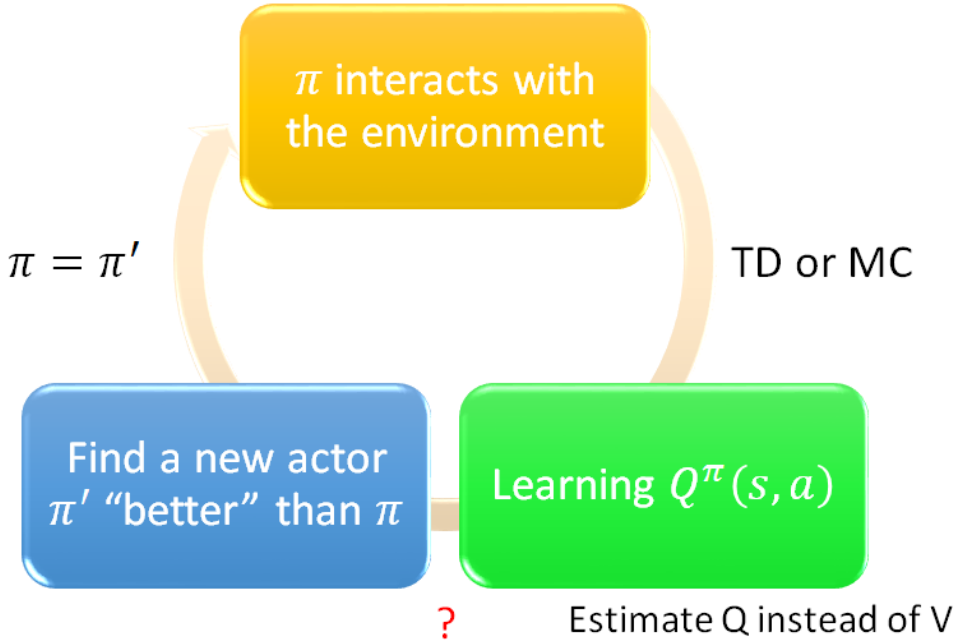
- State-action value function $Q^{\pi}(s, a)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation s and taking a



Q-Learning


π interacts with the environment
TD or MC
Learning $Q^{\pi}(s, a)$
?
Find a new actor π' "better" than π
$\pi = \pi'$

Q-Learning



- Given $Q^{\pi}(s,a)$, find a new actor π' "better" than π
  - "Better": $V^{\pi'}(s) \geq V^{\pi}(s)$, for all state s

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

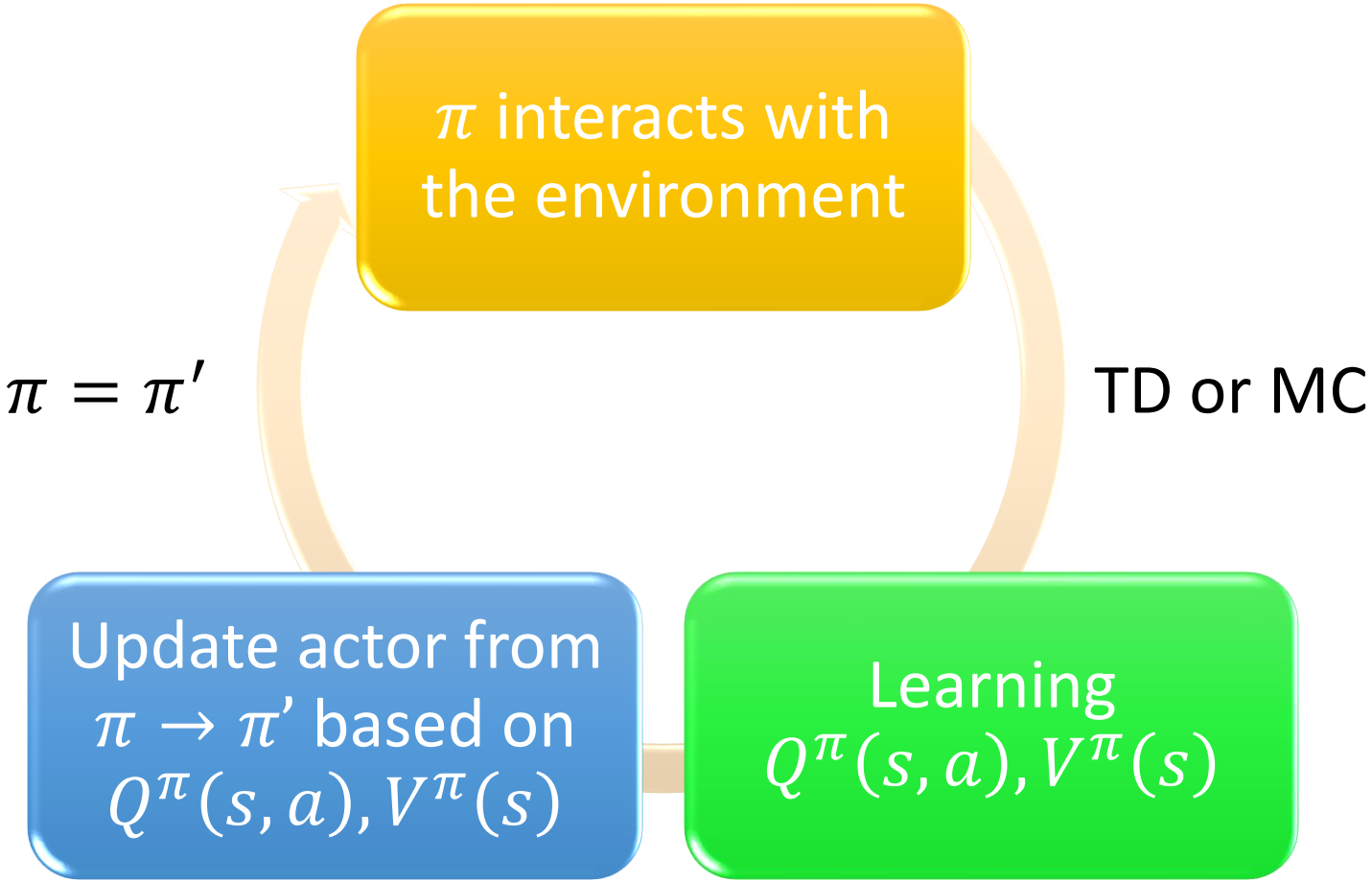
➢ π' does not have extra parameters. It depends on Q

➢ Not suitable for continuous action a

Deep Reinforcement Learning

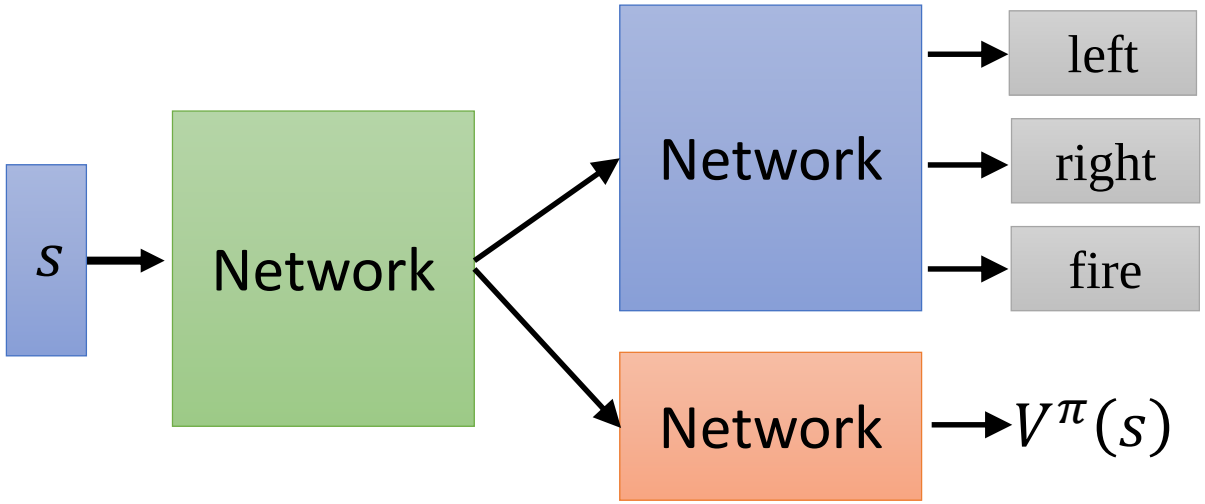
Actor-Critic

Actor-Critic



Actor-Critic

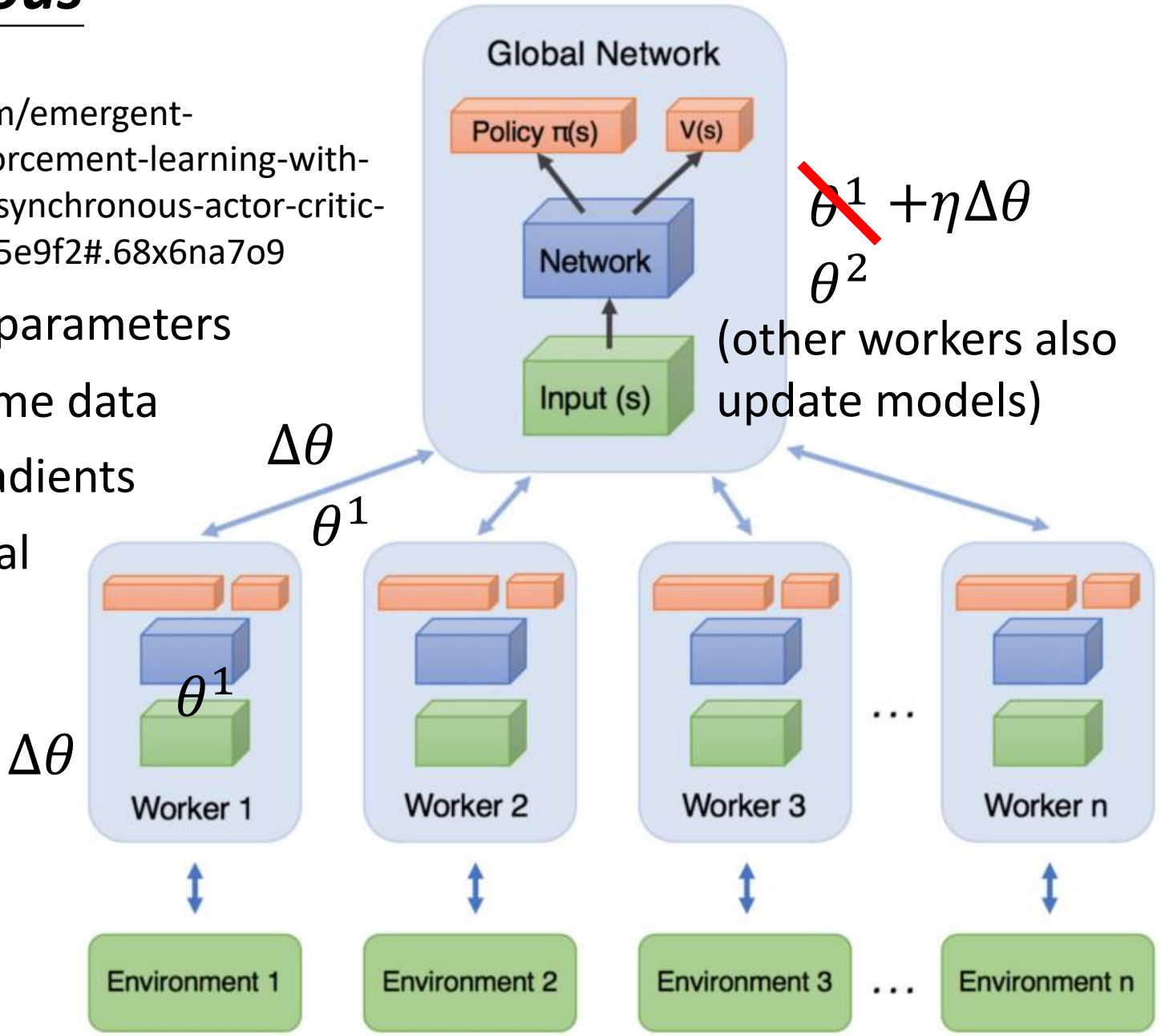
- Tips
  - The parameters of actor $\pi(s)$ and critic $V^{\pi}(s)$ can be shared



Asynchronous

Source of image: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2#.68x6na7o9

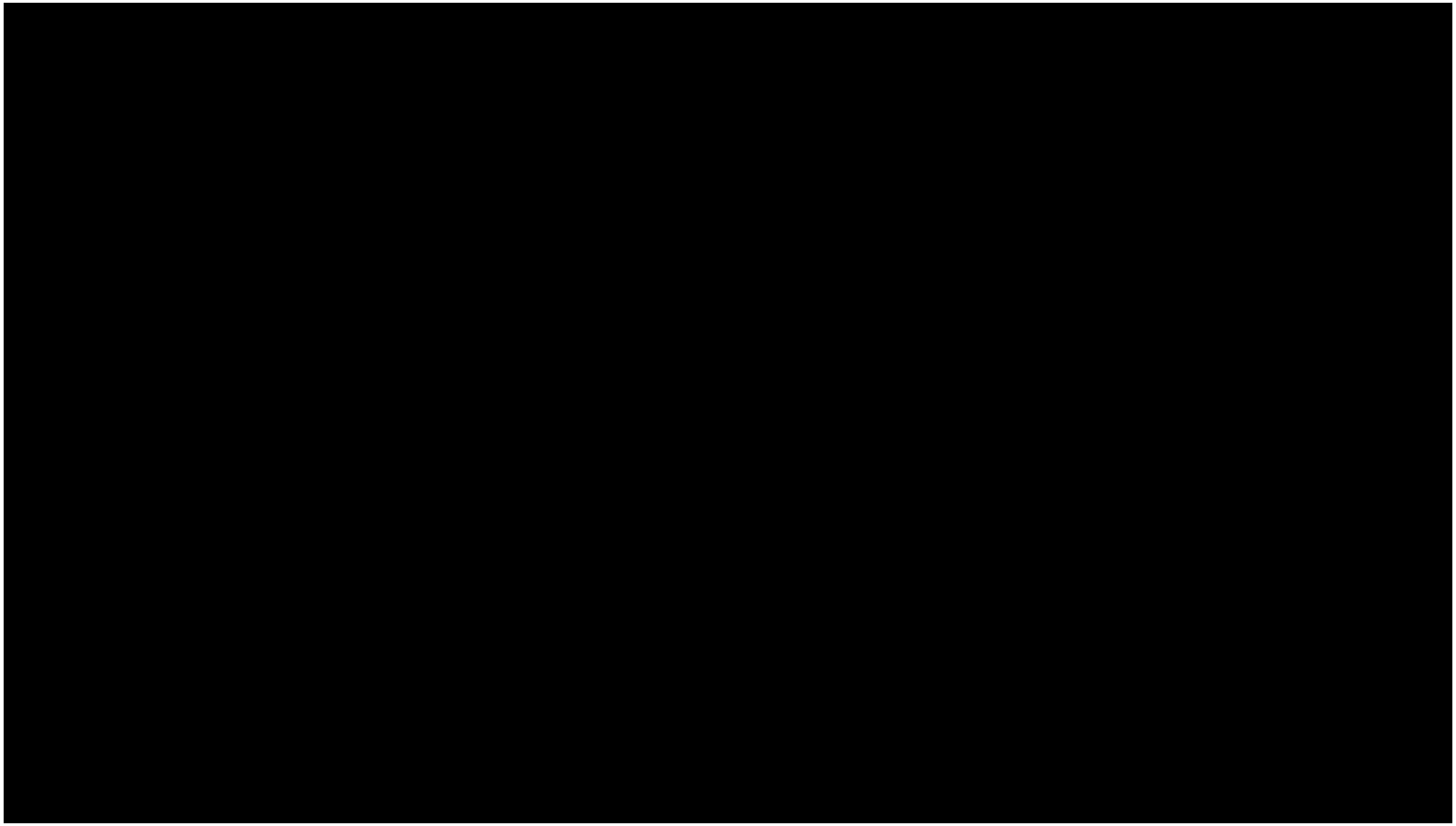
1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models



Demo of A3C

https://www.youtube.com/watch?v=0xo1Ldx3L5Q

- Racing Car (DeepMind)



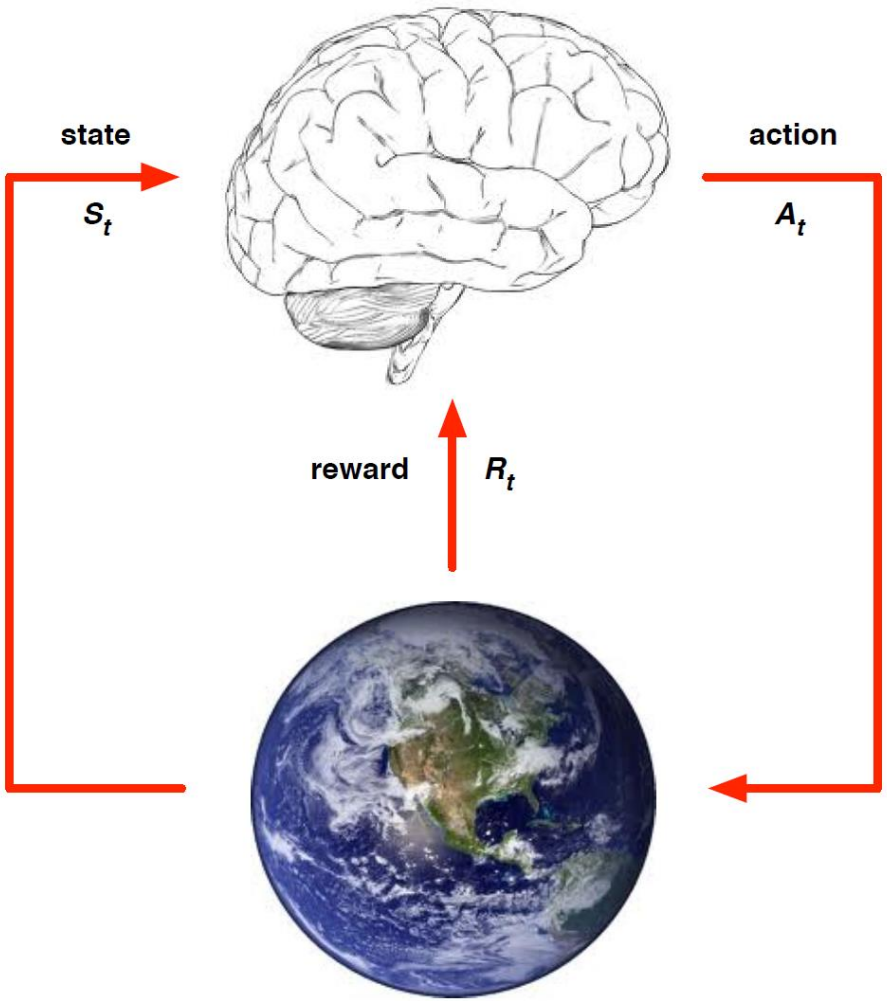
Outline

1. Course Review
2. Background
3. Model-free learning
4. Model-based learning
5. Applications

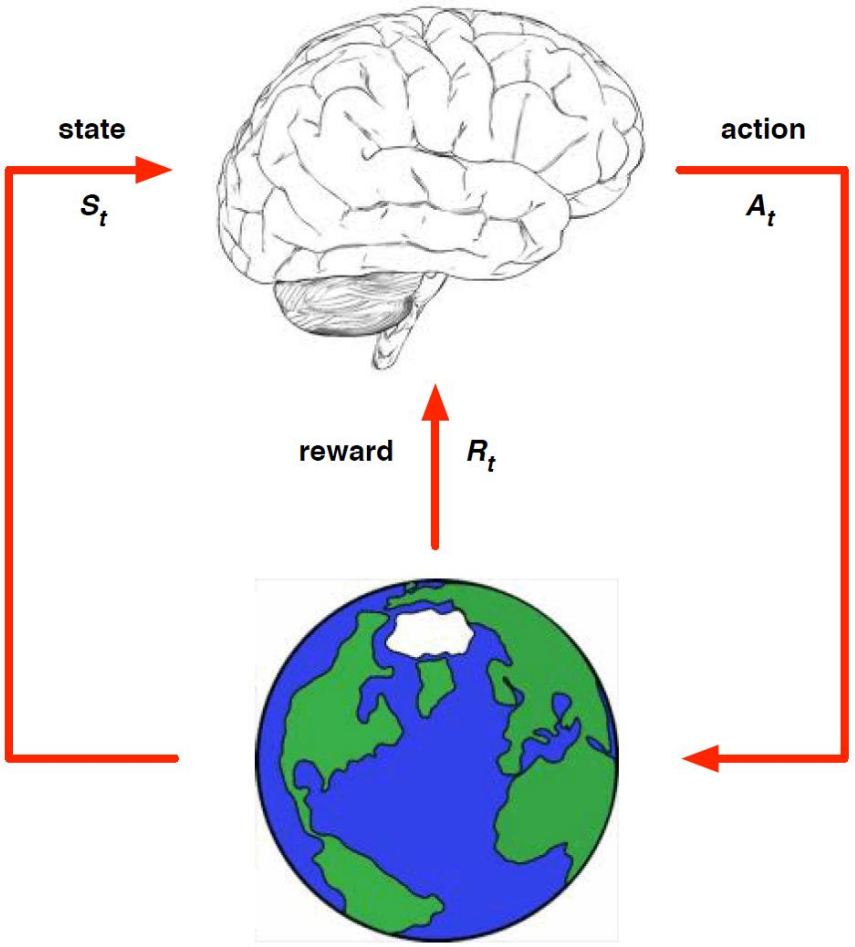
Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model

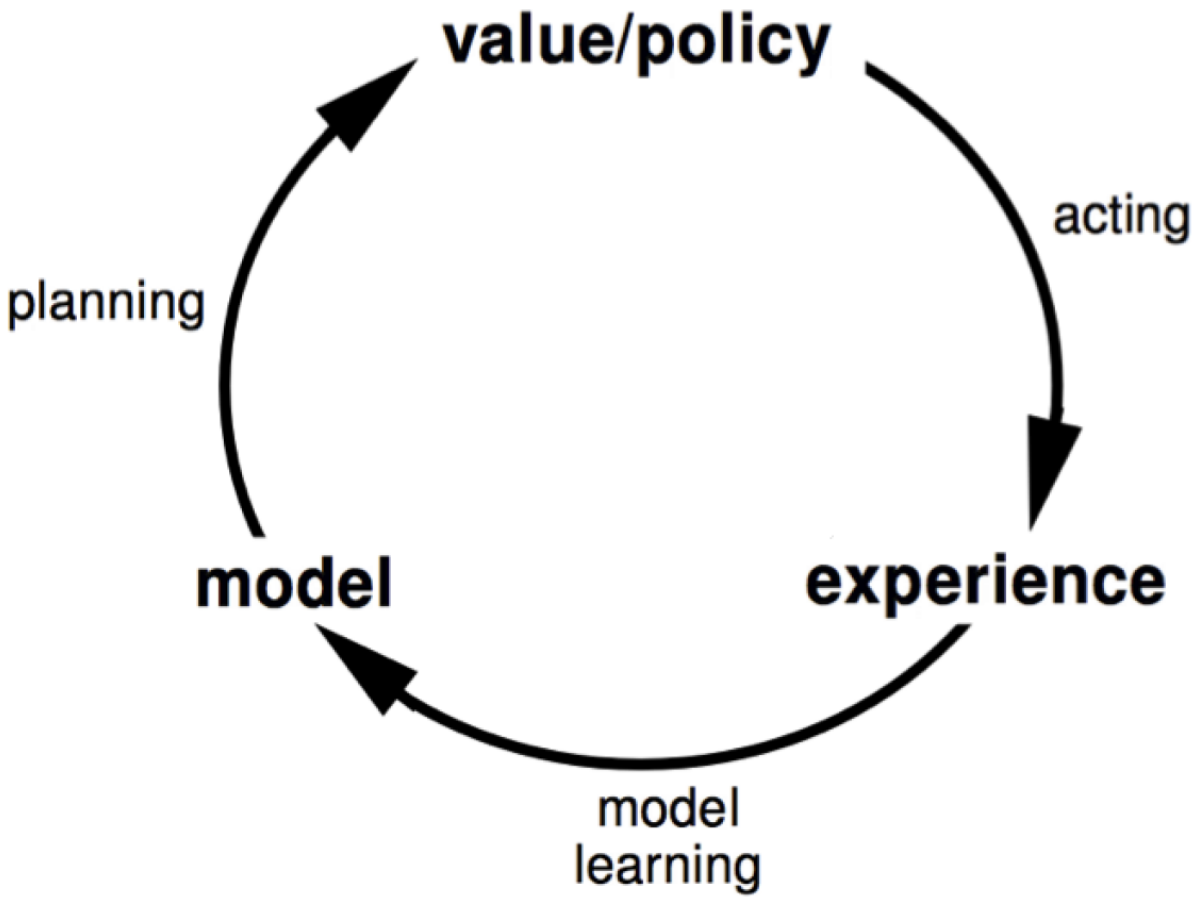
Model-Free RL



Model-Based RL



Model-Based RL



Advantages of Model-Based RL

Advantages:

- Can efficiently learn model by supervised learning methods
- Can reason about model uncertainty

Disadvantages:

- First learn a model, then construct a value function
  $\Rightarrow$ two sources of approximation error

What is a Model?

- A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η
- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
- So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

- Typically assume conditional independence between state transitions and rewards

$$\mathbb{P}\left[S_{t+1}, R_{t+1} \mid S_t, A_t\right] = \mathbb{P}\left[S_{t+1} \mid S_t, A_t\right] \mathbb{P}\left[R_{t+1} \mid S_t, A_t\right]$$

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
- This is a supervised learning problem

$$S_1, A_1 \rightarrow R_2, S_2$$
$$S_2, A_2 \rightarrow R_3, S_3$$
$$\vdots$$
$$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$$

- Learning $s, a \rightarrow r$ is a *regression* problem
- Learning $s, a \rightarrow s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- ...

Table Lookup Model

- Model is an explicit MDP, $\hat{\mathcal{P}}, \hat{\mathcal{R}}$
- Count visits $N(s, a)$ to each state action pair

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

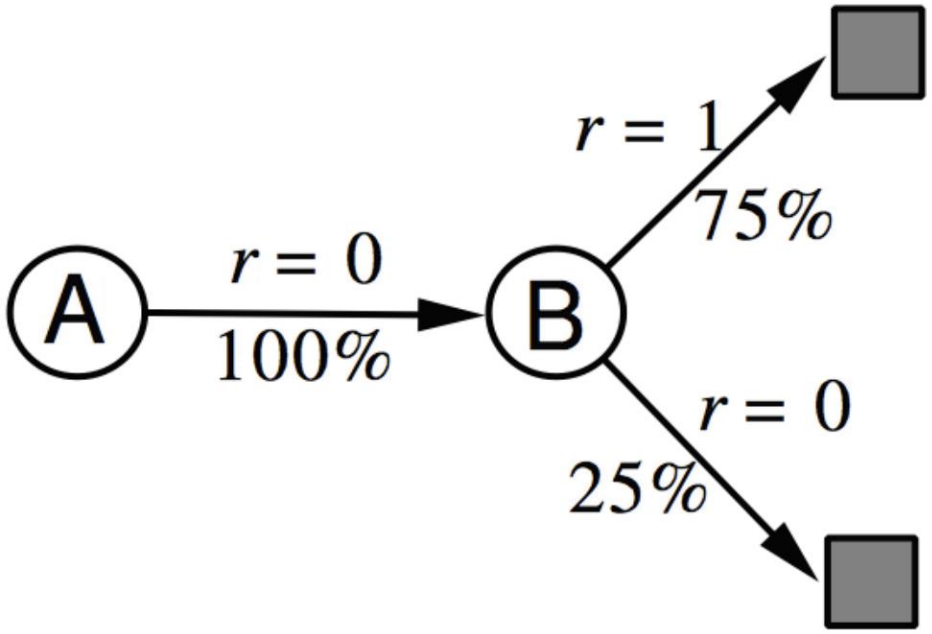
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

AB Example

Two states A, B; no discounting; 8 episodes of experience

$$\begin{array}{l} \mathrm{A}, 0, \mathrm{B}, 0 \\ \mathrm{B}, 1 \\ \mathrm{B}, 1 \\ \mathrm{B}, 1 \\ \mathrm{B}, 1 \\ \mathrm{B}, 1 \\ \mathrm{B}, 1 \\ \mathrm{B}, 0 \end{array}$$



We have constructed a table lookup model from the experience

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Using favourite planning algorithm
  - Value iteration
  - Policy iteration
  - Tree search
  - ...

Sample-Based Planning

- A simple but powerful approach to planning
- Use the model only to generate samples
- Sample experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

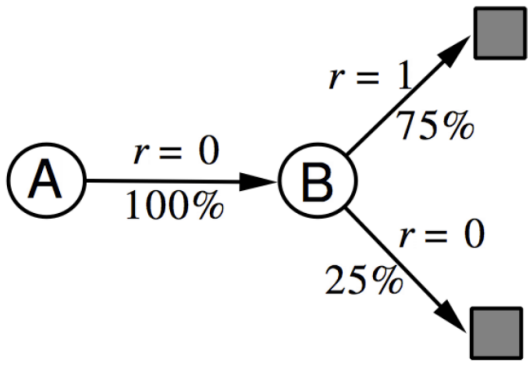
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Sampled experience

B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Planning with an Inaccurate Model

- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
- Solution 1: when model is wrong, use model-free RL
- Solution 2: reason explicitly about model uncertainty

Outline

1. Course Review
2. Background
3. Model-free learning
4. Model-based learning
5. Applications

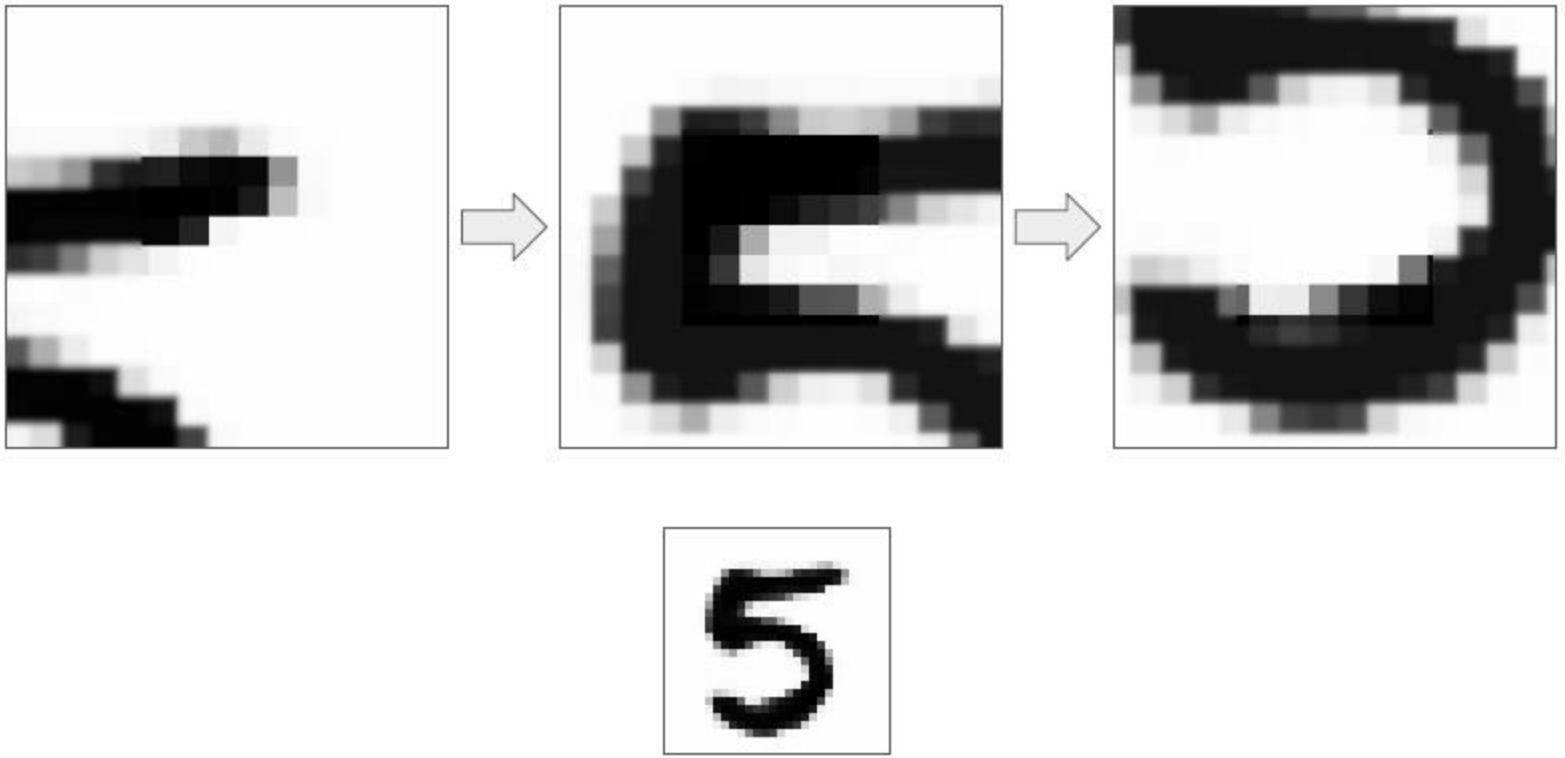
Applications

Recurrent Models of Visual Attention

Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu

Motivation

- **Task**: Classify digits in MNIST
- **Motivation**: Full image convolution is expensive!
  - Humans focus attention selectively on parts of an image
  - Combine information from different fixations over time



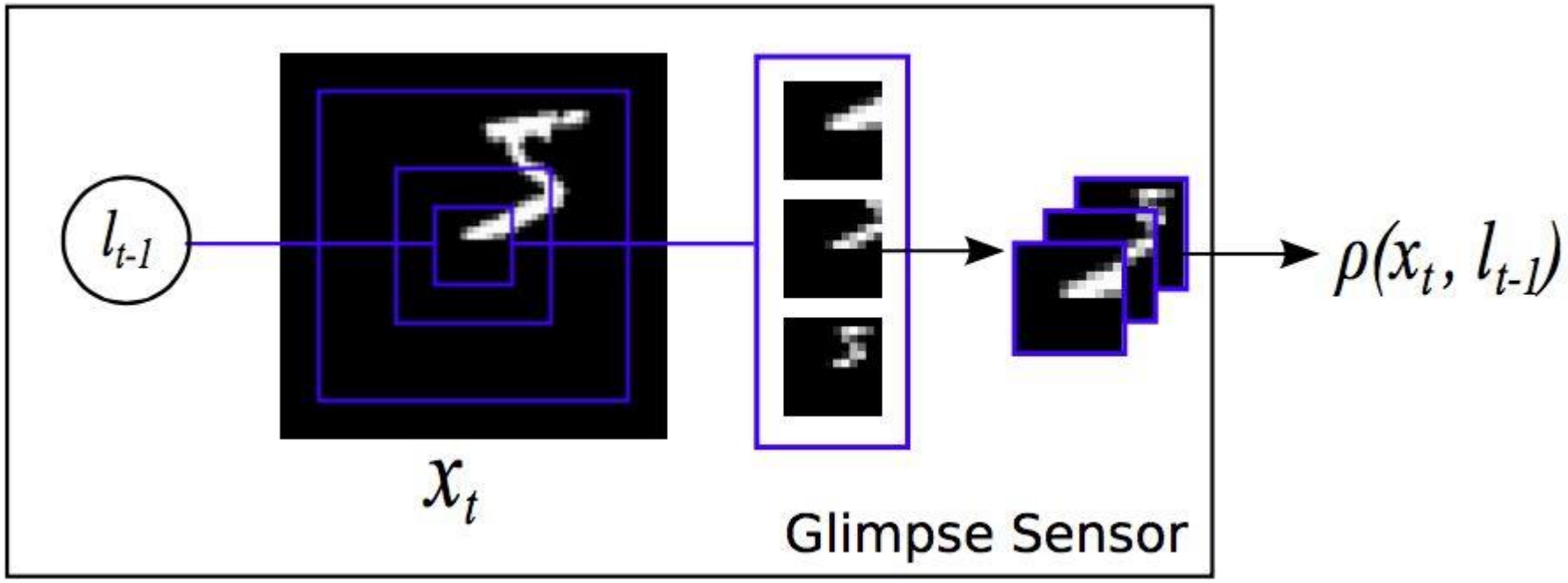
Overview

- True state of the environment is unobserved
  - **Glimpses** can be seen as a partial view of the state
- **State:** $h_t = f_h\,(h_{t-1}\,,\, g_t\,;\, \vartheta_h\,)$
- **Actions:**
  - Location: $l_t \sim p(.|f_l\,(h_t\,;\, \vartheta_l\,))$
  - An environment action: $a_t \sim p(.|f_a\,(h_t\,;\, \vartheta_a\,))$

- **Reward: Cross-Entropy Loss**
  - Agent needs to learn a stochastic policy
  - Policy π is defined by the Location Network in the RNN

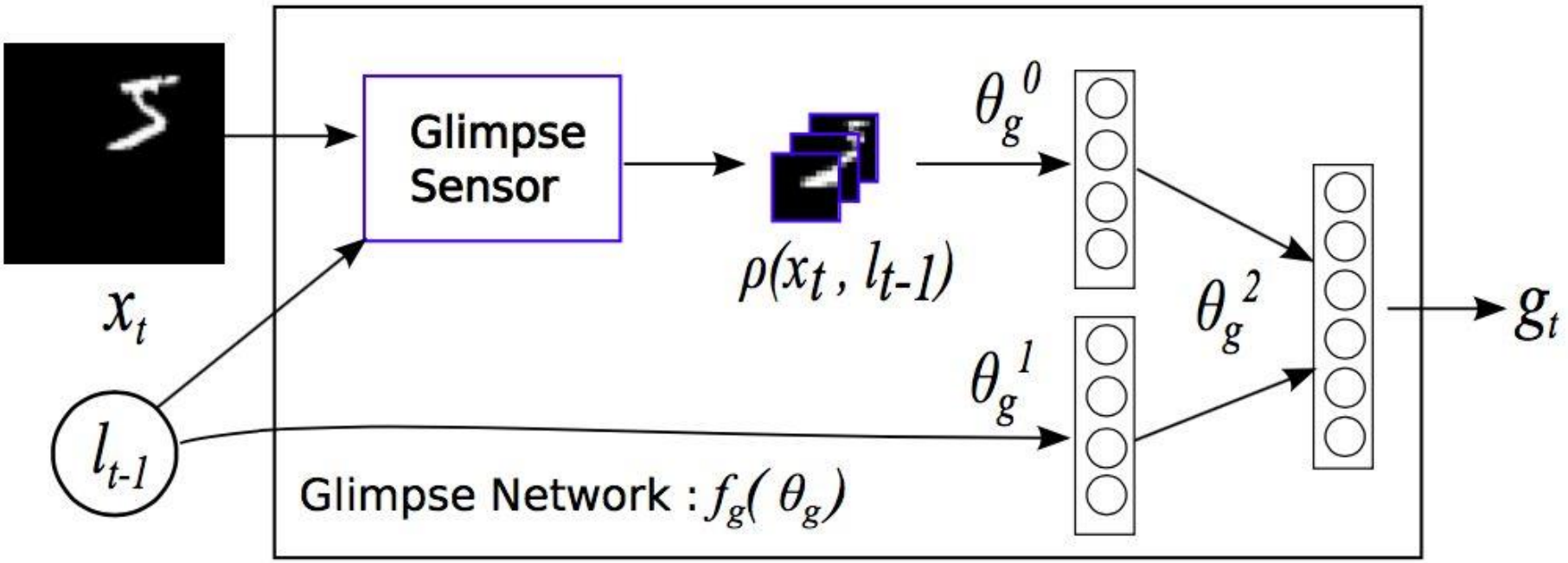
Glimpse

- Retina-like representation $\rho(x_t, l_{t-1})$
  - Contains multiple resolution patches
- Centered at location l_{t-1} of image x_t

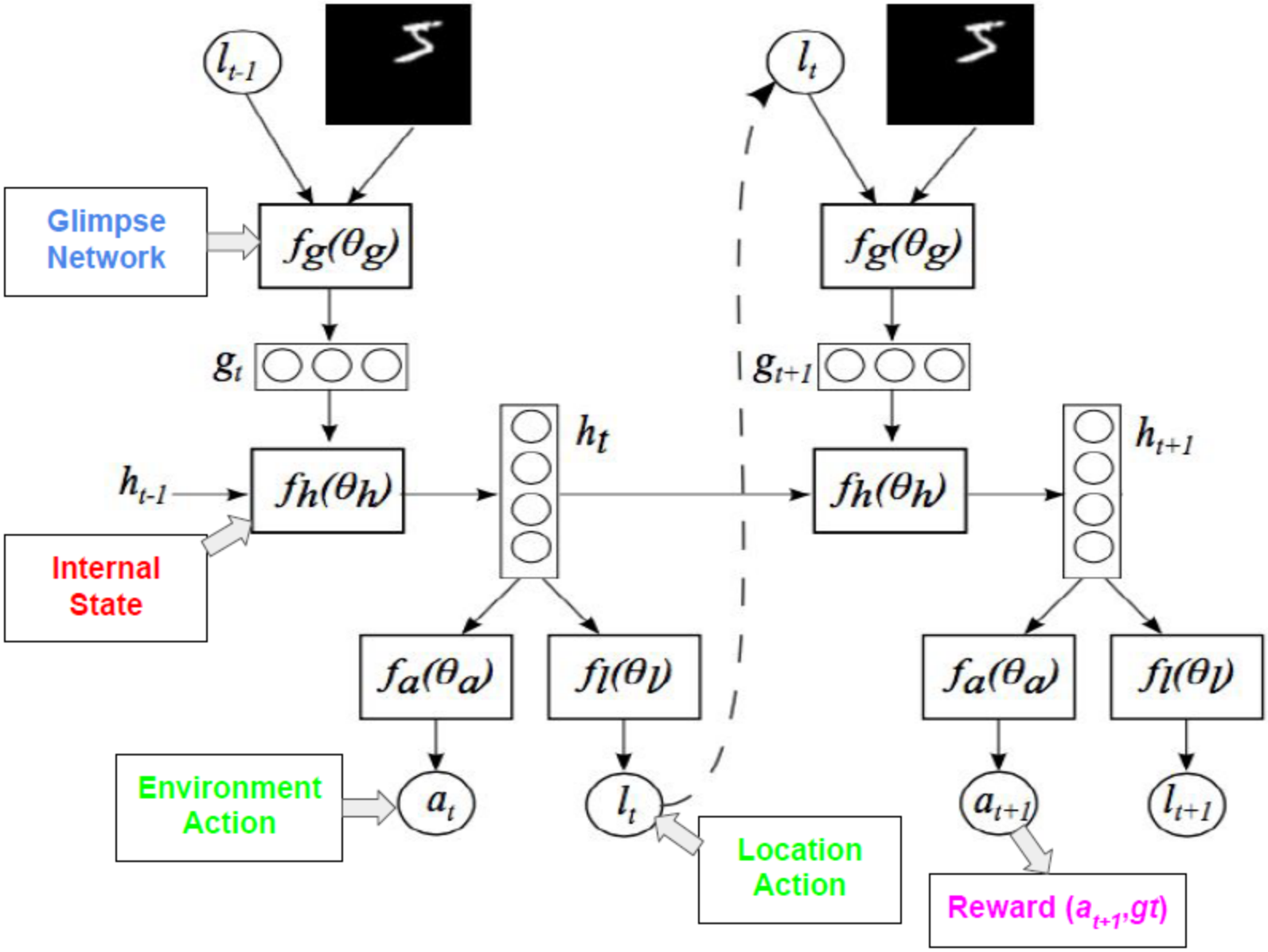


Glimpse Network

- ***ρ(x$_t$, l$_{t-1}$)*** *and* ***l$_{t-1}$*** *are mapped into a hidden space*



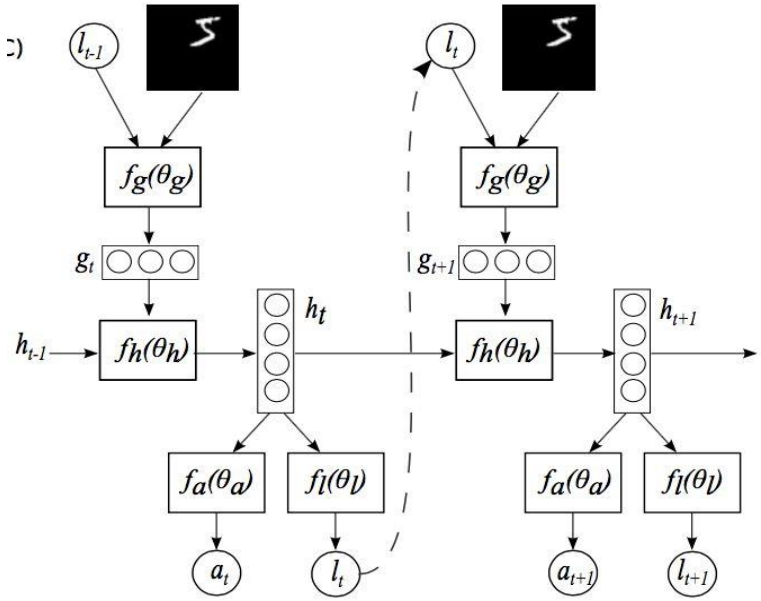
Model Architecture



Training

"OK, I've shown you the ropes, given you the low down, and gotten you up to speed. All that's left is actually training you."

Training



- Parameters of the agent are: $\vartheta = \{\vartheta_g, \vartheta_h, \vartheta_a\}$
  - Can be trained using standard backpropagation
- **RL Objective:** Maximize the reward given by: $J(\vartheta) = E[R]$
  - Can maximize $J(\vartheta)$ using REINFORCE

$$\nabla_\theta J = \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta)} \left[\nabla_\theta \log \pi(u_t | s_{1:t}; \theta) R\right] \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i | s_{1:t}^i; \theta) R^i$$

Results

Results

(a) 60x60 Cluttered Translated MNIST

Model	Error
FC, 2 layers (64 hiddens each)	28.58%
FC, 2 layers (256 hiddens each)	11.96%
Convolutional, 2 layers	8.09%
RAM, 4 glimpses, 12 × 12, 3 scales	4.96%
RAM, 6 glimpses, 12 × 12, 3 scales	4.08%
RAM, 8 glimpses, 12 × 12, 3 scales	4.04%
RAM, 8 random glimpses	14.4%

(b) 100x100 Cluttered Translated MNIST

Model	Error
Convolutional, 2 layers	14.35%
RAM, 4 glimpses, 12 × 12, 4 scales	9.41%
RAM, 6 glimpses, 12 × 12, 4 scales	8.31%
RAM, 8 glimpses, 12 × 12, 4 scales	8.11%
RAM, 8 random glimpses	28.4%

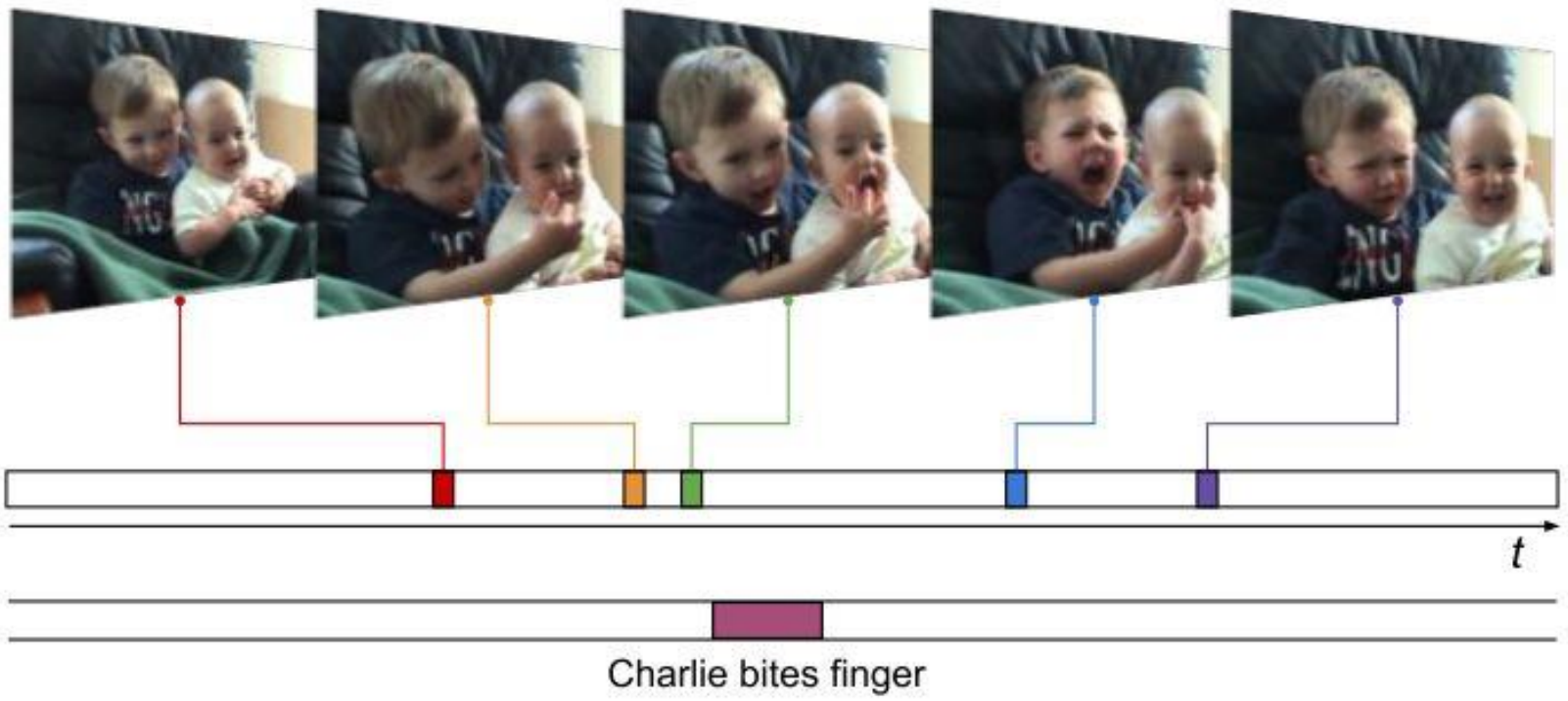
Applications

End-to-end Learning of Action Detection from Frame Glimpses in Videos

Serena Yeung, Olga Russakovsky, Greg Mori, Li Fei-Fei

Motivation

- **Task:** Detect and classify moments in an untrimmed video
- **Motivation**: Looking at all frames in a video is slow!
  - Process of detecting actions is one of observation and refinement

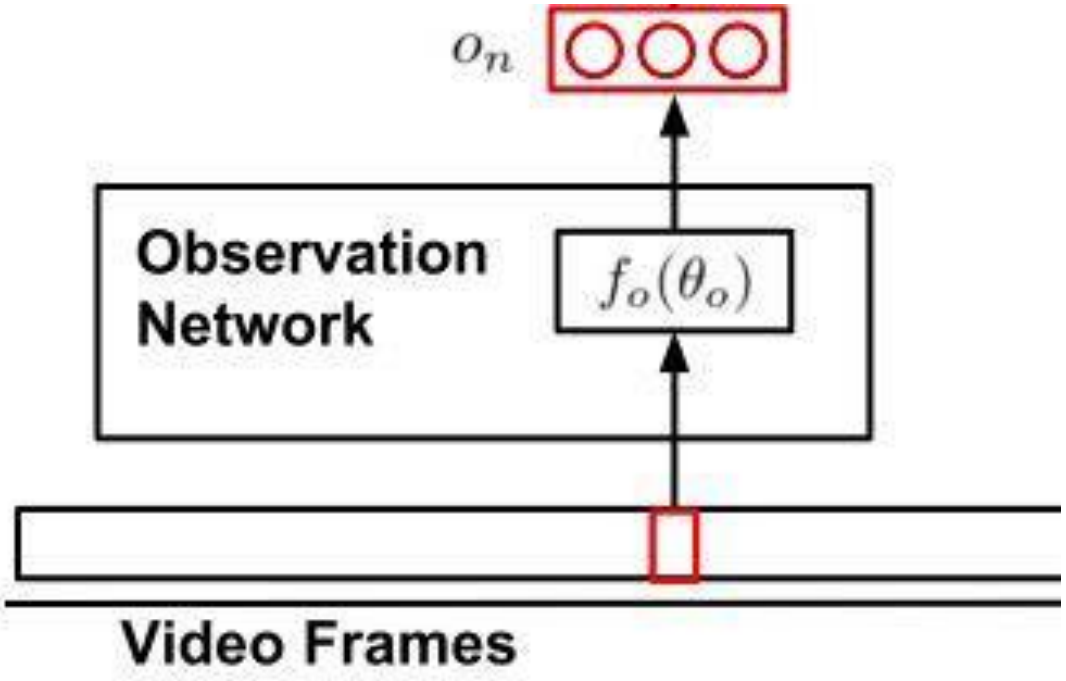


Overview

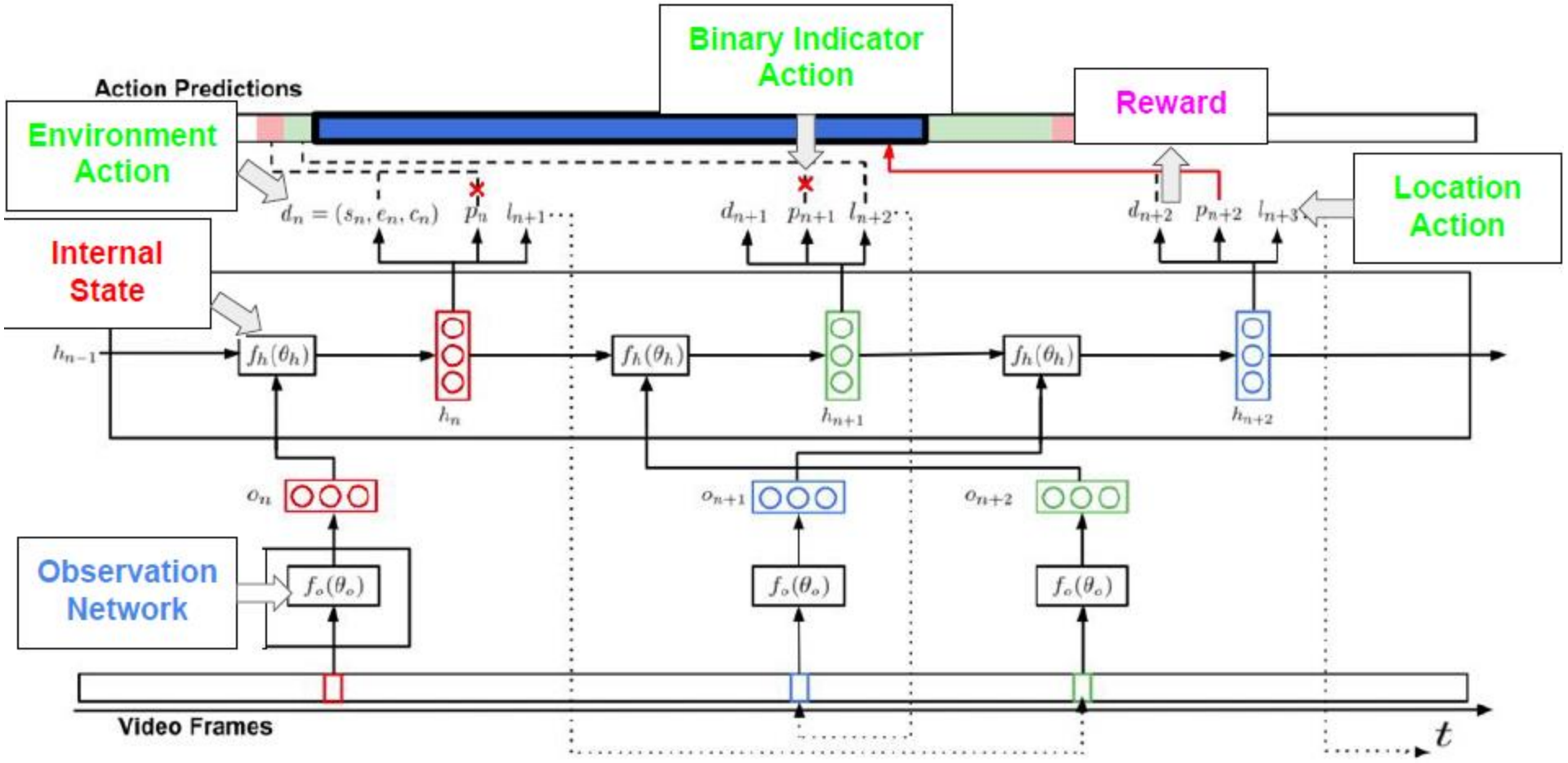
- True state of the environment is unobserved
  - **Observation Network** can be seen as a partial view of the state
- **State:** $h_n = f_h(h_{n-1}, o_n; \vartheta_h)$
- **Actions:**
  - Candidate detection: $d_n = f_d(h_n; \vartheta_d)$
  - Binary indication: $p_n = f_p(h_n; \vartheta_p)$
  - Temporal location: $l_{n+1} = f_l(h_n; \vartheta_l)$
- **Reward** $$R_N = \begin{cases} R_0 & \text{if } M > 0 \text{ and } N_p = 0 \\ N_+ R_+ + N_- R_- & \text{otherwise} \end{cases}$$
- Agent needs to learn a stochastic policy
  - Policy π is defined by the Location Network in the RNN

Observation Network

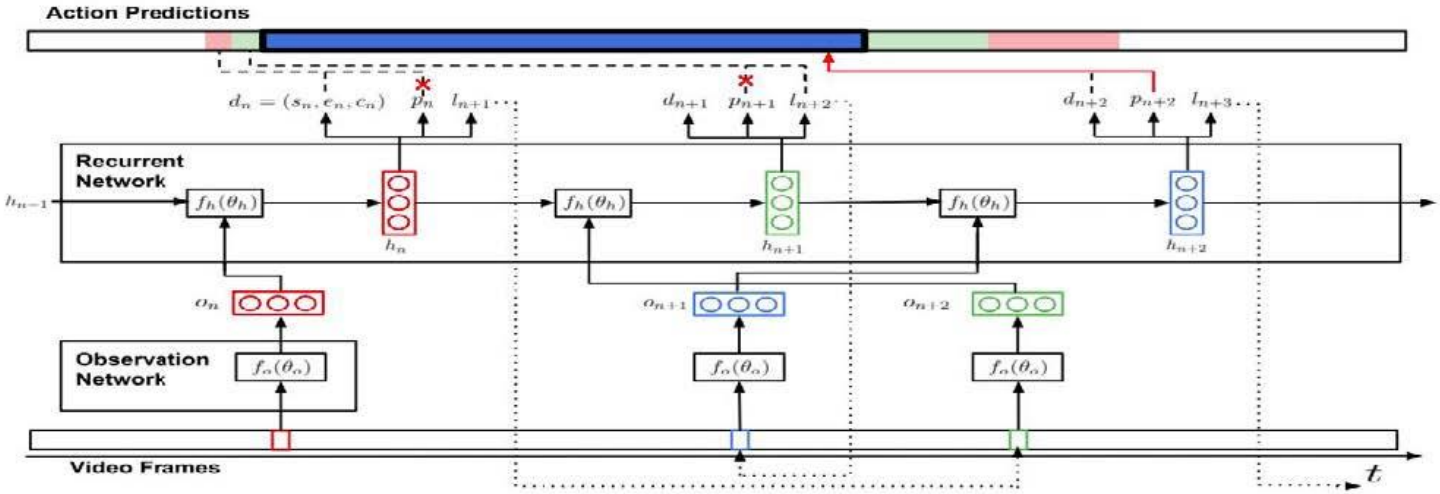
- Observes a single video frame at each timestep and encodes the frame and it's location into a feature vector $\boldsymbol{o_n}$
  - Inspired by the **Glimpse network**



Model Architecture



Training



- Parameters of the agent are: ***ϑ = { ϑo, ϑh, ϑd }***
  - Can be trained using standard backpropagation
- **RL Objective: Maximize the reward given by: *J(ϑ) = E[R]***

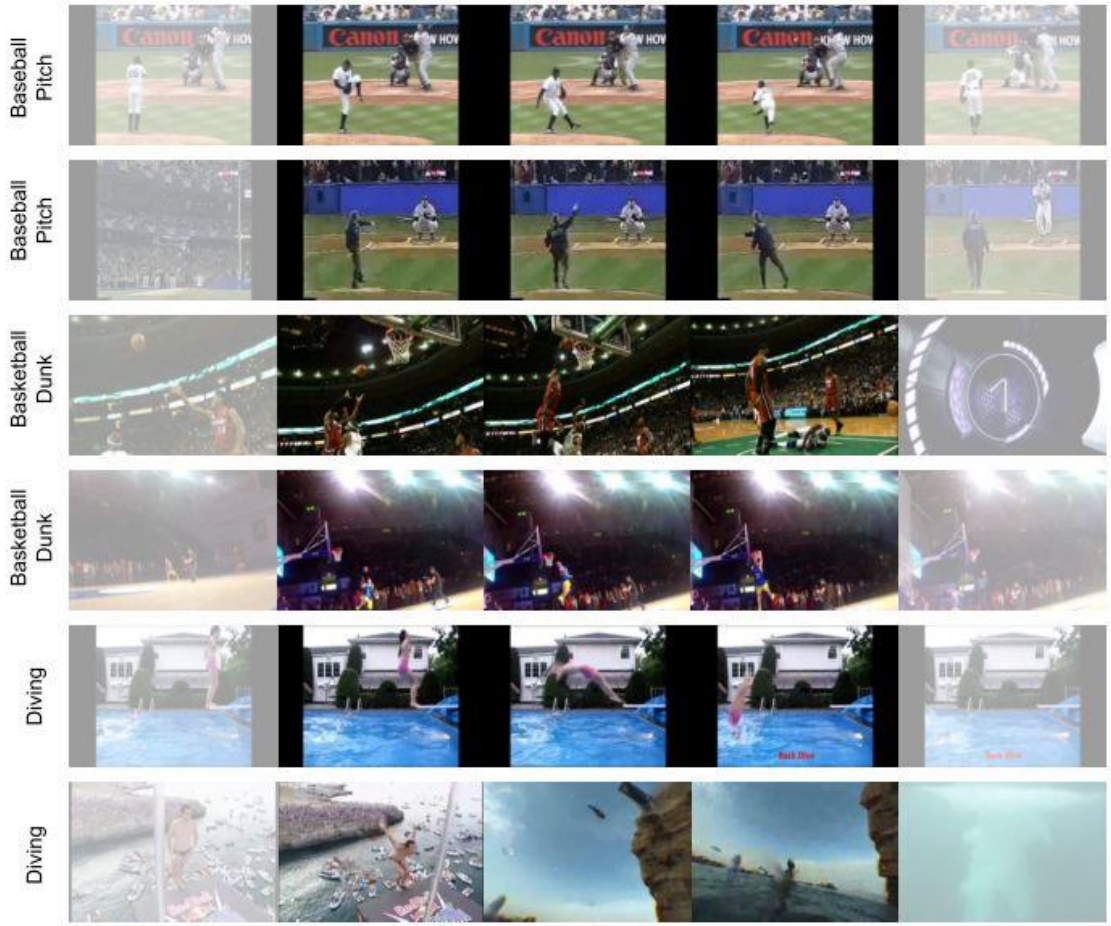
$$L(D) = \sum L_{cls}(d_n) + \gamma \sum \sum \mathbb{1}[y_{nm} = 1] L_{loc}(d_n, g_m)$$

Can maximize $J(\theta)$ using **REINFORCE**

$$\nabla_\theta J = \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta)} \left[\nabla_\theta \log \pi(u_t | s_{1:t}; \theta) R \right] \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i | s_{1:t}^i; \theta) R^i$$

Results - I

- THUMOS 14’ Dataset
  - Correct Predictions



Results - II

	[23]	Ours		[23]	Ours
Baseball Pitch	8.6	**14.6**	Hamm. Throw	**34.7**	28.9
Basket. Dunk	1.0	**6.3**	High Jump	17.6	**33.3**
Billiards	2.6	**9.4**	Javelin Throw	**22.0**	20.4
Clean and Jerk	13.3	**42.8**	Long Jump	**47.6**	39.0
Cliff Diving	**17.7**	15.6	Pole Vault	**19.6**	16.3
Cricket Bowl.	9.5	**10.8**	Shotput	11.9	**16.6**
Cricket Shot	2.6	**3.5**	Soccer Penalty	**8.7**	8.3
Diving	4.6	**10.8**	Tennis Swing	3.0	**5.6**
Frisbee Catch	1.2	**10.4**	Throw Discus	**36.2**	29.5
Golf Swing	**22.6**	13.8	Volley. Spike	1.4	**5.2**
mAP				14.4	**17.1**

- **Key Takeaways:**
  - Accuracy is comparable to state-of-the-art
  - Less frames observed

Disentangle latent factor

AlphaGo:

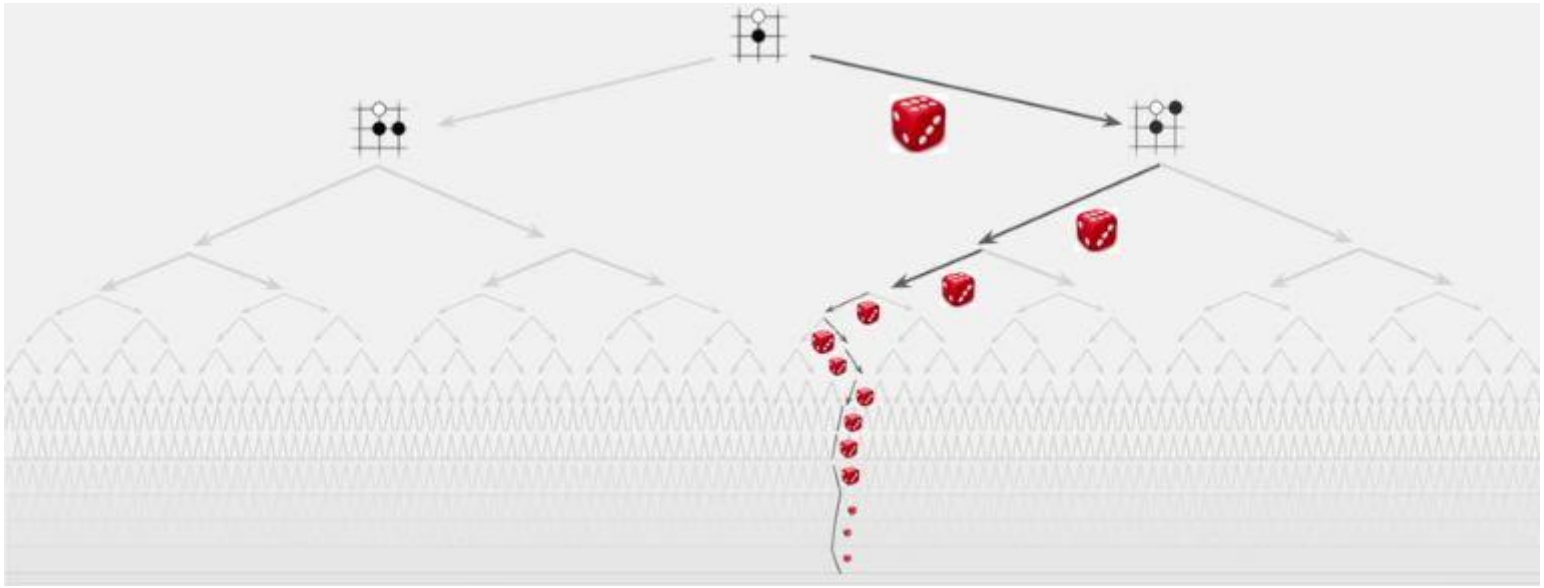
A bit of everything

(but mostly plain PG + planning)

https://www.youtube.com/watch?v=4D5yGiYe8p4

Background: Monte-Carlo Tree Search

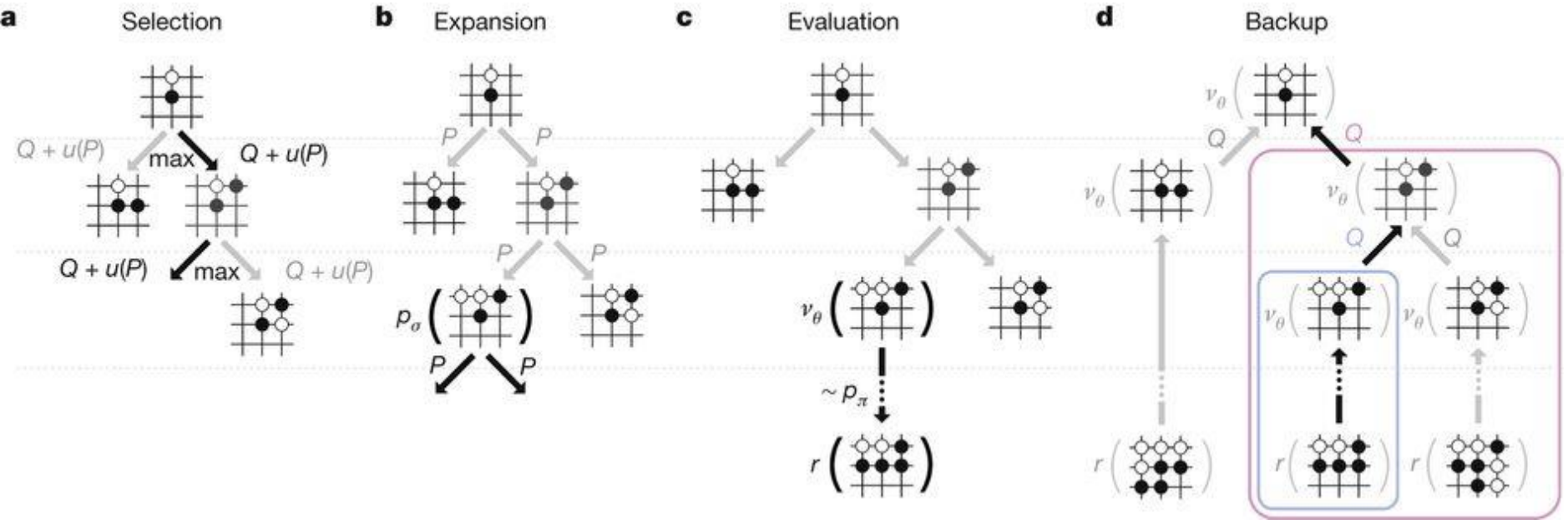
- Another planning method.
- Sample future paths using stochastic policy
  - Biased towards reasonable moves
  - The predictron paper may do this if they modeled the environment $\mathbb{P}(s'|s,a)$.



Background: Monte-Carlo Tree Search

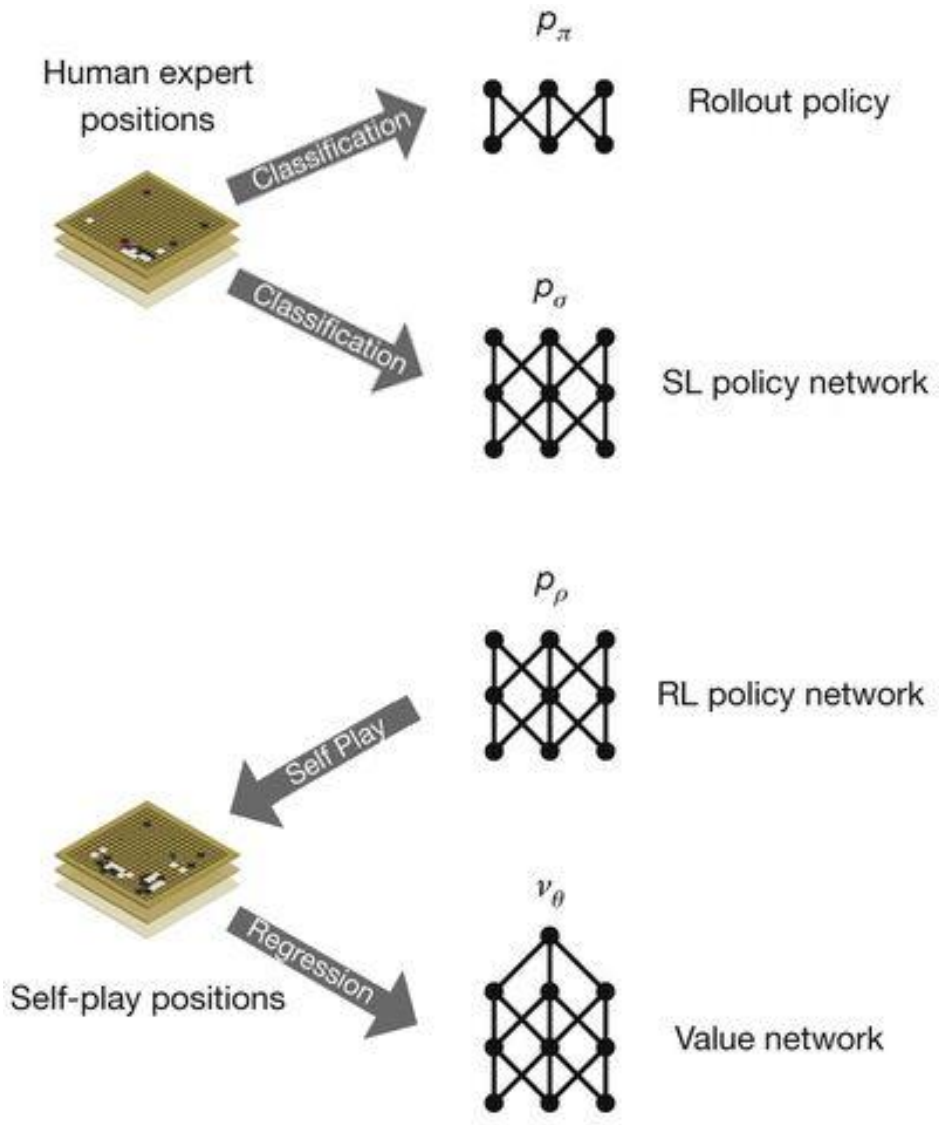
Deterministic environment version.

- 1. Select path according to π plus exploration
- 2. Expand leaf node s (compute children and their $\mathbb{P}(\cdot)$)
- 3. Evaluate V(s) by rolling out (play till the end)
- 4. Backup: update Q(s,a) along the path (count)



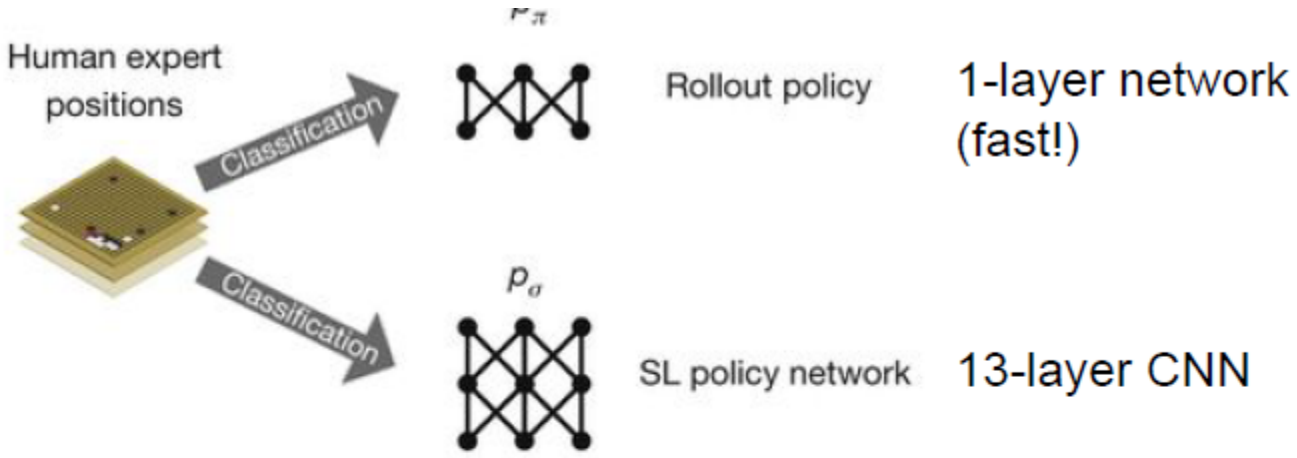
D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

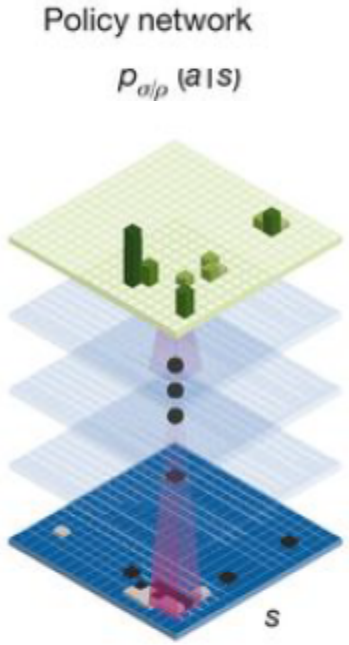
AlphaGo models overview



D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models

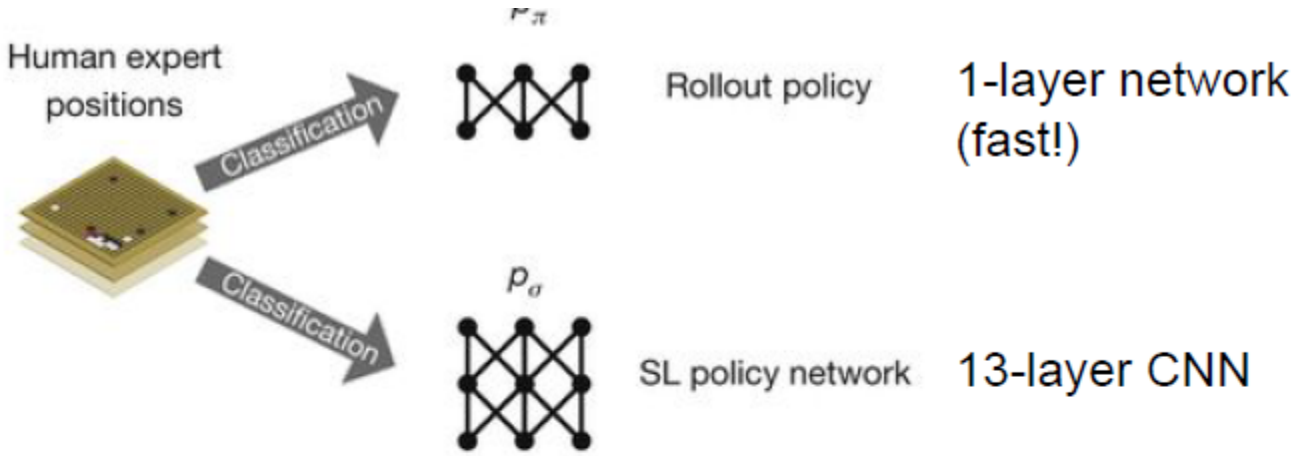


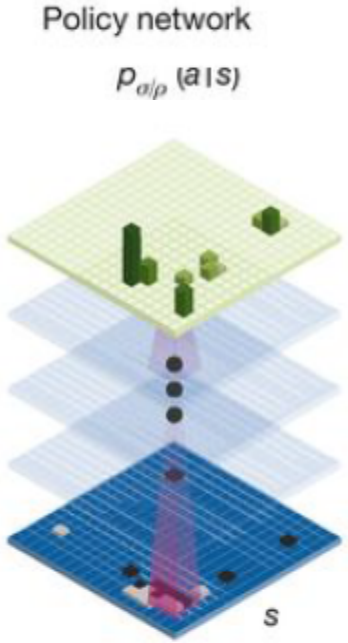


- Supervised learning
  - On human expert moves
  - One small (very fast rollout)
    - 2μs; 24.2% accuracy
  - One deeper
    - 57% accuracy w/ handcrafted features;
    - 55.7% using only raw board + past move

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

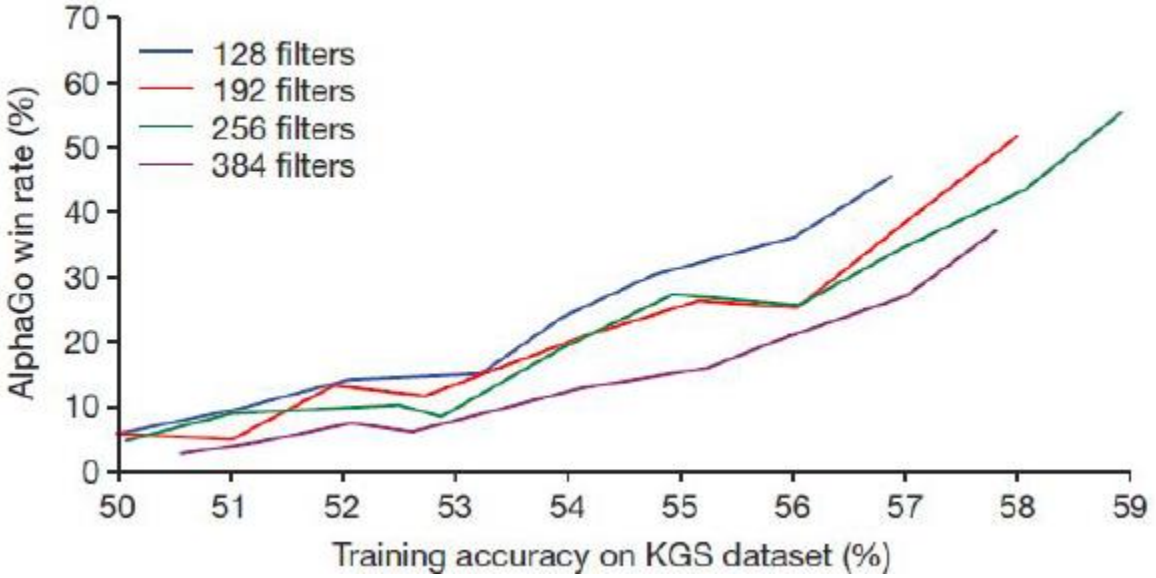
AlphaGo models





Importance of classification accuracy

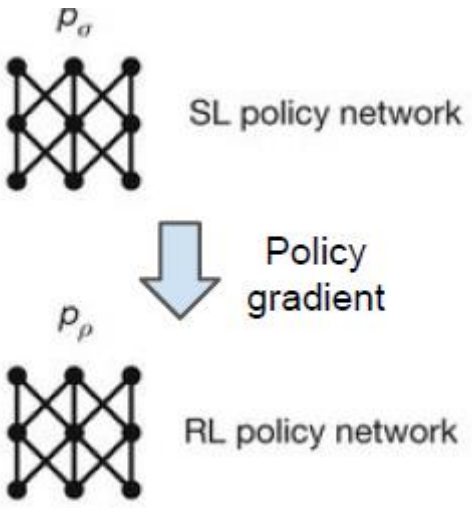
(win rate against final AlphaGo)



D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models

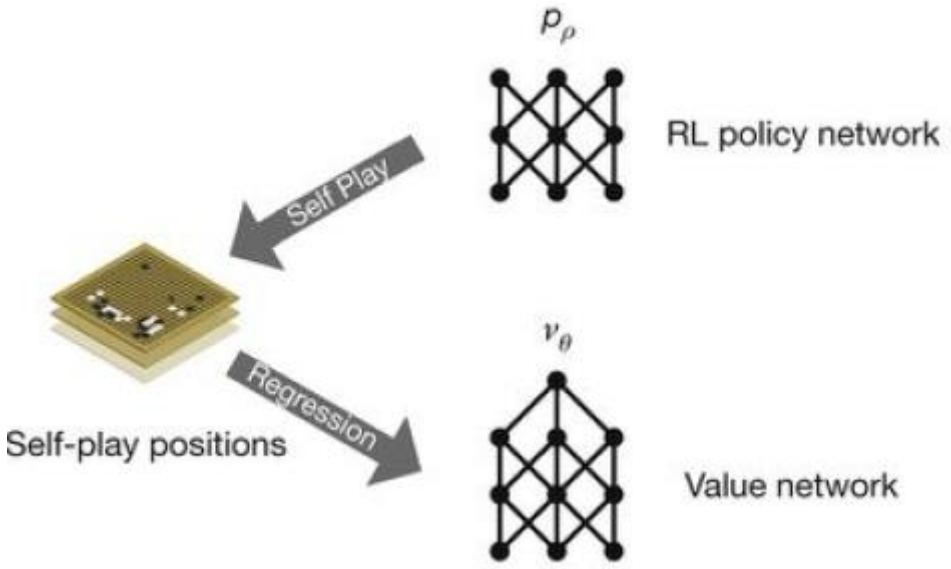
- Policy gradient
  - Improve SL policy to RL policy
    - Playing against its past iterations (less overfitting)

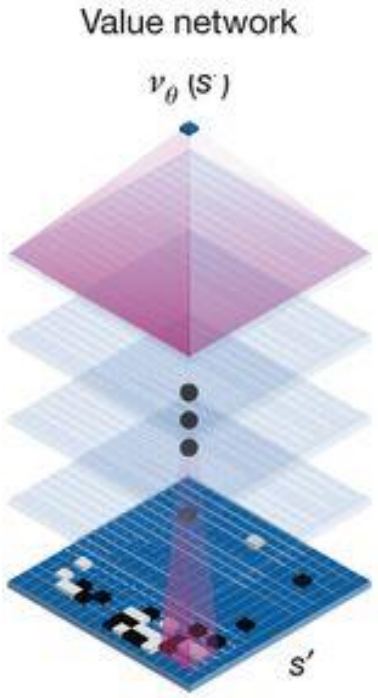


  - Training: PG w/o discount (rewards R_{win} = +1;R_{lose}= -1)
- Wins 80% against SL policy
  - 85% to Pachi (open source s-o-t-a)
  - Ranks ~ 3 amateur dan

AlphaGo models

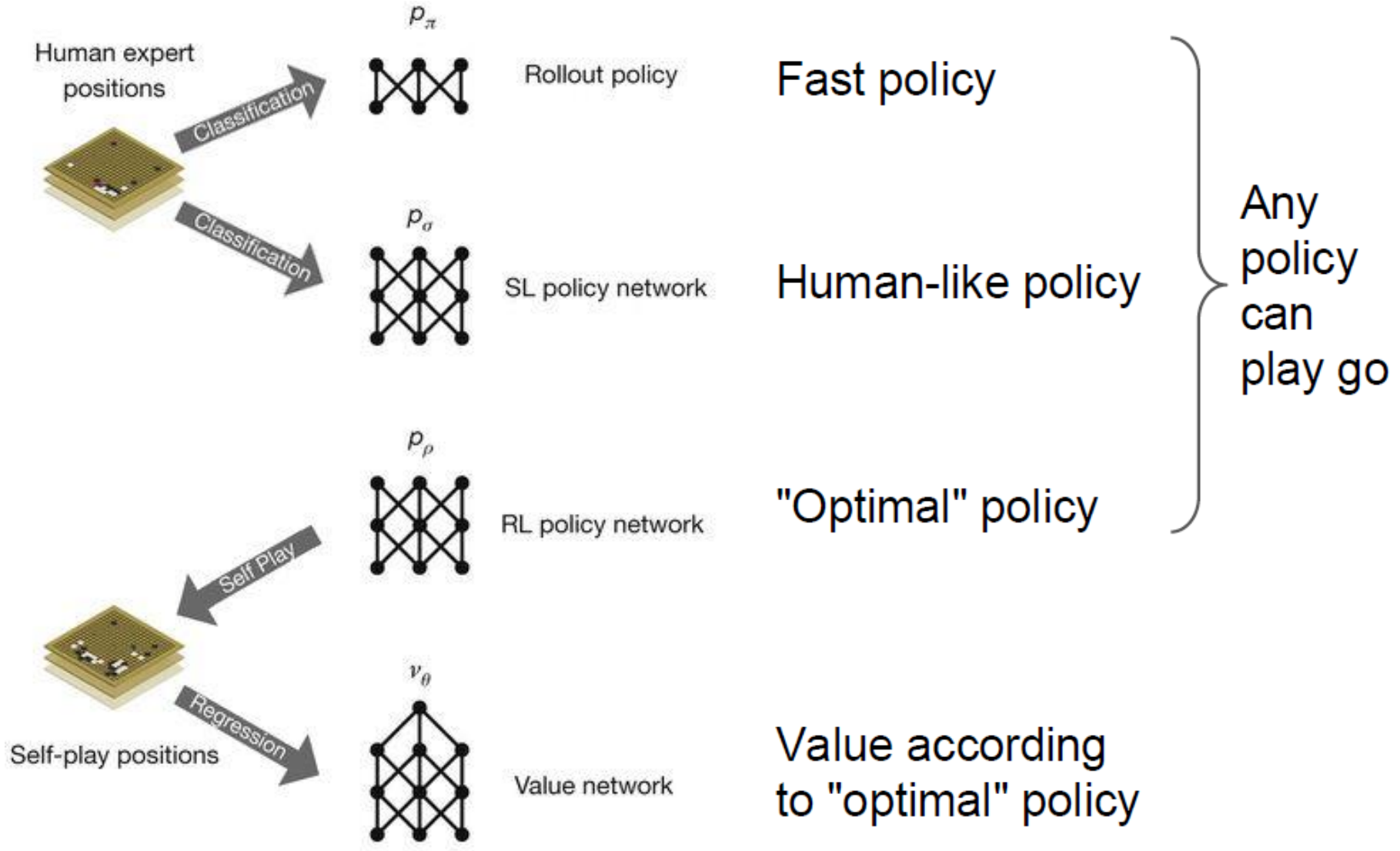
- Value network: evaluate the win-rate of state
  - Use self-play instead of human moves (less overfit)
  - Under "optimal policy" (the RL one)
  - David: "perhaps the key of AlphaGo dev.“
    - (first strong state evaluator)





D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

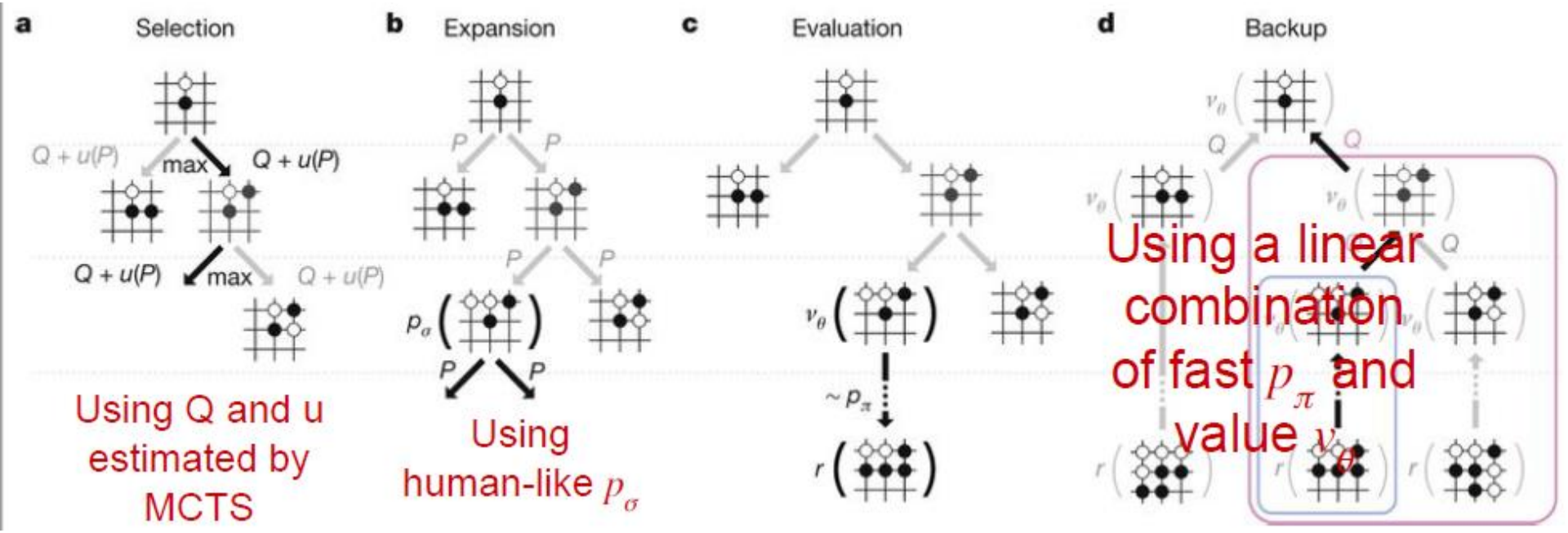
AlphaGo models recap



D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

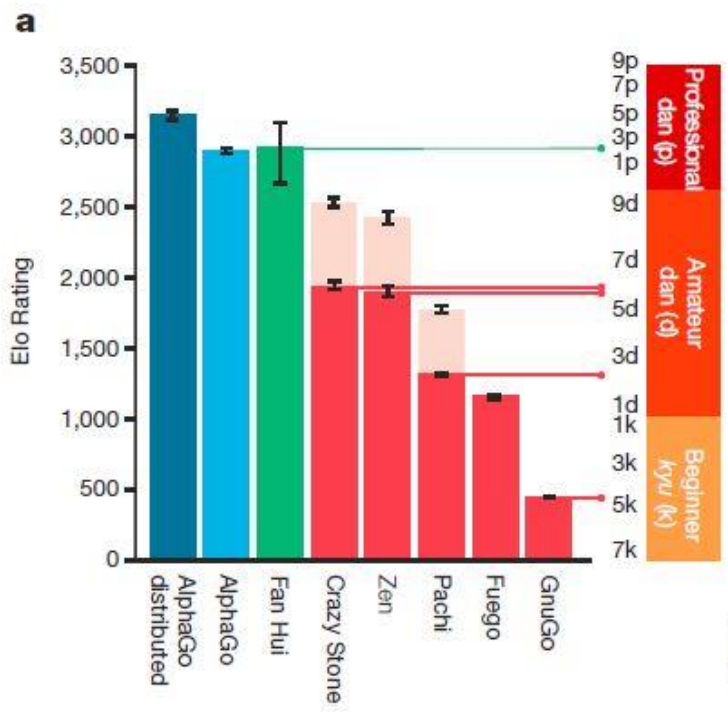
Putting everything together w/ MCTS

- - Deterministic environment version.
- 1. Select path by maximizing estimated *Q* and exploration *u*
- 2. Expand leaf node s(compute children's $\mathbb{P}(\cdot)$ using P_σ),
- 3. Evaluate V(s) by rolling out (using fast P_π and value v_θ)
- 4. Backup: update *Q*(s,*a*) along the path (using count)

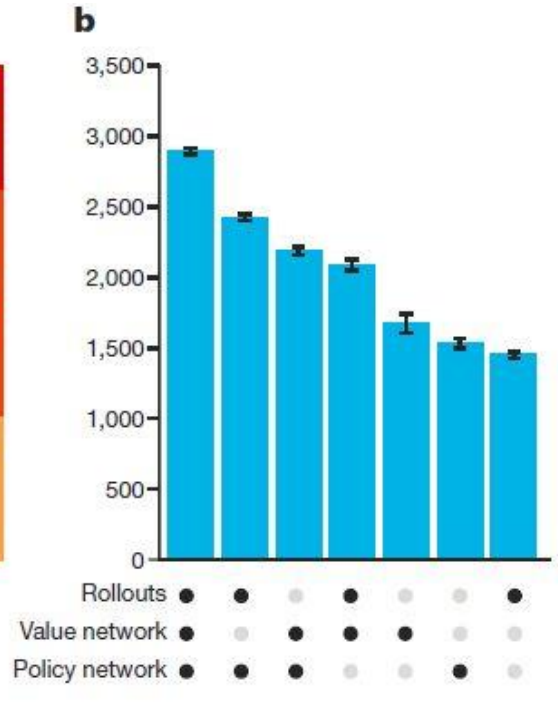


D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

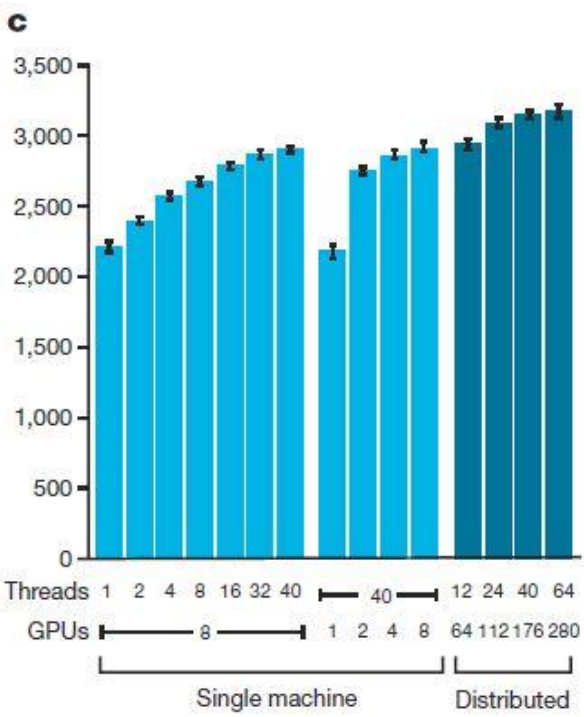
AlphaGo results



AlphaGo vs human or related work



Ablation study of the components



Ablation study of distributed MCTS

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course

- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course

- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course

- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course

Next time

- Attention and Memory

Questions?

Thank You !

WeChat Group for Deep Learning