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

2022

"Deep Learning Lecture"

Lecture 9: Reinforcement Learning

Yan Huang

Center for Research on Intelligent Perception and Computing (CRIPAC)

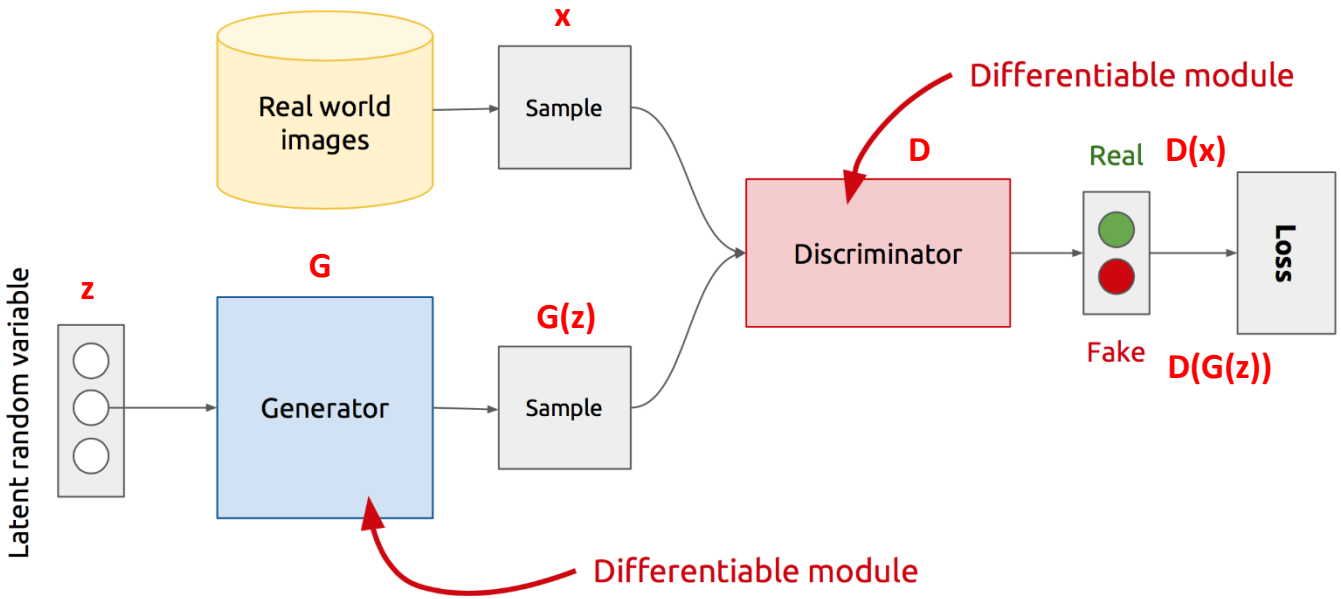
National Laboratory of Pattern Recognition (NLPR)

Institute of Automation, Chinese Academy of Science (CASIA)

Outline

1. Course Review
2. Background
3. Model-free Learning
4. Applications

GAN's Architecture



- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

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:
  - $P_{data}(x) = P_{gen}(x) \;\; \forall x$
  - $\mathrm{D}(x) = \frac{1}{2} \;\; \forall x$

Adversarial Training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

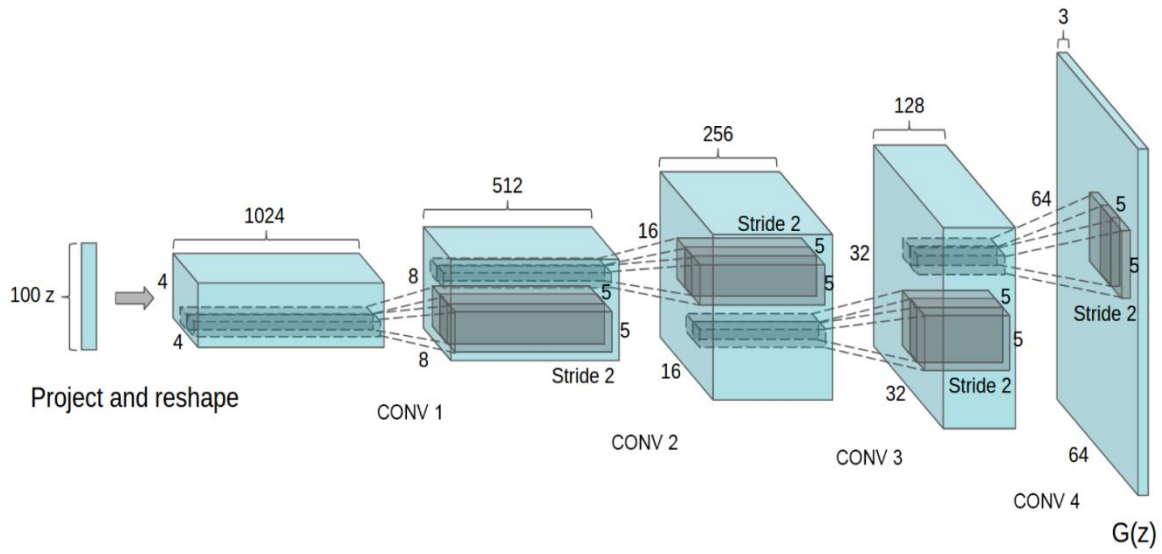
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Discriminator updates

Generator updates

Deep Convolutional GANs (DCGANs)

Generator Architecture



Key ideas:

- Replace FC hidden layers with Convolutions
  - **Generator:** Fractional-Strided convolutions
- Use Batch Normalization after each layer
- **Inside Generator**
  - Use ReLU for hidden layers
  - Use Tanh for the output layer

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

DCGAN: Bedroom images

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

Text-to-Image Synthesis

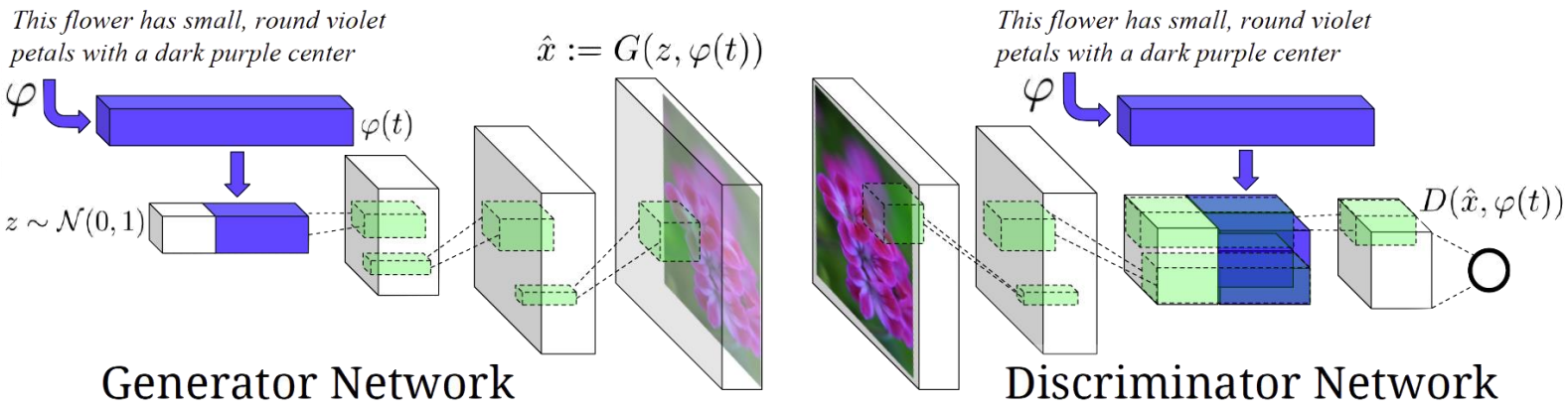


Figure 2 in the original paper.

Positive Example:
Real Image, Right Text

Negative Examples:
Real Image, Wrong Text
Fake Image, Right Text

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. "**Generative adversarial text to image synthesis**". ICML (2016).

Text-to-Image Synthesis

Given a text description, generate images closely associated.

Uses a conditional GAN with the generator and discriminator being condition on “dense” text embedding.

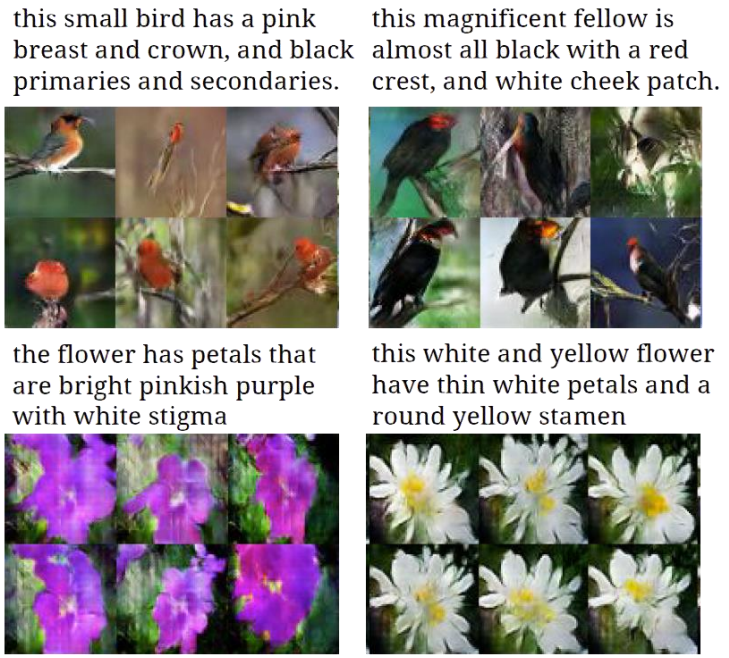
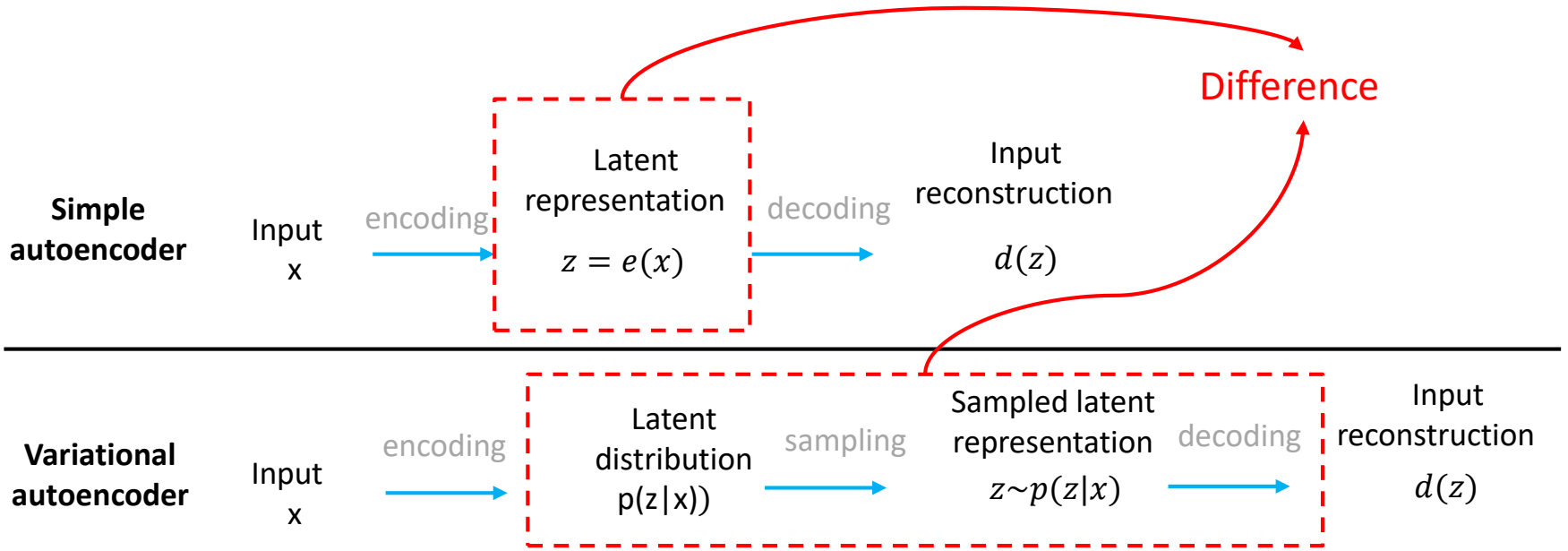


Figure 1 in the original paper.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. “**Generative adversarial text to image synthesis**”. ICML (2016).

Autoencoder vs Variational Autoencoder

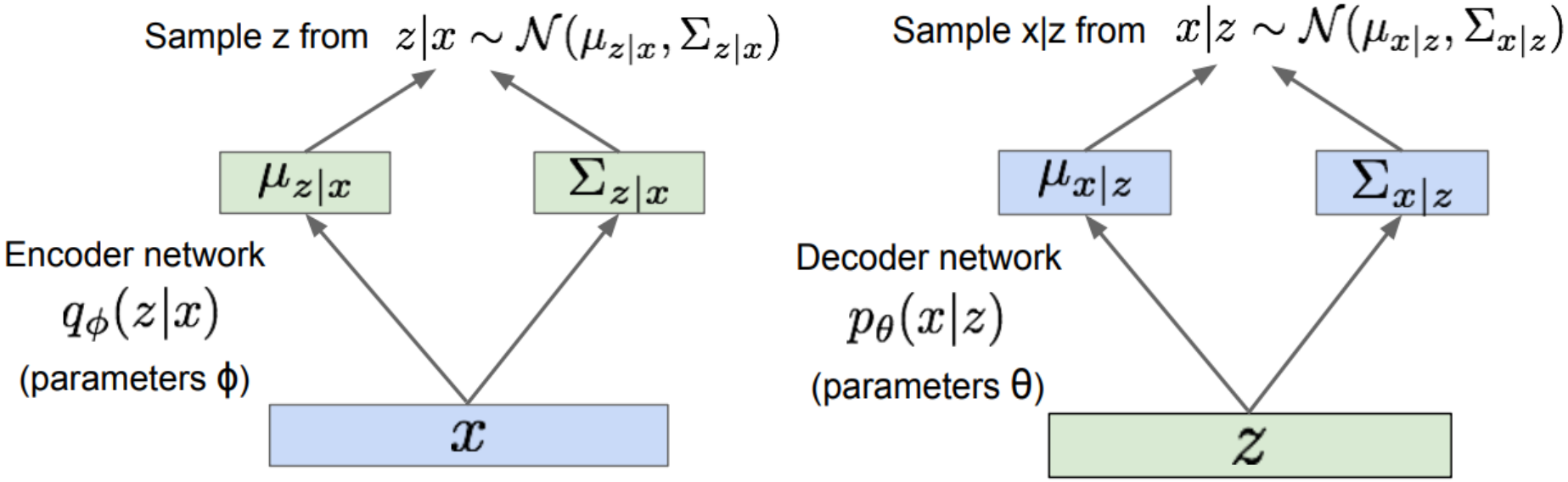


Difference between autoencoder (deterministic) and variational autoencoder (probabilistic)

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders

- Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Encoder and decoder networks also called "inference" and "generation" networks

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

Taking expectation wrt. z (using encoder network) will come in handy later

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{>0}$$

Reconstruct the input data

Make approximate posterior distribution close to prior

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Variational Autoencoders

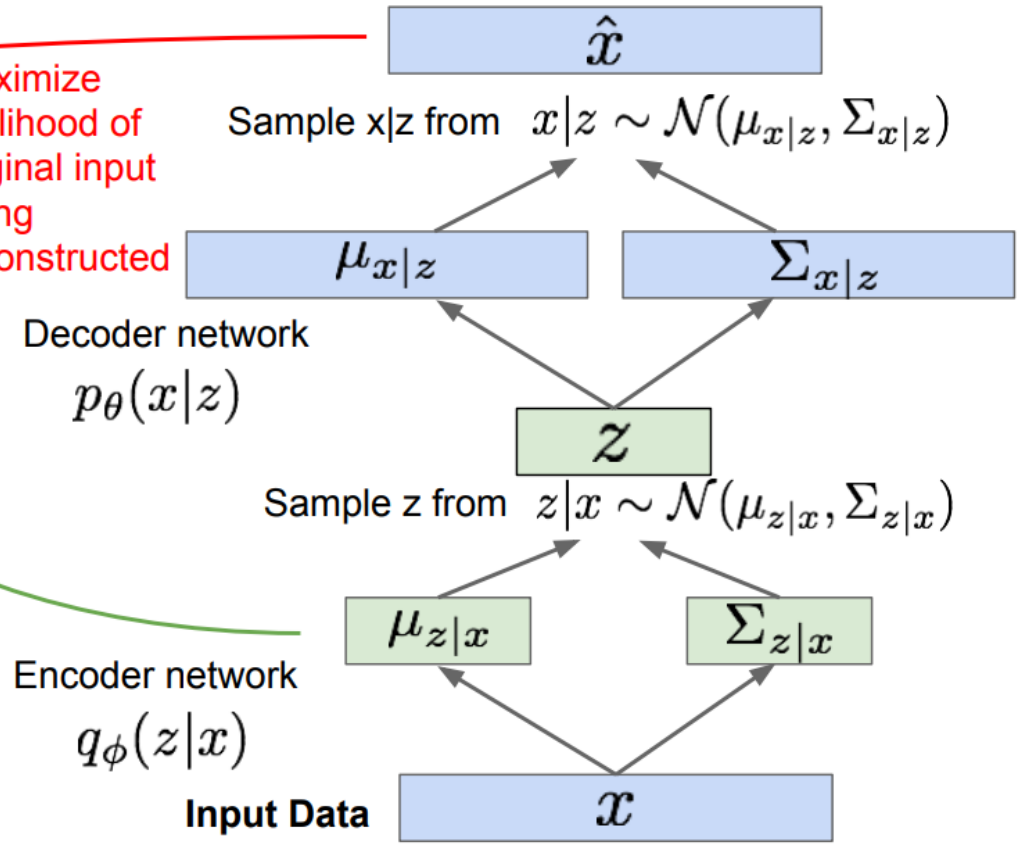
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!



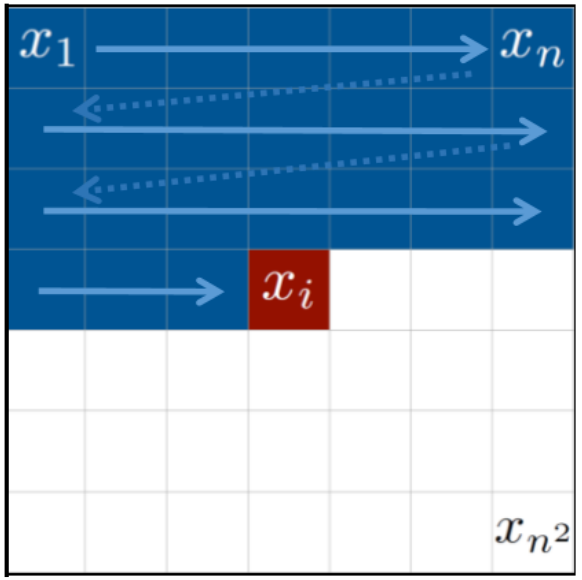
Pixel Generation

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

Bayes Theorem:

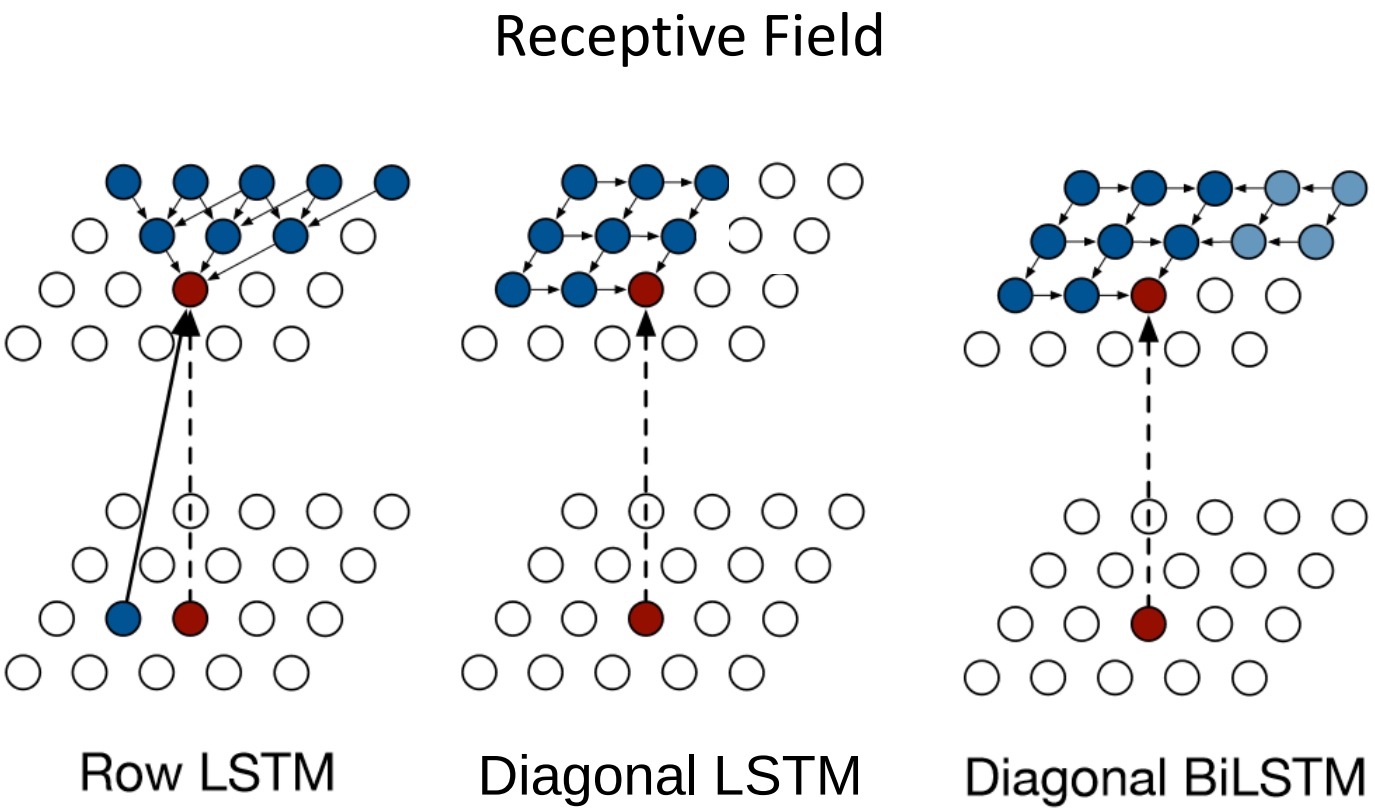
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

A sequential model!



Pixel recurrent neural networks, ICML 2016

PixelRNN: A Specific Multidimensional LSTM



Pixel recurrent neural networks, ICML 2016

Results

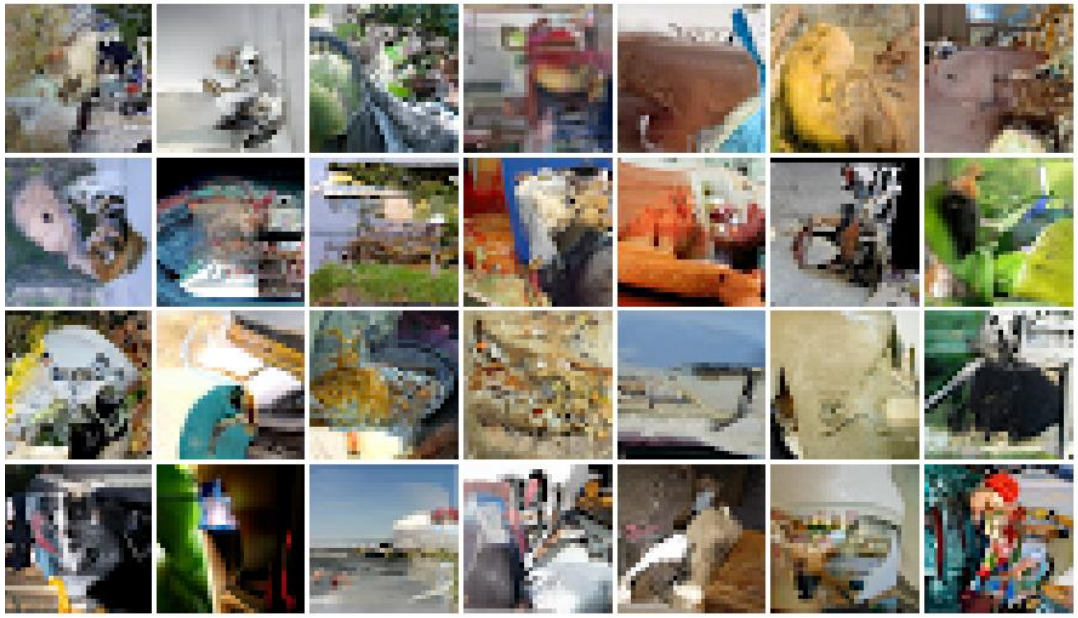


Figure: 32x32 ImageNet results from Diagonal BiLSTM model.

Figure from: Oord et al.

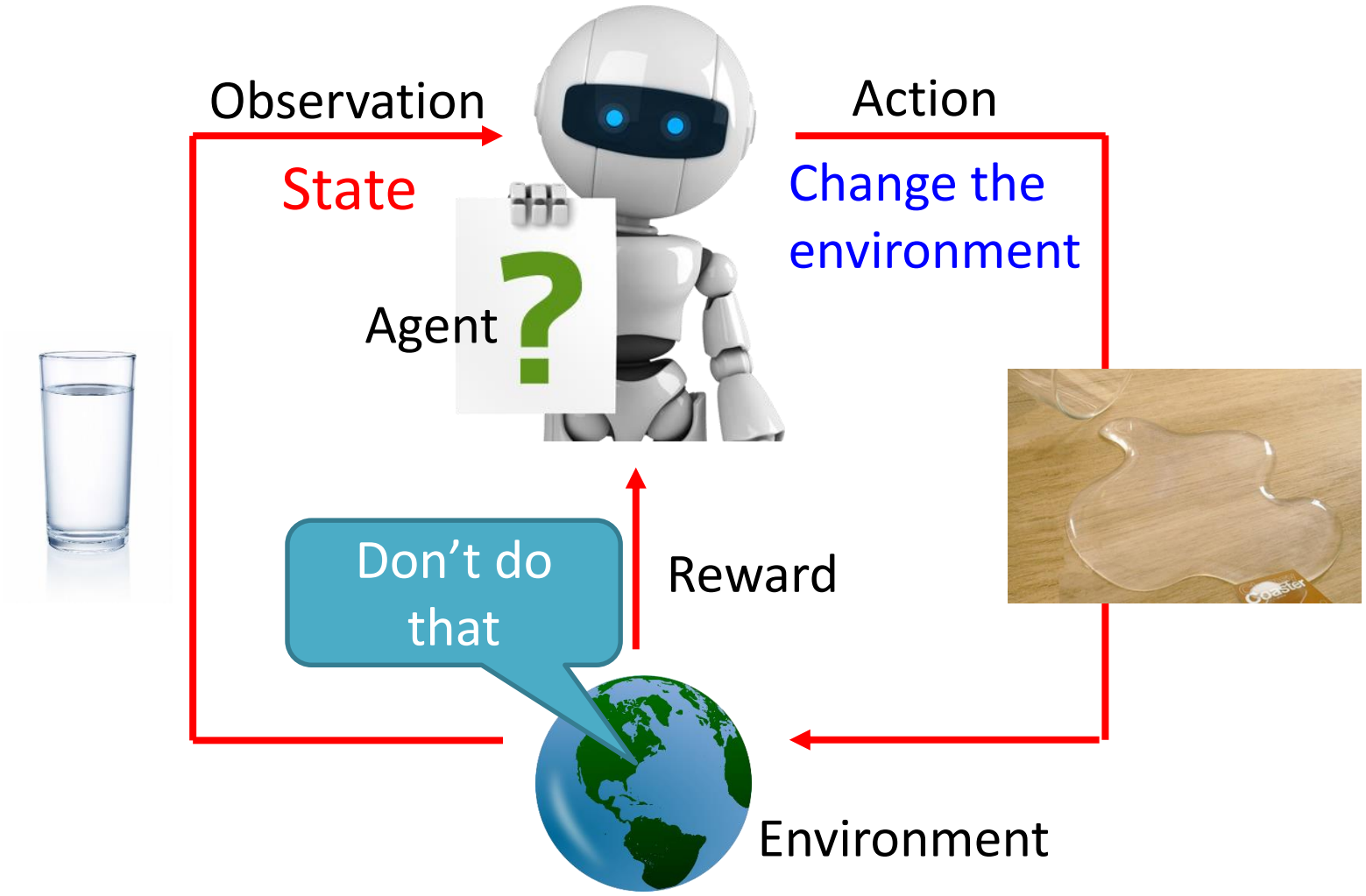
Outline

1. Course Review
2. Background
3. Model-free Learning
4. Applications

Background

Deep Reinforcement Learning: AI = RL + DL

Scenario of Reinforcement Learning



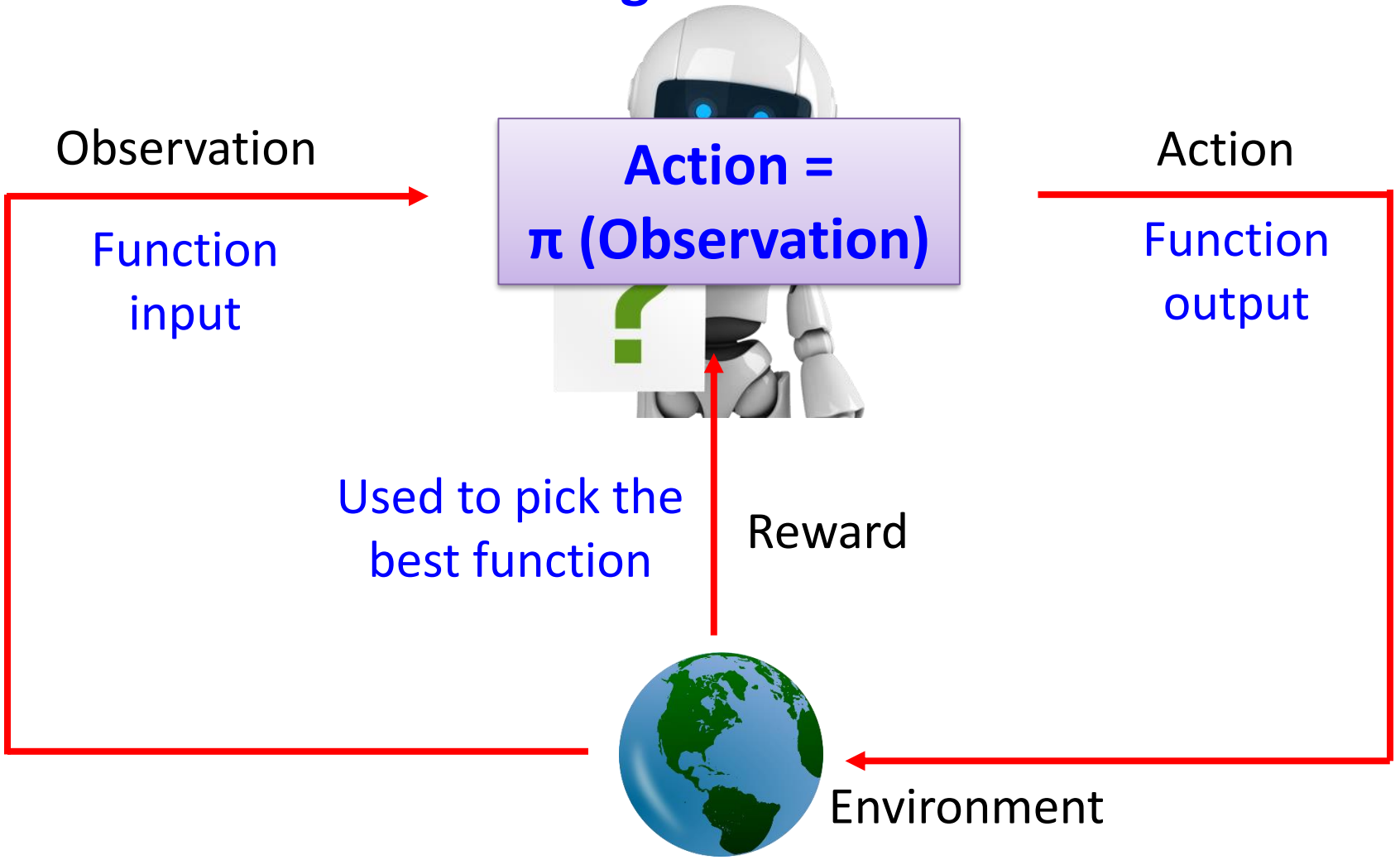
Scenario of Reinforcement Learning

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

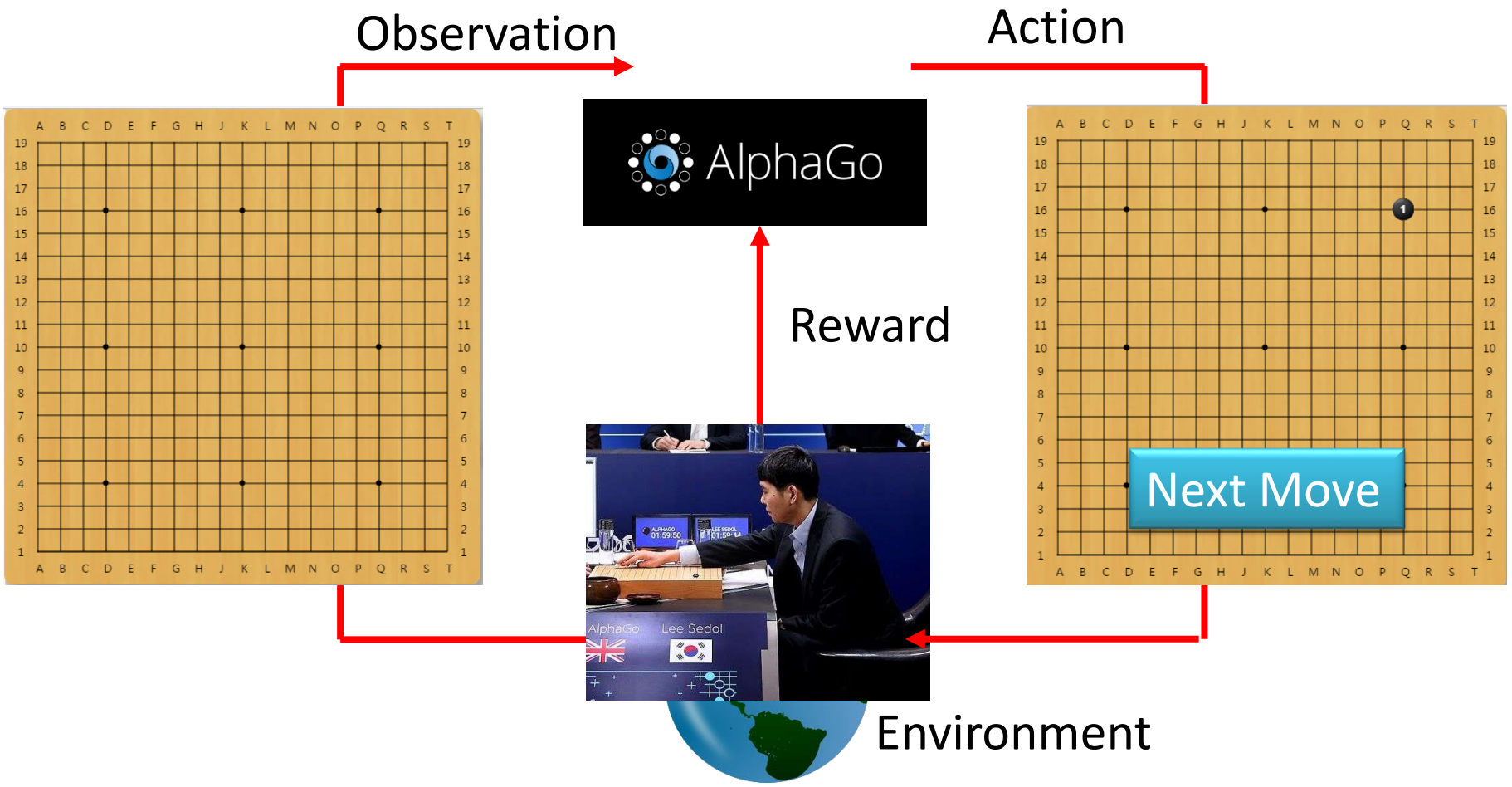
Scenario of Reinforcement Learning

Reinforcement Learning

≈ **Looking for a Function**



Learning to Play Go

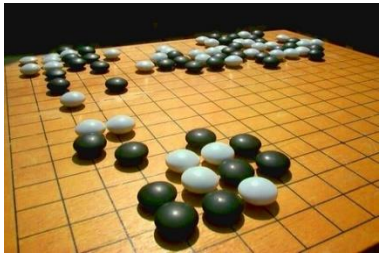


Learning to Play Go

Learning to Play Go

- Supervised: Learning from teacher

Next move: "5-5"



Next move: "3-3"

- Reinforcement Learning Learning from experience

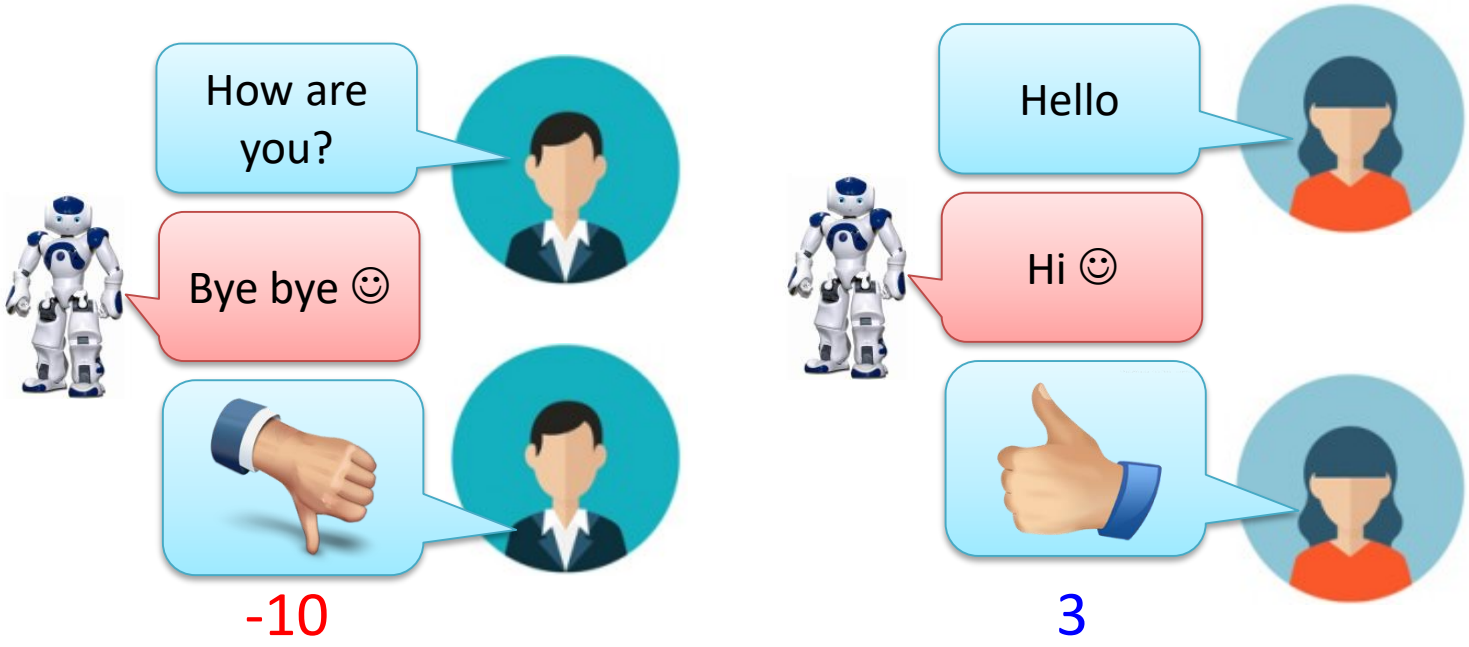
First move → …… many moves …… → Win!

(Two agents play with each other)

AlphaGo is supervised learning + reinforcement learning

Learning A Chat-bot

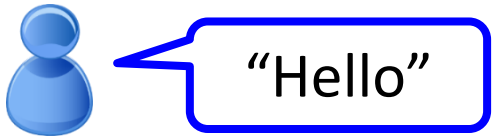
- Machine obtains feedback from user



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

Learning A Chat-bot

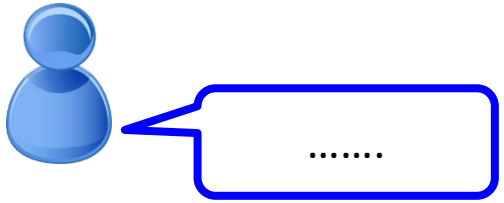
- Supervised

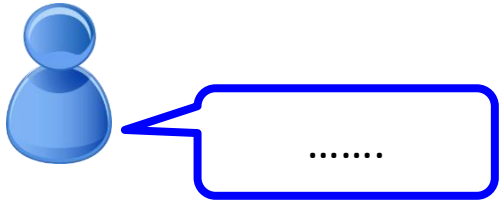


Say “Hi”

Say “Good bye”

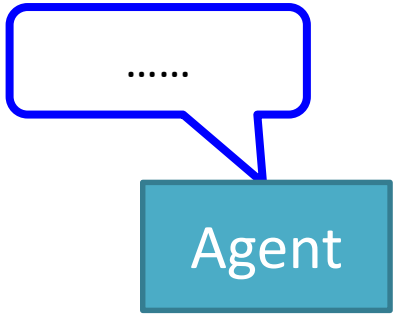
- Reinforcement





……

Bad



Playing Video Game

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

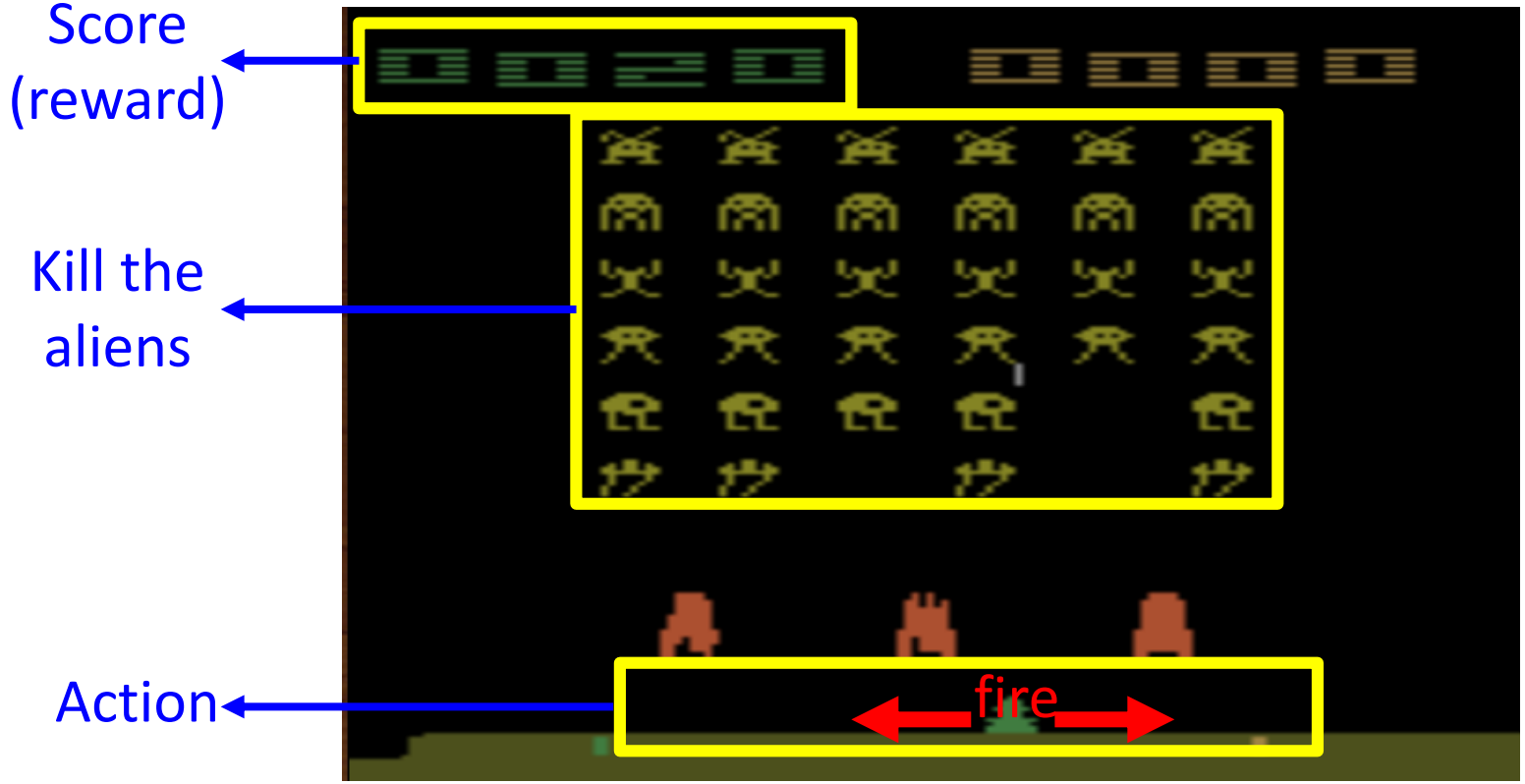
Machine learns to play video games as human players

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

Playing Video Game

- Space invader

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

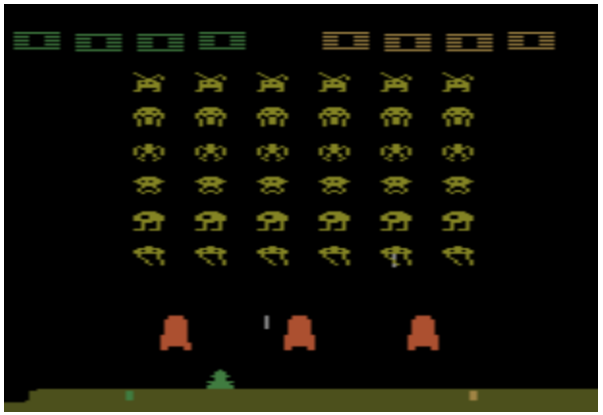


Playing Video Game

Start with observation s_1

Observation s_2

Observation s_3



Obtain reward $r_1 = 0$

Action a_1: "right"

Obtain reward $r_2 = 5$

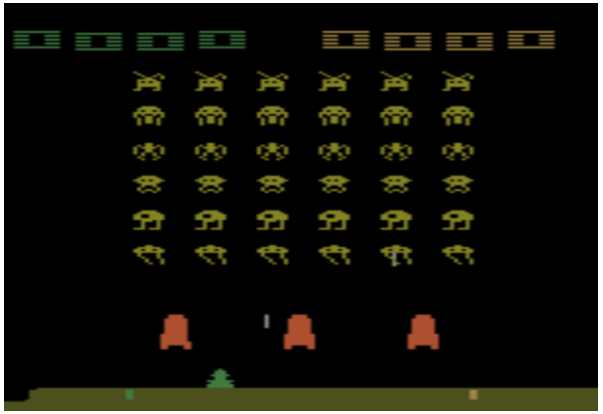
Action a_2 : "fire" (kill an alien)

Usually there is some randomness in the environment

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

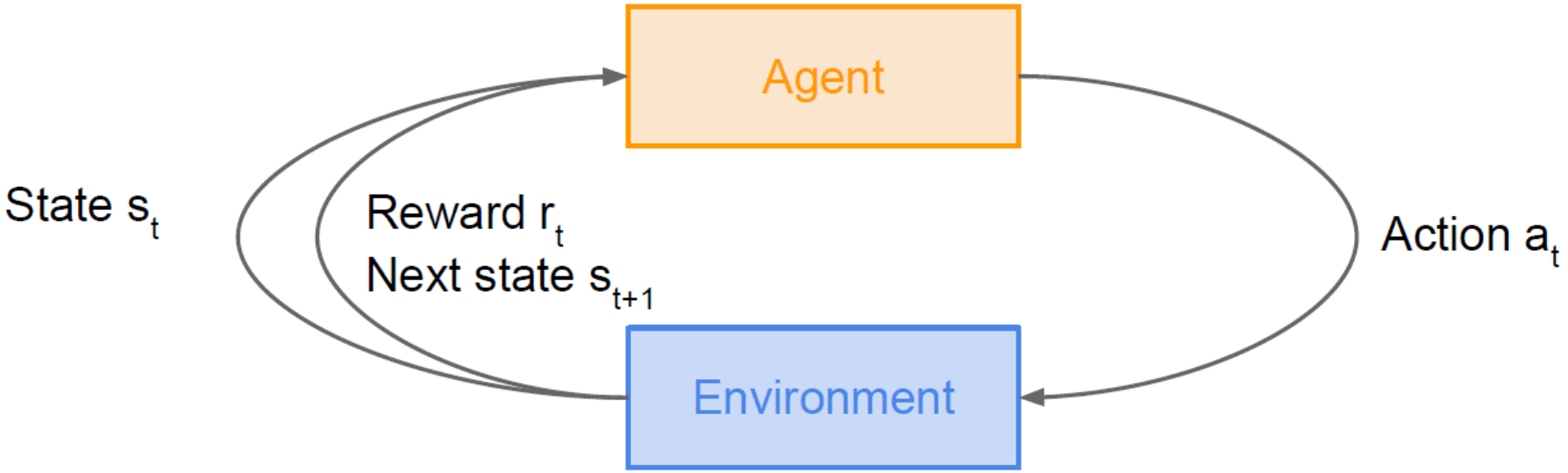
More Applications

- 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

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. butterfly effect

Markov Decision Process

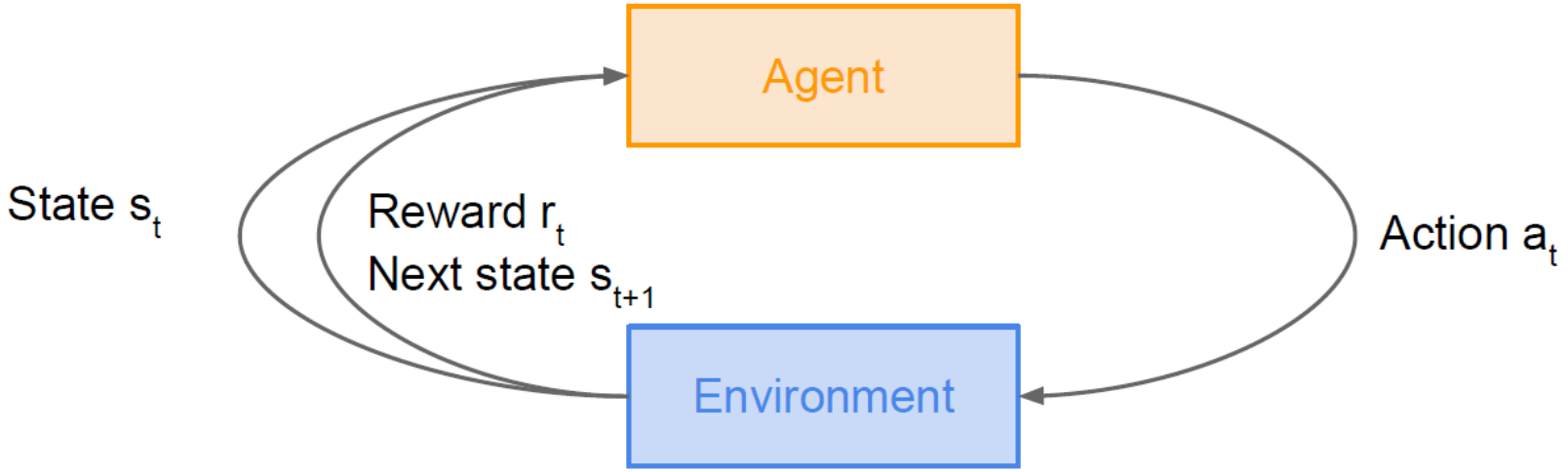


- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

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

Markov Decision Process

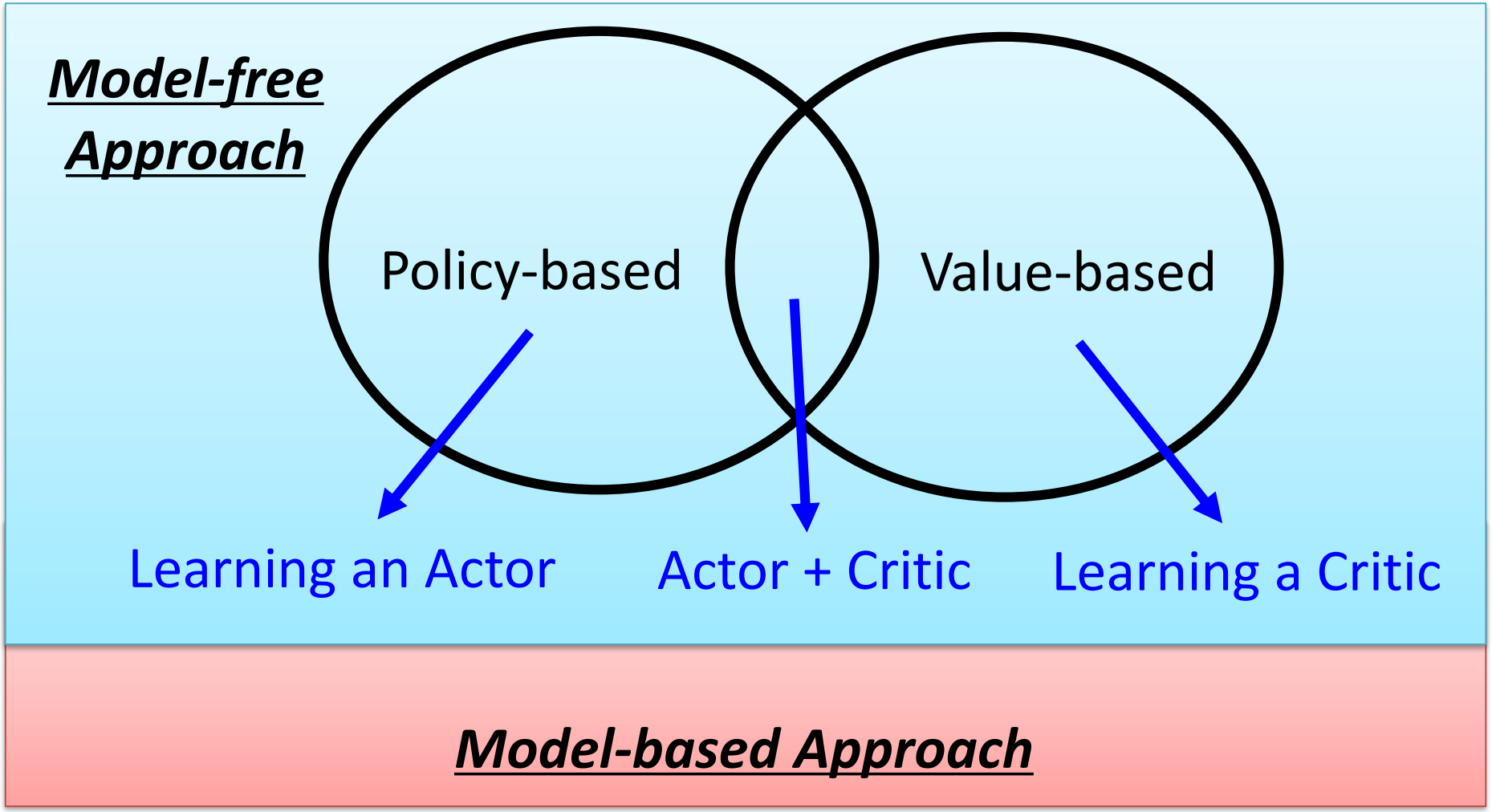


- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

Model-based vs. model-free

- 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: $\sum_{t \geq 0} \gamma^t r_t$

Methods of Reinforcement Learning



AlphaGo: policy-based + value-based + model-based

Outline

1. Course Review
2. Background
3. Model-free Learning
4. Applications

Policy-based Approach

Learning an Actor

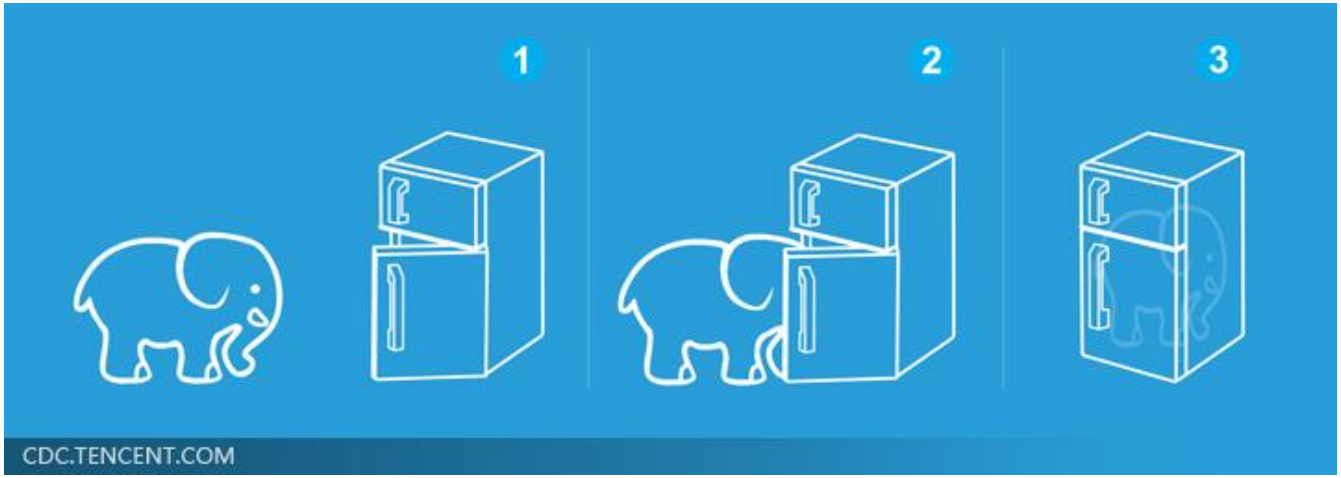
Three Steps for Deep Learning

Step 1: Neural Network as Actor

Step 2: goodness of function

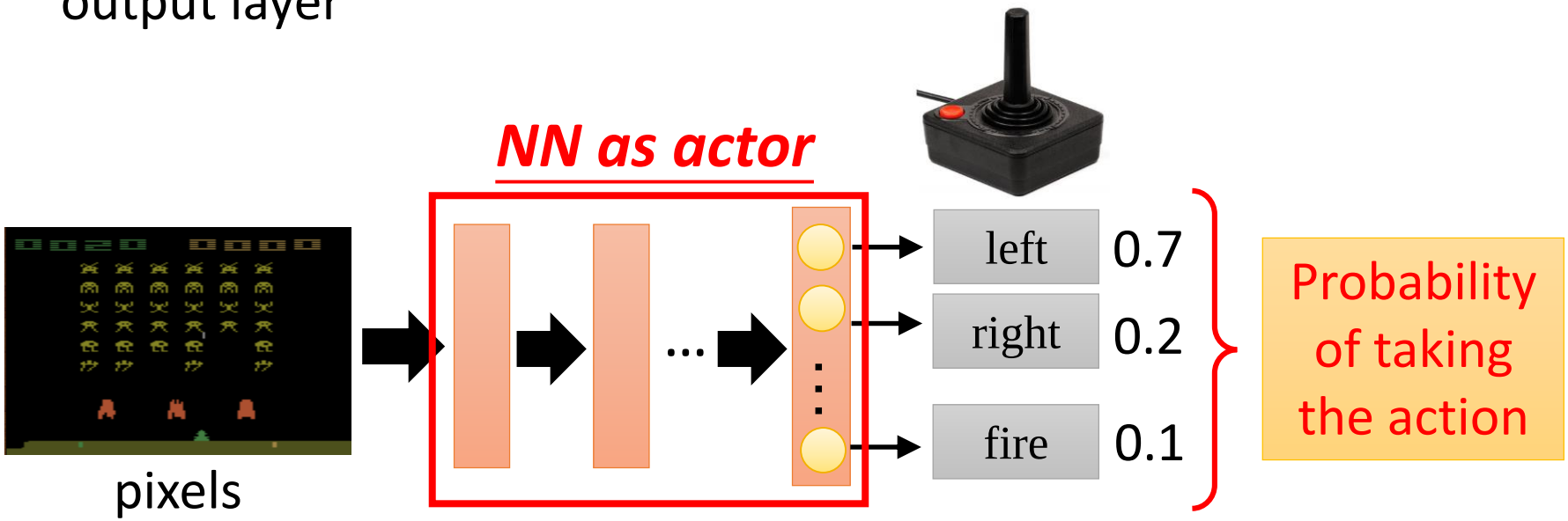
Step 3: pick the best function

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

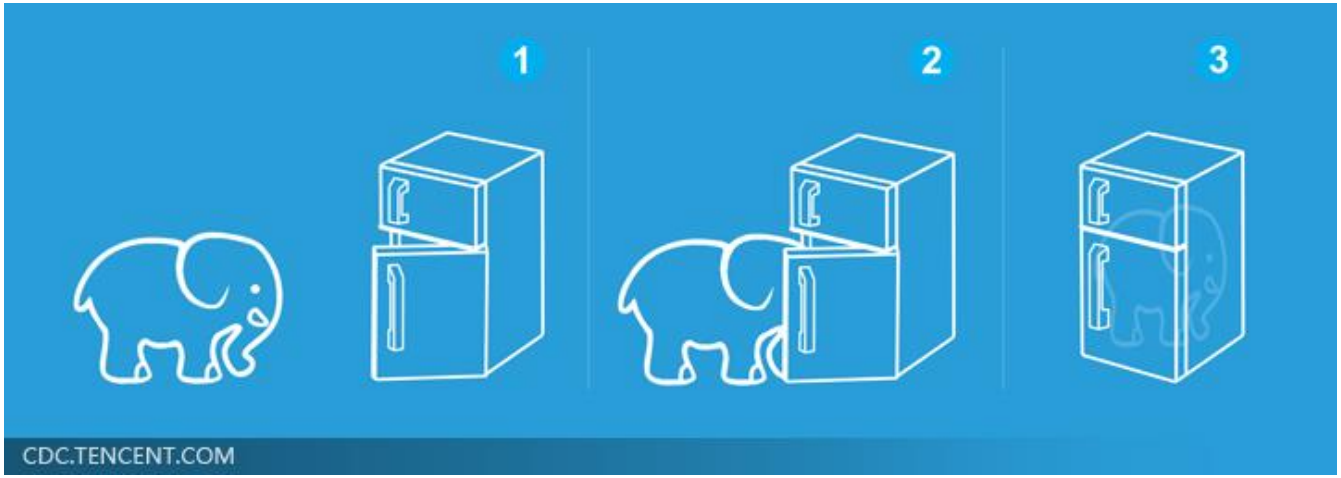
Three Steps for Deep Learning

Step 1: Neural Network as Actor

Step 2: goodness of function

Step 3: pick the best function

Deep Learning is so simple ……

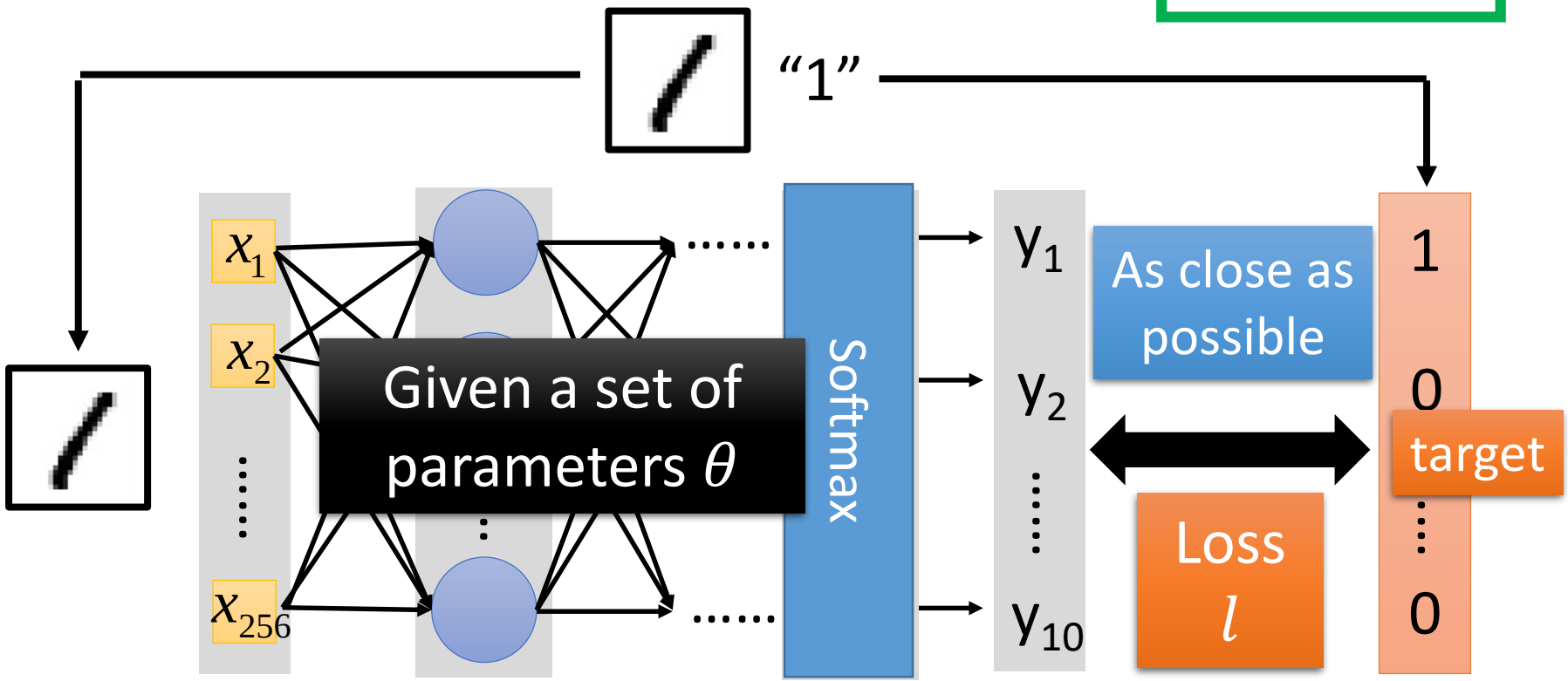


Goodness of Actor

- 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, due to randomness in the actor and the game

We define $\bar{R}_\theta$ as the *expected value* of R_θ, which evaluates the goodness of an actor $\pi_\theta(s)$

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, which depends on actor parameter θ: $P(\tau|\theta)$

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

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

Sum over all possible trajectory

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

Goodness of Actor

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

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

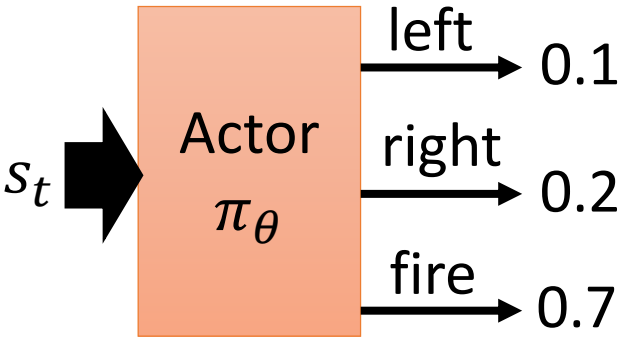
$$p(s_1)p(a_1|s_1,\theta)p(r_1,s_2|s_1,a_1)p(a_2|s_2,\theta)p(r_2,s_3|s_2,a_2)\cdots$$

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

not related to your actor

Control by your actor π_θ

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



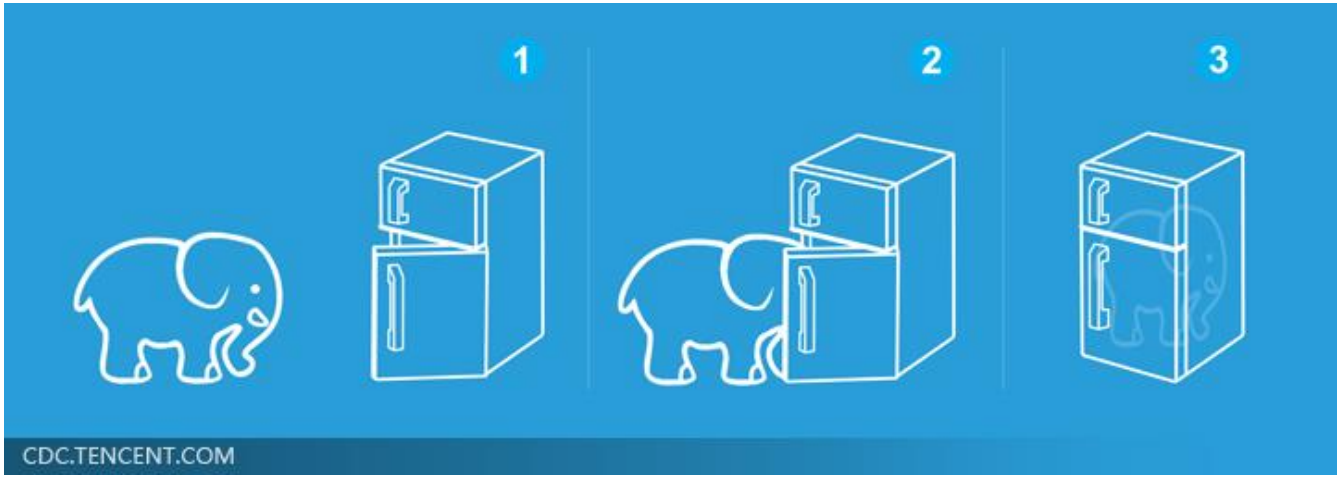
Three Steps for Deep Learning

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

$$\theta^* = arg\max_{\theta} \bar{R}_\theta \,, \;\; \bar{R}_\theta = \sum_{\tau} R(\tau) P(\tau|\theta)$$

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

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

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

Policy Gradient

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

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

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

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

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

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

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

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

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

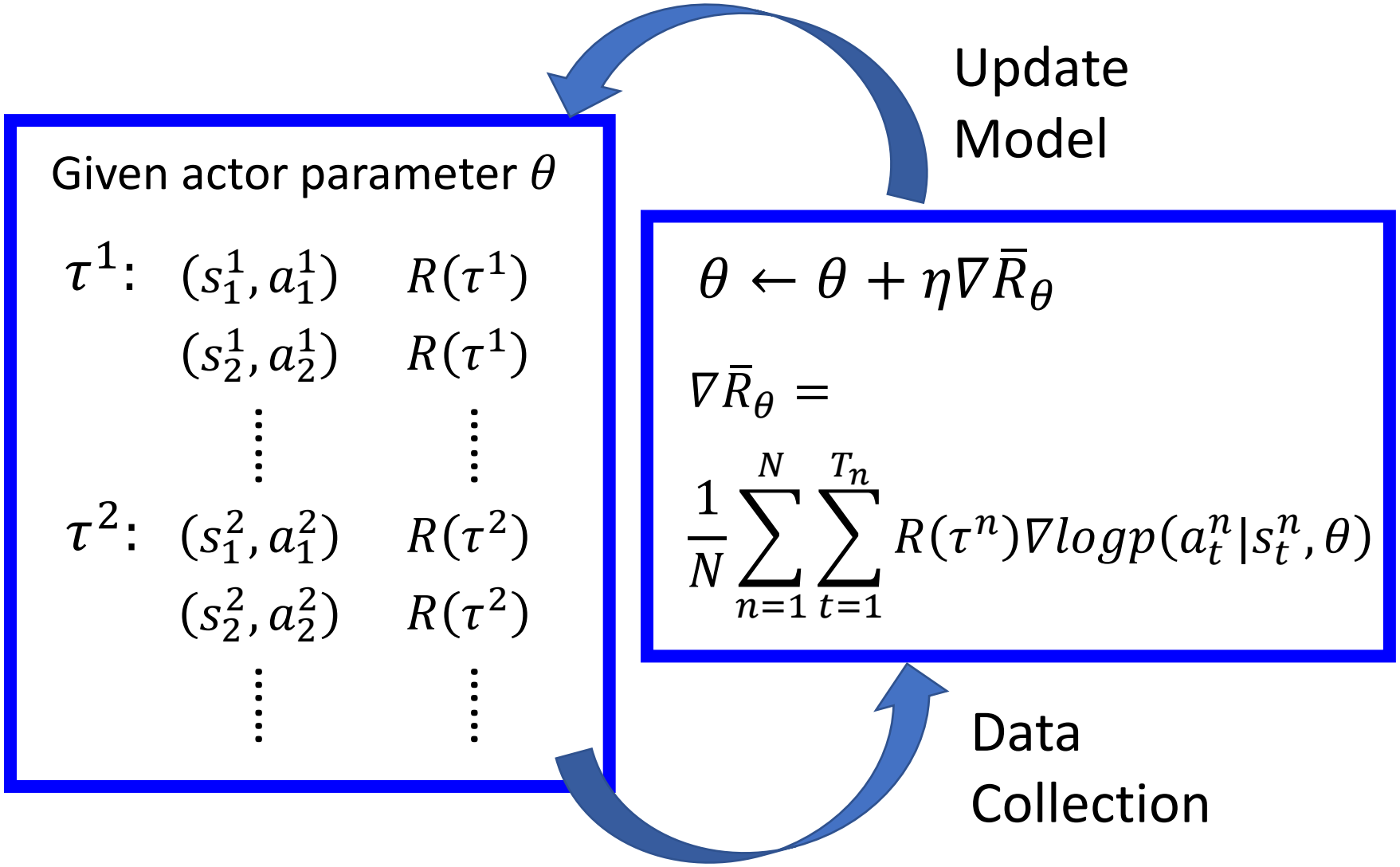
$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla logp(a_t^n|s_t^n,\theta)$$

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

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

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

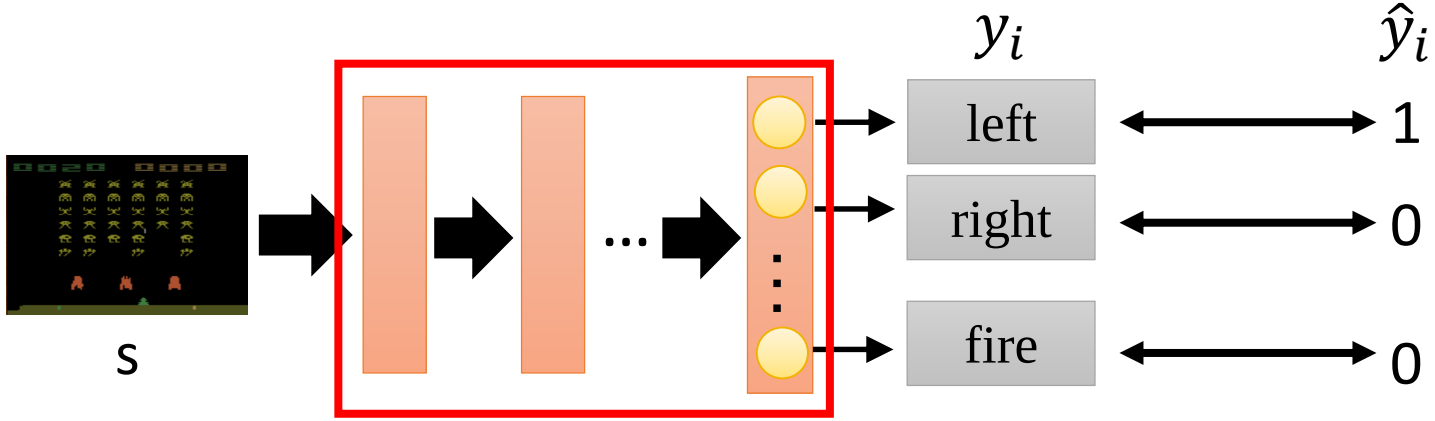
Policy Gradient



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

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

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

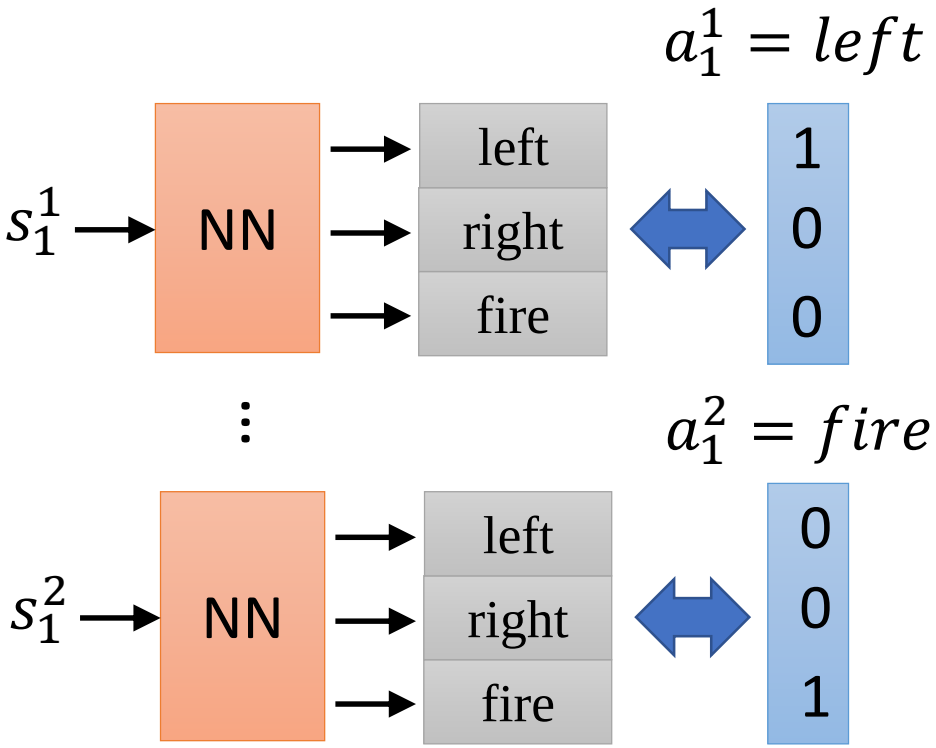
Given actor parameter θ

τ^1: (s_1^1, a_1^1) $R(\tau^1)$

(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

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

⋮ ⋮

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

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

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

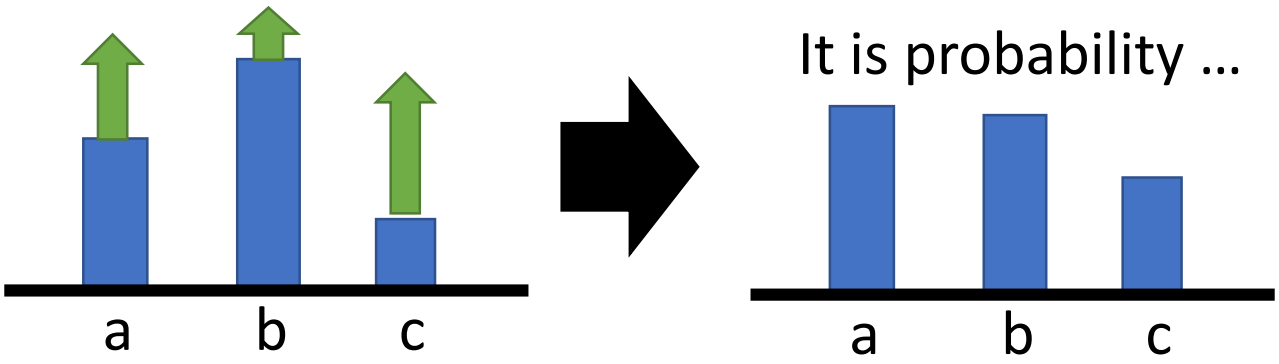
Add a Baseline

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

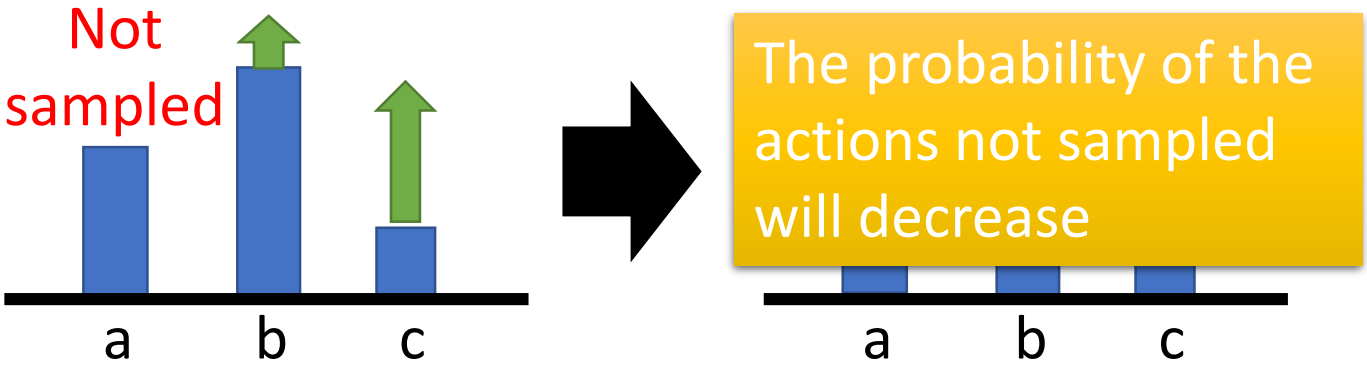
$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

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

Ideal case



Sampling ……



Value-based Approach

Learning a Critic

Critic

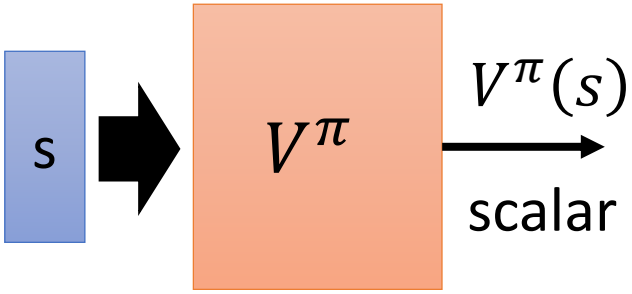
- A critic does not determine the action
- Given an actor π, it evaluates how good the actor is

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

Critic

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

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



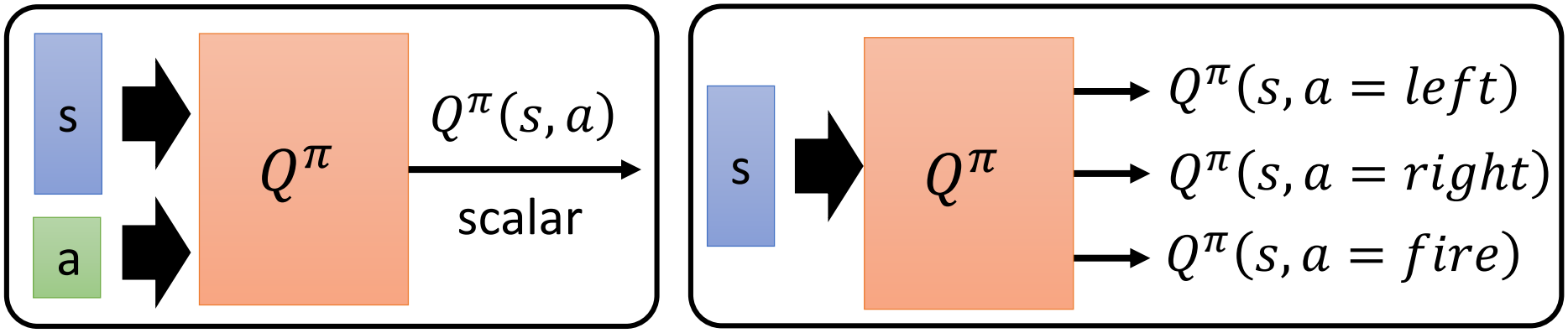
$V^{\pi}(s)$ is large

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

Another Critic

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

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



for discrete action only

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 previous 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*

Bellman Equation

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!

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**!

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):

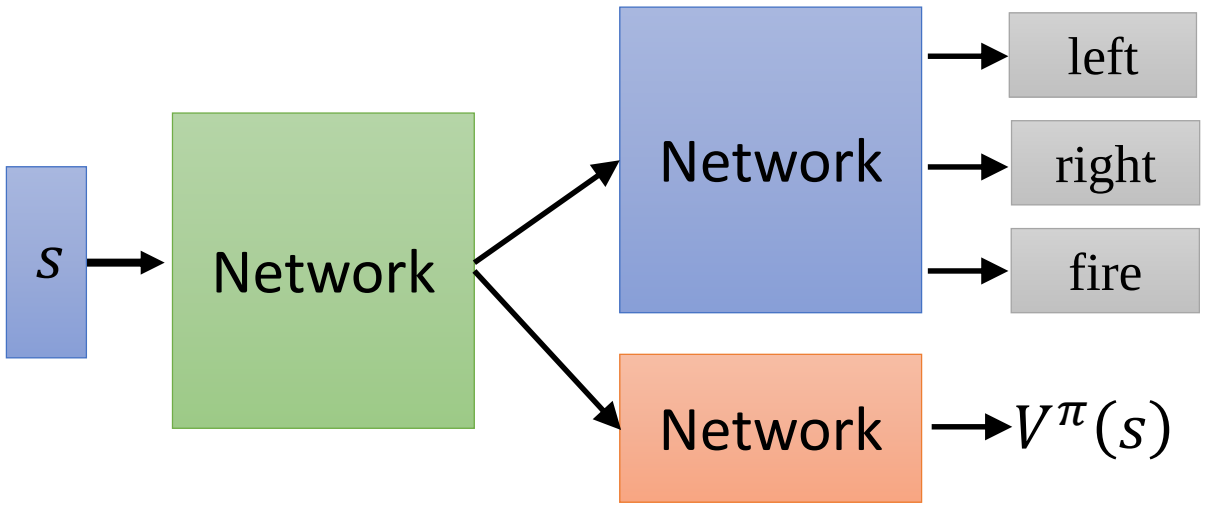
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Deep Reinforcement Learning

Actor-Critic

Actor-Critic

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



Outline

1. Course Review
2. Background
3. Model-free Learning
4. Applications

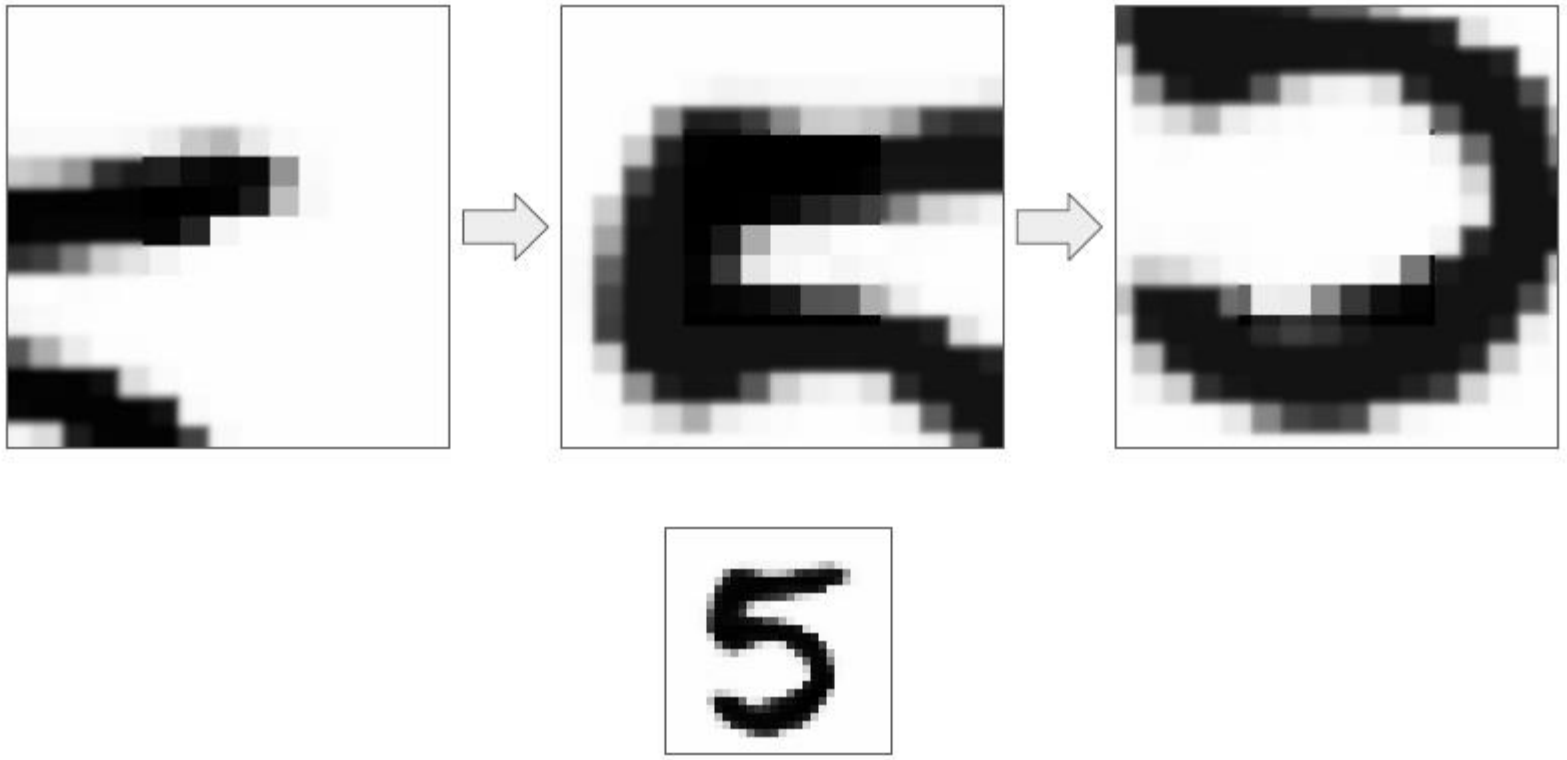
Applications-1

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



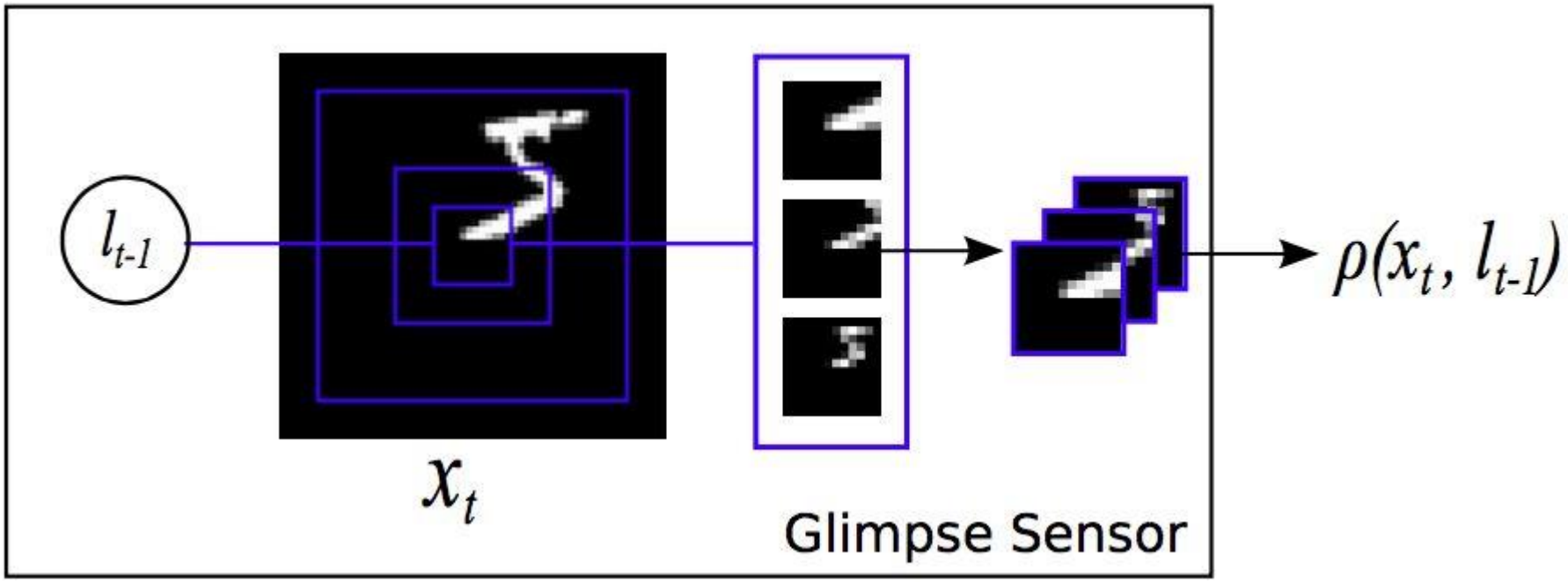
Overview

- 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

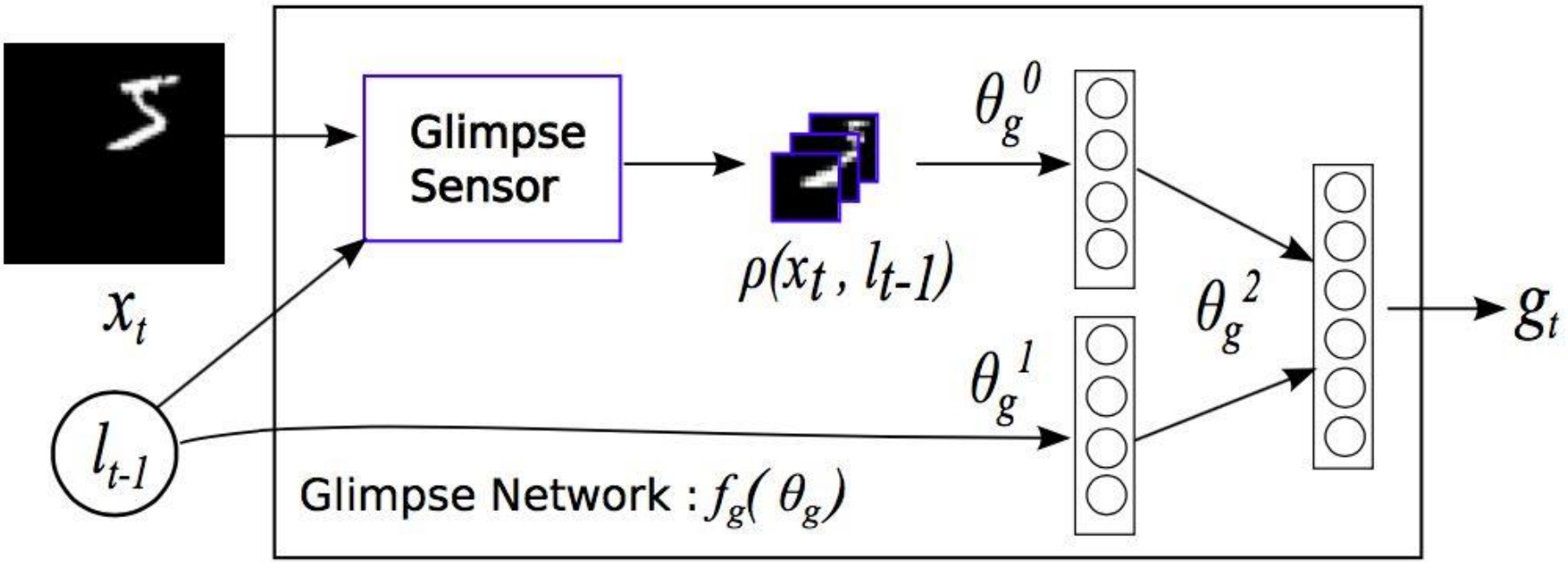
Glimpse

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

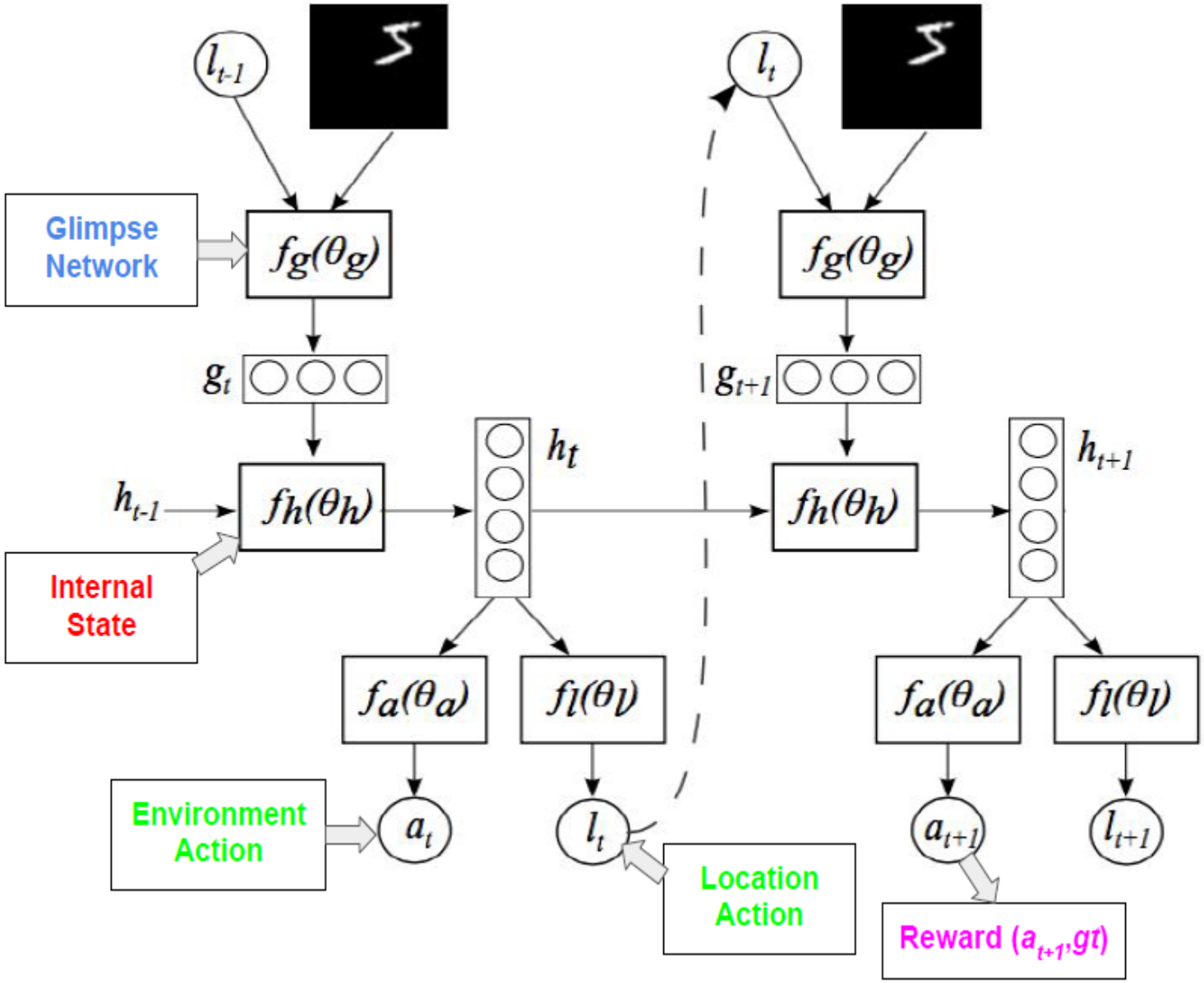


Glimpse Network

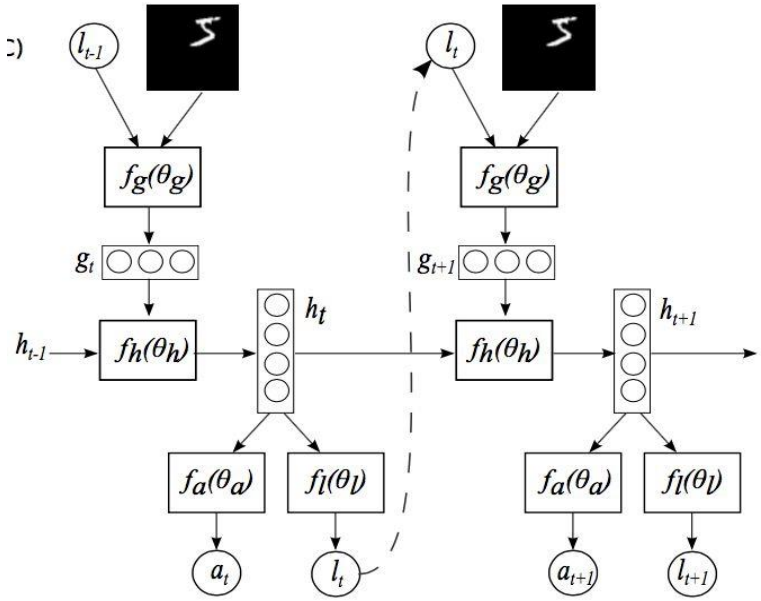
- $\rho(x_t, l_{t-1})$ and l_{t-1} are mapped into a hidden space



Model Architecture



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

(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%

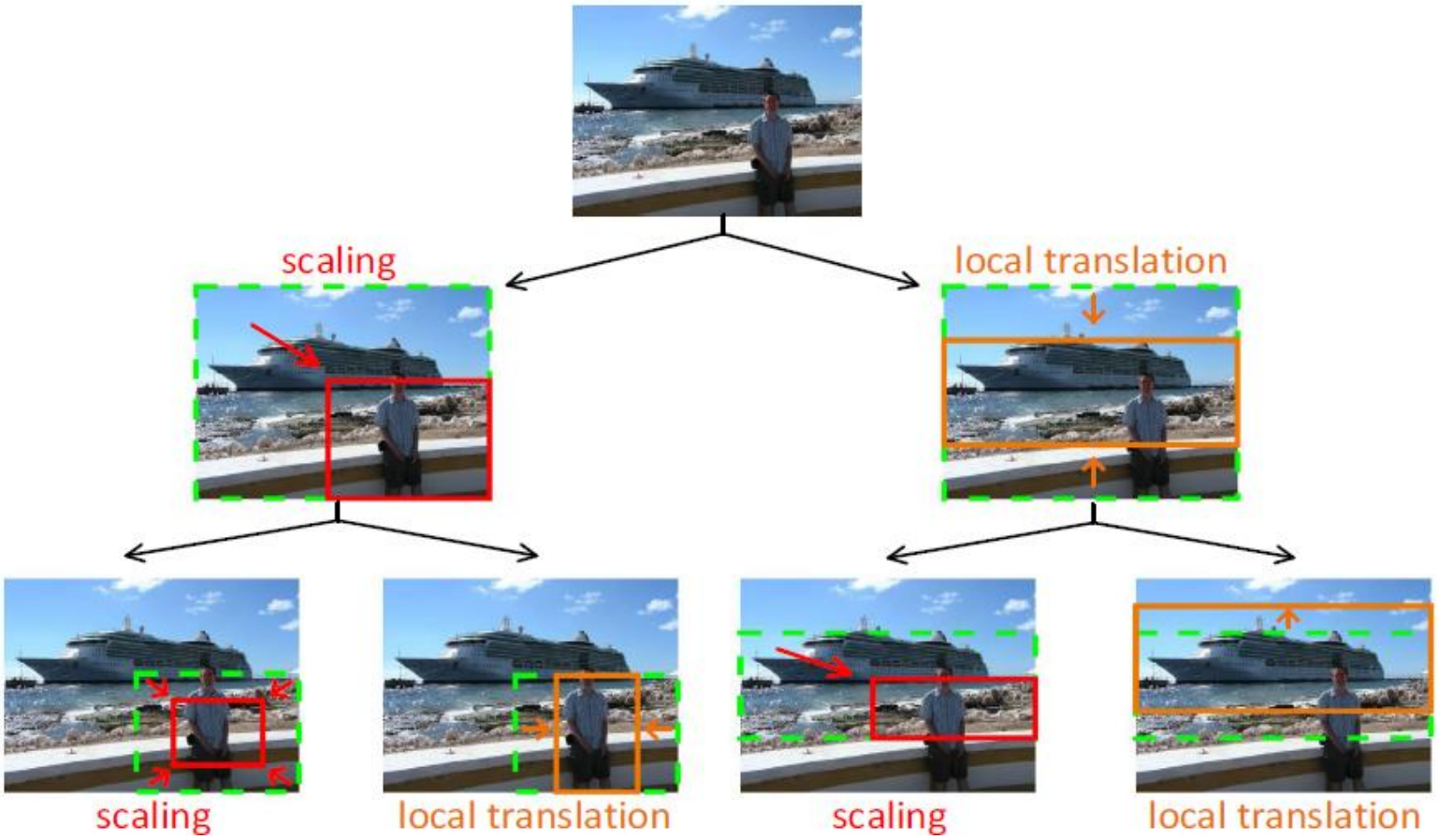
Applications-2

Tree-Structured Reinforcement Learning for Sequential Object Localization

Zequn Jie, Xiaodan Liang, Jiashi Feng, Xiaojie Jin, Wen Feng Lu, Shuicheng Yan

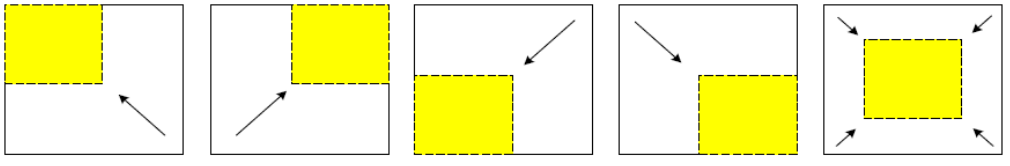
Sequential Object Localization

Starting from the **entire** image, the approach **sequentially** acts on the current search window either to: 1) **refine** the object location prediction, or 2) **discover** new objects by following a learned policy

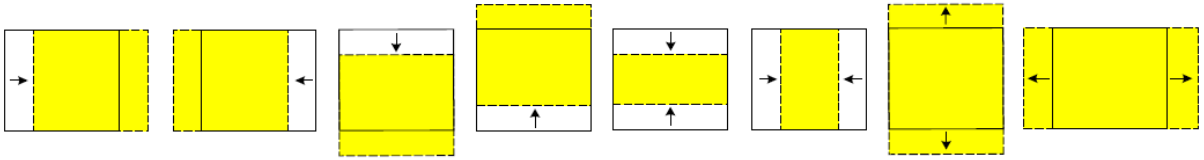


The Design of Action, State, Reward

- Two groups of **actions**:



scaling actions



local translation actions

- The **state** is the concatenation of three components: the feature vector of the current window, the feature vector of the whole image, and the history of taken actions

- The **reward** function r(s, a) reflects the localization accuracy improvements of all the objects by taking the action a under the state s

$$r(s,a) = \max_{1 \le i \le n} \text{sign}(\text{IoU}(w', g_i) - \text{IoU}(w, g_i)).$$

Basically, if any ground-truth object bounding box has a higher IoU with the next window than the current one, the reward of the action moving from the current window to the next one is +1, and -1 otherwise

Intersection-over-Union (IoU)

Tree-Structured Search & Deep Q-learning

For each window, the actions with the highest predicted value in **both** the scaling action group and the local translation action group are selected respectively

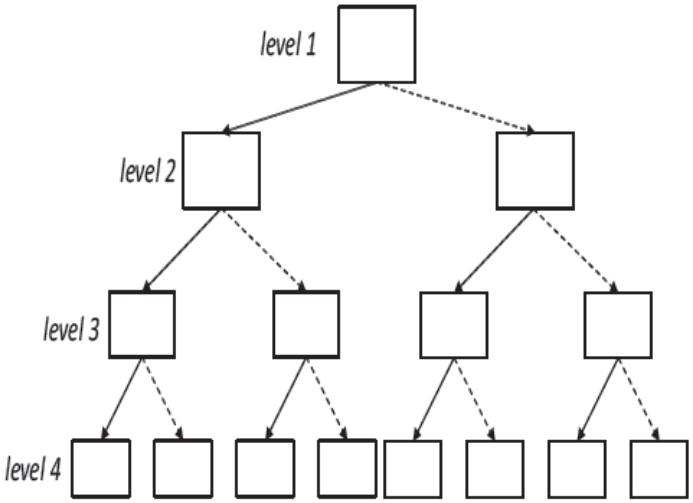


Figure 3: Illustration of the top-down tree search. Starting from the whole image, each window recursively takes the best actions from both action groups. Solid arrows and dashed arrows represent scaling actions and local translation actions, respectively.

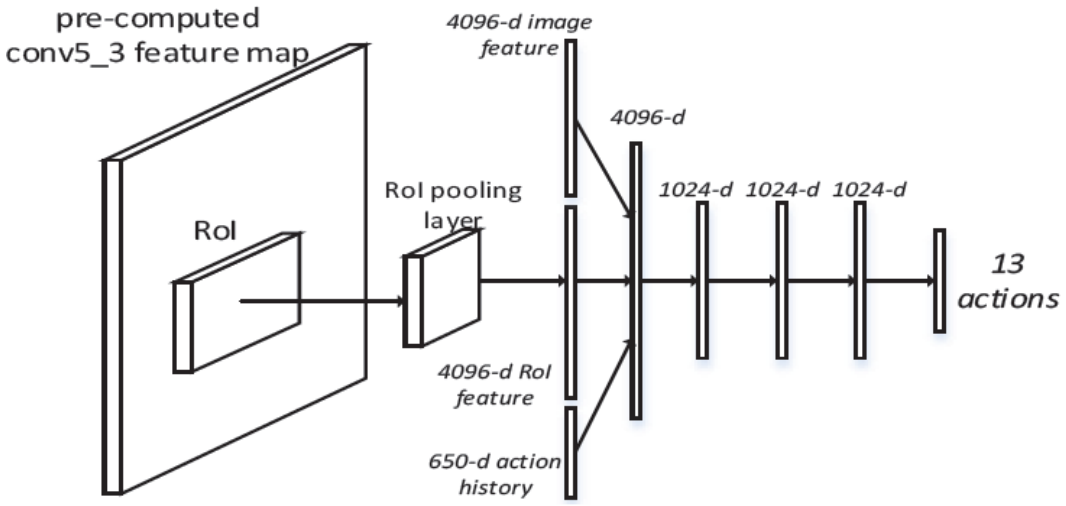


Figure 4: Illustration of our Q-network. The regional feature is computed on top of the pre-computed "conv5_3" feature maps extracted by VGG-16 pre-trained model. It is concatenated with the whole image feature and the history of past actions to be fed into an MLP. The MLP predicts the estimated values of the 13 actions.

Tree-Structured Search & Deep Q-learning

Table 4: Detection results comparison on PASCAL VOC 2012 testing set.

method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
RPN (VGG-16)+ Fast R-CNN (ResNet-101)	86.9	83.3	75.6	55.4	50.8	79.2	76.9	92.8	48.8	79.0	57.2	90.2	85.4	82.1	79.4	46.0	77.0	66.4	83.3	66.0	73.1
Faster R-CNN (ResNet-101) [26]	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6	73.8
Tree-RL (VGG-16)+ Fast R-CNN (ResNet-101)	85.9	79.3	77.1	62.1	53.4	77.8	77.4	90.1	52.3	79.2	56.2	88.9	84.5	80.8	81.1	51.7	77.3	66.9	82.6	68.5	73.7

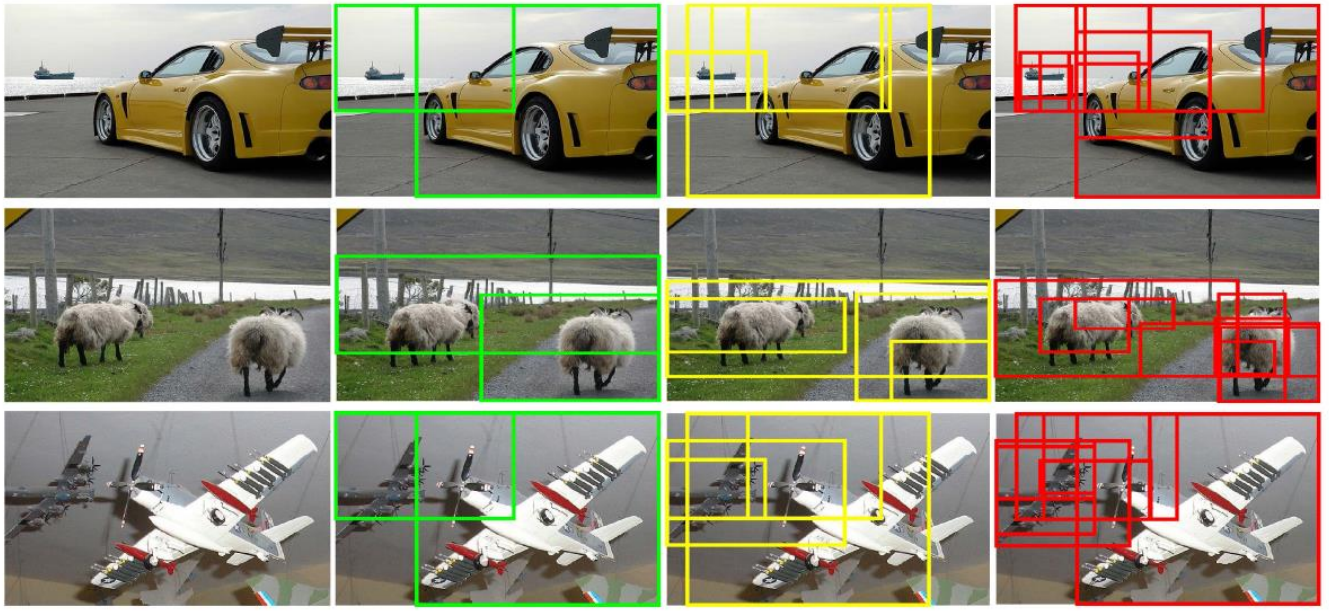


Figure 6: Examples of the proposals generated by Tree-RL. We only show the proposals of level 2 to level 4. Green, yellow and red windows are generated by the 2nd, 3rd and 4th level respectively. The 1st level is the whole image.

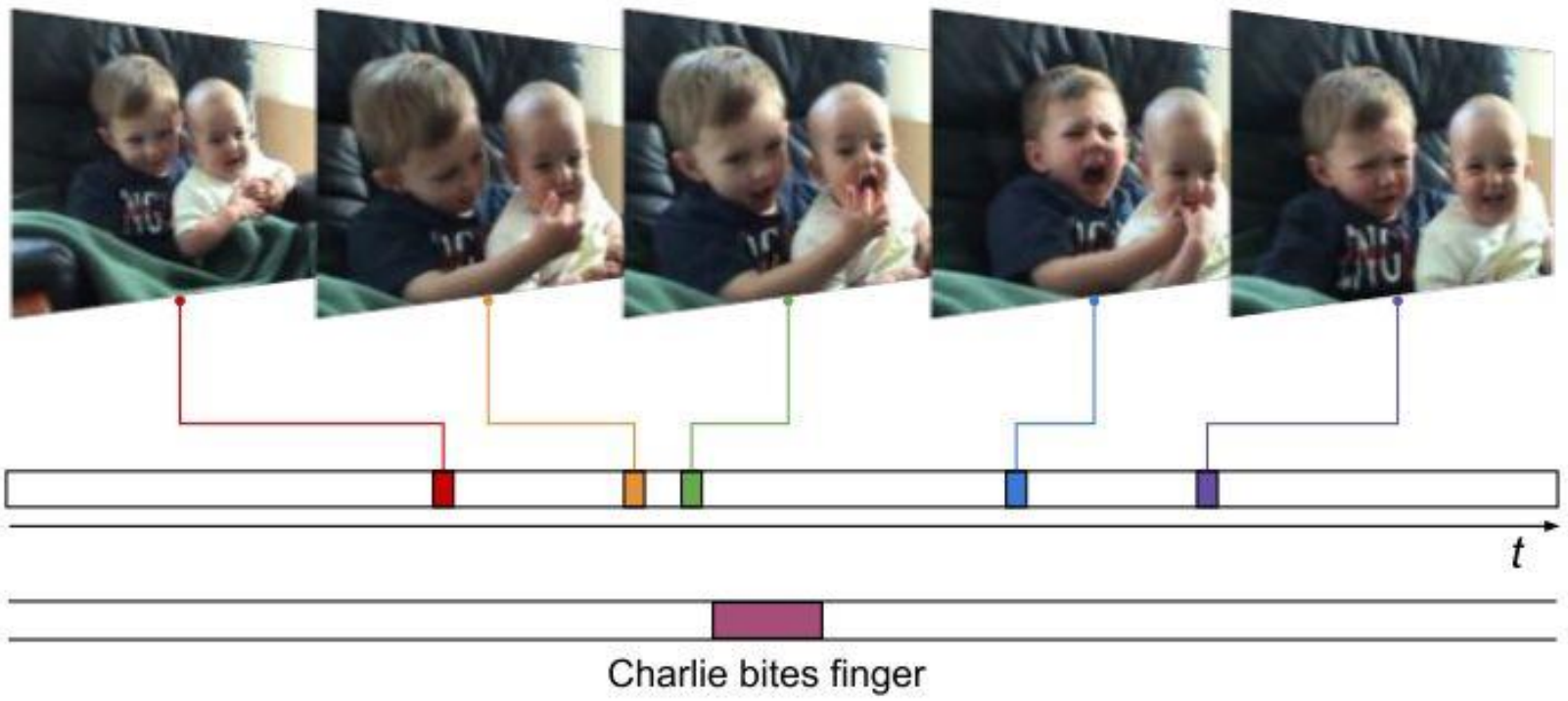
Applications-3

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

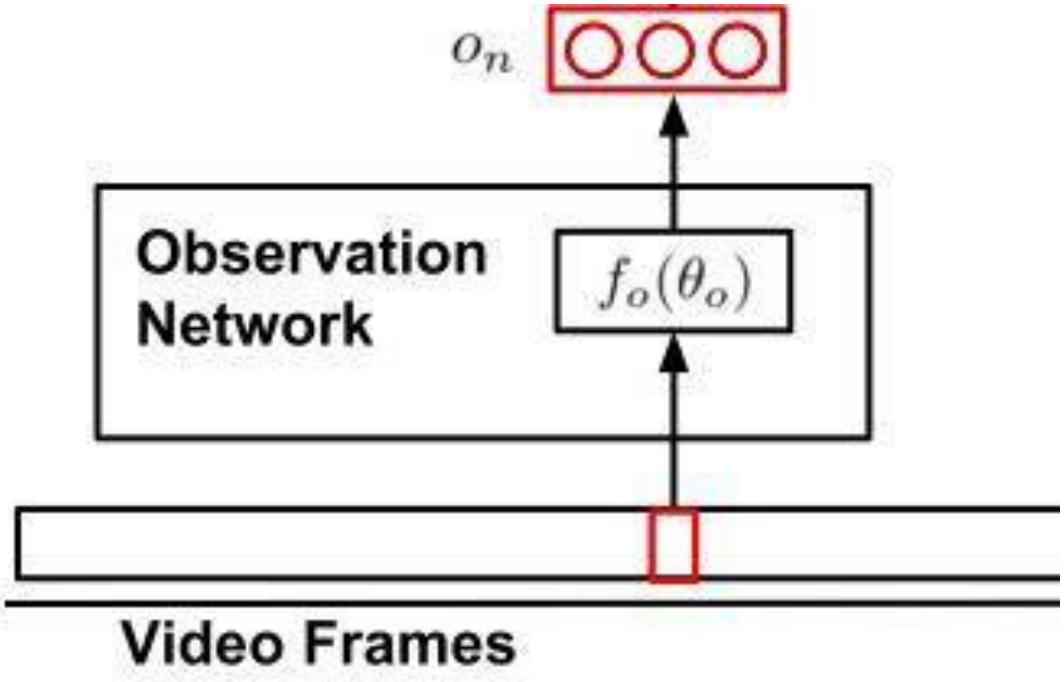


Overview

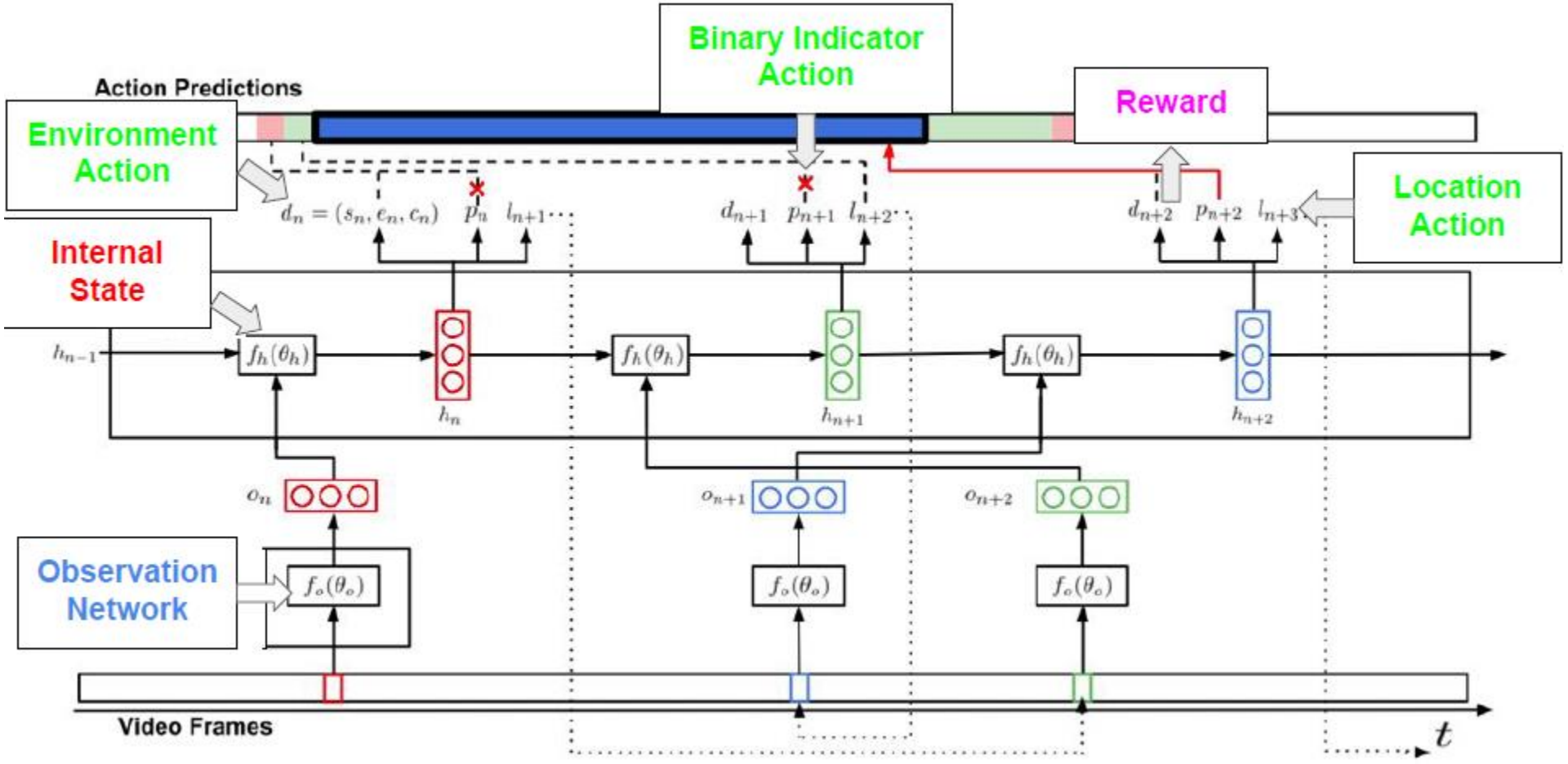
- True state of the environment is unobserved
  - **Observation Network** can be seen as a partial view of the state
- **State:** $h_n = f_h\,(h_{n-1}\,,\, o_n\,;\, \vartheta_h\,)$
- **Actions:**
  - Candidate detection: $d_n = f_d(h_n; \vartheta_d)$
  - Binary indication: $p_n = f_p(h_n; \vartheta_p)$
  - 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

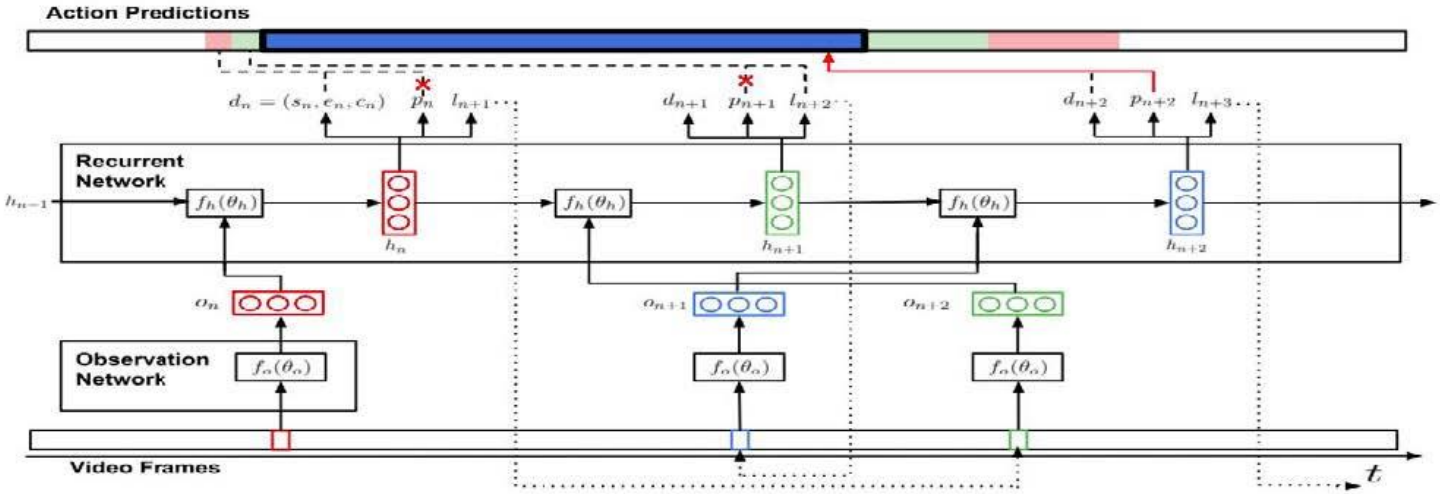
- Observes a single video frame at each timestep and encodes the frame and it's location into a feature vector o_n
  - Inspired by the **Glimpse network**



Model Architecture



Training



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

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

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

	[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 the state-of-the-art
  - Less frames observed

Applications-3

Language-driven Temporal Activity Localization: A Semantic Matching Reinforcement Learning Model

Weining Wang, Yan Huang, Liang Wang

Action Localization with Language

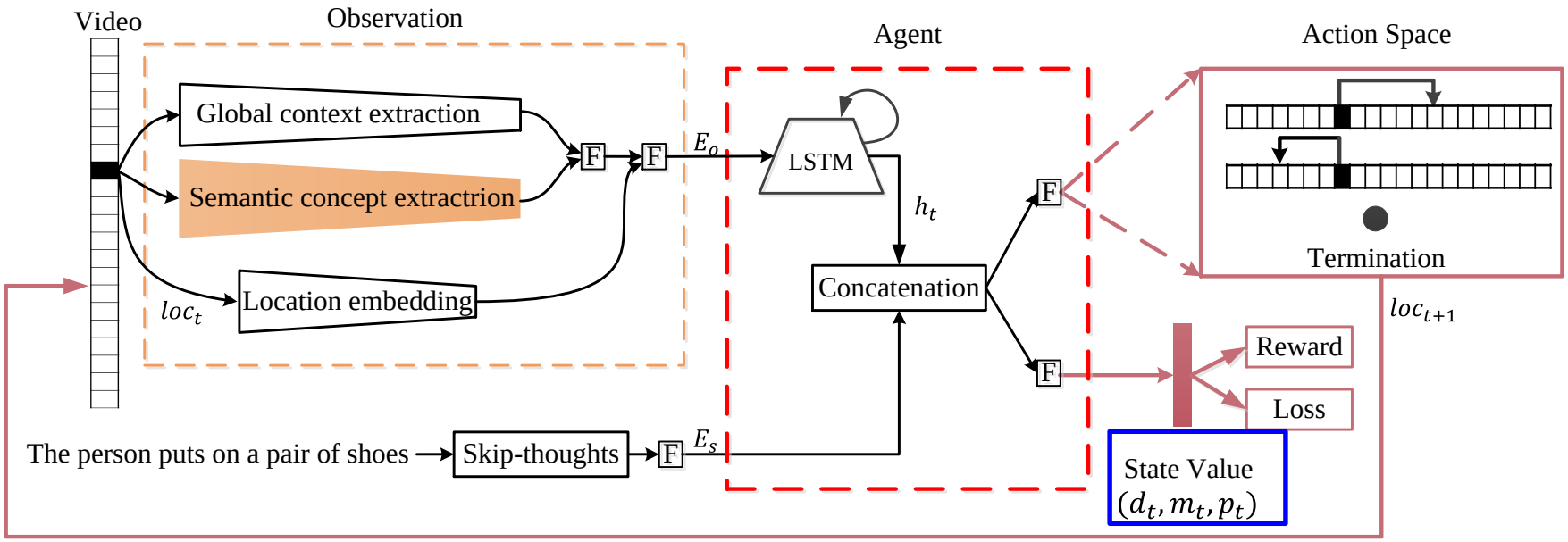
Longboarding

Only class annotation, single action

A person runs to the window and then look out

Language annotation, multiple actions

Semantic Matching Reinforcement Learning



$$L(D) = \omega_1 \sum_t L_{cls}(m_t) + \omega_2 \sum_t L_{loc}(d_t, g_t)$$

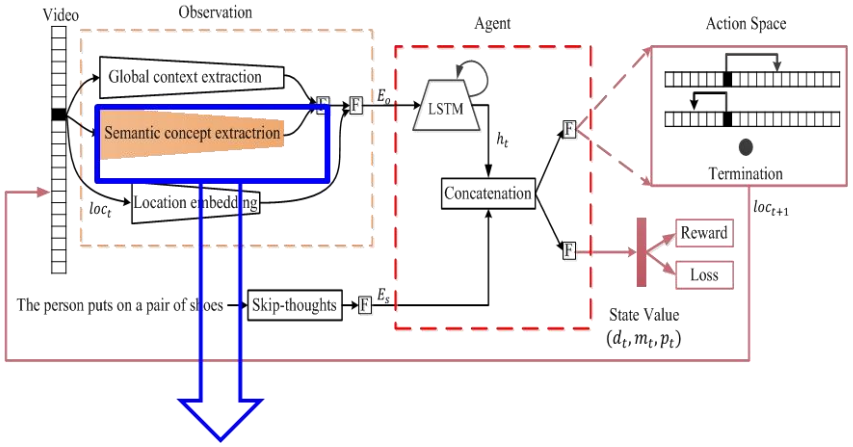
$$\begin{cases} L_{cls}(m_t; \theta_m) = -\sum_i (r_i) log P(r_i | m_i; \theta_m) \\ L_{loc} = \frac{1}{n} \sum_i^n \sum_j^M [x_{ij} \log(p_{ij}) + (1 - x_{ij}) \log(1 - p_{ij})] \end{cases}$$

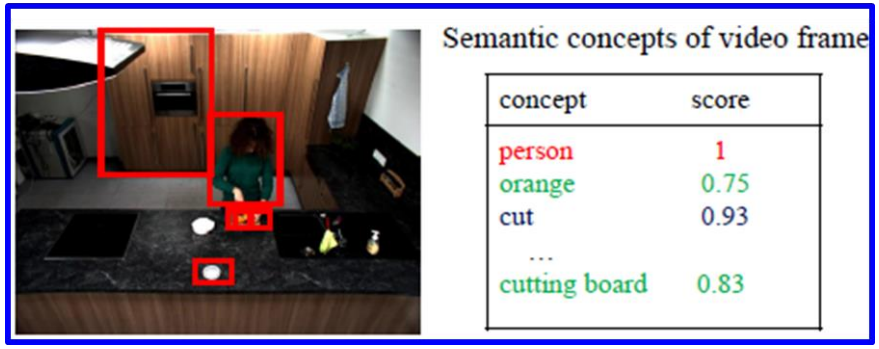
d_t: temporal boundary prediction

m_t: semantic matching score

p_t: binary prediction indicator

Sematic Concept Extraction





The person cuts oranges on a cutting board

⇩

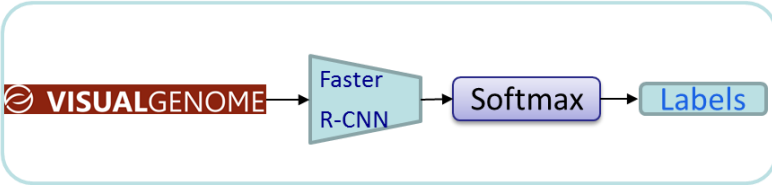
The, person, cuts, oranges, on, a, cutting board

⇩

~~The~~, person, cuts, oranges, ~~on~~, ~~a~~, cutting board

⇩

⇔ person, cuts, oranges, cutting board



Results

Table 1: Results on TACoS and Charades-STA datasets

Method	TACoS							Charades-STA				
	R@1 IoU=0.5	R@1 IoU=0.3	R@1 IoU=0.1	R@5 IoU=0.5	R@5 IoU=0.3	R@5 IoU=0.1	mR	R@1 IoU=0.5	R@1 IoU=0.7	R@5 IoU=0.5	R@5 IoU=0.7	mR
Random	0.83	1.81	3.28	3.57	7.03	15.09	5.27	8.51	3.03	37.12	14.06	15.68
CTRL [8]	13.30	18.32	24.32	25.42	36.69	48.73	27.80	23.63	8.89	58.92	29.52	30.24
RL(b)	11.76	17.70	22.42	22.61	33.24	45.10	25.47	19.78	5.60	55.65	25.07	26.53
RL(f)	12.79	18.53	23.87	24.56	35.30	47.64	27.15	21.18	7.33	56.01	27.85	28.09
SM-RL(attr+b)	13.50	18.83	23.72	24.01	34.19	46.56	26.80	21.00	7.63	57.25	28.06	28.49
SM-RL(attr+f)	14.01	19.02	23.96	24.55	36.42	47.14	27.51	22.54	8.56	58.95	29.74	29.95
SM-RL(attr*+b)	14.20	19.79	25.17	25.38	36.69	48.22	28.24	23.56	9.52	60.17	32.53	31.45
SM-RL(attr*+f)	**15.95**	**20.25**	**26.51**	**27.84**	**38.47**	**50.01**	**29.84**	**24.36**	**11.17**	**61.25**	**32.08**	**32.22**

Table 2: Results on DiDeMo dataset

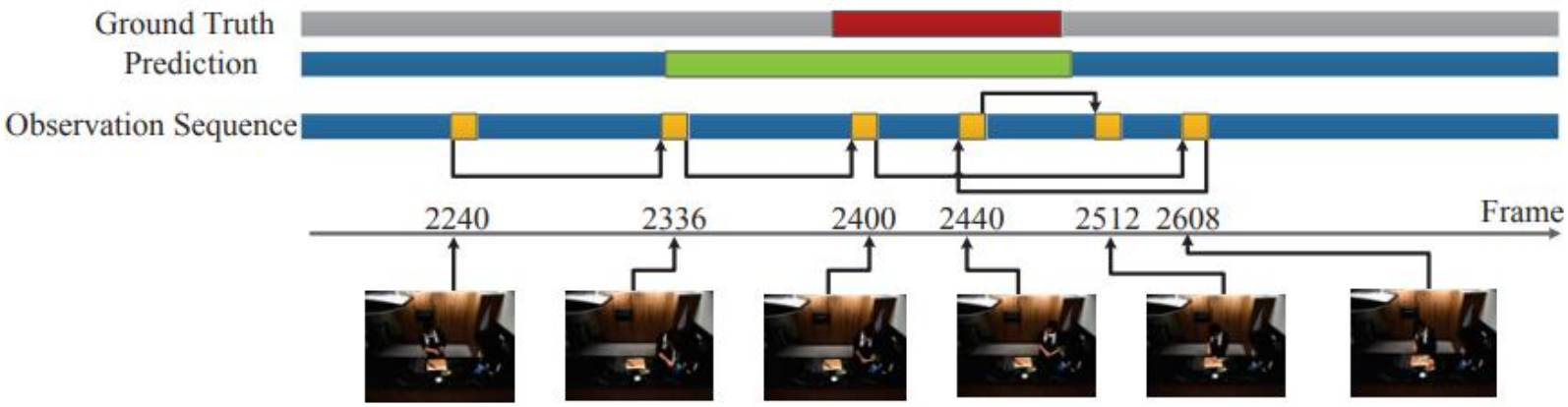
Method	Rank@1	Rank@5	mIoU
MCN([11])	28.10	78.21	41.08
SM-RL(attr*+b)	29.64	79.38	42.17
SM-RL(attr*+f)	**31.06**	**80.45**	**43.94**

Table 3: Detection speed comparison

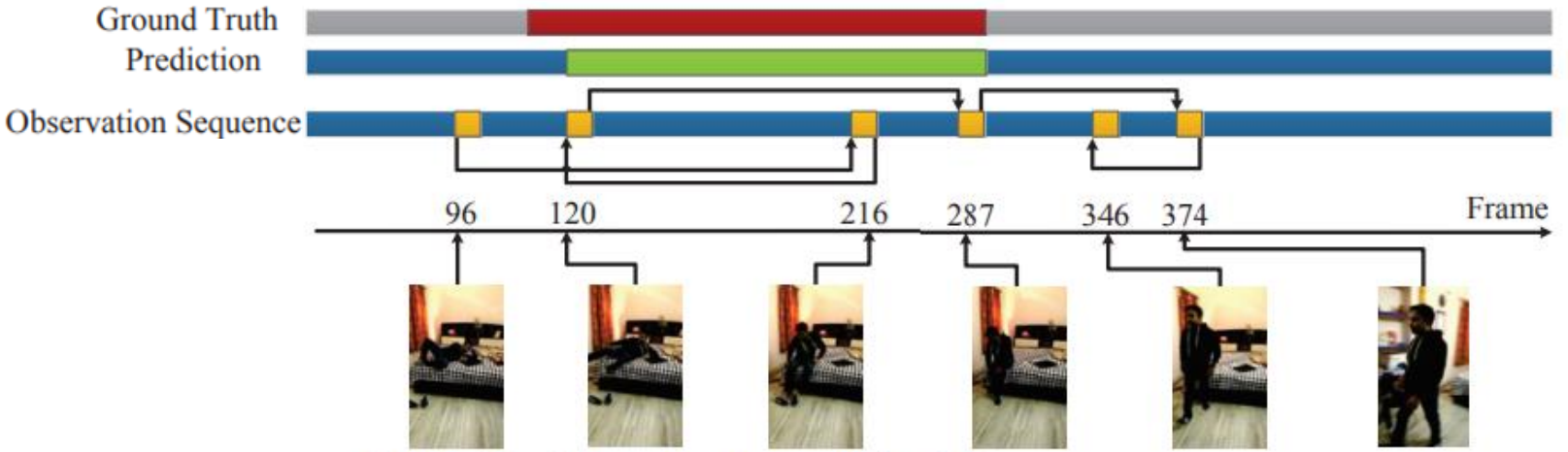
Method	Average running time (per minute video)
CTRL	202ms
Ours	**32ms**

- Simantic concepts lead to significant performance improvement
- Achieve the best performance with 6× faster speed

Example Analysis



Query A: The person washes the leeks in the sink



Query B: Person put on a pair of shoes

Skip in both forward and backward directions in a video

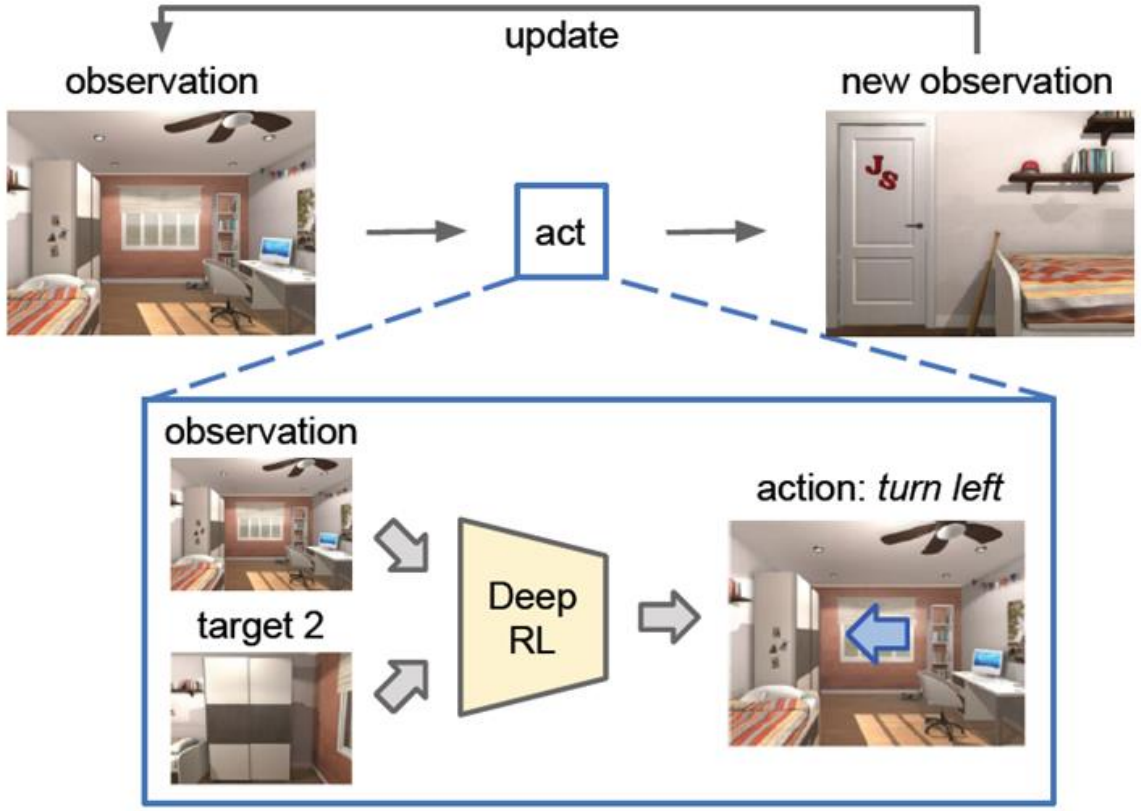
Applications-4

Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, Ali Farhadi

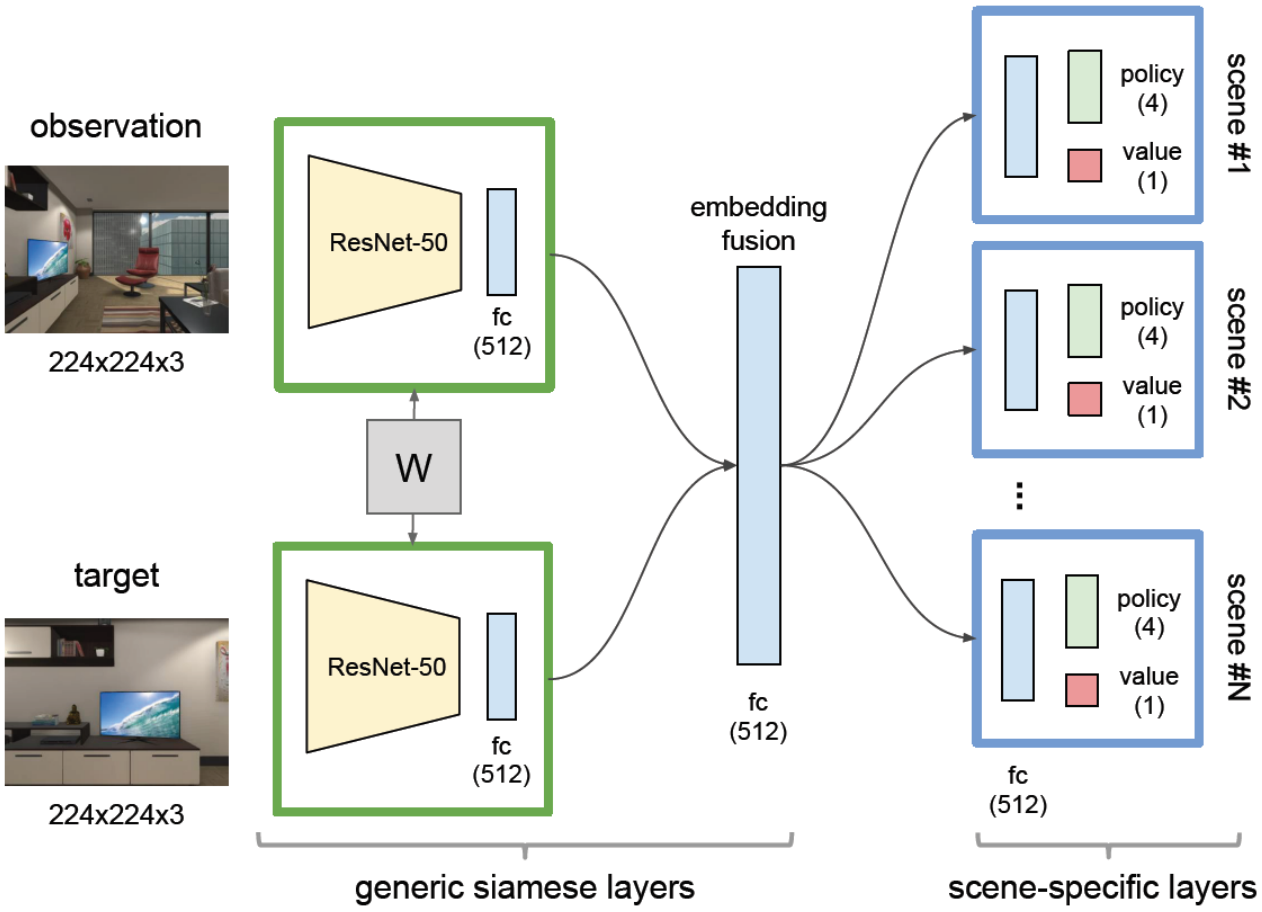
Visual Navigation

Navigate a space to find a given target using only visual input



learn a policy that jointly embeds the target goal and the current state

The Design of Action, State, Reward



Actions: moving forward, moving backward, turning left, and turning right

States: RGB images when navigating to a location

Reward: a goal-reaching reward (10) upon task completion, add a small time penalty (-0.01) as immediate reward

Results

TABLE I

PERFORMANCE OF TARGET-DRIVEN METHODS AND BASELINES

Type	Method	Avg. Trajectory Length
Heuristic	Random walk	2744.3
	Shortest path	17.6
Purpose-built RL	One-step Q	2539.2
	A3C (1 thread)	1241.3
	A3C (4 threads)	723.5
Target-driven RL (Ours)	Single branch	581.6
	Final	**210.7**

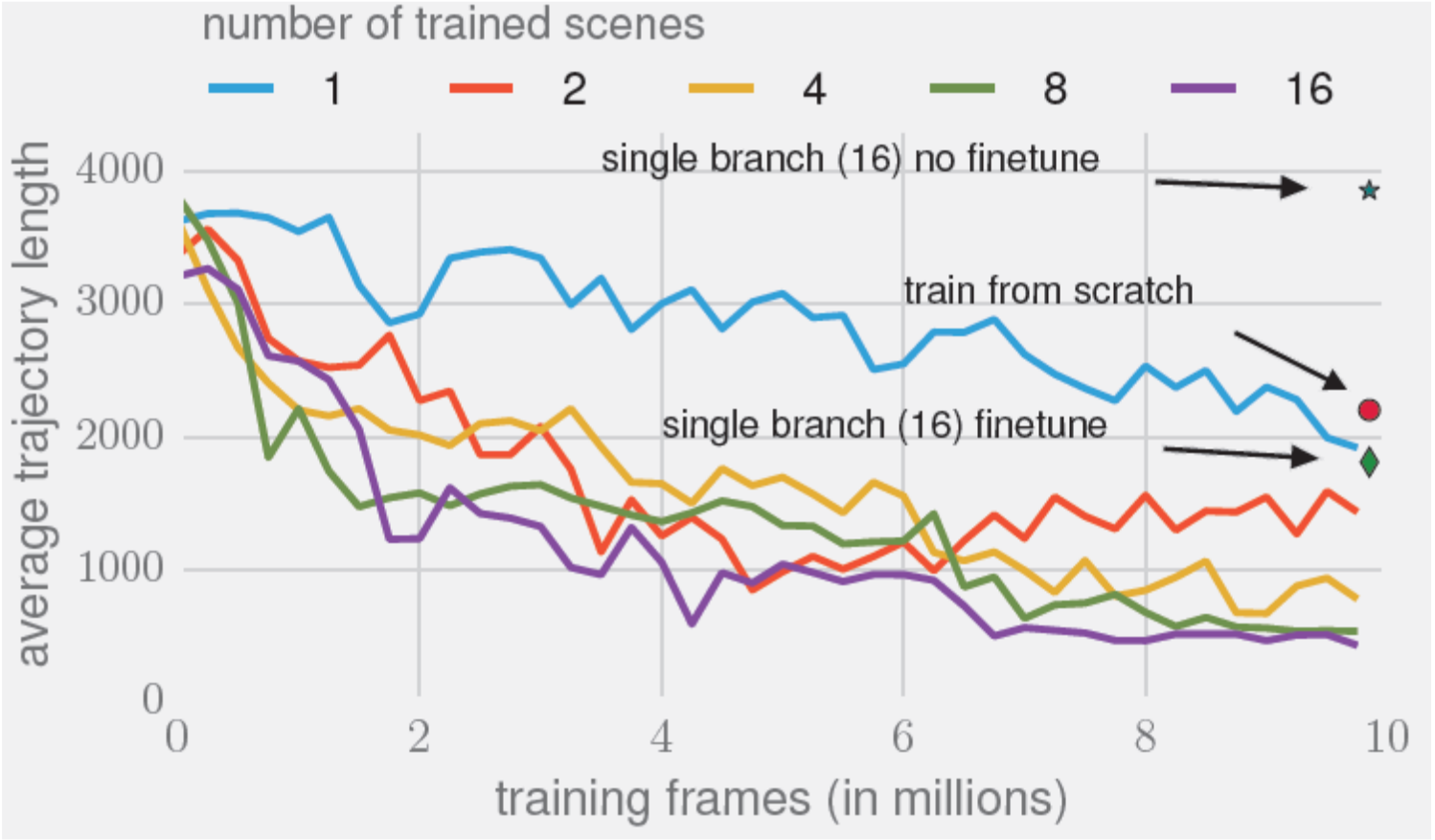
SCITOS

Test environment

Robot experiment setup. Our experiments are conducted on a SCITOS mobile robot

On the right, we show the test environment and one target (microwave) that we have used for evaluation

Scene Generalization



As the number of trained scene instances increases, fine-tuning the scene-specific layers becomes faster

Applications-5

Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments

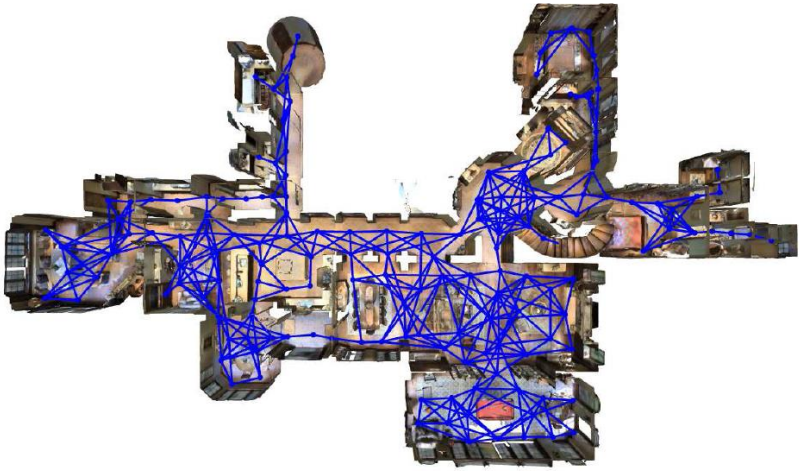
Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sunderhauf, Ian Reid, Stephen Gould, Anton van den Hengel

Vision-and-Language Navigation

Given language instructions, navigate in a 3D environment

Instruction: Head upstairs and walk past the piano through an archway directly in front. Turn right when the hallway ends at pictures and table. Wait by the moose antlers hanging on the wall.

R2R Dataset



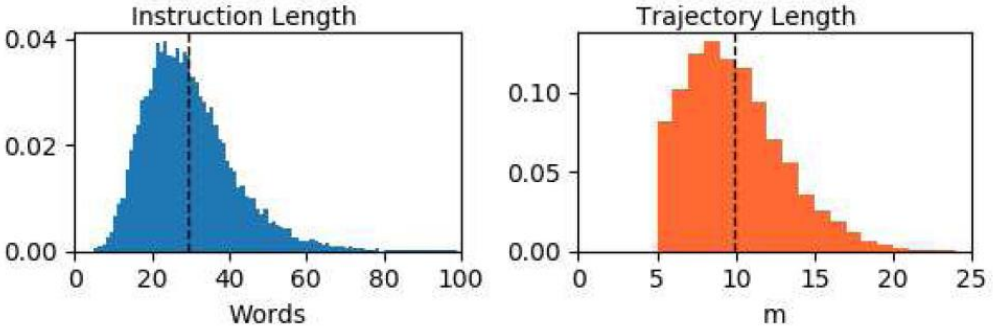


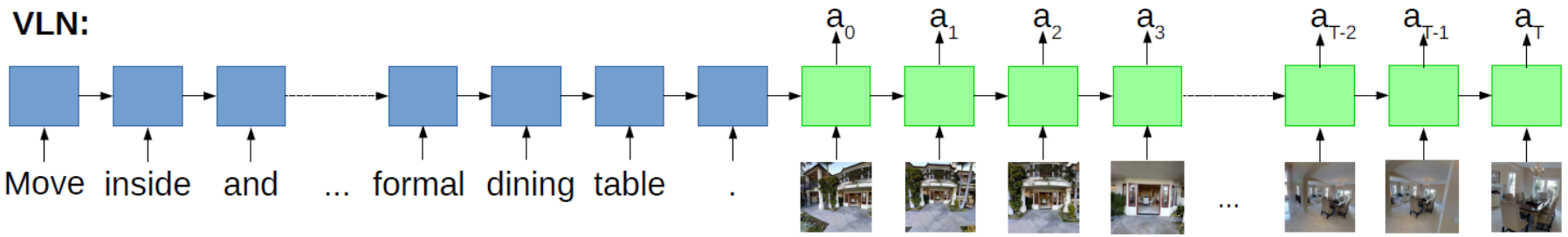
Figure 5. Distribution of instruction length and navigation trajectory length in the R2R dataset.

Pass the pool and go indoors using the double glass doors. Pass the large table with chairs and turn left and wait by the wine bottles that have grapes by them.

Walk straight through the room and exit out the door on the left. Keep going past the large table and turn left. Walk down the hallway and stop when you reach the 2 entry ways. One in front of you and one to your right. The bar area is to your left.

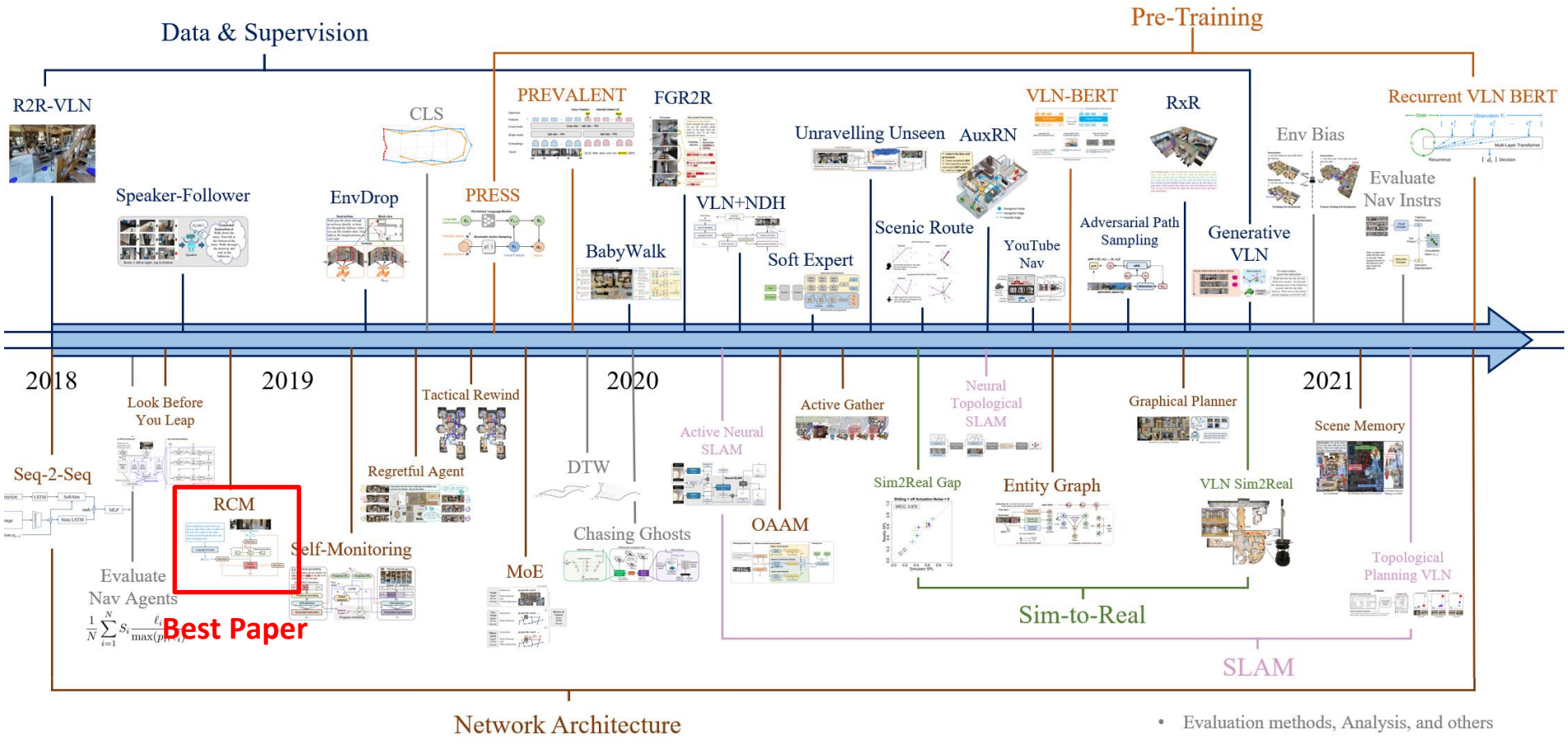
Enter house through double doors, continue straight across dining room, turn left into bar and stop on the circle on the ground.

Models

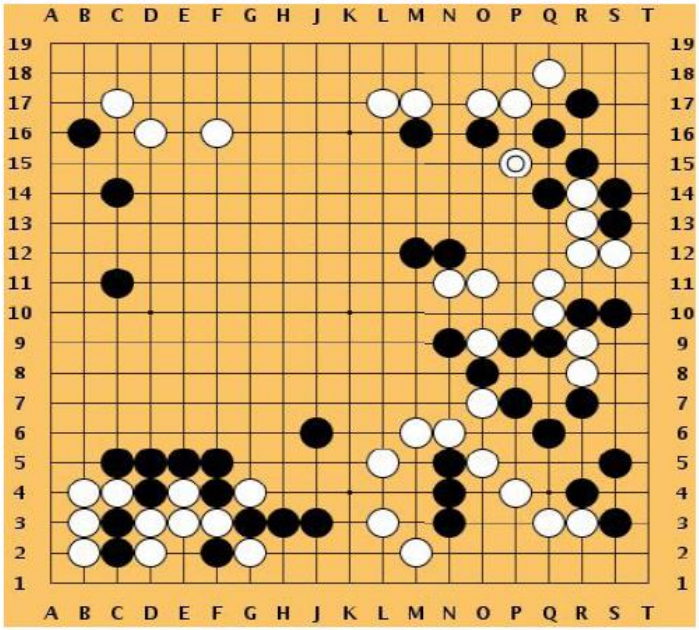


Leave the bedroom, and enter the kitchen. Walk forward, and take a left at the couch. Stop in front of the window.

VLN Timeline



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 policy and value networks in a Monte Carlo Tree Search algorithm to select actions by look ahead search

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 !