A Survey on Deep Reinforcement Learning

PhD Qualifying Examination

Siyi LI

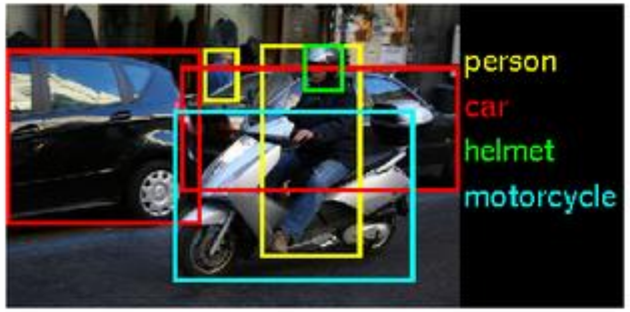
Supervisor:

Prof. Dit-Yan Yeung

2017-01-13

Background

- Deep learning methods have making major advances in solving many low-level perceptual tasks.



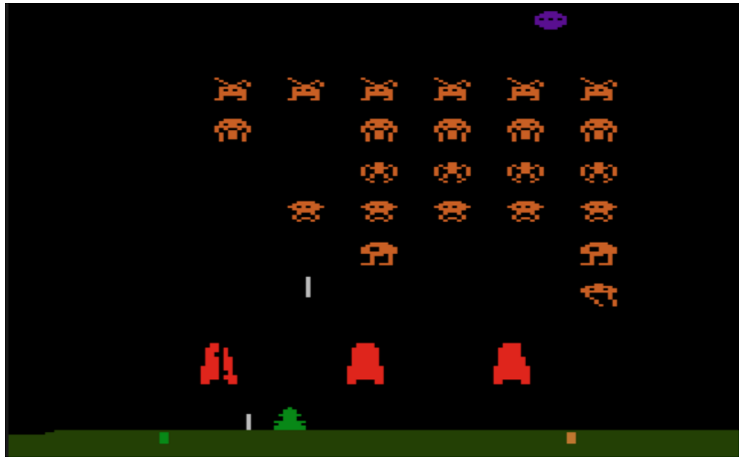
See (visual object recognition)

Read (text understanding)

Hear (speech recognition)

Background

- More sophisticated tasks that involve decision and planning require an higher level of intelligence.
- Real Artificial Intelligence system also requires the ability of reasoning, thinking and planning.



Playing Atari game

Robotic navigation

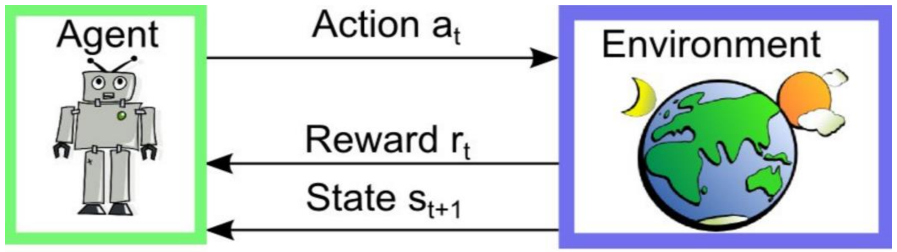
Limitations of Deep Learning

- Supervised learning assumptions
  - Training and testing instances are i.i.d variables
  - Training data are labeled data with strong supervision

- Reality of most real-world tasks
  - Strong supervision is expensive and scarce
  - Sequential interactive process violates the i.i.d assumption

Reinforcement Learning (RL) in a nutshell

- RL is a general-purpose framework for decision making
  - RL is for an agent to act with an environment
  - Each action influences the agent's future state
  - Feedback is given by a scalar reward signal
  - Goal: select actions to maximise future reward



Deep RL: Deep Learning + RL

- Traditional RL action approaches have been limited to domains with low-dimensional state spaces or handcrafted features
- By combining deep learning and RL, we want to embrace both the representation power of deep learning and generalization ability from RL
  - RL defines the objective
  - Deep Learning learns the representation

Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Deep Representations

- A deep representation is a composition of many functions, where each composition level is learning representations at different level of abstraction

$$\begin{tikzcd}
x \arrow[r] & h_1 \arrow[r] & \dots \arrow[r] & h_n \arrow[r] & y \arrow[r] & l \\
& \mathbf{w_1} \arrow[u] & \dots & \mathbf{w_n} \arrow[u] & &
\end{tikzcd}$$

- The weights are learned using backpropagation by chain rule

$$\begin{tikzcd}
\frac{\partial l}{\partial x} & \frac{\partial l}{\partial h_1} \arrow[l, "\frac{\partial h_1}{\partial x}"'] \arrow[d, "\frac{\partial h_1}{\partial w_1}"'] & \dots \arrow[l, "\frac{\partial h_2}{\partial h_1}"'] & \frac{\partial l}{\partial h_n} \arrow[l, "\frac{\partial h_n}{\partial h_{n-1}}"'] \arrow[d, "\frac{\partial h_n}{\partial w_n}"'] & \frac{\partial l}{\partial y} \arrow[l, "\frac{\partial y}{\partial h_n}"'] \\
& \frac{\partial l}{\partial \mathbf{w_1}} & \dots & \frac{\partial l}{\partial \mathbf{w_n}} &
\end{tikzcd}$$

Deep Neural Network

- A deep neural network typically consists of:
  - Linear transformations

$$h_{k+1} = Wh_k$$

  - Nonlinear activation functions

$$h_{k+1} = \sigma(h_k)$$

$$\sigma(\cdot) = \tanh(\cdot), \frac{1}{1+\exp(\cdot)}, \cdots$$

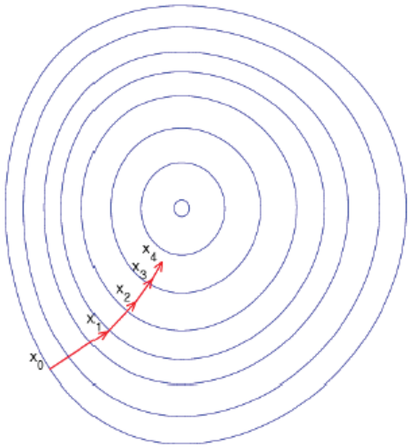
  - Loss function on the output
    - Mean squared error: $l = ||y - y^*||^2$
    - Log likelihood: $l = \log P(y^*)$

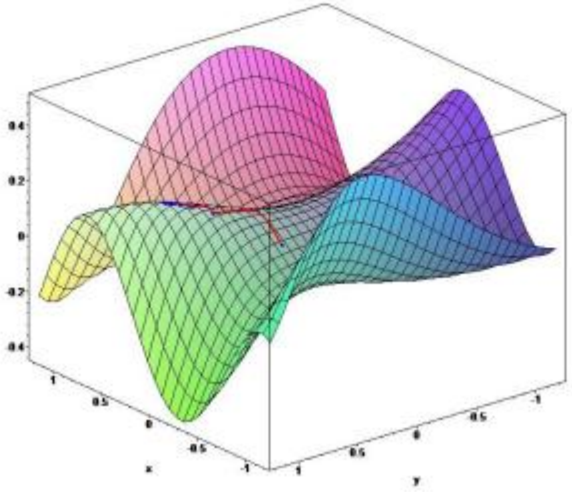
Training by Stochastic Gradient Descent

- Sample gradient of expected loss $L(\mathbf{w}) = \mathbb{E}[l]$ (better efficiency for large data)

$$\frac{\partial l}{\partial \mathbf{w}} \sim \mathbb{E}\left[\frac{\partial l}{\partial \mathbf{w}}\right] = \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

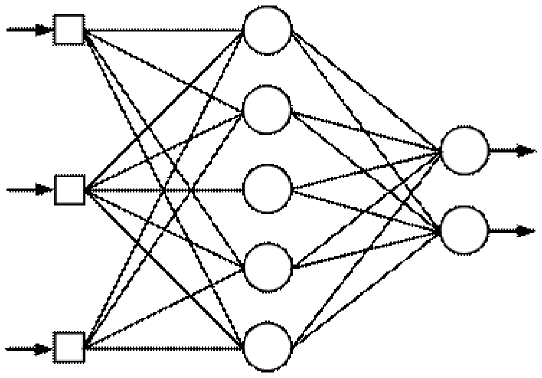
- Adjust $\mathbf{w}$ down the sampled gradient



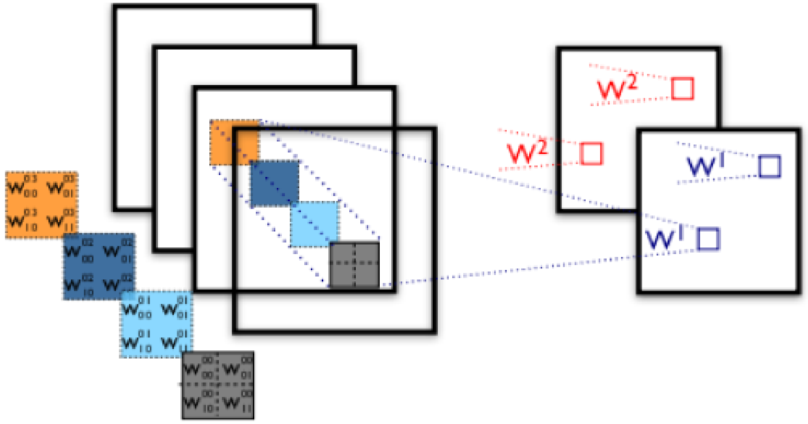


Deep Learning Models

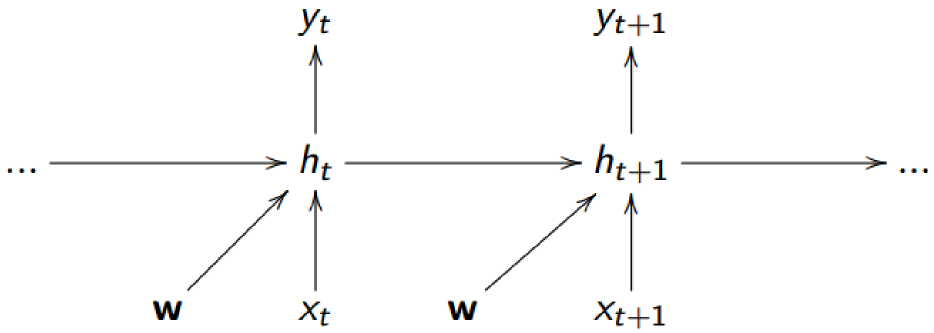
- Multilayer perceptrons (MLPs)
  - Fully-connected



- Convolutional neural networks (CNNs)
  - Weight sharing between local regions



- Recurrent neural networks (RNNs)
  - Weight sharing between time-steps



Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Markov Decision Processes (MDPs)

- MDPs formally describe an environment for RL, where the environment is fully observable

Definition

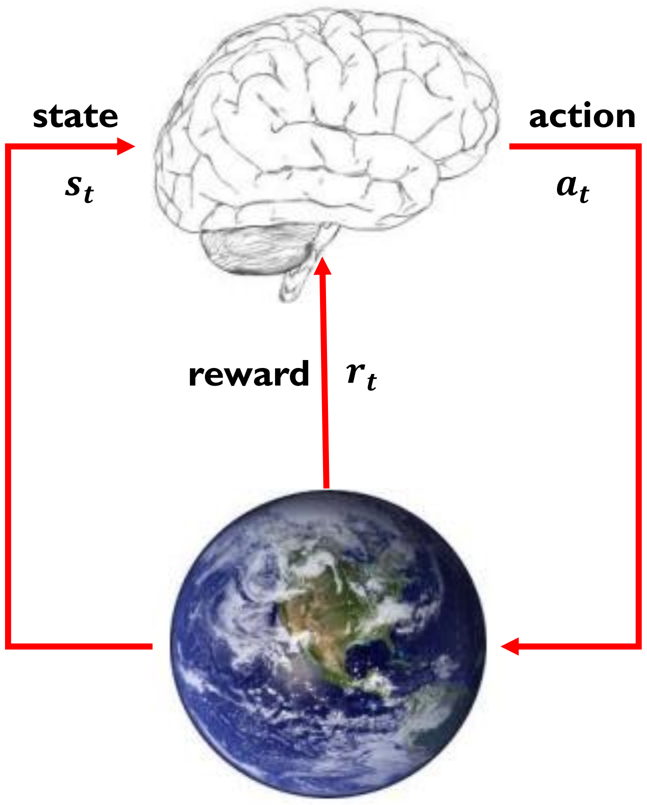
An MDP is a tuple $(\mathcal{S}, \mathcal{A}, f, R)$ consisting of:

- $\mathcal{S}$: The state space. In MDPs, the state is a sufficient statistic of the future.
- $\mathcal{A}$: The action space.
- f: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, \infty)$: The state transition probability density function.

$$P(s_{k+1} \in \mathcal{S}_{k+1} | s_k, a_k) = \int_{\mathcal{S}_{k+1}} f(s_k, a_k, s')ds'$$

- R: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$: The reward function.

$$r_k = R(s_k, a_k, s_{k+1})$$



Policy

A policy is the behavior of the agent.

- Stochastic policy π: $\mathcal{S} \times \mathcal{A} \mapsto [0, \infty)$.

$$P(a|s) = \pi(s, a)$$

- Deterministic policy π: $\mathcal{S} \mapsto \mathcal{A}$.

$$a = \pi(s)$$

Expected Return

- The goal of RL is to find the policy which maximizes the expected return

$$J(\pi) = \mathbb{E}\{g(r_0, r_1, \ldots)|\pi\}.$$

- In most cases, the function g is either the discounted sum of rewards or the average reward
- Discounted reward

$$g(r_0, r_1, \ldots) = \sum_{k=0}^{\infty} \gamma^k r_k$$

- Average reward

$$g(r_0, r_1, \ldots) = \lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} r_k$$

Value Function

- Consider the discounted reward case

$$J(\pi) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| d_0, \pi \right\}$$
$$= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi(s,a) \int_{\mathcal{S}} f(s,a,s') R(s,a,s') ds' da ds$$

$$d^\pi(s) = \sum_{k=0}^{\infty} \gamma^k p(s_k = s | d_0, \pi)$$

- A value function is the prediction of the above expected return
- Two definitions exist for the value function
  - State value function

$$V^\pi(s) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, \pi \right\}$$

  - State-action value function

$$V^\pi(s) = \mathbb{E}\left\{ Q^\pi(s,a) | a \sim \pi(s,\cdot) \right\}$$

$$Q^\pi(s,a) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| s_0 = s, a_0 = a, \pi \right\}$$

Bellman Equation and Optimality

- Value functions decompose into Bellman equations, i.e., the value functions can be decomposed into immediate reward plus discounted value of successor state

$$V^{\pi}(s) = \mathbb{E}\left\{R(s, a, s') + \gamma V^{\pi}(s')\right\}$$

$$Q^{\pi}(s, a) = \mathbb{E}\left\{R(s, a, s') + \gamma Q^{\pi}(s', a')\right\}$$

- An optimal value function is the maximum achievable value.

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

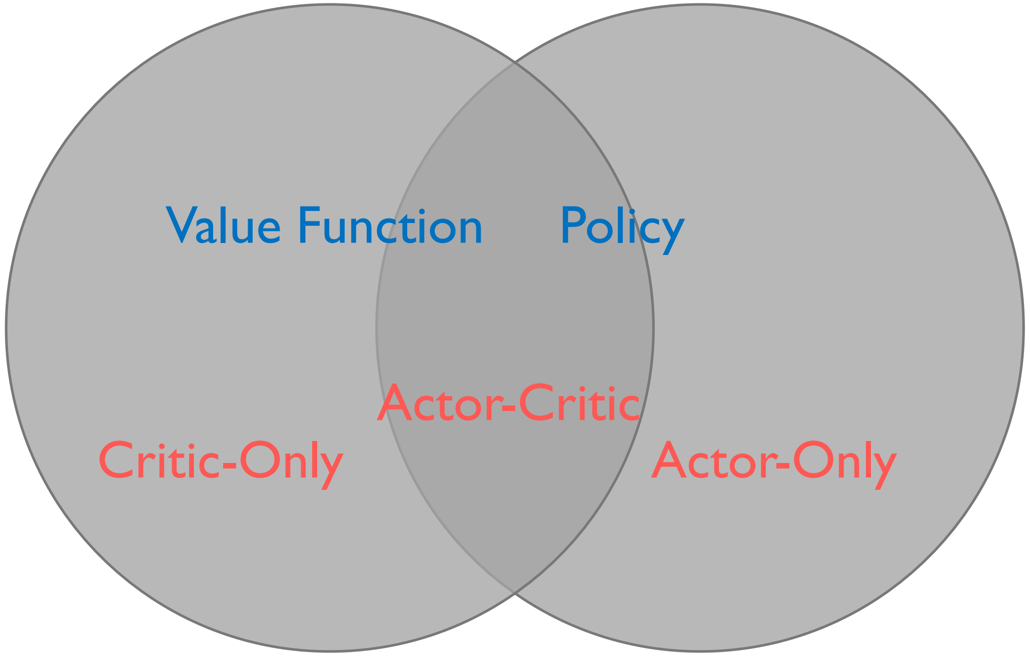
- Optimality for value functions are governed by the Bellman optimality equations.

$$V^{*}(s) = \max_{a} \mathbb{E}\left\{R(s, a, s') + \gamma V^{*}(s')\right\}$$

$$Q^{*}(s, a) = \mathbb{E}\left\{R(s, a, s') + \max_{a'} \gamma Q^{*}(s', a')\right\}$$

Approaches to RL

- Critic-Only Methods
  - Learnt value function
  - Implicit policy
- Actor-Only Methods
  - No value function
  - Learnt policy
- Actor-Critic Methods
  - Learnt value function
  - Learnt policy



Actor and *critic* are synonyms for the *policy* and *value* function.

Critic-Only

- A value function defines optimal policy.
- In critic-only methods, policy can be derived by selecting *greedy actions*

$$\pi^*(s) = \arg\max_a \mathbb{E}\left\{R(s,a,s') + \gamma V^*(s')\right\}$$

$$\pi^*(s) = \arg\max_a Q^*(s,a).$$

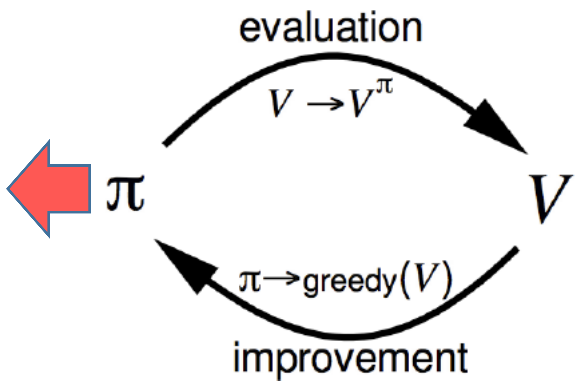
- Finding optimal policy from state-action value function is direct.
- Finding optimal policy from state value function is more complicated (requires the knowledge of transition model)

Critic-Only

- Dynamic programming-based methods
  - Policy iteration
  - Value iteration
- Monte Carlo (MC) methods
- Temporal difference (TD) learning methods
  - TD(λ)
  - Q-learning
  - SARSA

Dynamic Programming (DP)

- DP methods require a model of the state transition density function f and the reward function R to calculate the state value function
- DP is *model-based*
- Policy evaluation: updates the value function for the current policy
- Policy improvement: improve the policy by acting according to the current value function
- Typical methods:
  - Policy iteration
    - Alternates between policy evaluation and policy improvement
  - Value iteration
    - No explicit policy
    - Directly update value function



Monte Carlo Methods

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of transitions/rewards
- MC learns from complete episodes.
  - All episodes must terminate
- MC uses simple idea: empirical mean return
- The estimates are unbiased but have high variance.

Temporal Difference (TD) Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of transitions/rewards
- TD learns from incomplete episodes, it can learn online after every step
- TD uses temporal errors to update value function

$$V'(s) = V(s) + \alpha \left(R(s, a, s') + \gamma V(s') - V(s) \right)$$

$$Q'(s, a) = Q(s, a) + \alpha \left(R(s, a, s') + \gamma Q(s', a') - Q(s, a) \right)$$ SARSA

$$Q'(s, a) = Q(s, a) + \alpha \left(R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$ Q-learning

- The estimates are biased but have low variance.

Actor-Only

- Critic-only methods do not scale well to high-dimensional or continuous action spaces, since selecting greedy actions is computationally intensive.
- Actor-only methods work with a parameterized family of policies over which optimization procedures can be used directly.
- Advantages
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages
  - Evaluating a policy is inefficient and of high variance

Policy Gradient

- Given policy π_θ with parameters θ, the goal is find best θ to maximize the expected return
- Can use gradient descent

$$\theta_{k+1} = \theta_k + \alpha_{a,k} \nabla_\theta J(\theta_k)$$

- Policy gradient

$$\nabla_\theta J(\theta) = \frac{\partial J}{\partial \pi_\theta} \frac{\partial \pi_\theta}{\partial \theta}$$

- How to estimate the policy gradient?
  - Finite-difference methods
  - Likelihood ratio methods

Finite-Difference Methods

- Idea is simple, i.e., to vary the policy parameterization by small increments and evaluate the cost by rollouts
- Estimate k-th partial derivative of object function

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- Simple, works even if policy is not differentiable
- Noisy, inefficient

Likelihood Ratio Methods (REINFORCE)

- We can formulate the expected return from the view of trajectories generated by rollouts

$$\tau \sim p(\tau|\theta) \quad J^\tau = \sum_{k=0}^{H} \gamma^k r_k \quad J(\theta) = \int_{\mathbb{T}} p(\tau|\theta) J^\tau \, d\tau$$

- Use likelihood ratios to compute the policy gradient

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathbb{T}} \nabla_\theta p(\tau|\theta) J^\tau \, d\tau \\ &= \int_{\mathbb{T}} p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) J^\tau \, d\tau \\ &= \mathbb{E}\left\{ \underbrace{\nabla_\theta \log p(\tau|\theta)}\, J^\tau \right\}. \end{aligned}$$

Do not need to compute the system dynamics

$$\nabla_\theta \log p(\tau|\theta) = \sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(s_k, a_k)$$

$$\nabla_\theta J(\theta) = \mathbb{E}\left\{ \left(\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(s_k, a_k) \right) J^\tau \right\}$$

Likelihood Ratio Methods (REINFORCE)

- The above computation of policy gradient can be further reduced by replacing the trajectory return by the state-action value function

$$\nabla_\theta J(\theta) = \mathbb{E}\left\{\sum_{k=0}^{H} \nabla \log \pi_\theta(s_k, a_k) Q^\pi(s_k, a_k)\right\}$$

- The trajectory return or the state-action value can be estimated by the return v_t obtained from Monte Carlo rollouts
- Thus the estimated policy gradient may have large variance
- In practice, subtracting a baseline from the trajectory return or the state-action value helps a lot

Actor-Critic

- Critic-only
  - Pros: low variance
  - Cons: difficult for continuous action domains
- Actor-only
  - Pros: easy to handle continuous actions
  - Cons: high variance
- Actor-critic combines the advantages of actor-only and critic-only methods
  - Actions are generated by the parameterized actor
  - The critic supplies the actor with low variance gradient estimates

Policy Gradient Theorem

- Actor-critic methods rely on the following policy gradient theorem

Theorem

(Policy Gradient): For any MDP, in either the discounted reward or average reward setting, the policy gradient is given by

$$\nabla_\theta J(\theta) = \int_S d^\pi(s) \int_\mathcal{A} \nabla_\theta \pi_\theta(s,a) Q^\pi(s,a) dads$$

with $d^\pi(s)$ defined for the appropriate reward setting.

- The above theorem shows the relationship between the policy gradient and the exact critic function.

Policy Gradient With Function Approximation

- The following theorem shows that the state-action value function can be approximated with a certain function, without affecting the unbiasedness of the policy gradient estimate

Theorem

(Policy Gradient with Function Approximation): If the following two conditions are satisfied:

1. *Function approximator h_w is compatible to the policy*

$$\nabla_w h_w(s,a) = \nabla_\theta \log \pi_\theta(s,a),$$

2. *Function approximator h_w minimizes the following mean-squared error*

$$\varepsilon = \int_S d^\pi(s) \int_A \pi_\theta(s,a) \left\{ \left(Q^\pi(s,a) - h_w(s,a) \right)^2 \right\},$$

where $\pi_\theta(s,a)$ denotes the stochastic policy, parameterized by θ, then

$$\nabla_\theta J(\theta) = \int_S d^\pi(s) \int_A \nabla_\theta \pi_\theta(s,a) h_w(s,a) dads.$$

Reducing Variance Using a Baseline

- The policy gradient theorem generalizes to the case where a state-dependent baseline function is taken into account. This can reduce variance, without changing expectation

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left\{\nabla_\theta \pi_\theta(s,a) b(s)\right\} &= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(s,a) b(s) da ds \\ &= \int_{\mathcal{S}} d^\pi(s) b(s) \nabla_\theta \int_{\mathcal{A}} \pi_\theta(s,a) da ds = 0\end{aligned}$$

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(s,a) \left[h_w(s,a) - b(s)\right] da ds$$

- A good baseline is the state value function
- The policy gradient can be formulated by both the Q function and the advantage function

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \; A^{\pi_\theta}(s,a)\right]$$

Standard Actor-Critic Algorithms

- If both conditions in the above theorem are met, then the resulting algorithm is equivalent to the REINFORCE algorithm
- Practical actor-critic algorithms often relax the second condition: use TD learning to update the critic approximator.
- TD(0) actor-critic

$$
\begin{aligned}
\delta_k &= r_k + \gamma V_{w_k}(s_{k+1}) - V_{w_k}(s_k) \\
w_{k+1} &= w_k + \alpha_{c,k} \delta_k \nabla_w V_{w_k}(s_k) \\
\theta_{k+1} &= \theta_k + \alpha_{a,k} \delta_k \nabla_\theta \log \pi_\theta(s_k, a_k)
\end{aligned}
$$

- The TD error is actually an estimate of the advantage function

Practical Actor-Critic Variants

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic}
\end{aligned}
$$

Natural Policy Gradient

- The vanilla gradient is sensitive to policy parameterizations
- The natural policy gradient is parameterization independent
- It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount

$$\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

Fisher information matrix (FIM)

$$G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T\right]$$

- Natural policy gradient methods converges faster in most practical cases. However, estimating the FIM may induce large computation cost

Natural Actor-Critic

- Using compatible function approximation

$$\nabla_w A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

- The natural policy gradient is then

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T w\right] \\ &= G_\theta w\end{aligned}$$

$$\nabla_\theta^{nat} J(\theta) = w$$

Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy

- Optimize loss function by stochastic gradient descent

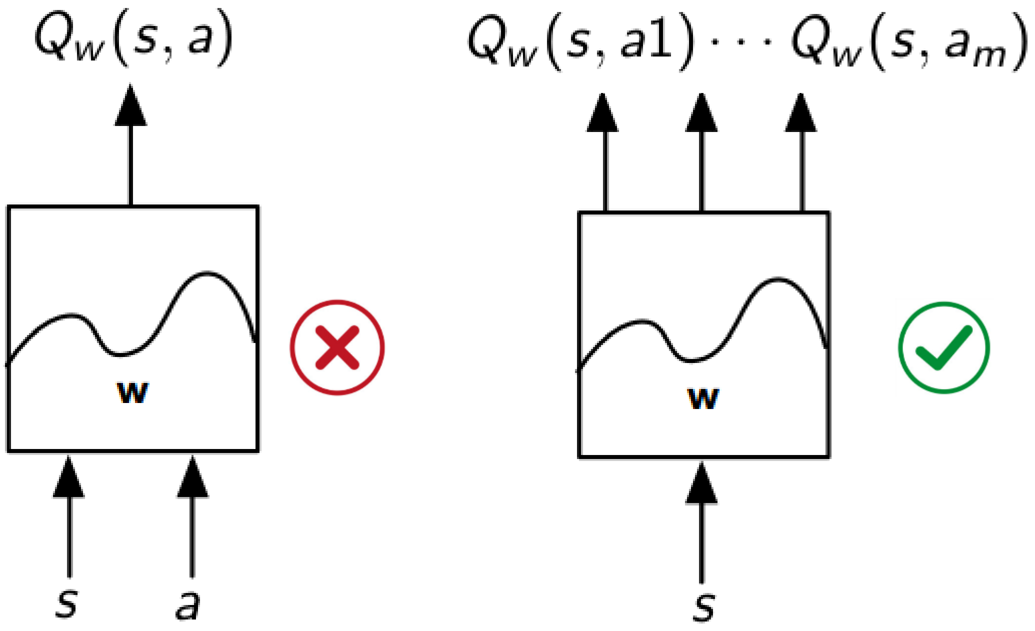
Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Q-Networks

- Represent the state-action value function by Q-network with weights w

$$Q_w(s, a) \approx Q^*(s, a)$$



Q-Learning

- Optimal Q-values obey Bellman equation

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

- Treat right-hand size as a target and minimize MSE loss by SGD

$$l = \left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right)^2$$

- Converge guarantee using table lookup representation
- But diverges using neural networks due to
  - Correlations between samples
  - Non-stationary targets

Deep Q-Networks (DQN)

- Experience replay
  - Build data set from agent's own experience
  - Sample experiences uniformly from data set to remove correlations

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

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

$\rightarrow \quad s, a, r, s'$

- Target Network
  - To deal with non-stationarity, target parameters $\hat{w}$ are held fixed

$$l = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left\{\left(r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a)\right)^2\right\}$$

Double DQN

- Q-learning is known to overestimate state-action values
  - The max operator uses the same values to select and evaluate an action

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

- The upward bias can be removed by decoupling the selection from the evaluation
  - Current Q-network is used to select actions
  - Older Q-network is used to evaluate actions

$$l = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left\{\left(r + \gamma Q_{\hat{w}_i}(s', \arg\max_{a'} Q_{w_i}(s',a')) - Q_{w_i}(s,a)\right)^2\right\}$$

Prioritized Replay

- Uniform experience replay samples transitions regardless of their significance
- Can weight experience according to their significance
- Prioritized replay store experience in priority queue according to the TD error

$$\left| r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a) \right|$$

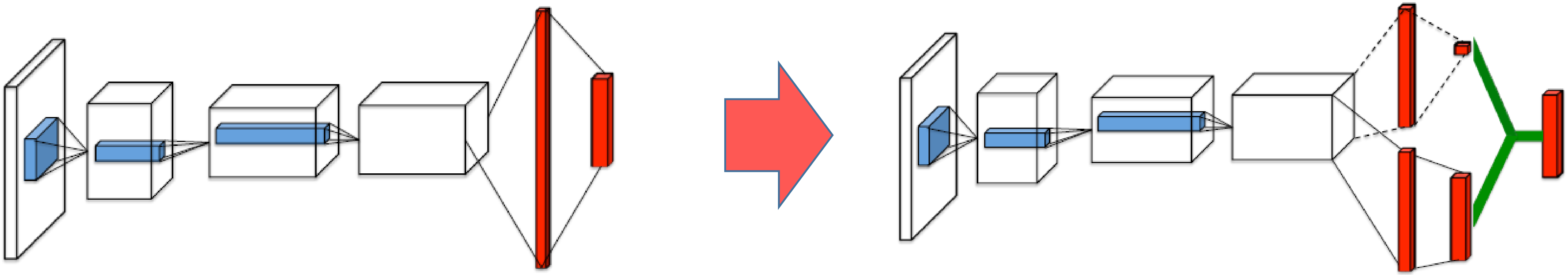
- Use stochastic sampling to increase sample diversity

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

$$p_i = |\delta_i| + \epsilon$$

Dueling Network

- Dueling network splits Q-network into two channels
  - Action-independent value function $V(s)$
  - Action-dependent advantage function $A(s, a)$

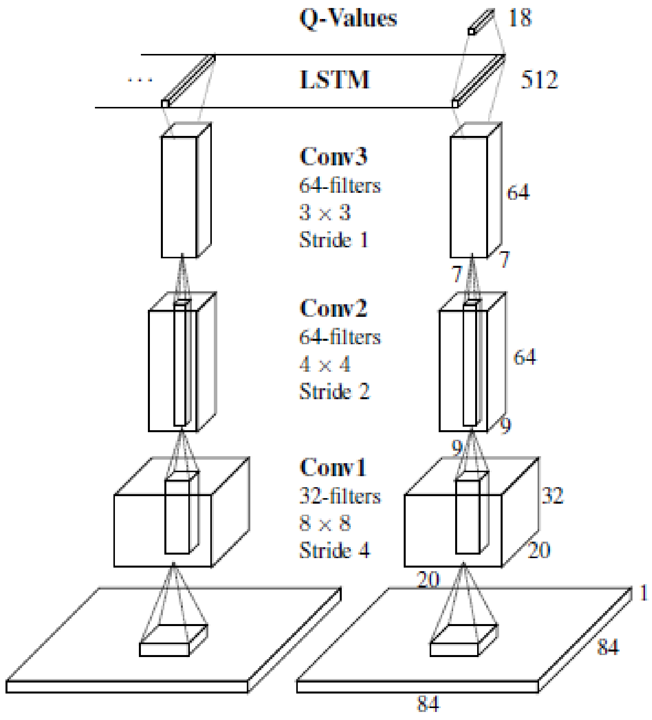


- The two stream are aggregated to get the Q function

$$Q(s, a; \beta, w_1, w_2) = V(s; \beta, w_1) + \left(A(s, a; \beta, w_2) - \max_{a'} A(s, a'; \beta, w_2) \right)$$

Deep Recurrent Q-Network (DRQN)

- DQNs learn a mapping from a limited number of past states.
- Most real world environments are Partially-Observable Markov Decision Process (POMDP)
- DRQN replaces DQN's first fully connected layer with a LSTM (one variant of RNN)



Asynchronous Q-Learning Variations

- Asynchronous RL
  - Exploits multithreading of standard CPU
  - Execute many instances of agent in parallel
  - Parallelism decorrelates data
    - Thus an alternative to experience replay, which is memory inefficient
  - Network parameters shared between threads
- Asynchronous one-step Q-learning
- Asynchronous one-step SARSA

$$r + \gamma Q(s', a')$$

- Asynchronous n-step Q-learning

$$r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)$$

Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Asynchronous Advantage Actor-Critic (A3C)

- Estimate state value function by neural networks

$$V_w(s) \approx \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \cdots | s\}$$

- Q-value estimated by an n-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_w(s_{t+n})$$

- Actor is updated by advantage policy gradient

$$\nabla_\theta J(\theta) = \nabla_\theta \pi_\theta(s_t, a_t)(q_t - V_w(s_t))$$

- Critic is updated by TD learning

$$l_v = (q_t - V_w(s_t))^2$$

Trust Region Policy Optimization (TRPO)

- Formulated as a trust region optimization problem, where each update of the policy is guaranteed to improve

$$\max \ L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$$
$$\text{subject to } \bar{D}_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta) \leq \delta$$

$$L_\pi(\tilde{\pi}) = J(\pi) + \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \tilde{\pi}(s,a) A^\pi(s,a)$$

- This provides a unifying perspective on a number of policy update schemes: standard policy gradient, natural policy gradient

$$\max_\theta \ \left[\nabla_\theta L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)|_{\theta=\theta_{\text{old}}}(\theta - \theta_{\text{old}})\right]$$

First order approximation to the objective

$$\text{subject to } \frac{1}{2}(\theta_{\text{old}} - \theta)^T \mathbf{F}(\theta_{\text{old}})(\theta_{\text{old}} - \theta) \leq \delta$$

$$\mathbf{F}(\theta_{\text{old}})_{ij} = \frac{\partial}{\partial \theta_i}\frac{\partial}{\partial \theta_j}\mathbb{E}\{D_{KL}(\pi_{\theta_{\text{old}}}(s,\cdot)||\pi_\theta(s,\cdot))\}$$

Natural policy gradient

$$\text{subject to } \frac{1}{2}||\theta - \theta_{\text{old}}||^2 \leq \delta$$

Standard policy gradient

Practical TRPO Algorithm

- Use the same approximation schemes as the natural policy gradient
- TRPO enforces the constraint by line search
  - Increases stability in practice

- Use a conjugate gradient algorithm to compute the natural gradient direction
  - Makes it practical for deep neural network policies

Deep Deterministic Policy Gradient (DDPG)

- Deterministic policy gradient

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} d^\pi(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) ds$$

- DDPG is the continuous analogue of DQN
  - Experience replay
  - Critic estimates value of current policy as in DQN

$$l_w = \left(r + \gamma Q_{\hat{w}}(s', \pi_{\hat{\theta}}(s')) - Q_w(s, a)\right)^2$$

  - Actor updates policy in the deterministic policy gradient direction

$$\nabla_a Q_w(s, a)|_{s=s_t, a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s)|_{s=s_t}$$

  - The critic provides loss function for the actor

Outline

- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Continuous Q-Learning

- General Q function parameterizations are difficult to find the maximum
- Specific Q function parameterizations can have analytic solution on the maximum

$$A(s, a; w^A) = -\frac{1}{2}(a - \mu(s; w^\mu))^T \mathbf{P}(s; w^P)(a - \mu(s; w^\mu))$$

$$Q(s, a; w^A, w^V) = A(s, a; w^A) + V(s; w^V)$$

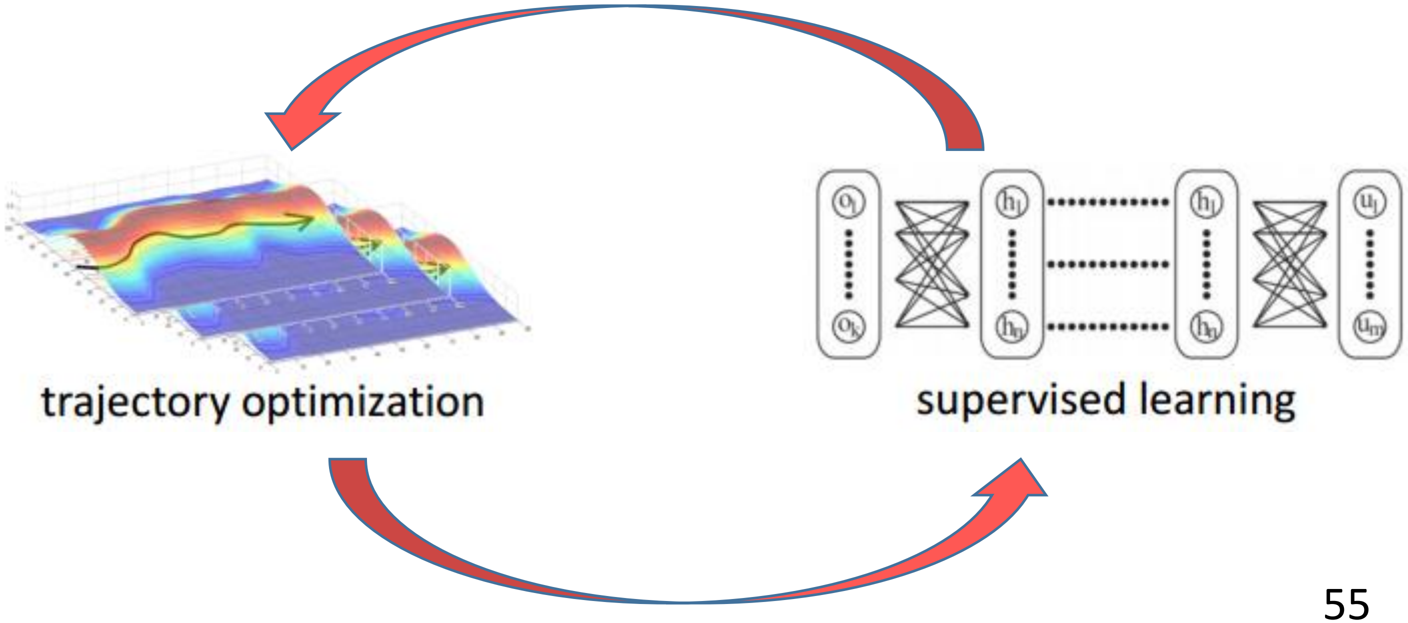
- The action that maximizes the Q function is always $\mu(s; w^\mu)$
- The parameterization can then be trained by DQN

Q-Learning with Model-Based Acceleration

- Use model-based methods to generate exploration behavior for Q-learning
  - In practice it often brings very small or no improvement
  - Off-policy model-based exploration is too different from the Q learning policy
- Imagination rollouts
  - Generate synthetic experiences under a learned model by model-based methods
  - Adding synthetic samples to replay buffer
  - Increases sample efficiency in practice

Guided Policy Search (GPS)

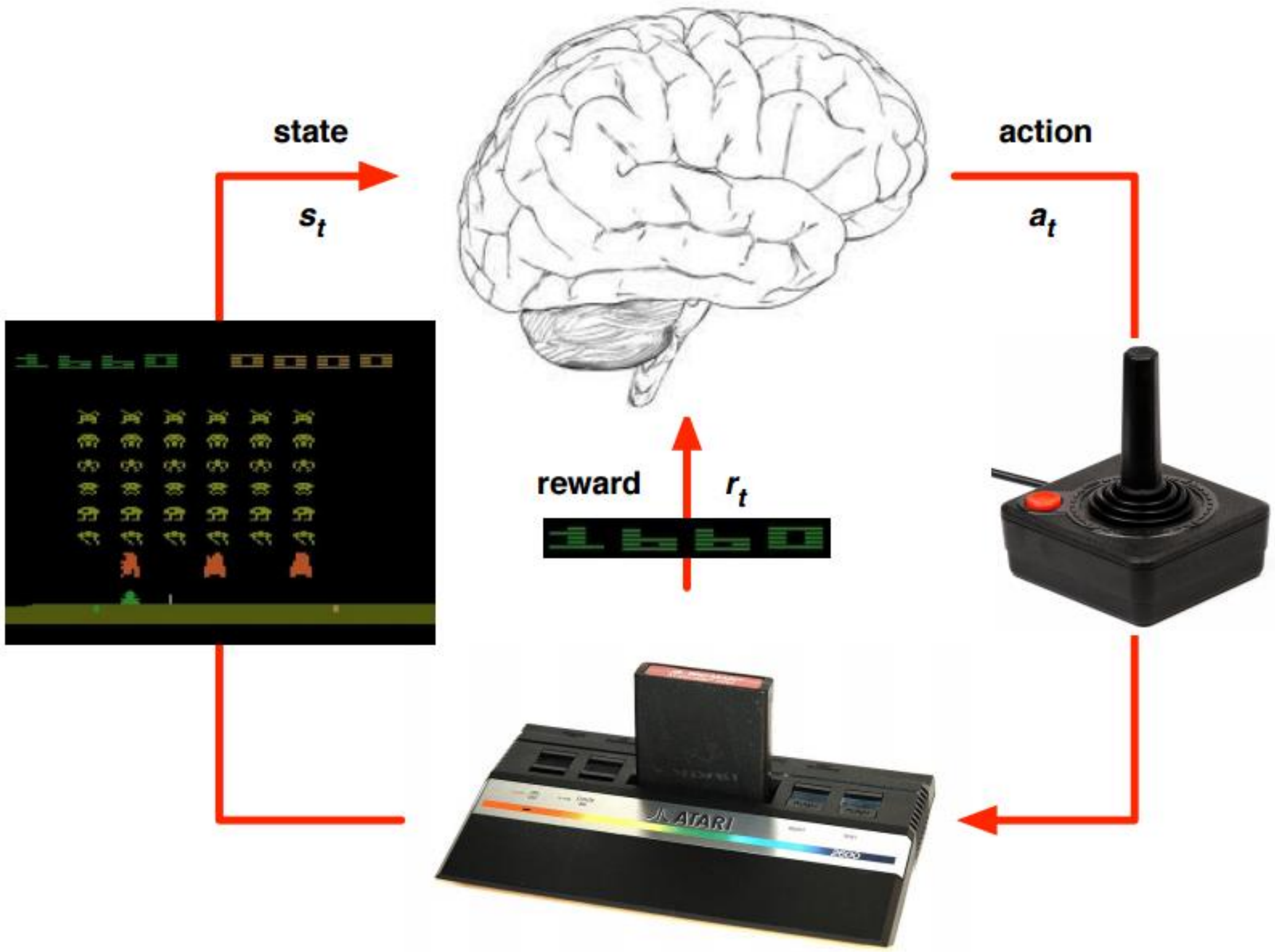
- GPS converts policy search into supervised learning
- Basically a model-based trajectory optimization algorithm generates training data to supervised train the neural network policy
- To enforce convergence, GPS alternates between trajectory optimization and supervised learning
- GPS is data efficient



Outline

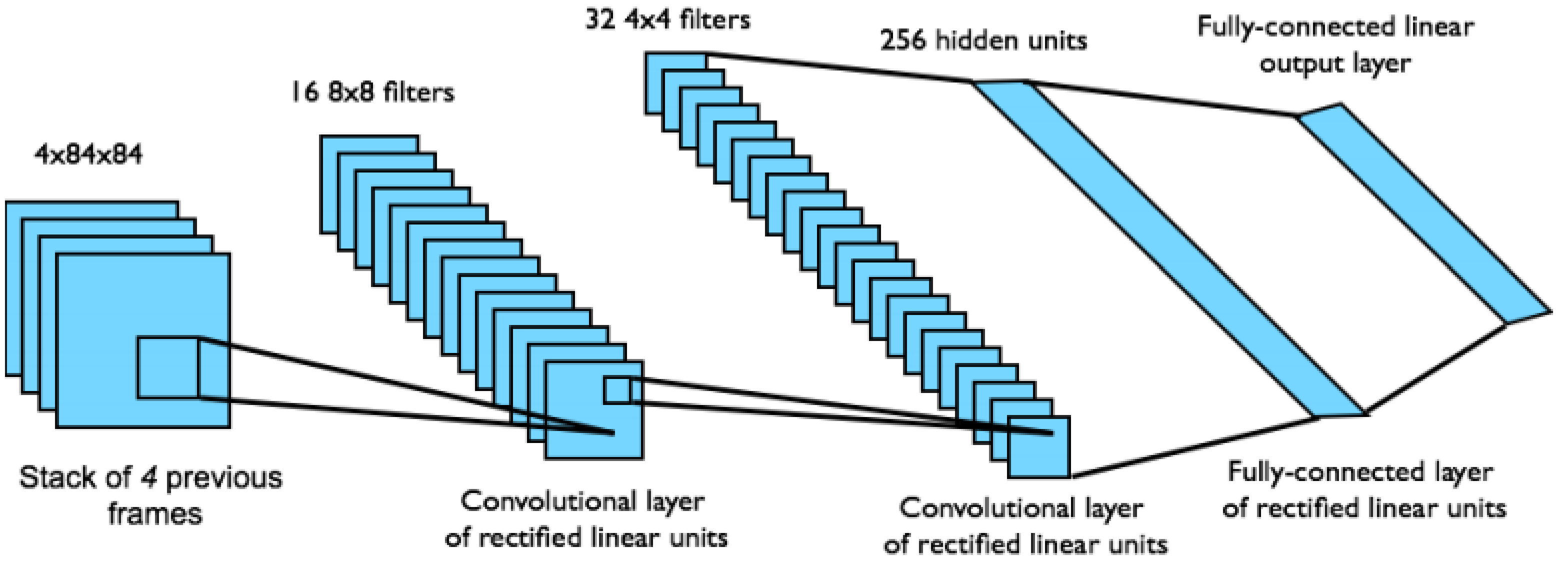
- Introduction to Deep Learning
- Introduction to Reinforcement Learning (RL)
- Value-Based Deep RL
- Policy-Based Deep RL
- Other Deep RL Extensions
- Deep RL Applications

Deep RL in Atari

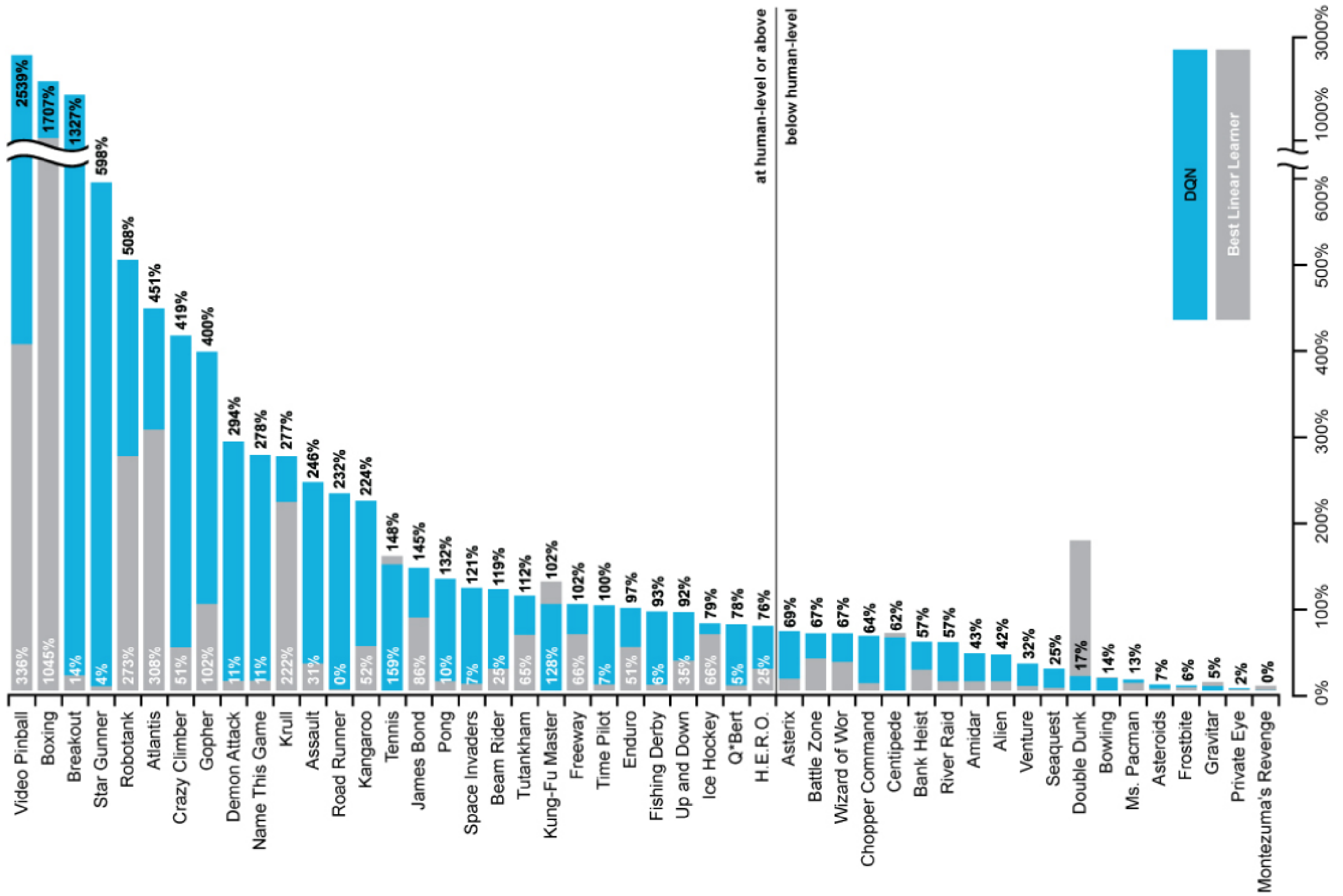


DQN in Atari

- End-to-end learning of state-action values from raw pixels
- Input state is stack of raw pixels from last 4 frames
- Output is state-action values from all possible actions
- Reward is change in score for that step

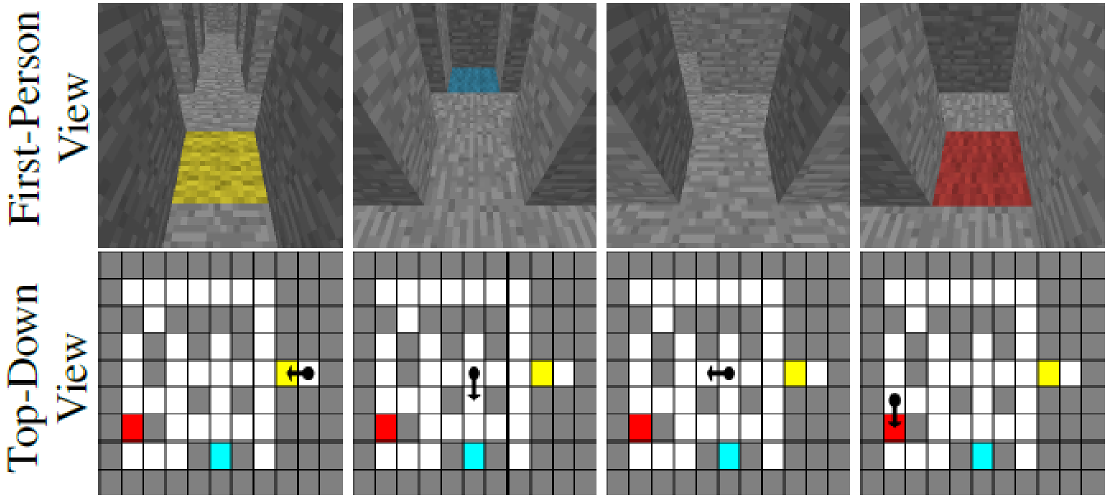


DQN Results in Atari



DQN Variants in Minecraft

- Challenges in environments
  - Partial observability (first-person visual observations)
  - Delayed rewards
  - High-dimensional perception



- Combine DQN with memory network to solve this kind of task

Other Games

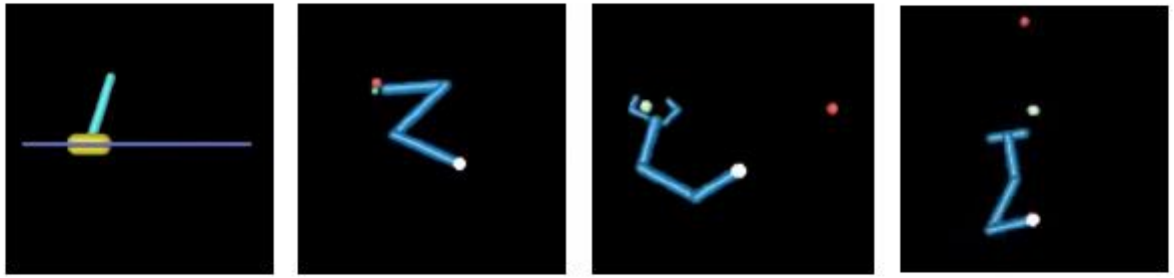
- First-person shooter games

- Car racing games

Deep RL in Go

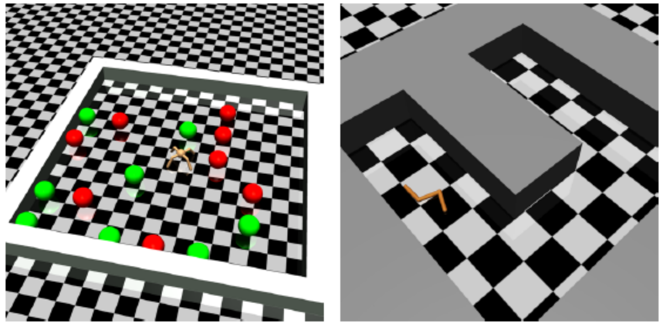
- Use supervised learning followed by deep RL
- Falls in the category of actor-critic framework
  - Use policy network to select moves
  - Use value network to evaluate board positions
- The learning approach is further combined with Monte Carlo search
- It beats the human world champion

Deep RL for Classic Control Tasks



Simple tasks

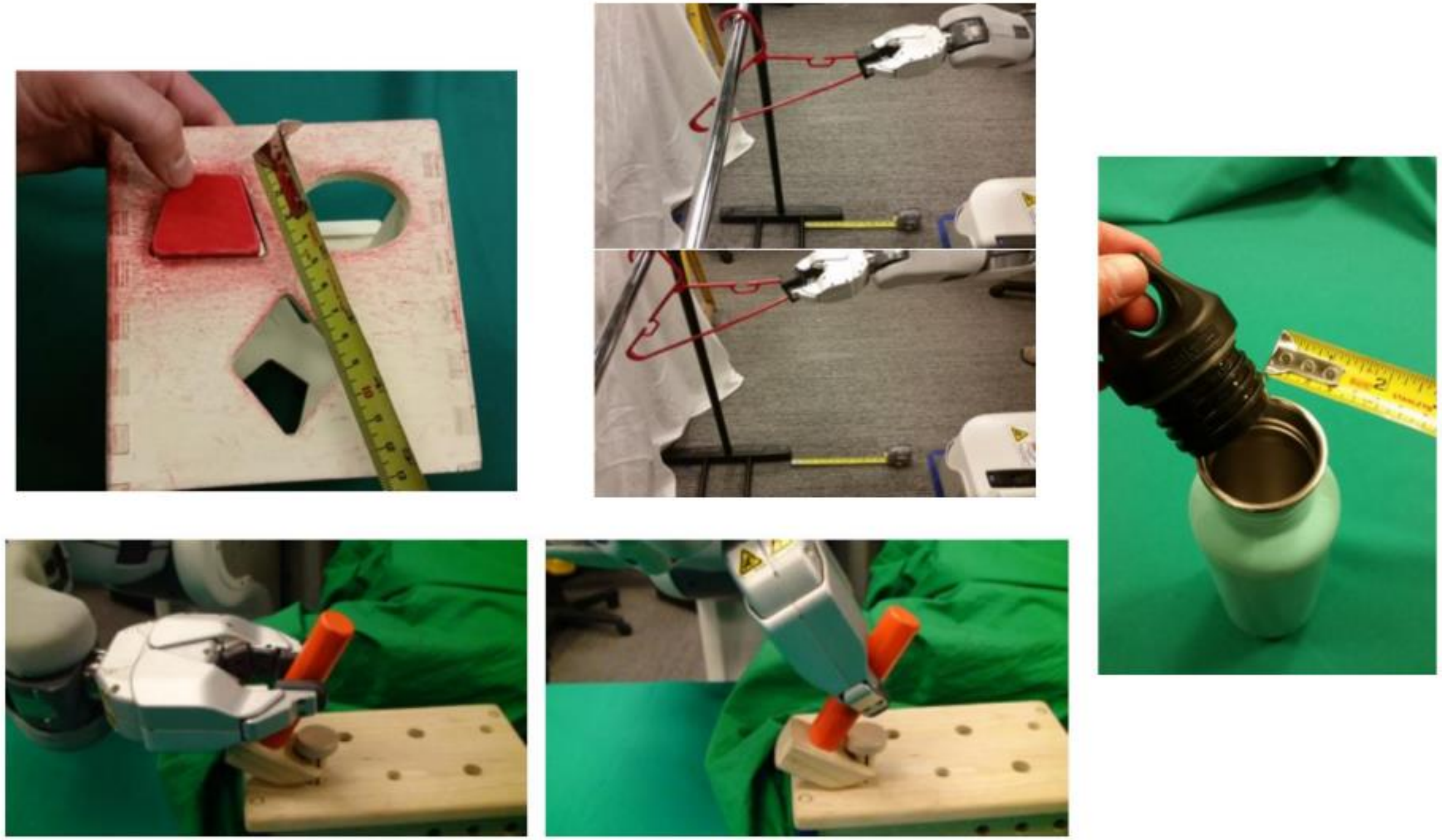
Locomotion tasks



Hierarchical tasks

Suitable for benchmark various algorithms

Deep RL in Real-World Robotics



Possible Future Research Directions

- Model-free deep RL methods are data inefficient
  - Successful applications focus on game playing and simulator
- How to design more efficient deep RL method
  - Combing with model-based methods or supervised learning
- How to transfer policies trained in simulator to real-world environments
- Transfer learning and multi-task learning of similar policies