Topics:

- Reinforcement Learning Part 3
  - **Policy Gradients**

CS 4644-DL / 7643-A

ZSOLT KIRA

RL: Sequential decision making in an environment with evaluative feedback.

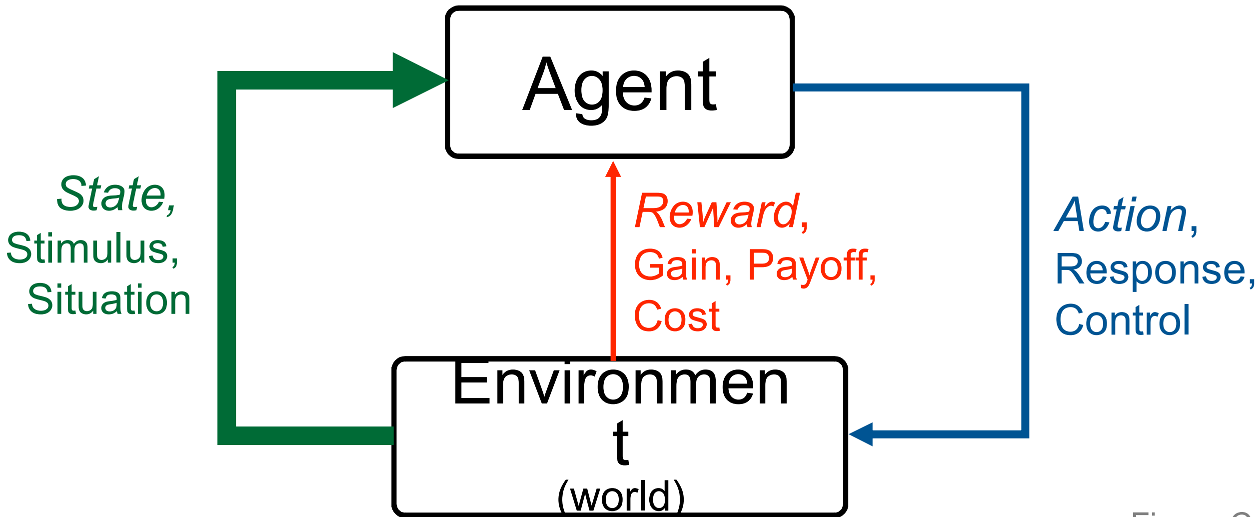


Figure Credit: Rich Sutton

- **Environment** may be unknown, non-linear, stochastic and complex.
- **Agent** learns a **policy** to map states of the environments to actions.
  - Seeking to maximize cumulative reward in the long run.

What is Reinforcement Learning?

- **MDPs**: Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
  - $\mathcal{S}$: Set of possible states
  - $\mathcal{A}$: Set of possible actions
  - $\mathcal{R}(s, a, s')$: Distribution of reward
  - $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as p(s'|s,a)
  - γ : Discount factor

Markov Decision Processes (MDPs)

Georgia Tech

- **MDPs**: Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
  - $\mathcal{S}$: Set of possible states
  - $\mathcal{A}$: Set of possible actions
  - $\mathcal{R}(s, a, s')$: Distribution of reward
  - $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as p(s'|s,a)
  - γ : Discount factor
- **Interaction trajectory**: $\ldots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots$

Markov Decision Processes (MDPs)

Georgia Tech

What we want

e.g.

State	Action
A $\longrightarrow$	2
B $\longrightarrow$	1

A policy π

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

Definition of **optimal policy**

Some intermediate concepts and terms

A **Value function** (how good is a state?)

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

A **Q-Value function** (how good is a state-action pair?)

$$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \quad Q^\pi(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

$$Q^*(s,a) = \mathop{\mathbb{E}}_{\sim p(s'|s,a)} [r(s,a) + \gamma V^*(s')]$$

(Math in previous lecture)

Equalities relating optimal quantities

$$\boxed{V^*(s) = \max_a Q^*(s,a)}$$

$$\boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

We can then derive the Bellman Equation

$$Q^*(s,a) = \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma \max_a Q^*(s',a')\right]$$

This must hold true for an optimal Q-Value!

-> Leads to dynamic programming algorithm to find it

Summary of Last Time

Q-Learning

- We'd like to do Q-value updates to each Q-state:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

  - But can't compute this update without knowing T, R

- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - But we want to average over results from (s,a)
  - So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

- **Q-Learning** with **linear function approximators**

$$Q(s, a; w, b) = w_a^\top s + b_a$$

  - Has some theoretical guarantees

- **Deep Q-Learning**: Fit a **deep Q-Network** $Q(s, a; \theta)$

  - Works well in practice

  - Q-Network can take RGB images

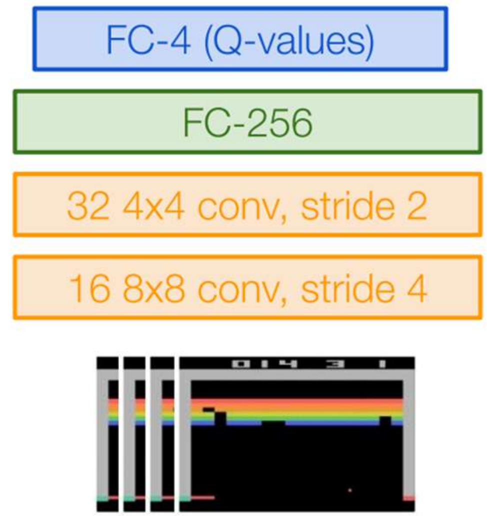


Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Deep Q-Learning

Algorithm 1 Deep Q-learning with Experience Replay

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

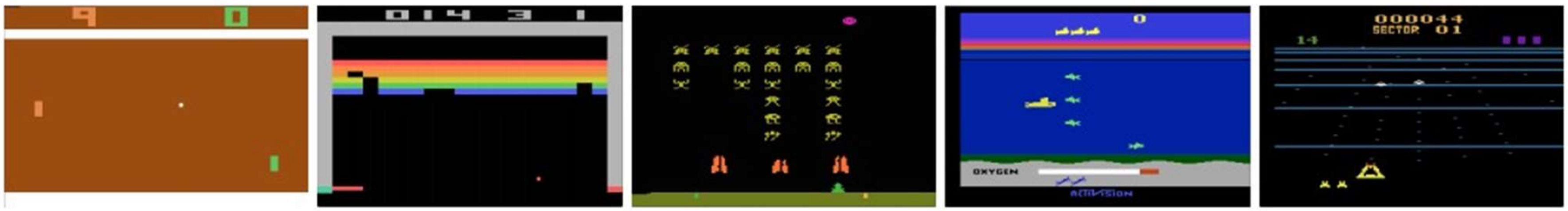
Experience Replay

Epsilon-greedy

Q Update

Deep Q-Learning Algorithm

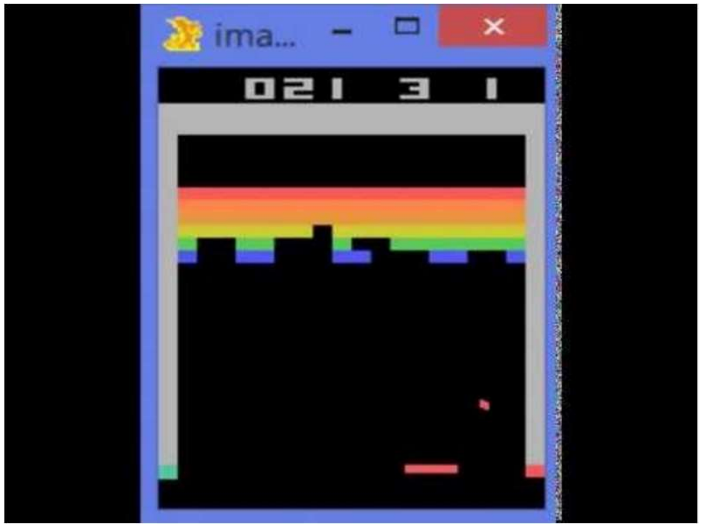
Atari Games

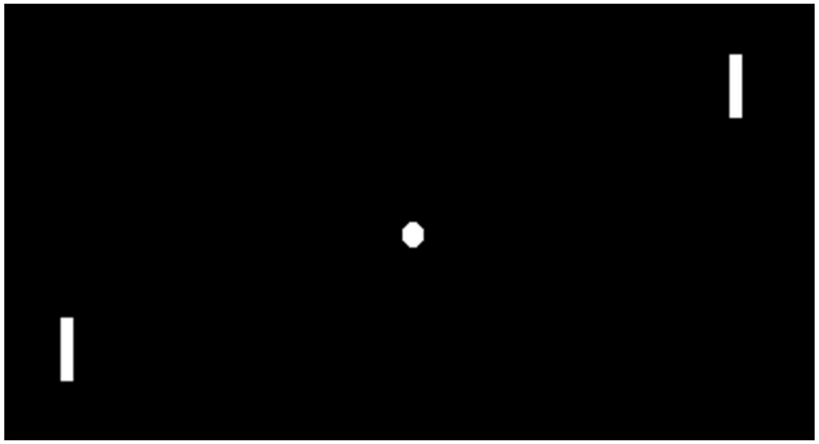


- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

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

Atari Games





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

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

In today's class, we looked at

- **Dynamic Programming**
  - Value, Q-Value Iteration

- **Reinforcement Learning (RL)**
  - The challenges of (deep) learning based methods
  - Value-based RL algorithms
    - Deep Q-Learning

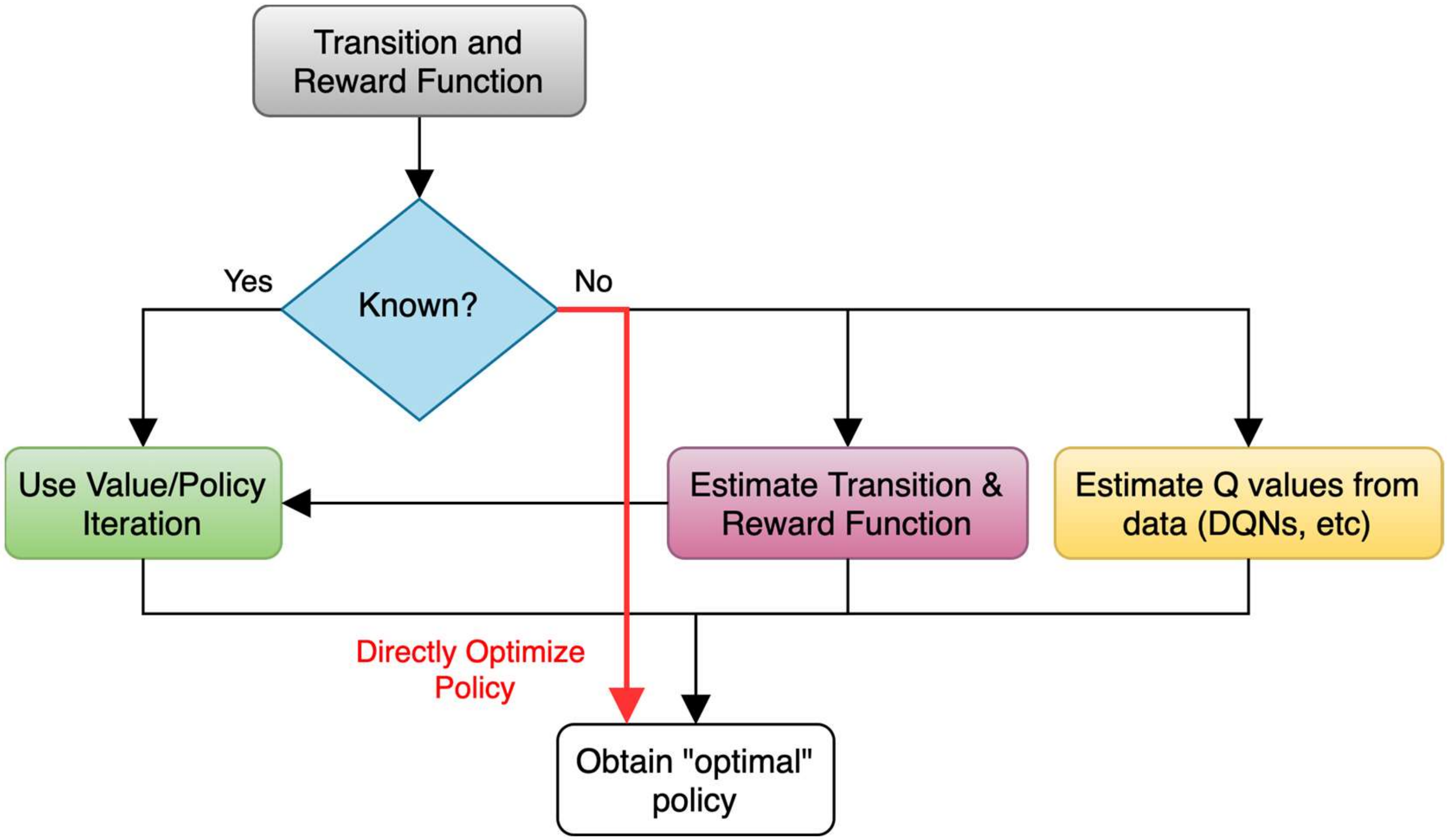
Now:

- **Policy-based RL algorithms** (policy gradients)

Summary

Georgia Tech

Policy Gradients, Actor-Critic



Overview

Georgia Tech

- Class of policies defined by parameters θ

$$\boxed{\pi_\theta(a|s) : \mathcal{S} \to \mathcal{A}}$$

  - Eg: θ can be parameters of linear transformation, deep network, etc.

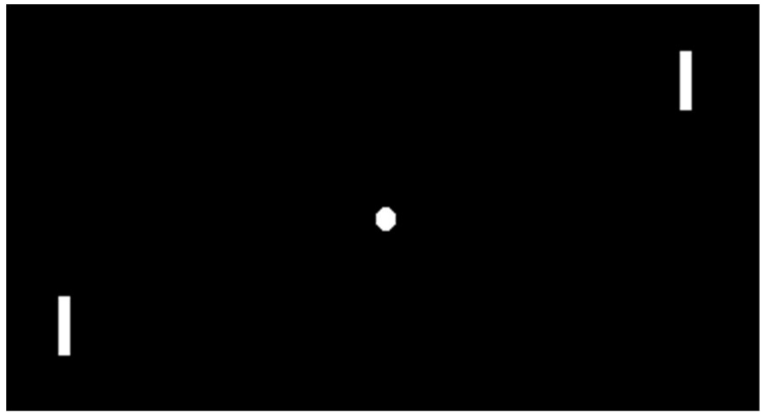
- Want to maximize:

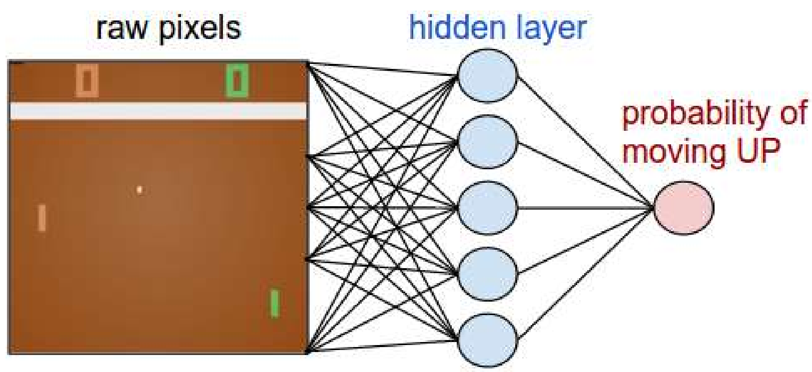
$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

- In other words,

$$\pi^* = \arg\max_{\pi:\mathcal{S}\to\mathcal{A}} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right] \quad \Rightarrow \quad \theta^* = \arg\max_{\theta} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

Parametrized Policy





Pong from Pixels

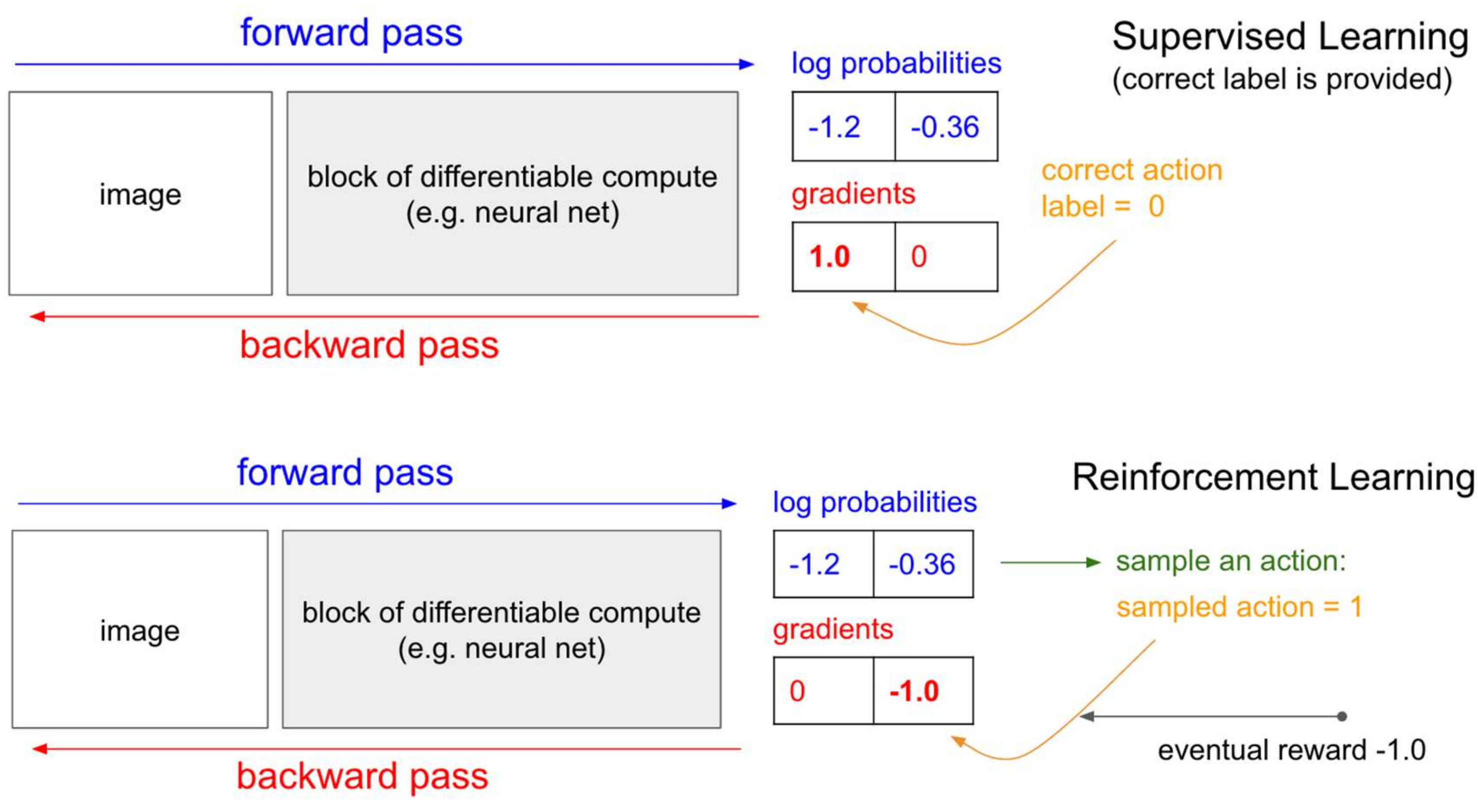


Image Source: http://karpathy.github.io/2016/05/31/rl/

Policy Gradient: Loss Function

- Slightly re-writing the notation

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

$$\pi_\theta(\tau) = p_\theta(\tau) = p_\theta\left(s_0, a_0, \ldots s_T, a_T\right)$$
$$= p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)$$

$$\boxed{\arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\mathcal{R}(\tau)\right]}$$

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\mathcal{R}(\tau)]$$

$$= \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)} \left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t) \right]$$

- How to gather data?
  - We already have a policy: π_θ
  - Sample N trajectories $\{\tau_i\}_{i=1}^N$ by acting according to π_θ

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} r(s_t^i, a_t^i)$$

Gathering Data/Experience

The REINFORCE Algorithm

- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \ ?$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Run the policy and sample trajectories

Compute policy gradient

Update policy

Slide credit: Sergey Levine

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[\mathcal{R}(\tau)]$$

$$= \nabla_\theta \int \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Expectation as integral

$$= \int \nabla_\theta \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Exchange integral and gradient

$$= \int \nabla_\theta \pi_\theta(\tau) \cdot \frac{\pi_\theta(\tau)}{\pi_\theta(\tau)} \cdot \mathcal{R}(\tau)d\tau$$

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

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

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)]$$

Deriving The Policy Gradient

$$\pi_\theta(\tau) = p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)$$

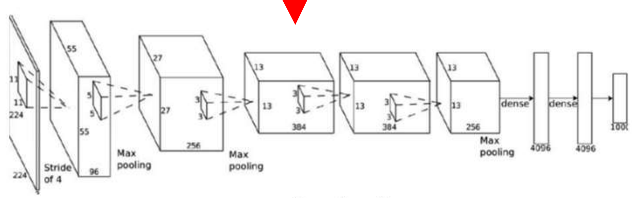
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\underbrace{\nabla_\theta \log \pi_\theta(\tau)}\mathcal{R}(\tau)]$$

$$\nabla_\theta \left[\cancel{\log p(s_0)} + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \cancel{\log p(s_{t+1} \mid s_t, a_t)}\right]$$

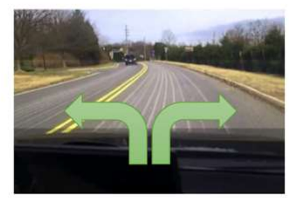
Doesn't depend on Transition probabilities!

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

$\mathbf{s}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$



$\mathbf{a}_t$

Continuous Action Space?

Deriving The Policy Gradient

Georgia Tech

The REINFORCE Algorithm

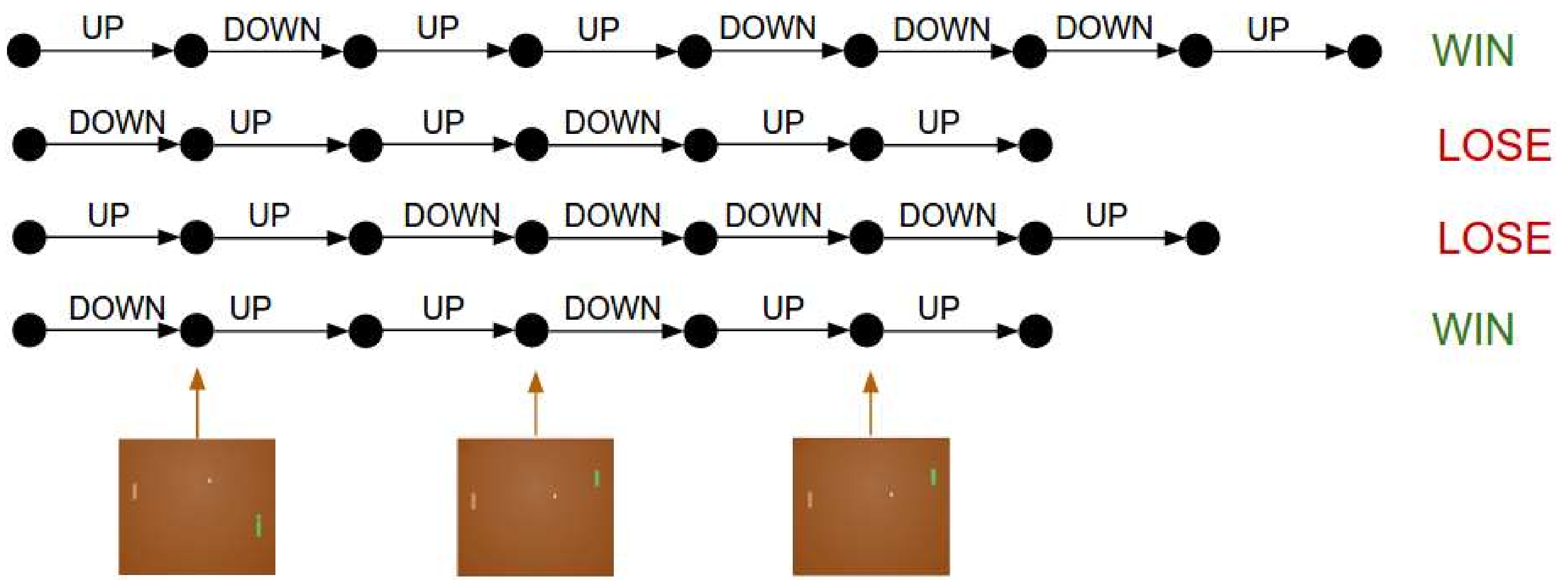
- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta \left(a_t^i \mid s_t^i \right) \cdot \sum_{t=1}^T \mathcal{R} \left(s_t^i \mid a_t^i \right) \right]$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Slide credit: Sergey Levine



Slide credit: Dhruv Batra

Drawbacks of Policy Gradients

Georgia Tech

Issues with Policy Gradients

- Credit assignment is hard!
  - Which specific action led to increase in reward
  - Suffers from high variance → leading to unstable training

Variance reduction

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

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

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

Variance reduction

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

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

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

Second idea: Use discount factor γ to ignore delayed effects

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

- Credit assignment is hard!
  - Which specific action led to increase in reward
  - Suffers from **high variance**, leading to unstable training

- How to reduce the variance?
  - Subtract an action independent baseline from the reward

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) \cdot \sum_{t=1}^{T}\left(\mathcal{R}\left(s_t, a_t\right) - b(s_t)\right)\right]$$

  - Why does it work? **Normalization constant (expected value doesn't change)**
  - What is the best choice of b?

Drawbacks of Policy Gradients

How to choose the baseline?

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

Q: What does this remind you of?

How to choose the baseline?

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

Q: What does this remind you of?

A: Q-function and value function!

Actor-Critic

- Learn both policy and Q function
  - Use the “actor” to sample trajectories
  - Use the Q function to “evaluate” or “critic” the policy

Actor-Critic

- Learn both policy and Q function
  - Use the “actor” to sample trajectories
  - Use the Q function to “evaluate” or “critic” the policy

- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \mathcal{R}(s, a) \right]$

- Actor-critic: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right]$

Actor-Critic

- Learn both policy and Q function
  - Use the "actor" to sample trajectories
  - Use the Q function to "evaluate" or "critic" the policy
- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\mathcal{R}(s,a)\right]$
- Actor-critic: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)Q^{\pi_\theta}(s,a)\right]$
- Q function is unknown too! Update using $\mathcal{R}(s,a)$

Actor-Critic

- Initialize s, θ (policy network) and β (Q network)

Actor-Critic

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$

Actor-Critic

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$
- For each step:
  - Sample reward $\mathcal{R}(s, a)$ and next state $s' \sim p(s'|s, a)$

Actor-Critic

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$
- For each step:
  - Sample reward $\mathcal{R}(s,a)$ and next state $s' \sim p(s'|s,a)$
  - evaluate "actor" using "critic" $Q_\beta(s,a)$

Actor-Critic

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$
- For each step:
  - Sample reward $\mathcal{R}(s,a)$ and next state $s' \sim p(s'|s,a)$
  - evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s) Q_\beta(s,a)$$

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$
- For each step:
  - Sample reward $\mathcal{R}(s,a)$ and next state $s' \sim p(s'|s,a)$
  - evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s) Q_\beta(s,a)$$

  - Update "critic": $\mathrm{MSE\ Loss} := \Big(Q_{new}(s,a) - (r + \max_a Q_{old}(s',a)) \Big)^2$
    - Recall Q-learning

Actor-Critic

Georgia Tech

- Initialize s, θ (policy network) and β (Q network)
- sample action $a \sim \pi_\theta(\cdot|s)$
- For each step:
  - Sample reward $\mathcal{R}(s,a)$ and next state $s' \sim p(s'|s,a)$
  - evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s) Q_\beta(s,a)$$

  - Update "critic":
    - Recall Q-learning $\text{MSE Loss} := \left(Q_{new}(s,a) - (r + \max_a Q_{old}(s',a))\right)^2$

$$a \leftarrow a', s \leftarrow s'$$

    - Update β Accordingly

Actor-Critic

Georgia Tech

How to choose the baseline?

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

Q: What does this remind you of?

A: Q-function and value function!

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

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

Actor-critic

- In general, replacing the policy evaluation or the "critic" leads to different flavors of the actor-critic
  - REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \mathcal{R}(s,a) \right]$
  - Q – Actor Critic $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \right]$
  - Advantage Actor Critic: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s,a) \right]$

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

"how much better is an action than expected?

Summary

- Policy Learning:
  - Policy gradients
  - REINFORCE
  - Reducing Variance (Homework!)
- Actor-Critic:
  - Other ways of performing "policy evaluation"
  - Variants of Actor-critic

Summary

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

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

- Sparse long-horizon tasks (Montezuma's revenge)
- Imitation Learning, inverse reinforcement learning
- Sim2Real – Simulation to real, domain randomization
- Lifelong Learning
- Safety
- World Models

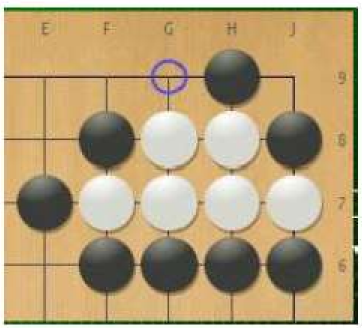
Open Problems / Challenges

Georgia Tech

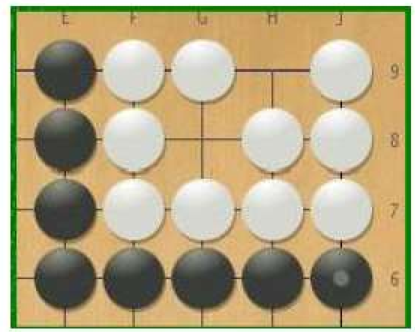
Playing Go

Rules

- Each player puts a stone on the goban, black first
- Each stone remains on the goban, except:



group w/o degree freedom is killed



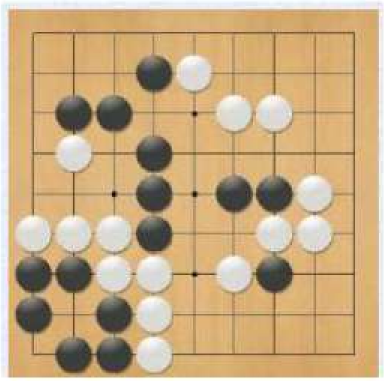
a group with two eyes can't be killed

- The goal is to control the max. territory

Go is a Difficult Game

Features

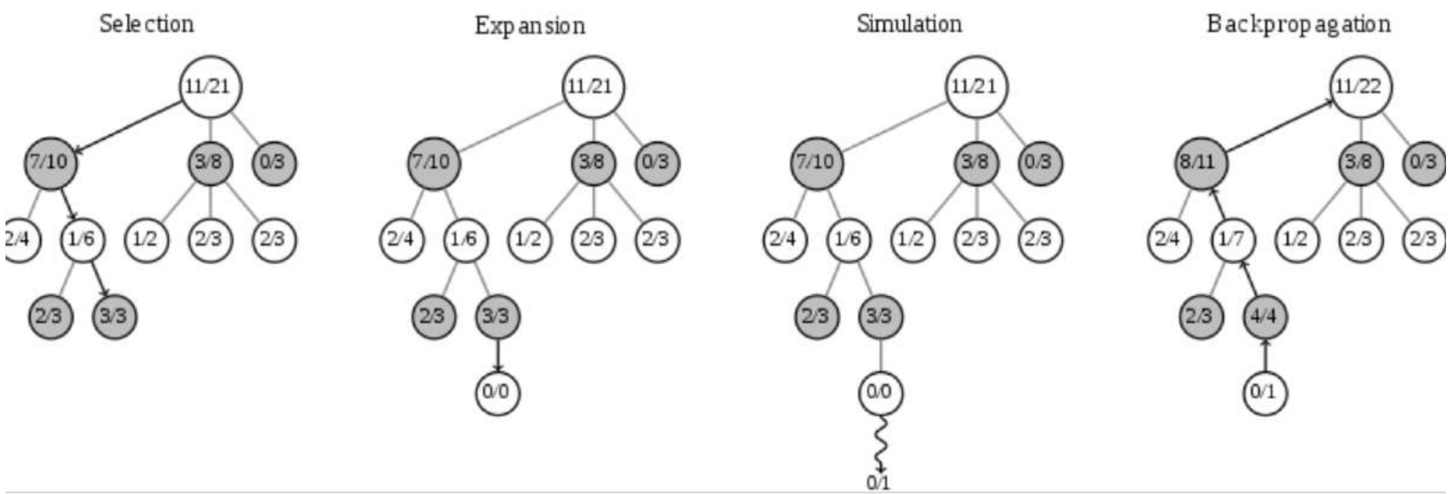
- Size of the state space 2.10^{170}
- Size of the action space 200
- No good evaluation function
- Local and global features (symmetries, freedom, ...)
- A move might make a difference some dozen plies later



AlphaGo

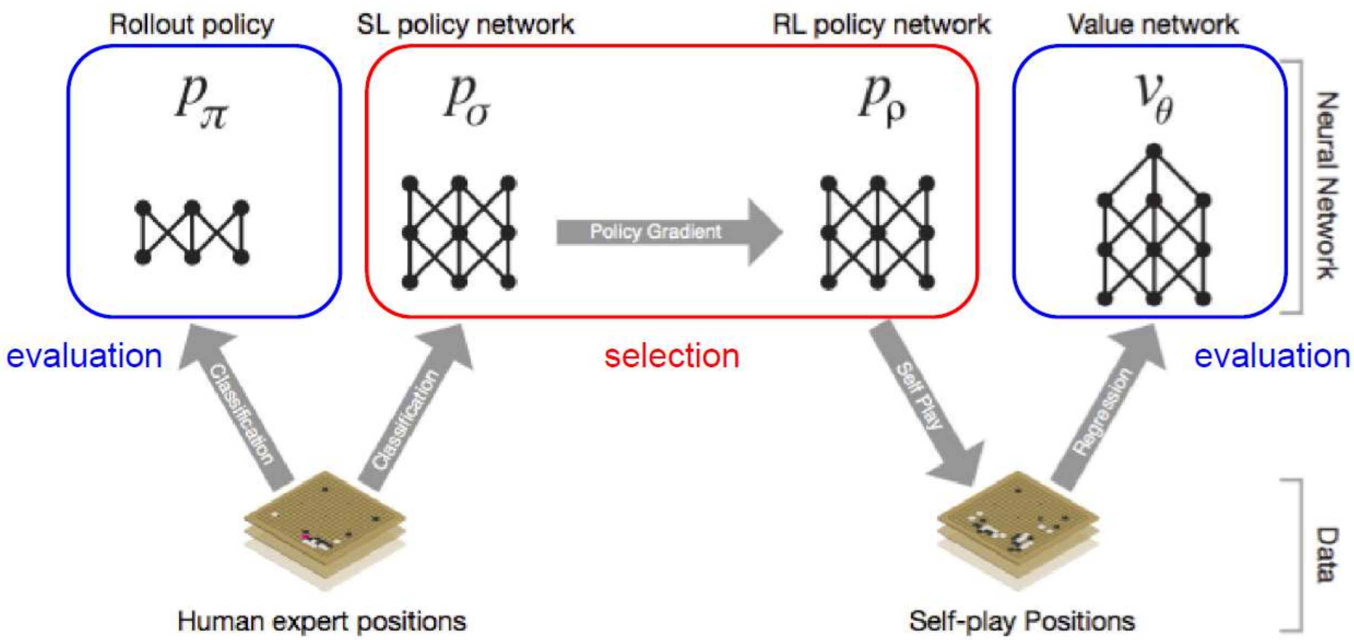
- Go is a perfect information game
  - See entire board at all times
  - Has an optimal value function!
- Key idea: We cannot unroll search tree to learn a policy/value for a large number of states, instead:
  - Reduce depth of search via **position evaluation**: Replace subtrees with estimated value function *v(s)*
  - Reduce breadth of search via **action sampling**: Don't perform unlikely actions
    - Start by predicting expert actions, gives you a probability distribution
- Use Monte Carlo rollouts, with a policy, selecting children with higher values
  - As policy improves this search improves too

Monte-Carlo Tree Search



Rollout
(Random Search)

From Wikipedia



Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy Gradient
Neural Network
evaluation
selection
evaluation
Classification
Classification
Self Play
Regression
Data
Human expert positions
Self-play Positions