Artificial Intelligence Methods for Social Good

M4-2 [Sequential Decision Making]:

Policy Gradient and Its Applications

08-537 (9-unit) and 08-737 (12-unit)

Instructor: Fei Fang

feifang@cmu.edu

Wean Hall 4126

Recap: Value Iteration and Policy Iteration

- Bellman Equation

$$V_t^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s'} P(s'|s,\pi(s)) V_{t-1}^\pi(s')$$
$$V_0^\pi = 0$$

- Value Iteration

$$V^*(s) = \max_{a\in A} [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$$

- Policy Iteration
  - Policy evaluation

$$V_{i+1}^\pi(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s'} P(s'|s,a) V_i^\pi(s'), V_0^\pi(s) \leftarrow 0$$

  - Policy update

$$\pi(s) \coloneqq \operatorname{argmax}_{a\in A} [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')]$$

Q-value

Policy Gradient

- Policy gradient
  - Most popular class of continuous action reinforcement learning algorithms
  - Also provides an alternative approach for discrete action problems
- Parameterize the policy
- Greedy policy update: Potentially unstable learning process with large policy jumps
- Soft policy update: Stable learning process with smooth policy improvement
  - Update the parameters towards the direction that increase the objective function (e.g., expected reward)
  - Challenge: hard to compute the gradient w.r.t. policy parameters due to uncertainty in MDPs
    - Finite difference methods
    - Likelihood ratio methods

Policy Gradient – Finite Difference Methods

- Perturb one parameter by a small amount and approximate the gradient
- Perturb all parameters by a small but different amount n times and approximate the gradient
- Slow, noisy and inefficient

Policy Gradient – Likelihood Ratio Gradient

- Policy Gradient Theorem

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta})] \qquad g(X)$$

- Can be approximated by sampling X and compute average $g(X)$!

Policy Gradient – Likelihood Ratio Gradient

- Now rewrite the gradient of the objective function with respect to policy parameters

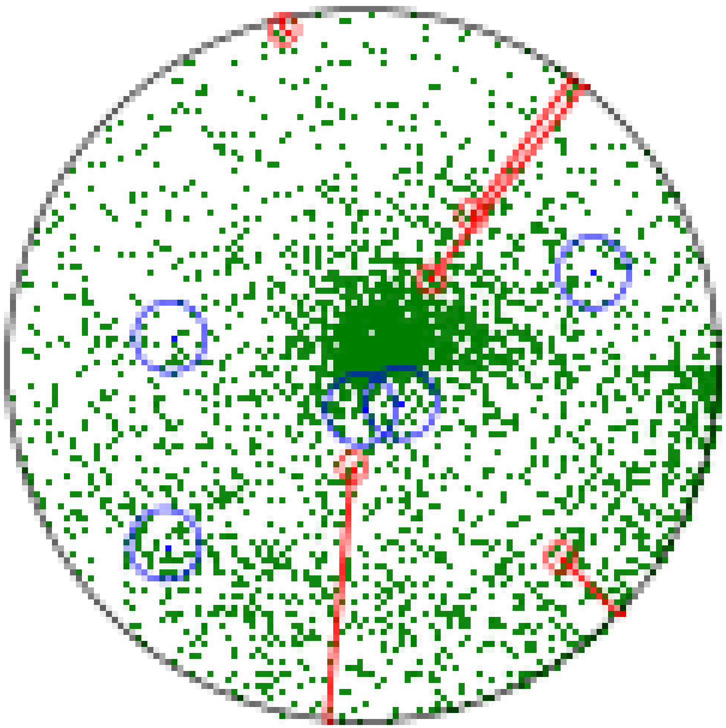
- Estimate gradient through sampling
  - Sample possible histories of actions (dependent on both policy and environment)
  - If probability of getting such history is a known differentiable function w.r.t. policy parameters, compute the gradient
  - Estimate the gradient of objective function w.r.t. policy parameters

Policy Gradient: Beyond MDPs

- Essentially a way to improve a parameterized policy/strategy through gradient descent
- Instead of writing down the full objective function and compute gradient, use finite difference or likelihood ratio + sampling to estimate the gradient

Forest Protection

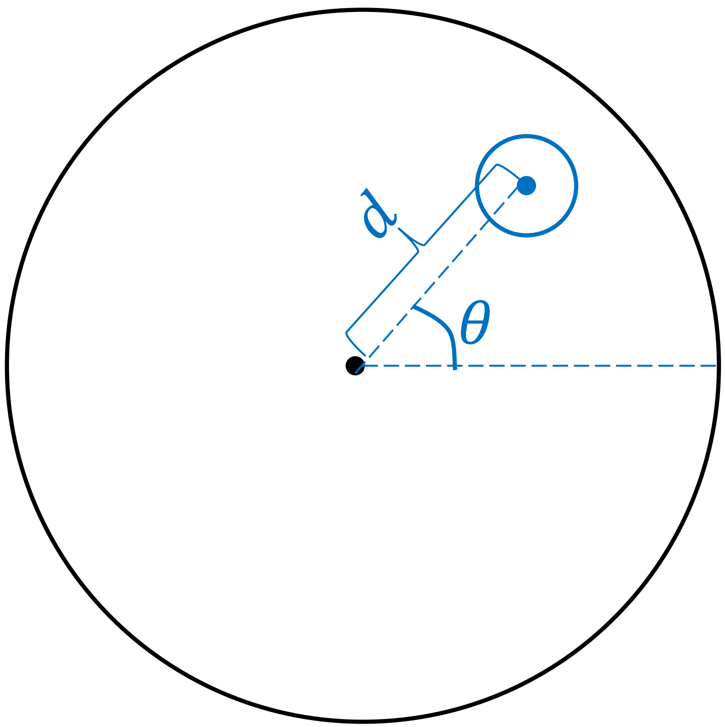
- Green dots: Valuable trees
- Blue dots: Defender location
- Red dots: Logging locations
- Zero-sum game
- Goal: Find defender strategy or defender policy

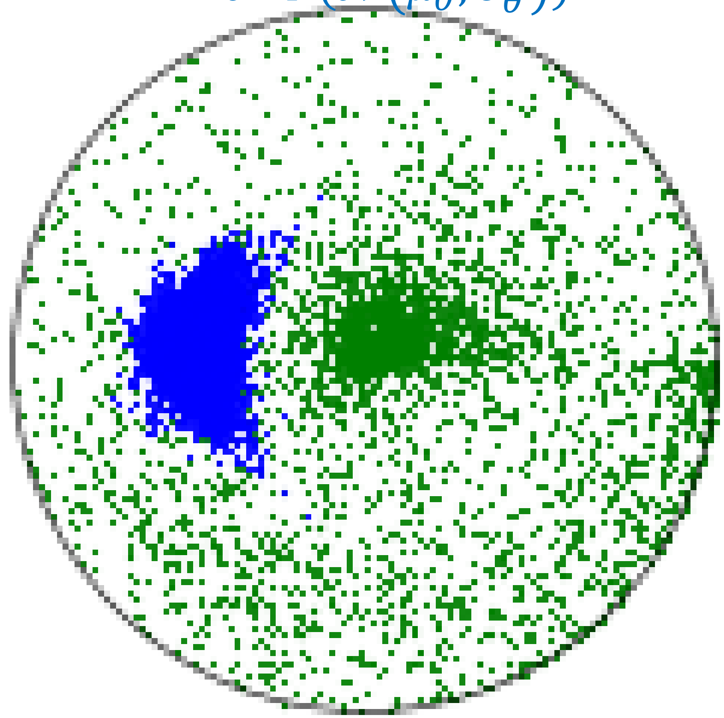


- Key idea 1: Represent defender strategy using logit normal distribution in polar coordinate system

$$d \sim P\left(\mathcal{N}\left(\mu_d, \sigma_d^2\right)\right)$$

$$\theta \sim P(\mathcal{N}(\mu_\theta, \sigma_\theta^2))$$





Forest Protection

- If attacker's mixed strategy is fixed (but unknown to the defender), how to find the best defender strategy? In this case, the best value of $\mu_d, \sigma_d, \mu_\theta, \sigma_\theta$?

- Use policy gradient!
  - Randomly initialize $\mu_d, \sigma_d, \mu_\theta, \sigma_\theta$
  - Compute the gradient of the objective function (defender's utility) w.r.t. to the parameters
  - Update the parameters
  - Repeat

Compute Gradient using Policy Gradient Theorem

- Recall

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta})]$$

- X: defender location
- θ: parameters representing defender strategy $(\mu_d, \sigma_d, \mu_\theta, \sigma_\theta)$
- $f(X)$: utility for the defender
- p: probability that the defender chooses this location

Compute Gradient using Policy Gradient Theorem

- m defenders
- Gradient of defender's expected utility w.r.t. $\theta_D = (\mu_d, \sigma_d, \mu_\theta, \sigma_\theta)$:

$$\nabla_{\theta_D} J_D = E_{a_D}[r_D \nabla_{\theta_D} \log \pi_D]$$

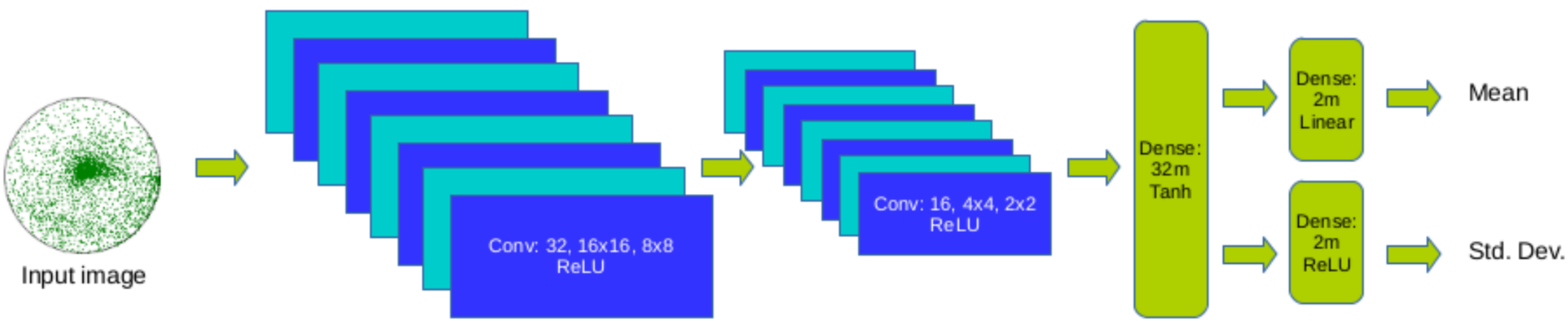
- The probability of taking action $a_D = (d, \theta), d \in R^m$

$$\pi_D(\boldsymbol{d}, \boldsymbol{\theta} | s) = \prod_{i \in [m]} p_{ln}(d_i; \mu_{d,i}, \nu_{d,i}) p_{ln}\left(\frac{\theta_i}{2\pi}; \mu_{\theta,i}, \nu_{\theta,i}\right)$$

$$p_{ln}(X; \mu, \nu) = \frac{1}{\sqrt{2\pi\nu}} \frac{1}{x(1-x)} e^{-\frac{(\text{logit}(x)-\mu)^2}{2\nu^2}}$$

Solving Game through Learning from Self Play

- More advanced version
- Key idea 2: Represent a "policy" with Convolutional Neural Network
  - Policy: mapping from game setting to strategy
  - CNN: Tree Distribution →Mean/Std of d and θ



Compute Gradient using Policy Gradient Theorem

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta})]$$

- X: defender location
- θ: parameters representing the defender policy (weights in CNN)
- $f(X)$: utility for the defender
- p: probability that the defender chooses this location

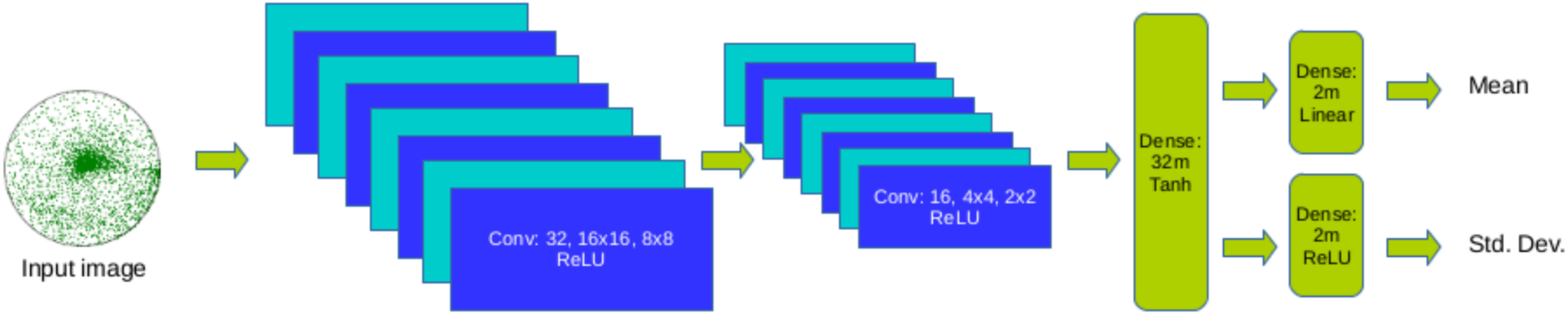
Compute Gradient using Policy Gradient Theorem

- m defenders
- Gradient of defender's expected utility w.r.t. w_D:

$$\nabla_{w_D} J_D = E_{a_D}[r_D \nabla_{w_D} \log \pi_D]$$

Solving Game through Learning from Self Play

- Key idea 3: Approximate Fictitious Play
  - Fictitious Play: Best responds to opponent's average strategy
  - Average strategy → Random samples from history
  - Best response → Update neural network



Solving Game through Learning from Self Play

- Put them together

Algorithm 1: OptGradFP

Initialization. Initialize policy parameters $\boldsymbol{w_D}$ and $\boldsymbol{w_O}$, replay memory *mem*;

for *ep in* $\{0, \ldots, ep_{max}\}$ **do**

Simulate n_s game play. Sample game setting and actions from current policy π_D and π_O n_s times, save in *mem*;

Replay for defender. Draw n_b samples from *mem*, resample defender action from current policy π_D;

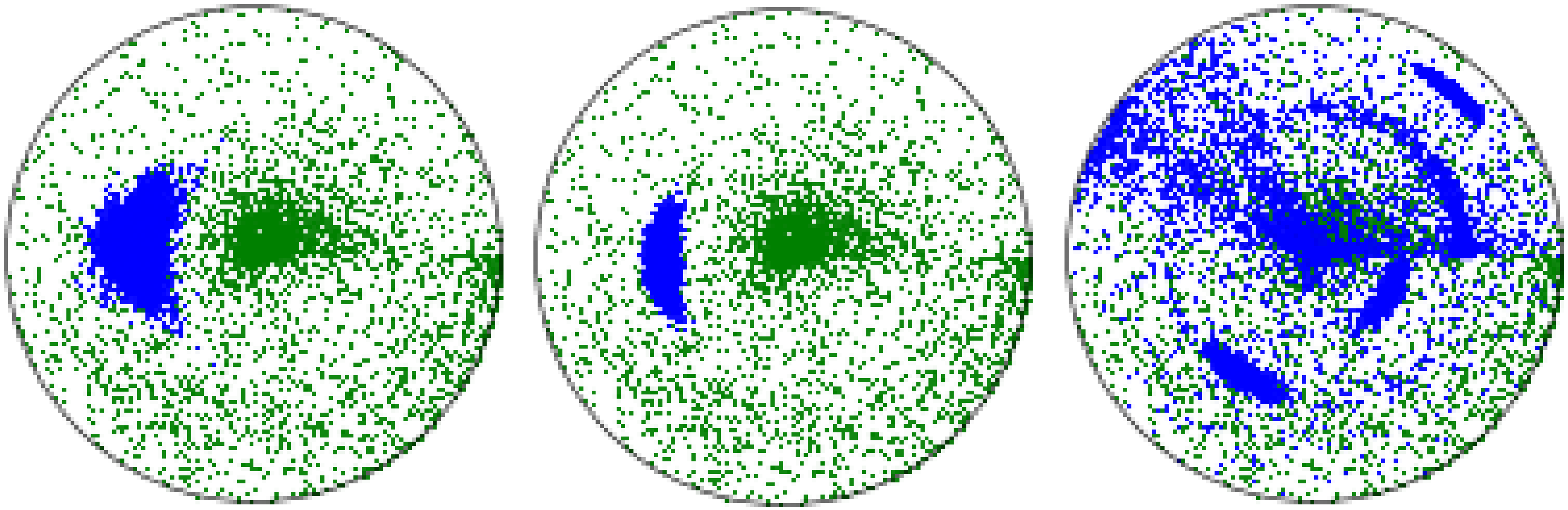
Update parameter for defender. Update defender policy parameter $\boldsymbol{w_D} := \boldsymbol{w_D} + \frac{\alpha_D}{1+ep\,\beta_D} * \nabla_{\boldsymbol{w_D}} J_D$;

Replay for attacker. Draw n_b samples from *mem*, resample attacker action from current policy π_O;

Update parameter for attacker. Update attacker policy parameter $\boldsymbol{w_O} := \boldsymbol{w_O} + \frac{\alpha_O}{1+ep\,\beta_O} * \nabla_{\boldsymbol{w_O}} J_O$

Solving Game through Learning from Self Play

- Single game setting



Cournot Adjustment StackGrad OptGradFP

- Multiple game setting
  - Train on 1000 forest states, predict on unseen forest state
  - 7 days for training, Prediction time 90 ms
  - Shift computation from online to offline

Solving Game through Learning from Self Play

- OptGradFP (Kamra et al., 2018)
  - Pro
    - Can predict defender strategy for unseen setting
  - Con
    - Restricted to specific parameterization + Slow convergence

Policy Learning for Continuous Space Security Games using Neural Networks
Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, Milind Tambe
In AAAI-18: The Thirty-Second AAAI Conference on Artificial Intelligence, February 2018