Reinforcement Learning

Lecture 9:
Policy Gradient

Yishay Mansour, Tel-Aviv University

Lecture 9: outline

- Policy optimization
  - When is optimizing policy better
- Policy Gradient Theorem
- Finite Differences Method

- Reinforce
  - Basic idea
  - Baseline
- Actor-Critic

- Applications
  - Aibo: Robo-soccer
  - AlphaGo

Function Approximation

- Previous Lecture
- Parametrized value functions
  - $\hat{V}(s;w) \approx V^{\pi}(s)$
  - $\hat{Q}(s,a;w) \approx Q^{\pi}(s,a)$
- Policy generated from value functions
  - ϵ-greedy, soft-max, etc.
- Today: learn directly $\pi(a|s;\theta)$

Advantages of direct Policy Optimization

Advantages

- Continuous action space
- Better convergence
- Effective in high dimension
- Stochastic policy

drawbacks

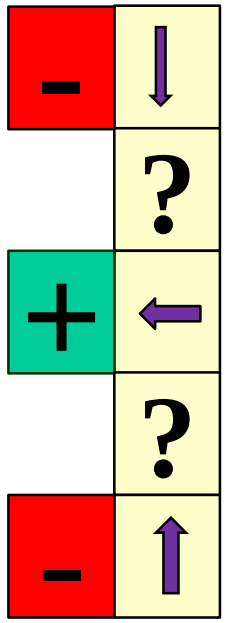
- Typically reaches local minimum
  - Contrast to linear Func. Approx.
- Evaluation typically time consuming

Stochastic Policies

- Why do we need stochastic policies?
  - Optimal policy is deterministic!
- Main issue:
  - State encoding
  - Tradeoff between:
    - Expressiblity
    - Efficiency
      - Lower dimension, aliased states

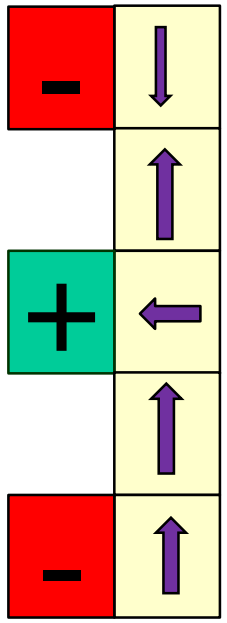
Example: Aliased Grid-world

- State encoding:
  - Where are the walls
  - Some state indistinguishable
    - Caused by encoding



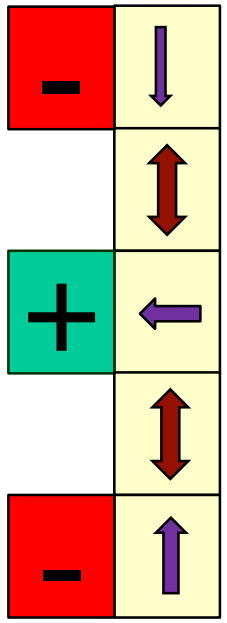
Example: Aliased Grid-world

- Deterministic policy
  - Need to be identical for equivalent states
  - Example:
    - Good for lower state
    - Bad for upper state



Example: Aliased Grid-world

- Randomized policy
  - Will get fairly fast to center state.
  - Much better
- How can this happen?
  - After state encoding, system not Markovian!



Example: zero-sum game

- Simple zero sum game
  - Penny-matching
  - Each player selects bit
  - If the bits match
    - Player 1 wins
  - If they do not match
    - Player 2 wins
- Best policy:
  - Play at random (1/2,1/2)
- Any near deterministic policy is bad
  - Say, ϵ-greedy
  - The other side can exploit it
- Again, states do not encode all information

Policy Parametrization

- State encoding
  - $\phi \in \mathbb{R}^d$
  - State s, or state-action (s, a)
- Policy parameter
  - $\theta \in \mathbb{R}^d$
- Combine ϕ and θ
  - Derive policy $\pi(a|s;\theta)$

Policy Optimization

- Optimization problem
  - Find θ that maximizes $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- Gradient based:
  - Gradient descent
  - Conjugate gradient
  - Quasi-Newton
- We focus on gradient descent.

Log Linear Policy

- Encode each (s, a) as $\phi(s, a)$
- Linear combination of features
  - $\phi(s, a)^\top \theta$
- Policy soft-max in values
  - $\pi(a|s, \theta) \propto e^{\phi(s,a)^\top \theta}$

Gaussian Policy

- Encode each state s as $\phi(s)$
- Average action per state
  - $\mu(s) = \phi(s)^\top \theta$
- Policy Gaussian:
  - $a \sim \mathcal{N}(\mu(s), \sigma^2) = \mu(s) + \mathcal{N}(0, \sigma^2)$
- Continuous action space!

Policy Gradient

- Have a differential return
  - $J(\theta)$
    - Maximizing $J(\theta)$ is our TRUE objective!
      - Contrast to minimizing a loss function …
- Iterative optimization
  - $\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} J(\theta_t)$
- Challenge:
  - Related $\nabla_{\theta_t} J(\theta_t)$ to $\nabla \pi(a|s;\theta_t)$

Finite Differences Methods

- Component-wise gradient estimates:
  - Fix a dimension i
    - Add perturbation δe_i
      - $e_i = (0, \dots, 0, 1, 0, \dots, 0)$
- Approximation
  - $\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{\hat{J}(\theta + \delta e_i) - \hat{J}(\theta)}{\delta}$
  - $\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{\hat{J}(\theta + \delta e_i) - \hat{J}(\theta - \delta e_i)}{2\delta}$

Finite Differences Methods

- Estimates $\hat{J}(\theta \pm \delta e_i)$ noisy
  - Need to average many samples
  - Need to be done per-dimension
- Selection of δ critical
  - Tradeoff between
    - Relative noise
    - Non-linearity of J

Finite Differences: Least Squares

- Assume more general perturbations
  - $J(\theta + \delta u_i) \approx J(\theta) + \delta u_i^{\mathsf{T}} \nabla J(\theta)$
  - $\Delta J^{(i)} = J(\theta + \delta u_i) - J(\theta) \approx \left(\Delta\theta^{(i)}\right)^{\mathsf{T}} \nabla J(\theta)$
    - $\Delta\theta^{(i)} = \delta u_i$
- Compute least squares
  - $G = \underset{x}{\operatorname{argmin}} \sum_i \left(J(\theta + \delta u_i) - J(\theta) - \delta u_i^{\mathsf{T}} x\right)^2$
  - $G = \underset{x}{\operatorname{argmin}} \sum_i \left(\Delta J^{(i)} - \left(\Delta\theta^{(i)}\right)^{\mathsf{T}} x\right)^2$

Finite Differences: Least Squares

- Matrix notation
  - $\Delta J = \left(\cdots, \Delta J^i, \cdots\right)^\mathsf{T}$
  - $[\Delta\Theta] = \left(\cdots, \Delta\theta^i, \cdots\right)^\mathsf{T}$
    - Matrix
  - $\Delta J \approx [\Delta\Theta]x$
- Least squares solution
  - $G = ([\Delta\Theta]^\mathsf{T}[\Delta\Theta])^{-1}[\Delta\Theta]^\mathsf{T}\Delta J$

Finite Differences: Least Squares

- One more issue
  - $J(\theta)$ unknown
    - Solve also for $J(\theta)$
- Solution
  - $\vec{J} = (\cdots, J(\theta + \delta u_i), \cdots)^{\mathsf{T}}$
  - $\vec{J} \approx M \begin{bmatrix} J(\theta) \\ \nabla J(\theta) \end{bmatrix}, \quad M = [1, \Delta\Theta]$

Policy Gradient Theorem

- For the return
  - Episodic $J(\theta) = V^{\pi}(s_1)$
    - $V^{\pi}(s) = E^{\pi(\theta)}[\sum_{t=1}^{T} r_t | s_1 = s]$
  - Discounted ; average reward
- We have
  - $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q^{\pi}(s,a) \nabla_{\theta} \pi(a|s;\theta)$
    - $\mu(s) = expected\ visits\ to\ state\ s$
      - $\eta(s) = \sum_{t=1}^{T} \Pr(s_t = s | s_1, \pi), \mu(s) = \frac{\eta(s)}{T}$

- For all states s
  - $\nabla V^{\pi}(s) = \nabla[\sum_a \pi(a|s;\theta)Q^{\pi}(s,a)]$
  - $= \sum_a[\nabla\pi(a|s;\theta)Q^{\pi}(s,a) + \pi(a|s;\theta)\nabla Q^{\pi}(s,a)]$
    - $\nabla Q^{\pi}(s,a) = \nabla \sum_{s'} p(s'|s,a)(r(s,a) + V^{\pi}(s')) = \sum_{s'} p(s'|s,a)\nabla V^{\pi}(s')$
  - $= \sum_a[\nabla\pi(a|s;\theta)Q^{\pi}(s,a) + \pi(a|s;\theta)\sum_{s'} p(s'|s,a)\nabla V^{\pi}(s')]$
    - Unrolling $\nabla V^{\pi}(s')$
  - $= \sum_a[\nabla\pi(a|s;\theta)Q^{\pi}(s,a) + \pi(a|s;\theta)\sum_{s'} p(s'|s,a)\Gamma]$
    - $\Gamma = \sum_a[\nabla\pi(a|s';\theta)Q^{\pi}(s',a) + \pi(a|s';\theta)\sum_{s''} p(s''|s',a)\Gamma']$
- $\nabla V^{\pi}(s) = \sum_{x\in S}\sum_{t=0}^{T} \Pr[s_t = x|s,\pi]\sum_a \nabla\pi(a|x;\theta)Q^{\pi}(x,a)$

- For the return
  - $\nabla J(\theta) = \nabla V^{\pi}(s_1)$
  - $= \sum_s (\sum_{t=0}^{T} \Pr(s_t = s | s_1, \pi)) \sum_a \nabla \pi(a|s;\theta) Q^{\pi}(s,a)$
  - $= \sum_s \eta(s) \sum_a \nabla \pi(a|s;\theta) Q^{\pi}(s,a)$
  - $= (\sum_x \eta(x)) \sum_s \frac{\eta(s)}{\sum_x \eta(x)} \sum_a \nabla \pi(a|s;\theta) Q^{\pi}(s,a)$
  - $\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s;\theta) Q^{\pi}(s,a)$
- Q.E.D.

Using Gradient Policy Theorem

- Statement
  - $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q^{\pi}(s,a) \nabla_{\theta} \pi(a|s;\theta)$
- We can sample state s
  - Using policy π
- What about the action?

Using Gradient Policy Theorem

- Simple identity
  - For any differential function $f(x)$
  - $\nabla f(x) = f(x)\frac{\nabla f(x)}{f(x)} = f(x)\nabla \log \mathrm{f(x)}$
- Back to Policy Gradient theorem:
  - $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q^\pi(s,a)\pi(a|s;\theta)\nabla_\theta \log \pi(a|s;\theta)$
  - $= E^\pi[Q^\pi(s,a)\nabla_\theta \log \pi(a|s;\theta)]$

Applying the theorem: log-linear

- Log linear policy
  - $\pi(a|s,\theta) \propto e^{\phi(s,a)^\top \theta}$
- Derivative:
  - $\nabla \log \pi(a|s;\theta) \propto \phi(s,a)$
    - $\nabla \log \pi(a|s;\theta) = \phi(s,a) - E_{b\sim\pi(\cdot|s;\theta)}\left[\phi(s,b)\right]$
- Update
  - $\Delta\theta = \alpha\, U\, \phi(s,a)$
    - Assume $E[U] = Q^\pi(s,a)$

Applying the theorem: Gaussian

- Gaussian policy:
  - $\mu(s) = \phi(s)^\top \theta; \quad a \sim \mathcal{N}(\mu(s), \sigma^2)$
- Derivative
  - $\nabla \log \pi(a|s;\theta) \propto \frac{(a-\mu(s))\phi(s)}{\sigma^2}$
- Update:
  - $\Delta\theta = \alpha U \frac{(a-\mu(s))\phi(s)}{\sigma^2}$
    - Assume $E[U] = Q^\pi(s,a)$

REINFORCE: Monte-Carlo

- Use Monte-Carlo estimates for U
  - Every visit, or first visit
- Algorithm (Do Forever):
  - Generate episode $(s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$
    - Using $\pi(\cdot \mid \cdot; \theta)$
  - For $t = 1, \ldots, T$
    - $R_t \leftarrow \sum_{i=t}^{T} r_i$
    - $\theta \leftarrow \theta + \alpha R_t \nabla \log \pi(a_t | s_t; \theta)$

REINFORCE with Baseline

- Baseline function
  - $b(s)$ depends only on the state
- Observation
  - $\sum_a b(s)\nabla\pi(a|s;\theta) =$
  - $= b(s)\nabla\sum_a \pi(a|s;\theta) = b(s)\nabla 1 = 0$
    - Note that $b(s)$ does not depend on action a !

REINFORCE with Baseline

- Implications
  - $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (Q^\pi(s,a) - b(s)) \nabla_\theta \pi(a|s;\theta)$
    - Since $\sum_a b(s) \nabla \pi(a|s;\theta) = 0$
- Restating
  - $\nabla J(\theta) \propto E^\pi \left[\left(Q^\pi(s,a) - b(s)\right) \nabla_\theta \log \pi(a|s;\theta) \right]$
- How to set baseline
  - $b(s)$

REINFORCE with Baseline

- Reduce variance
  - $Var(Z) = E[Z^2] - E^2[Z]$
- Leap of faith
  - Assume all $\nabla\pi$ are same magnitude
- Minimize $E^{\pi}\left[\left(Q^{\pi}(s,a) - b(s)\right)^2\right]$
  - $b(s) = E^{\pi}[Q^{\pi}(s,a)] = V^{\pi}(s)$
  - $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$

REINFORCE with Baseline

- How can we get $V^{\pi}(s)$?
  - Seems like part of our goal.
- Approximate $V^{\pi}(s) \approx V(s; w)$
  - Note: any $b(s)$ given unbiased estimates!
    - Having errors does not influence that
    - It does influence the variance.

REINFORCE with Baseline

- Algorithm idea:
  - Estimate $V^{\pi}(s)$
    - Using Monte-Carlo
  - Use estimate as $b(s)$
  - Update using $Q^{\pi}(s,a) - b(s)$
    - Actually $U - b(s)$

REINFORCE with Baseline

- Algorithm
- Inputs:
  - $\pi(a|s;\theta)$
  - $\hat{V}(s;w)$
  - α, β
- Initialize
  - $\theta_0 \in \mathbb{R}^{d_1}$
  - $w_0 \in \mathbb{R}^{d_2}$

- Given episode
  - Compute $R_{t:T}$
    - Return in $[t:T]$
  - $\Gamma_t = R_{t:T} - \hat{V}(s_t;w)$
  - $\Delta w = \alpha\Gamma_t \nabla_w \hat{V}(s_t;w)$
  - $\Delta\theta = \beta\Gamma_t \nabla_\theta \log\pi(a_t|s_t;\theta)$

Actor-Critic: TD approximation

- What would be the TD approach?
- We compute TD error
  - For updates
- Goal:
  - approximating $V^{\pi}(s)$

Actor-Critic: Q-approx

- Critic:
  - Maintains $\hat{Q}(\cdot; w)$
- Give an episode
  - For each t
  - $\Gamma_t = r_t + \hat{Q}(s_{t+1}, a_{t+1}; w) - \hat{Q}(s_t, a_t; w)$
  - $\Delta w = \alpha \Gamma_t \nabla_w \hat{Q}(s_t, a_t; w)$
- Send Actor Γ

- Actor:
  - Maintains $\pi(\cdot \mid \cdot; \theta)$
- Receive Γ_t
  - $\Delta\theta = \beta \Gamma_t \nabla_\theta \log \pi(a_t | s_t; \theta)$
- Select action
  - $a_{t+2} \sim \pi(\cdot \mid s_{t+2}; \theta)$
  - Observe reward r_{t+2}, s_{t+3}
- Send Critic $(s_{t+2}, r_{t+2}, s_{t+3})$

Bias in Actor-Critic

- Function approximation may introduce bias
  - We saw this before
- Bias approximation influence updates
  - Not anymore “pure” Gradient descent.

Compatible function approximation

- Value function compatible
  - $\nabla_w Q(s,a;w) = \nabla_\theta \log \pi(a|s;\theta)$
- Theorem:
- If parameter w minimize
  - $SE(w) = E^\pi[\left(Q^\pi(s,a) - Q(s,a;w)\right)^2]$
- Then gradient is exactly,
  - $\nabla_\theta J(\theta) = E^\pi[\nabla \log \pi(a|s;\theta)\, Q(s,a;w)]$

- Proof:
- Since w minimizes $SE(w)$
  - $0 = \nabla_w SE(w)$
  - $0 = E^{\pi}[\big(Q^{\pi}(s,a) - Q(s,a;w)\big)\nabla_w Q(s,a;w)]$
- By our assumption on the gradients
  - $0 = E^{\pi}[\big(Q^{\pi}(s,a) - Q(s,a;w)\big)\nabla_\theta \log \pi(a|s;\theta)]$
  - $E^{\pi}[Q^{\pi}(s,a)\nabla_\theta \log \pi(a|s;\theta)] = E^{\pi}[Q(s,a;w)\nabla_\theta \log \pi(a|s;\theta)]$
  - $\nabla_\theta J(\theta) = E^{\pi}[Q(s,a;w)\nabla_\theta \log \pi(a|s;\theta)]$
- Q.E.D.

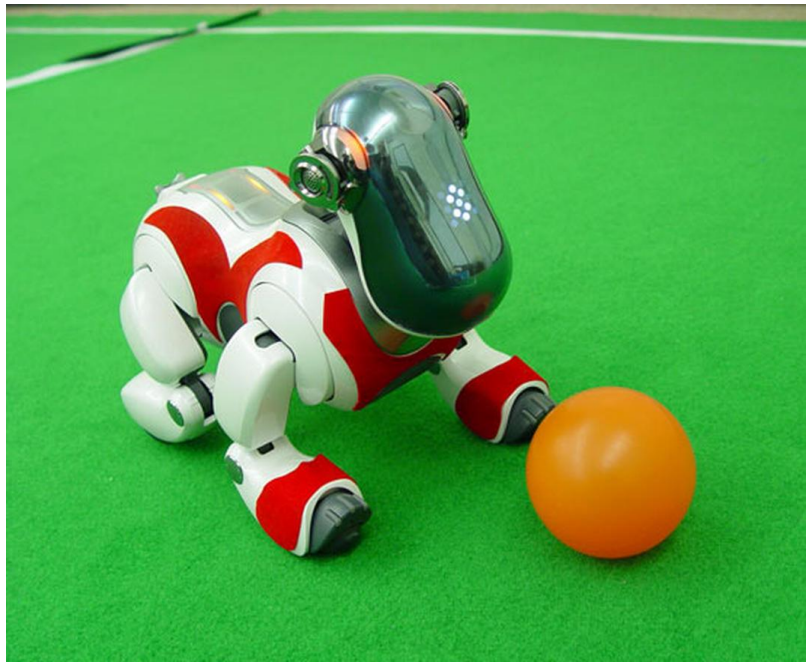
Summary: Policy Gradient

- Compute the gradient $\nabla J(\theta)$
- REINFORCE (Monte-Carlo):
  - $E^{\pi}[\nabla \log \pi(a|s;\theta) R_t]$
- Actor-Critic
  - Q-func: $E^{\pi}[\nabla \mathrm{Q}(s,a;w) Q(s,a;w)]$
  - A-func: $E^{\pi}[\nabla \log \pi(a|s;\theta) A(s,a;w)]$
    - $A(s,a;w) = Q(s,a;w) - V(s;w)$
  - TD: $E^{\pi}[\nabla \log \pi(a|s;\theta) \Delta_t]$

Applications

RoboSoccer: training Aibo

- Need to train walking
- Optimize the parameters
  - Speed
  - Stability

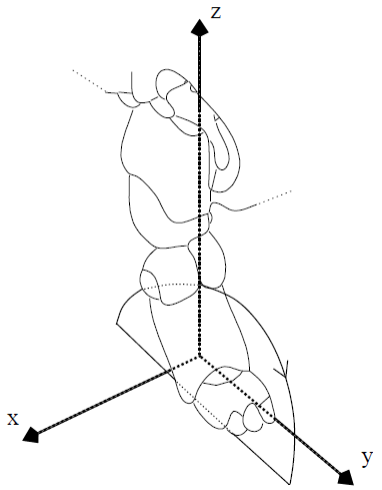


ONY

Before and After

Aibo policy (for gait)

- parameters
  - The front/rear locus
    - 3 parameters: height, x-pos., y-pos.
  - Locus:
    - length/skew multiplier
  - height of the body
    - Front/rear
  - Time (per foot)
    - On ground/locus

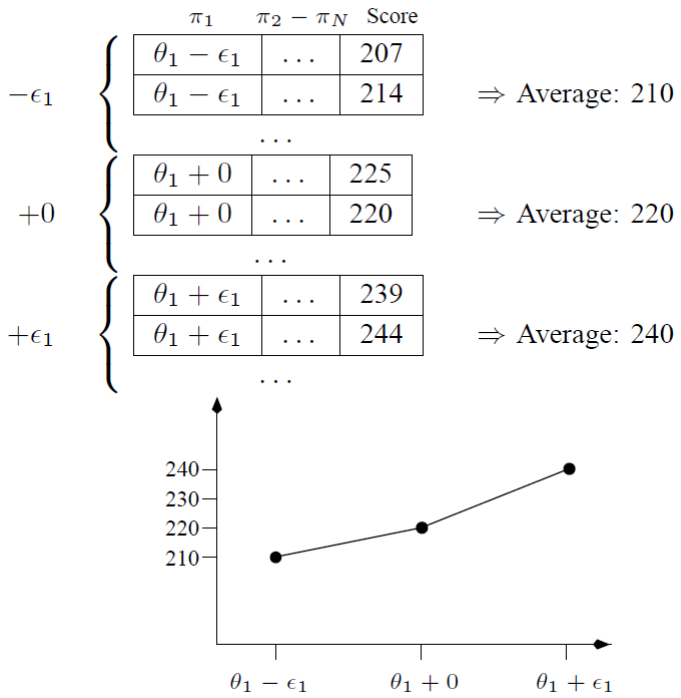


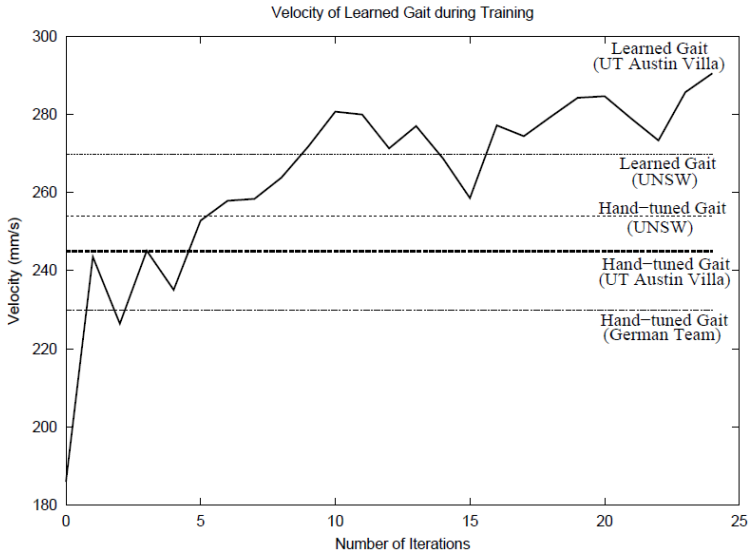
Training Aibo

- Episodic training
- Walk between landmarks
- Policy gradient improvement
- Finite difference method
  - Actual policy unknown!

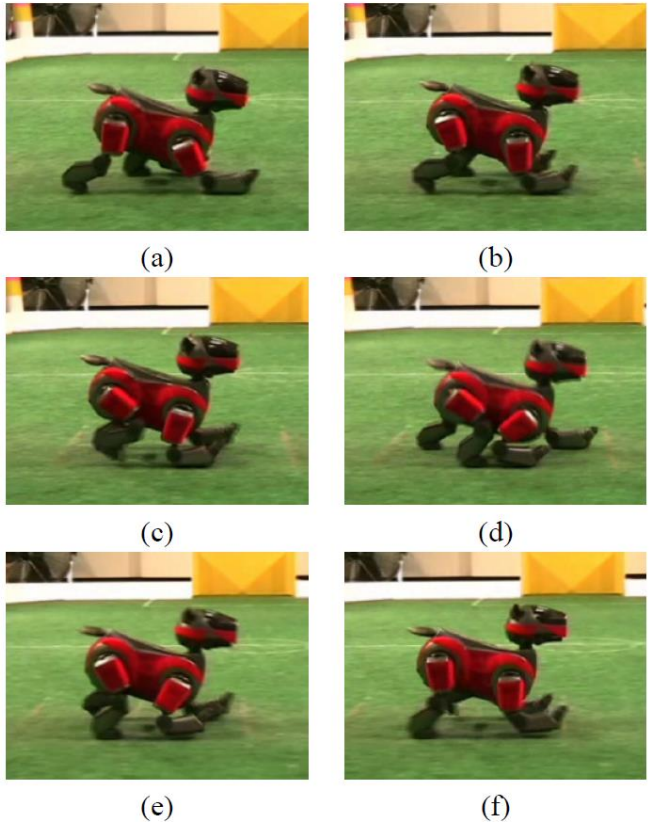
Perturbing a policy

- Given $\pi(\cdot \mid \cdot; \theta)$
  - Define π^i
  - Set $\theta_i \pm \Delta\theta_i$
    - $\Delta\theta_i \in \{+\epsilon_i, 0, -\epsilon_i\}$
  - Run π^i
    - Record speed sp^i
- Compute averages
  - Average per θ_i:
    - $\theta_i + \epsilon_i$
    - $\theta_i + 0$
    - $\theta_i - \epsilon_i$
  - If $\theta_i + 0$ $best$
    - $A_i = 0$
  - Otherwise
    - $A_i = avg(+\epsilon_i) - avg(-\epsilon_i)$
- $\theta = \theta + \alpha\, A / \|A\|$


π_1 $\pi_2 - \pi_N$ Score
$-\epsilon_1$
$\theta_1 - \epsilon_1$ ... 207
$\theta_1 - \epsilon_1$ ... 214
⇒ Average: 210
$+0$
$\theta_1 + 0$ ... 225
$\theta_1 + 0$ ... 220
⇒ Average: 220
$+\epsilon_1$
$\theta_1 + \epsilon_1$ ... 239
$\theta_1 + \epsilon_1$ ... 244
⇒ Average: 240
240
230
220
210
$\theta_1 - \epsilon_1$
$\theta_1 + 0$
$\theta_1 + \epsilon_1$


Velocity of Learned Gait during Training
Velocity (mm/s)
Number of Iterations
300
280
260
240
220
200
180
0
5
10
15
20
25
Learned Gait (UT Austin Villa)
Learned Gait (UNSW)
Hand−tuned Gait (UNSW)
Hand−tuned Gait (UT Austin Villa)
Hand−tuned Gait (German Team)

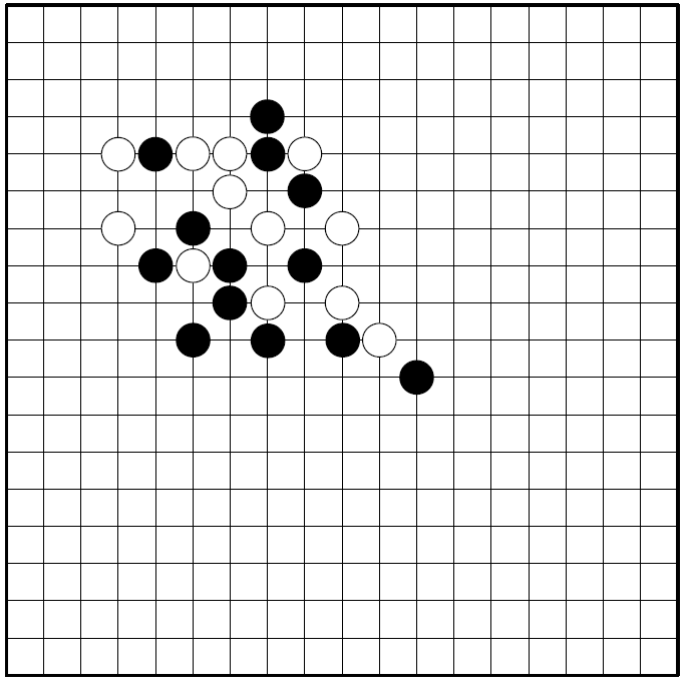
Parameter	Initial Value	ϵ	Best Value
Front locus:			
(height)	4.2	0.35	4.081
(x offset)	2.8	0.35	0.574
(y offset)	4.9	0.35	5.152
Rear locus:			
(height)	5.6	0.35	6.02
(x offset)	0.0	0.35	0.217
(y offset)	-2.8	0.35	-2.982
Locus length	4.893	0.35	5.285
Locus skew multiplier	0.035	0.175	0.049
Front height	7.7	0.35	7.483
Rear height	11.2	0.35	10.843
Time to move through locus	0.704	0.016	0.679
Time on ground	0.5	0.05	0.430

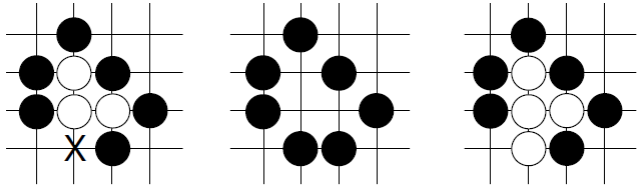


(a) (b) (c) (d) (e) (f)

ALPHA-GO

What is GO?





AlphaGo Architecture

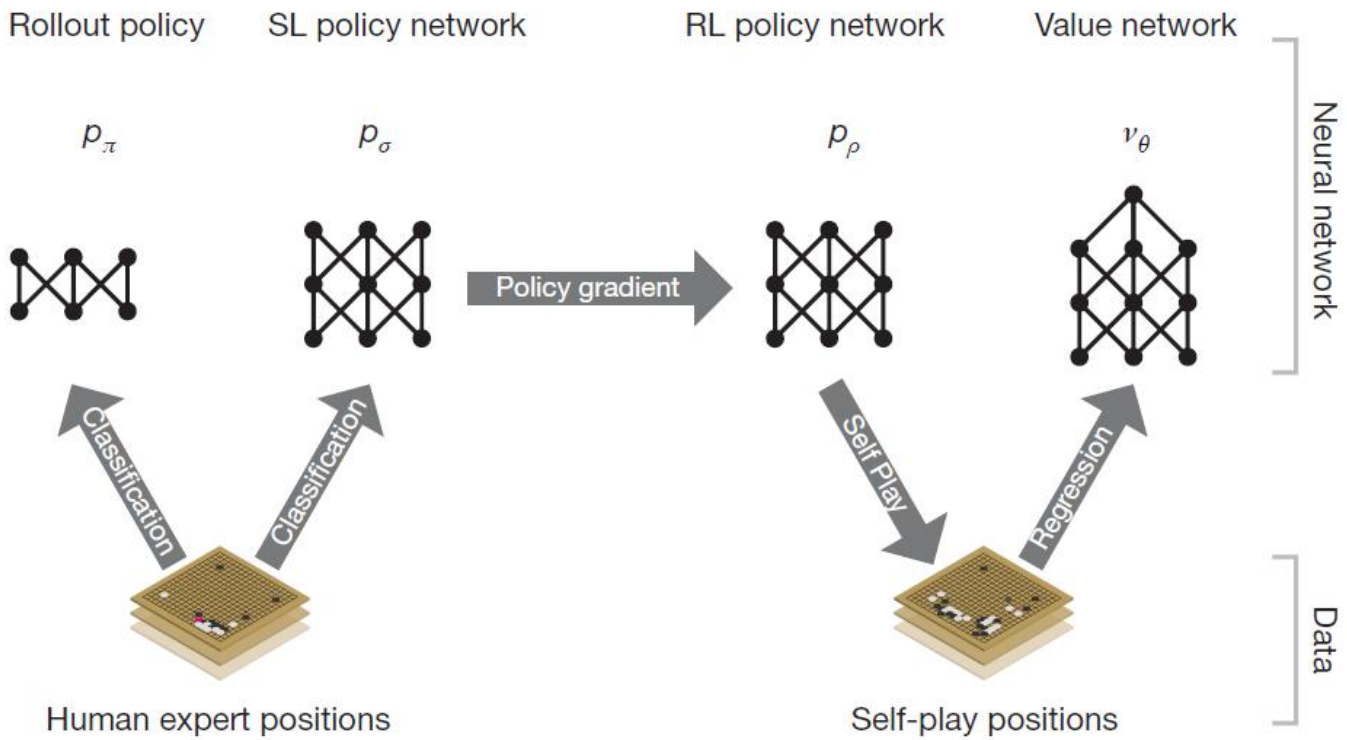
- Three phased training

- Phase 1:
  - Learn from human games (experts)
  - SL: predict next move
  - Rollout: fast prediction
    - 3 orders faster

- Phase 2:
  - Learn RL network
    - Start with SL weights
    - Optimized on self-play

- Phase 3:
  - Predict the winner
    - For the RL network
      - In self-play

How is Alpha Go design



Phase 1

- SL: Supervised Learning
- Deep network
  - Predict human moves
    - $p(a|s;\sigma)$
  - 13 layers
- Parameter update
  - $\Delta\sigma \propto \frac{\partial \log p(a|s;\sigma)}{\partial \sigma}$
- Increased accuracy
  - 55% vs. 44% (prev.)

- Rollout policy
- Linear soft-max
- Performance
  - Accuracy $\approx 24\%$
  - Very fast
  - $2\mu s$ vs. $3ms$

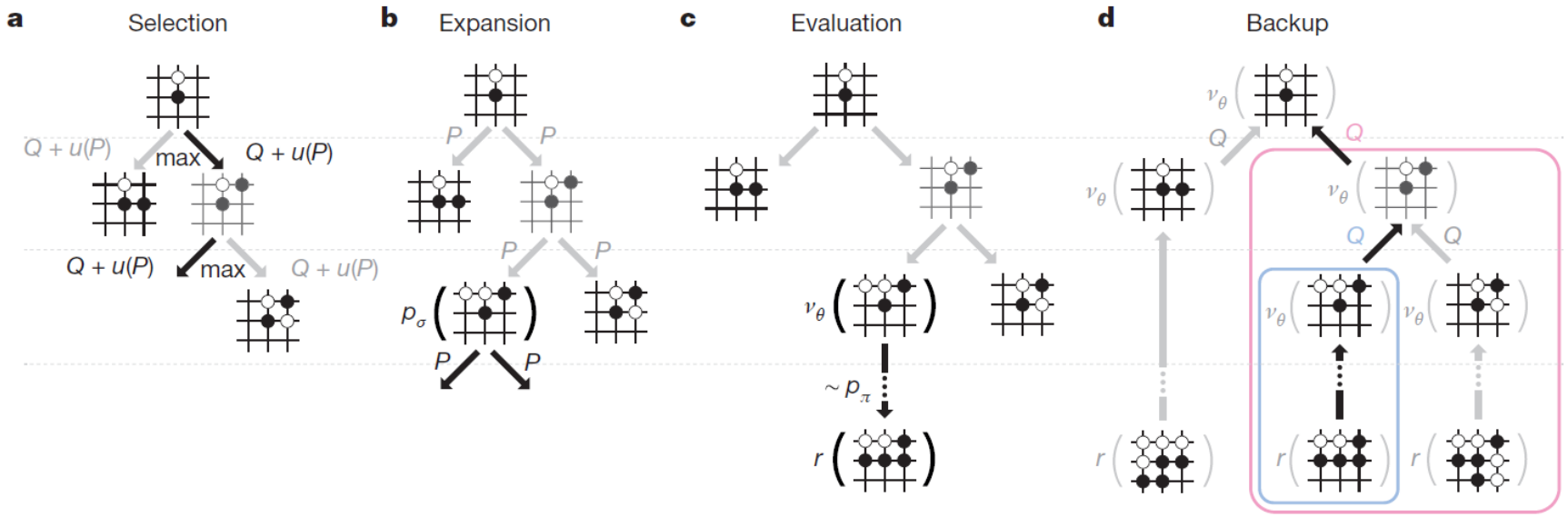
Phase 2

- RL policy $p(a|s;\rho)$
- Initialized to SL
- Self-play
  - Opponent: Random recent RL
- Rewards:
  - At the end (win/lose)
- Training
  - SGD
  - $\Delta\rho \propto \frac{\partial \log p(a|s;\rho)}{\partial \rho} z_t$
- Outperforms SL
  - Wins 80%
  - SL predict humans better

Phase 3

- Value network $V(s;\theta)$
  - $V^p(s) = \Pr[z_t = 1]$
  - SGD:
  - $\Delta\theta \propto \frac{\partial V(s;\theta)}{\partial\theta}(z - v(s;\theta)$
- Training:
  - Complete games
    - Overfitting (0.37/0.19)
  - One position per game
    - No overfitting (0.22)
    - 30M games
- Monte-Carlo search Tree
  - Look ahead
  - Uses rollout policy
    - Generates a trajectory from state
  - Score averages
    - $V(s;\theta)$
    - Outcome of a game

Monte-Carlo Search Tree



Lecture 9: outline

- Policy optimization
  - When is optimizing policy better
- Policy Gradient Theorem
- Finite Differences Method

- Reinforce
  - Basic idea
  - Baseline
- Actor-Critic

- Applications
  - Aibo: Robo-soccer
  - AlphaGo