Deep Reinforcement Learning

STAT946 Deep Learning

Guest Lecture by Pascal Poupart

University of Waterloo

October 19, 2017

Outline

- Introduction to Reinforcement Learning
- AlphaGo (Deep RL for Computer Go)
  - Mastering the Game of Go with Deep Reinforcement Learning and Tree Search, Nature 2016

Machine Learning

- Supervised Learning
  - Teacher tells learner what to remember
- Reinforcement Learning
  - Environment provides hints to learner
- Unsupervised Learning
  - Learner discovers on its own

What is RL?

- Reinforcement learning is learning what to do so as to maximize a numerical reward signal
  - Learner is not told what actions to take, but must discover them by trying them out and seeing what the reward is

Animal Psychology

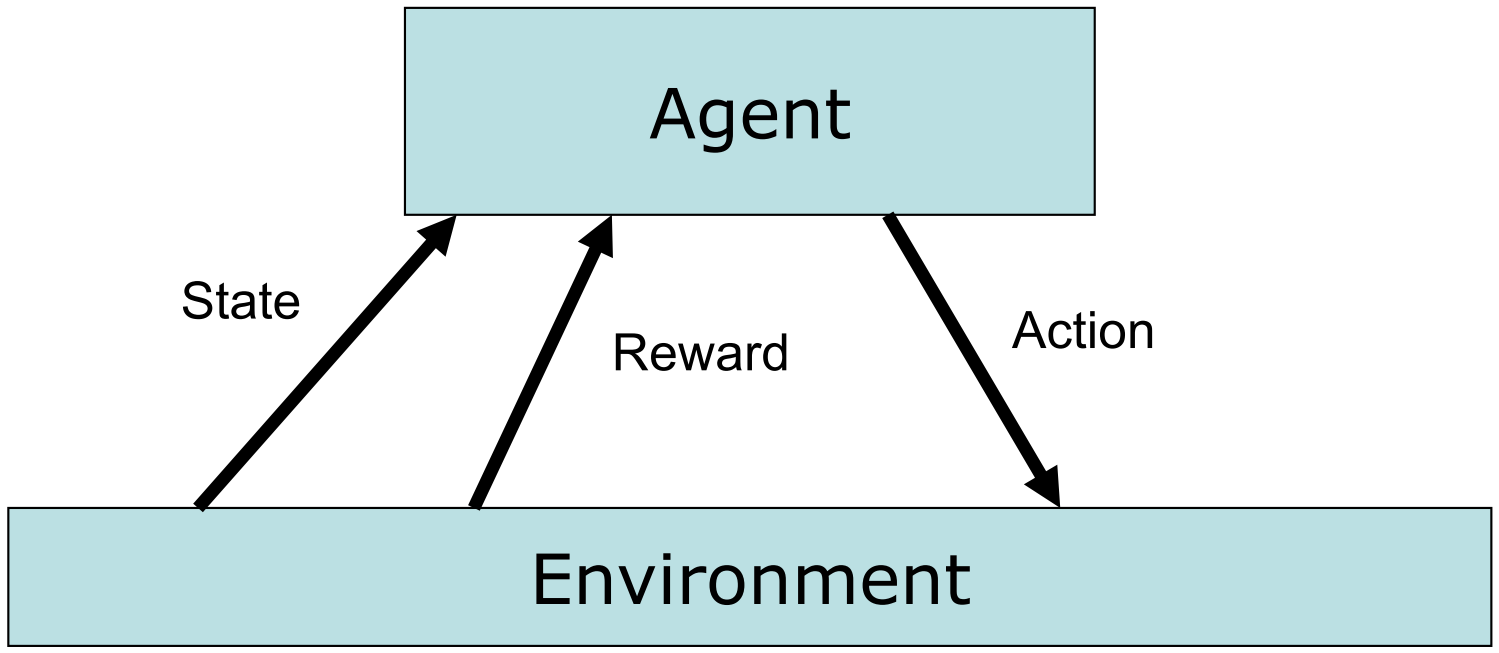
- Negative reinforcements:
  - Pain and hunger
- Positive reinforcements:
  - Pleasure and food
- Reinforcements used to train animals

- Let's do the same with computers!

RL Examples

- Game playing (go, atari, backgammon)
- Operations research (pricing, vehicle routing)
- Elevator scheduling
- Helicopter control
- Spoken dialog systems
- Data center energy optimization
- Self-managing network systems

Reinforcement Learning Problem



$$\begin{tikzcd} s0 \arrow[r, "a0", "r0"'] & s1 \arrow[r, "a1", "r1"'] & s2 \arrow[r, "a2", "r2"'] & \cdots \end{tikzcd}$$

Goal: Learn to choose actions that maximize $r_0+\gamma\, r_1+\gamma^2 r_2+\ldots$, where 0· γ <1

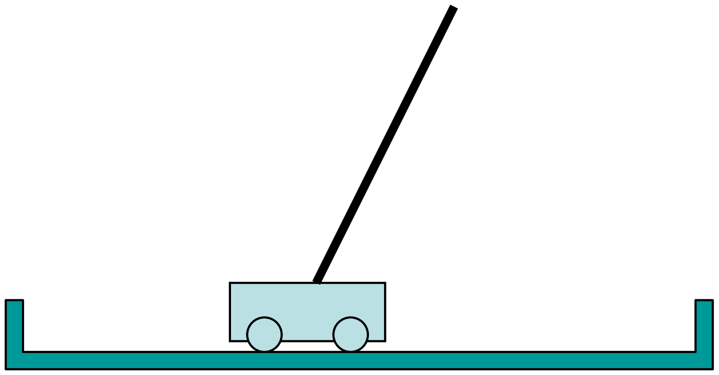
Reinforcement Learning

- Definition:
  - Markov decision process with unknown transition and reward models

- Set of states S
- Set of actions A
  - Actions may be stochastic
- Set of reinforcement signals (rewards)
  - Rewards may be delayed

Example: Inverted Pendulum

- State: $x(t)$, $x'(t)$, $\theta(t)$, $\theta'(t)$
- Action: Force F
- Reward: 1 for any step where pole balanced



Problem: Find $\delta: S \rightarrow A$ that maximizes rewards

Policy optimization

- Value-based techniques:
  - Find best possible $V(s) = \sum_t \gamma^t E_\delta[r_t|s_t, a_t]$
  - Then extract policy δ
  - Example: Q-learning

- Policy search techniques:
  - Search for δ that maximizes $V(s)$
  - Example: policy gradient

Supervised Learning

- Consider a stochastic policy $\Pr_w(a|s)$ parametrized by weights w.
- Data: state-action pairs $\{(s_1, a_1^*), (s_2, a_2^*), \dots\}$
- Maximize log likelihood of the data

$$w^* = argmax_w \sum_t \log \Pr_w(a_t^*|s_t)$$

- Gradient update

$$w_{t+1} \leftarrow w_t + \alpha\, \nabla_w \log \Pr_w(a_t^*|s_t)$$

Reinforcement Learning

- Consider a stochastic policy $\Pr_w(a|s)$ parametrized by weights w.
- Data: state-action-reward triples
  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$

- Maximize discounted sum of rewards
$$w^* = argmax_w \sum_t \gamma^t E_w[r_t|s_t, a_t]$$
- Gradient update
$$w_{t+1} \leftarrow w_t + \alpha \gamma^t R_t \nabla_w \log \Pr_w(a_t|s_t)$$
where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{i+t}$

Gradient Policy Theorem

- Gradient Policy Theorem

$$\boxed{\nabla V_w(s_0) = \sum_s \mu_w(s) \sum_a \nabla_w \Pr(a|s)\, Q_w(s,a)}$$

$\mu_w(s)$: stationary state distribution when executing policy parametrized by w

$Q_w(s,a)$: discounted sum of rewards when starting in s, executing a and following the policy parametrized by w thereafter.

Derivation

- $\nabla V_w(s) = \nabla \left[\sum_a \Pr_w(a|s)\, Q_w(s,a)\right] \qquad \forall s \in S$

$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s)\, \nabla Q_w(s,a)\right]$$
$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s)\, \nabla \sum_{s',r} \Pr(s',r|s,a)\left(r + \gamma V_w(s')\right)\right]$$
$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s) \sum_{s'} \gamma \Pr(s'|s,a)\, \nabla \mathrm{V_w}(s')\right]$$
$$= \sum_a [\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s) \sum_{s'} \gamma \Pr(s'|s,a)$$
$$\sum_{a'} [\nabla \Pr_w(a'|s') Q_w(s',a') + \Pr(a'|s') \sum_{s''} \gamma \Pr(s''|s',a')\, \nabla \mathrm{V_w}(s'')]]$$
$$= \sum_{x\in S} \sum_{k=0}^{\infty} \gamma^t \Pr(s \rightarrow x, t, w) \sum_a \nabla \Pr_w(a|x)\, Q_w(x,a)$$

- $\nabla V_w(s_0) = \sum_{x\in S} \sum_{k=0}^{\infty} \gamma^t \Pr(s_0 \rightarrow x, t, w) \sum_a \nabla \Pr_w(a|s)\, Q_w(s,a)$

$$= \sum_s \mu_w(s) \sum_a \nabla \Pr_w(a|s)\, Q_w(s,a)$$

REINFORCE: Monte Carlo Policy Gradient

- $\nabla V_w = \sum_s \mu_w(s) \sum_a Q_w(s,a) \nabla \Pr_w(a|s)$

$$= E_w\left[\gamma^t \sum_a Q_w(S_t,a) \nabla \Pr_w(a|S_t)\right]$$

$$= E_w\left[\gamma^t \sum_a \Pr_w(a|S_t)\, Q_w(S_t,a) \frac{\nabla \Pr_w(a|S_t)}{\Pr_w(a|S_t)}\right]$$

$$= E_w\left[\gamma^t Q_w(S_t,A_t) \frac{\nabla \Pr_w(A_t|S_t)}{\Pr_w(A_t|S_t)}\right]$$

$$= E_w\left[\gamma^t R_t \frac{\nabla \Pr_w(A_t|S_t)}{\Pr_w(A_t|S_t)}\right]$$

$$= E_w\left[\gamma^t R_t \nabla \log \Pr_w(A_t|S_t)\right]$$

- Stochastic gradient

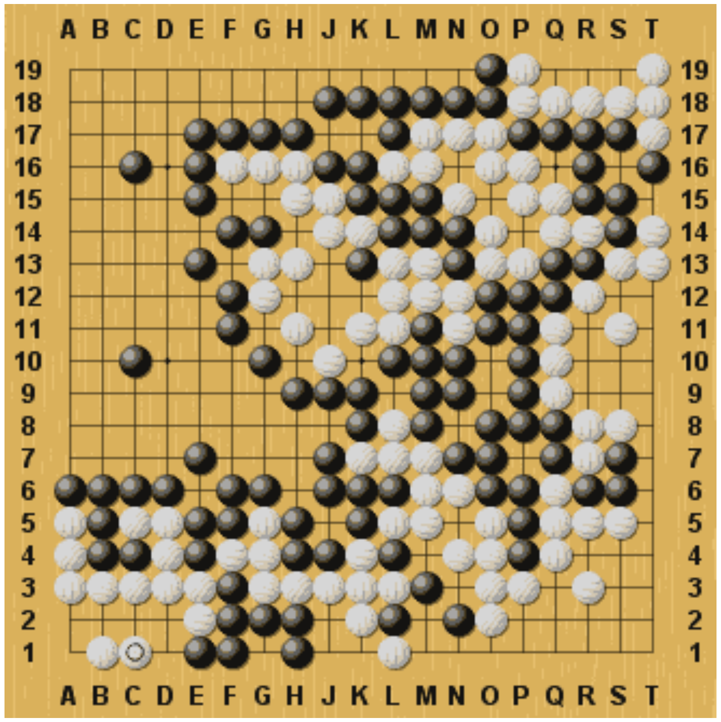
$$\nabla V_w = \gamma^t R_t \nabla \log \Pr_w(a_t|s_t)$$

Outline

- Introduction to Reinforcement Learning
- AlphaGo (Deep RL for Computer Go)
  - Mastering the Game of Go with Deep Reinforcement Learning and Tree Search, Nature 2016

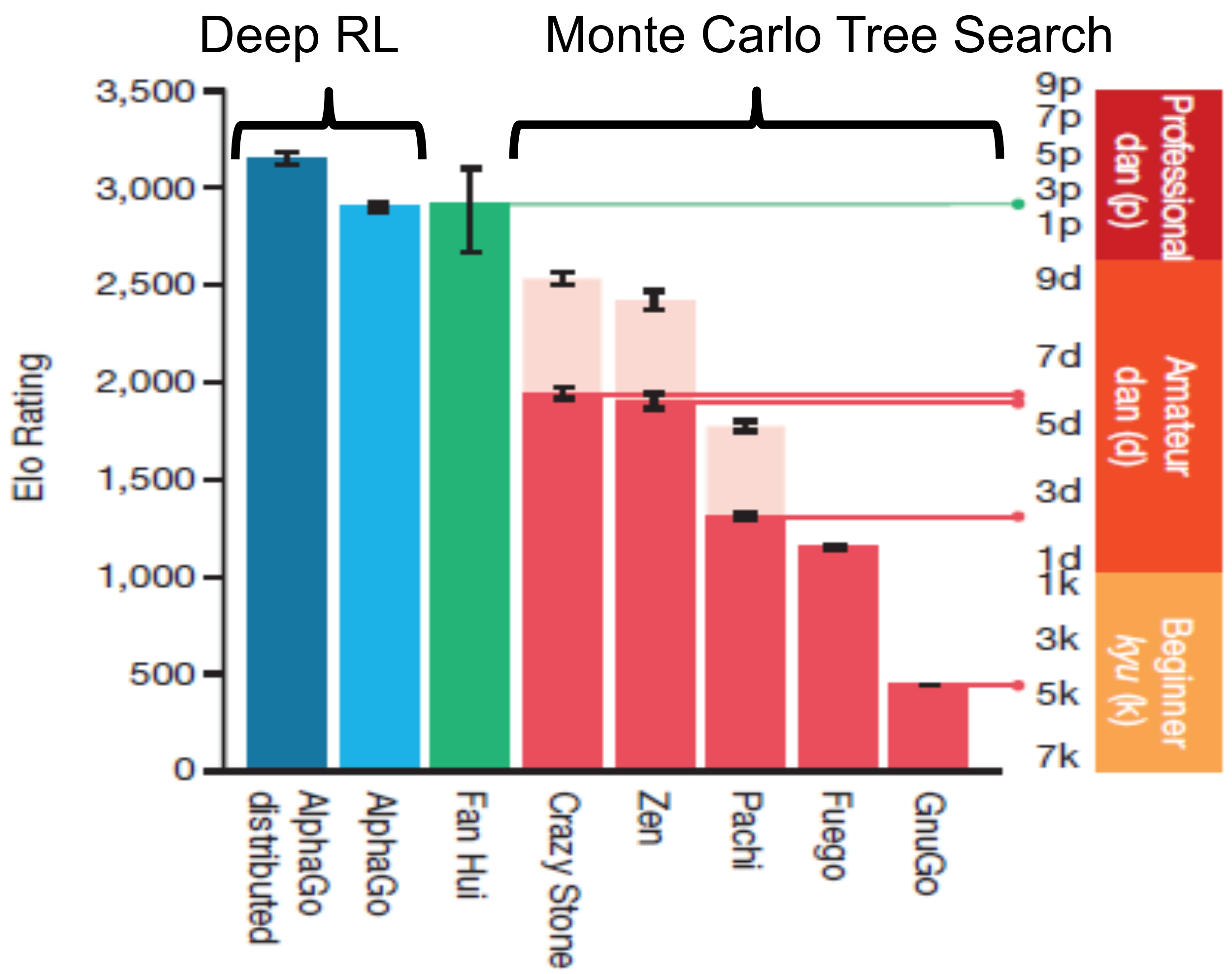
Game of Go

- (simplified) rules:
  - Two players (black and white)
  - Players alternate to place a stone of their color on a vacant intersection.
  - Connected stones without any liberty (i.e., no adjacent vacant intersection) are captured and removed from the board
  - Winner: player that controls the largest number of intersections at the end of the game



Computer Go

- Oct 2015:



Computer Go

- March 2016: AlphaGo defeats Lee Sedol (9-dan)

  "[AlphaGo] can't beat me" Ke Jie (world champion)

- May 2017: AlphaGo defeats Ke Jie (world champion)

  "Last year, [AlphaGo] was still quite humanlike when it played. But this year, it became like a god of Go" Ke Jie (world champion)

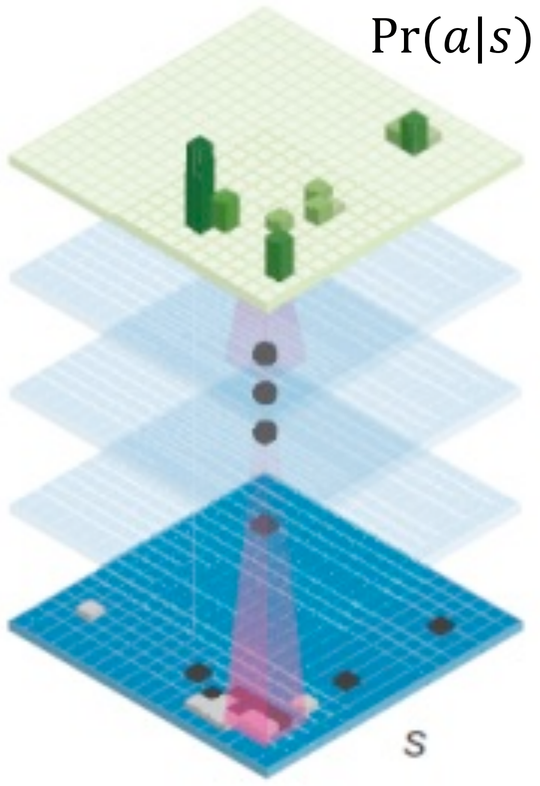
Winning Strategy

- Four steps:

1. Supervised Learning of Policy Networks
2. Reinforcement Learning of Policy Networks
3. Reinforcement Learning of Value Networks
4. Searching with Policy and Value Networks

Policy Network

- Train policy network to imitate Go experts based on a database of 30 million board configurations from the KGS Go Server.
- Policy network: $\Pr(a|s)$
  - Input: state s (board configuration)
  - Output: distribution over actions a (intersection on which the next stone will be placed)



Supervised Learning of the Policy Network

- Let $\boldsymbol{w}$ be the weights of the policy network
- Training:
  - Data: suppose a is optimal in s
  - Objective: maximize $\log \Pr_{\boldsymbol{w}}(\mathrm{a}|\mathrm{s})$
  - Gradient: $\nabla w = \frac{\partial \log \Pr_{\boldsymbol{w}}(a|s)}{\partial \boldsymbol{w}}$
  - Weight update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla \boldsymbol{w}$

Reinforcement Learning of the Policy Network

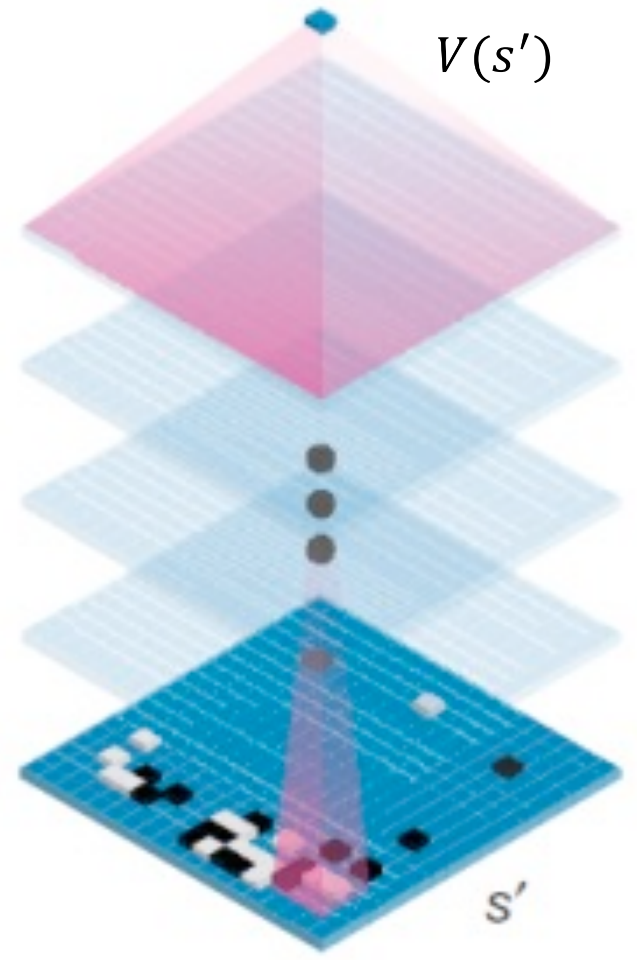
- How can we update a policy network based on reinforcements instead of the optimal action?
- Let $R_t = \sum_i \gamma^i r_{t+i}$ be the discounted sum of rewards in a trajectory that starts in s at time t by executing a.
- Gradient: $\nabla \boldsymbol{w} = \frac{\partial \log Pr_{\boldsymbol{w}}(a|s)}{\partial \boldsymbol{w}} \gamma^t R_t$
  - Intuition rescale supervised learning gradient by R
- Weight update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla \boldsymbol{w}$

Reinforcement Learning of the Policy Network

- In computer Go, program repeatedly plays games against its former self.
- For each game $R_t = \begin{cases} 1 & win \\ -1 & lose \end{cases}$
- For each (s_t, a_t) of turn t of the game, assume $\gamma = 1$ then compute
  - Gradient: $\nabla \boldsymbol{w} = \frac{\partial \log Pr_{\boldsymbol{w}}(a_t|s_t)}{\partial \boldsymbol{w}} R_t$
  - Weight update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla \boldsymbol{w}$

Value Network

- Predict $V(s')$ (i.e., who will win game) in each state s' with a value network
  - Input: state s (board configuration)
  - Output: expected discounted sum of rewards $V(s')$



Reinforcement Learning of Value Networks

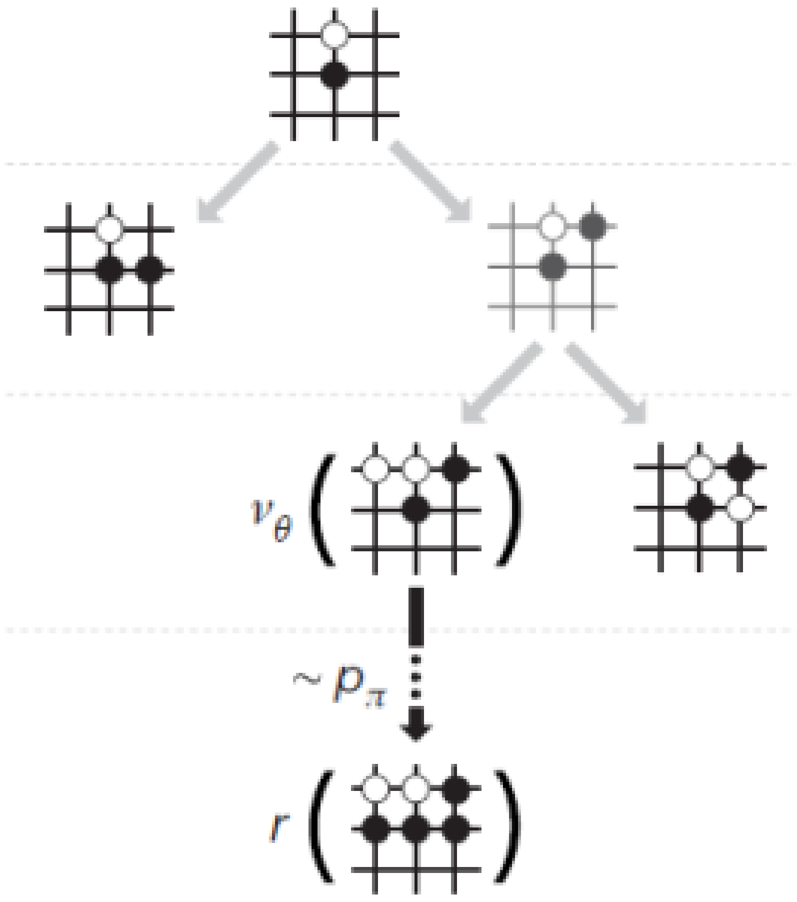
- Let $\boldsymbol{v}$ be the weights of the value network

- Training:
  - Data: (s, R) where $R = \begin{cases} 1 & win \\ -1 & lose \end{cases}$
  - Objective: minimize $\frac{1}{2}(V_{\boldsymbol{v}}(s) - R)^2$
  - Gradient: $\nabla \boldsymbol{v} = \frac{\partial V_{\boldsymbol{v}}(s)}{\partial \boldsymbol{v}}(V_{\boldsymbol{v}}(s) - R)$
  - Weight update: $\boldsymbol{v} \leftarrow \boldsymbol{v} - \alpha \nabla \boldsymbol{v}$

Searching with Policy and Value Networks

- AlphaGo combines policy and value networks into a Monte Carlo Tree Search algorithm

- Idea: construct a search tree
  - Node: s
  - Edge: a

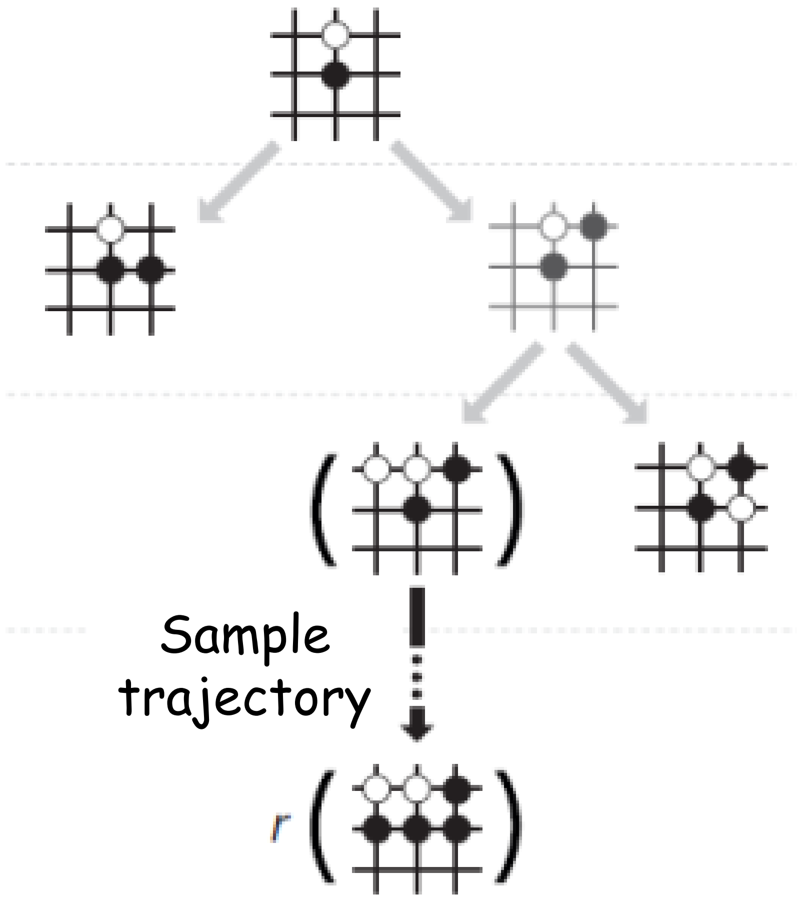


Search Tree

- At each edge store

  $Q(s,a), \Pr_{\boldsymbol{w}}(a|s), N(s,a)$

- Where $N(s,a)$ is the visit count of (s,a)

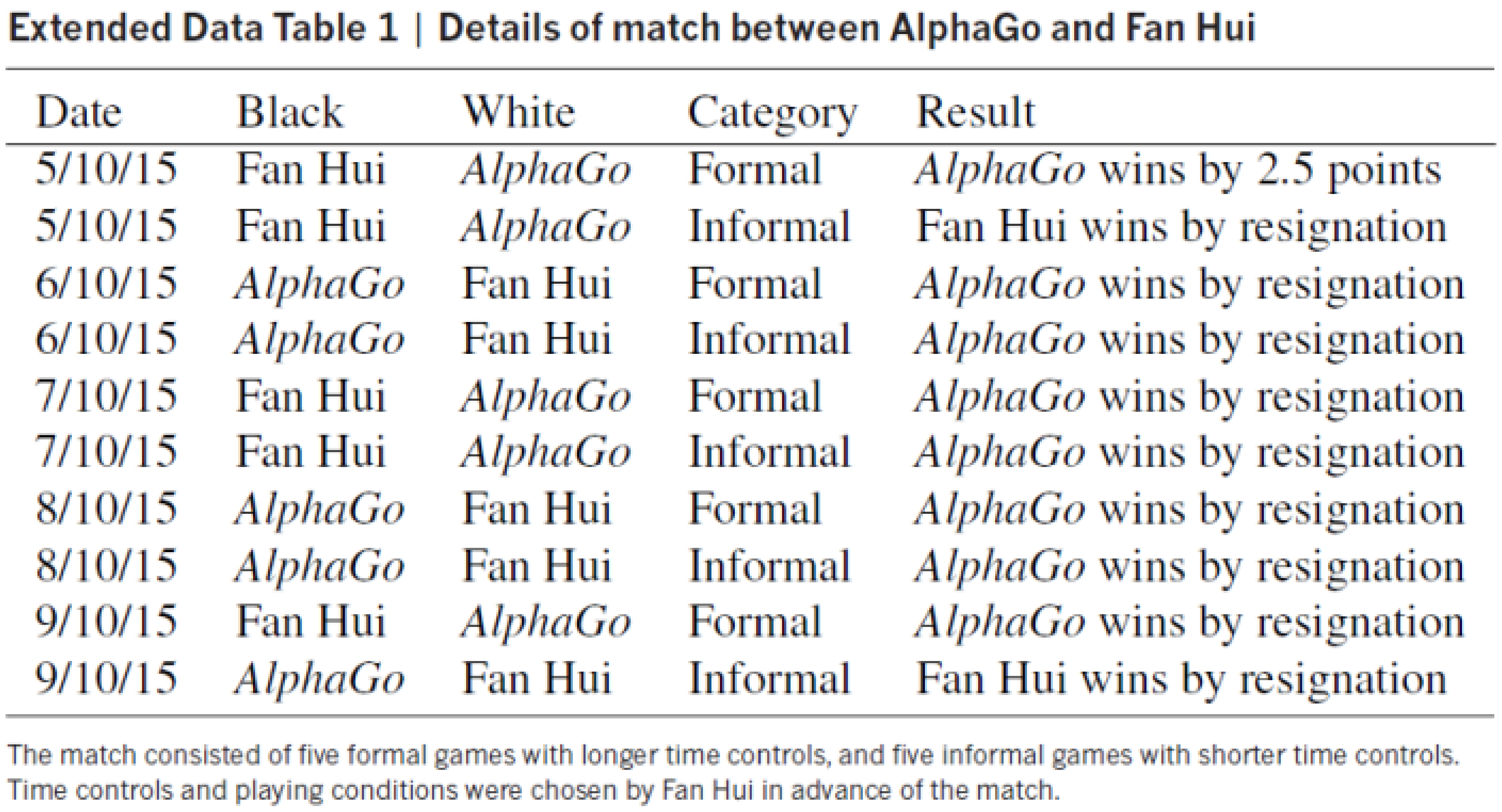


Simulation

- At each node, select edge a^* that maximizes
$$a^* = argmax_a \, Q(s,a) + u(s,a)$$

- where $u(s,a) \propto \frac{P(s|a)}{1+N(s,a)}$ is an exploration bonus
$$Q(s,a) = \frac{1}{N(s,a)} \sum_i 1_i(s,a) \, [\lambda V_v(s) + (1-\lambda) R_i]$$
$$1_i(s,a) = \begin{cases} 1 & if \ (s,a) \ was \ visited \ at \ iteration \ i \\ 0 & otherwise \end{cases}$$

Competition



Extended Data Table 1 | Details of match between AlphaGo and Fan Hui

Date	Black	White	Category	Result
5/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by 2.5 points
5/10/15	Fan Hui	*AlphaGo*	Informal	Fan Hui wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Informal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
9/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
9/10/15	*AlphaGo*	Fan Hui	Informal	Fan Hui wins by resignation

The match consisted of five formal games with longer time controls, and five informal games with shorter time controls. Time controls and playing conditions were chosen by Fan Hui in advance of the match.

Summary

- Policy gradient technique
  - Example: AlphaGo

- Upcoming course:
  - CS885: Deep Reinforcement Learning
    - Instructor: Pascal Poupart
    - Spring 2018
    - Deep Q Networks, Recurrent deep RL, memory network, Conversational systems, robotics