Reinforcement Learning
An Introduction and
Robustness Issues

Waterloo ML + Security + Verification

Workshop, August 26, 2019

Pascal Poupart, CIFAR AI Chair

BOREALIS AI

RBC Institute for Research

WATERLOO.AI

WATERLOO ARTIFICIAL INTELLIGENCE INSTITUTE

Borealis AI Principal Researcher

- Research institute funded by RBC

- 5 research centers:
  - Montreal, Toronto, Waterloo, Edmonton and Vancouver

- 80 researchers:
  - Integrated (applied & fundamental) research model
    - ML, RL, NLP, computer vision, private AI, fintech

ML Professor at U of Waterloo

- Deep Learning
  - Automated structure learning, sum-product networks
- Reinforcement learning
  - Constrained RL, motion-oriented RL, Bayesian RL, sport analytics
- NLP
  - Conversational agents, machine translation, fake news detection
- Theory
  - Satisfiability, local optima in mixture models, consistency in ML

Outline

- Reinforcement Learning
  - Introduction
  - REINFORCE algorithm
  - Application: Game of Go

- Robustness
  - Safe reinforcement learning
  - Imprecise model/simulator
  - Adversarial attacks

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

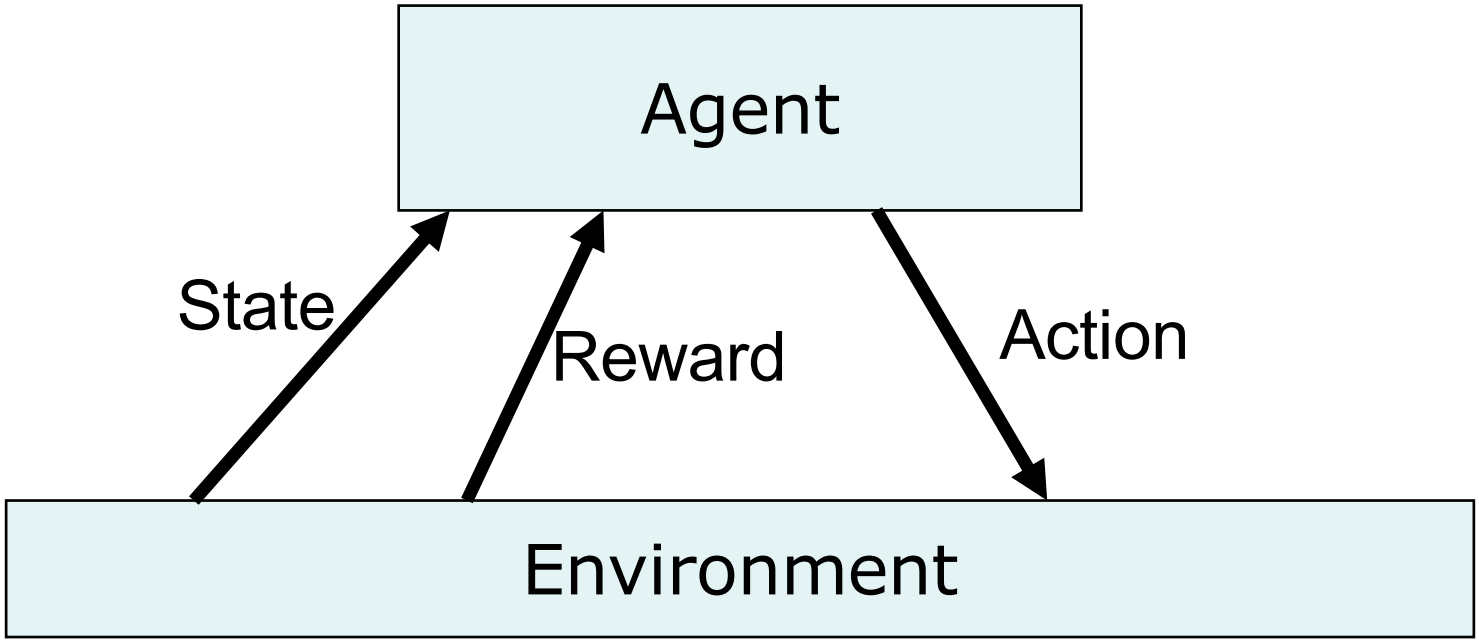
- Reinforcement learning: learn to select actions that maximize a numerical reward signal
  - Learner is not told what actions to take, but must discover them by trying them out and seeing what the reward is

- Animal training
  - **Reinforcements:** food (positive), pain (negative)
  - Let's do the same with computers

RL Applications

- Game playing (backgammon, go, video games)
- Operations research (pricing, vehicle routing)
- Finance (automated trading)
- Robotics (control, path planning)
- Conversational agents (dialog management)
- Recommender systems (ad placement, product recommendation)
- Verification (satisfiability)
- Optimization (neural architecture search)

Reinforcement Learning Problem



Goal: Learn to choose actions that maximize rewards

RL Framework

- Definition
  - States: $s \in S$
  - Actions: $a \in A$
  - Rewards: $r \in \mathbb{R}$
  - Transition model: $\Pr(s_t|s_{t-1}, a_{t-1})$
  - Reward model: $\Pr(r_t|s_t, a_t)$

  } unknown model

  - Discount factor: $0 \leq \gamma \leq 1$
    - discounted: $\gamma < 1$ undiscounted: $\gamma = 1$
  - Horizon (i.e., # of time steps): h
    - Finite horizon: $h \in \mathbb{N}$ infinite horizon: $h = \infty$

- Goal: find optimal policy π^* such that

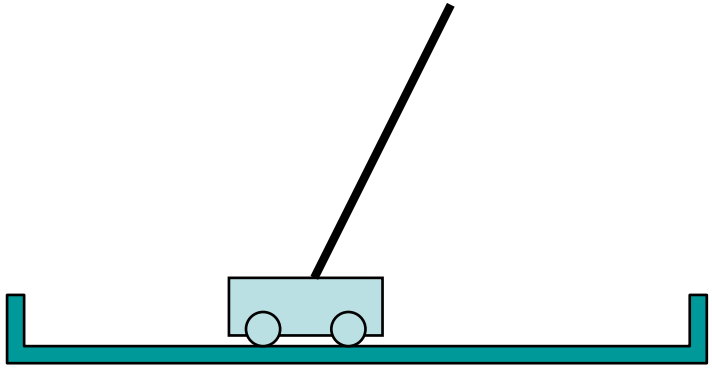
$$\pi^* = argmax_\pi \sum_{t=0}^{h} \gamma^t E_\pi[r_t]$$

RL Applications

- Game playing (backgammon, go, video games)
- Operations research (pricing, vehicle routing)
- Finance (automated trading)
- Robotics (control, path planning)
- Conversational agents (dialog management)
- Recommender systems (ad placement, product recommendation)
- Verification (satisfiability)
- Optimization (neural architecture search)

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 $\pi: S \rightarrow A$ that maximizes rewards

Important Components in RL

RL agents may or may not include the following components:

- Model: $\Pr(s'|s,a)$, $\Pr(r|s,a)$
  - Environment dynamics and rewards
- Policy: $\pi(s)$
  - Agent action choices
- Value function: $V(s)$
  - Expected total rewards of the agent policy

Categorizing RL Agents

Value based

- No policy (implicit)
- Value function

Policy based

- Policy
- No value function

Actor critic

- Policy
- Value function

Model based

- Transition and reward model

Model free

- No transition and no reward model (implicit)

Policy Optimization

- Value-based techniques:
  - Find best possible $V(s_0) = \sum_t \gamma^t E_\pi[r_t | s_t, a_t]$
  - Then extract policy π
  - Example: Q-learning

- Policy search techniques:
  - Search for π 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 $\{(s_1, a_1^*), (s_2, a_2^*), \ldots\}$

- 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

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

$\mu_w(s)$: 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

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

$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s)\, \nabla Q_w(s,a) \right]$$

$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s)\, \nabla \sum_{s',r} \Pr(s',r|s,a) \left(r + \gamma V_w(s') \right) \right]$$

$$= \sum_a \left[\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s) \sum_{s'} \gamma \Pr(s'|s,a)\, \nabla \mathrm{V_w}(\mathrm{s}') \right]$$

$$= \sum_a [\nabla \Pr_w(a|s)\, Q_w(s,a) + \Pr_w(a|s) \sum_{s'} \gamma \Pr(s'|s,a)$$

$$\sum_{a'} [\nabla \Pr_w(a'|s') Q_w(s',a') + \Pr(a'|s') \sum_{s''} \gamma \Pr(s''|s',a')\, \nabla \mathrm{V_w}(\mathrm{s}'')]$$

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

Probability of reaching x from s at time step t

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

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

REINFORCE: Monte Carlo Policy Gradient

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

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

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

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

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

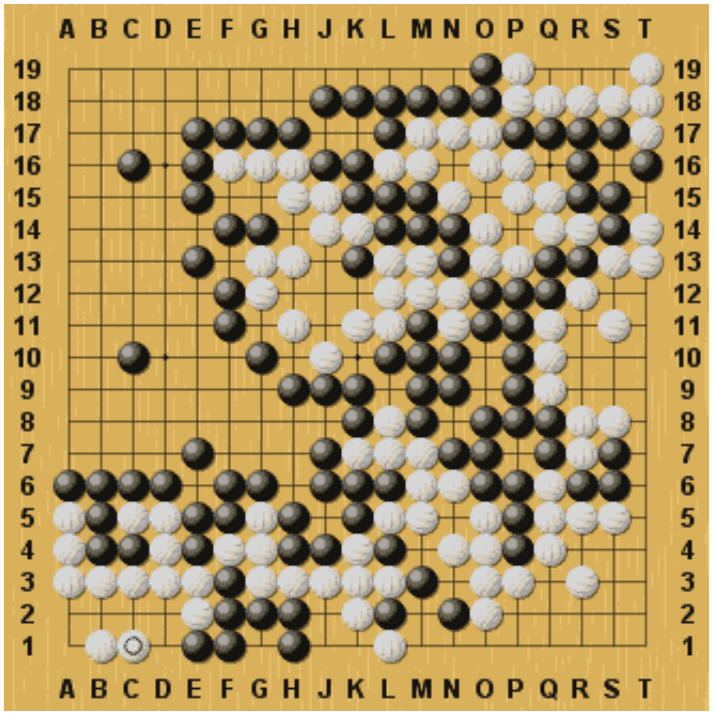
$$= E_w \left[\gamma^t R_t \nabla \log \Pr_w(A_t|S_t) \right]$$

- Stochastic gradient

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

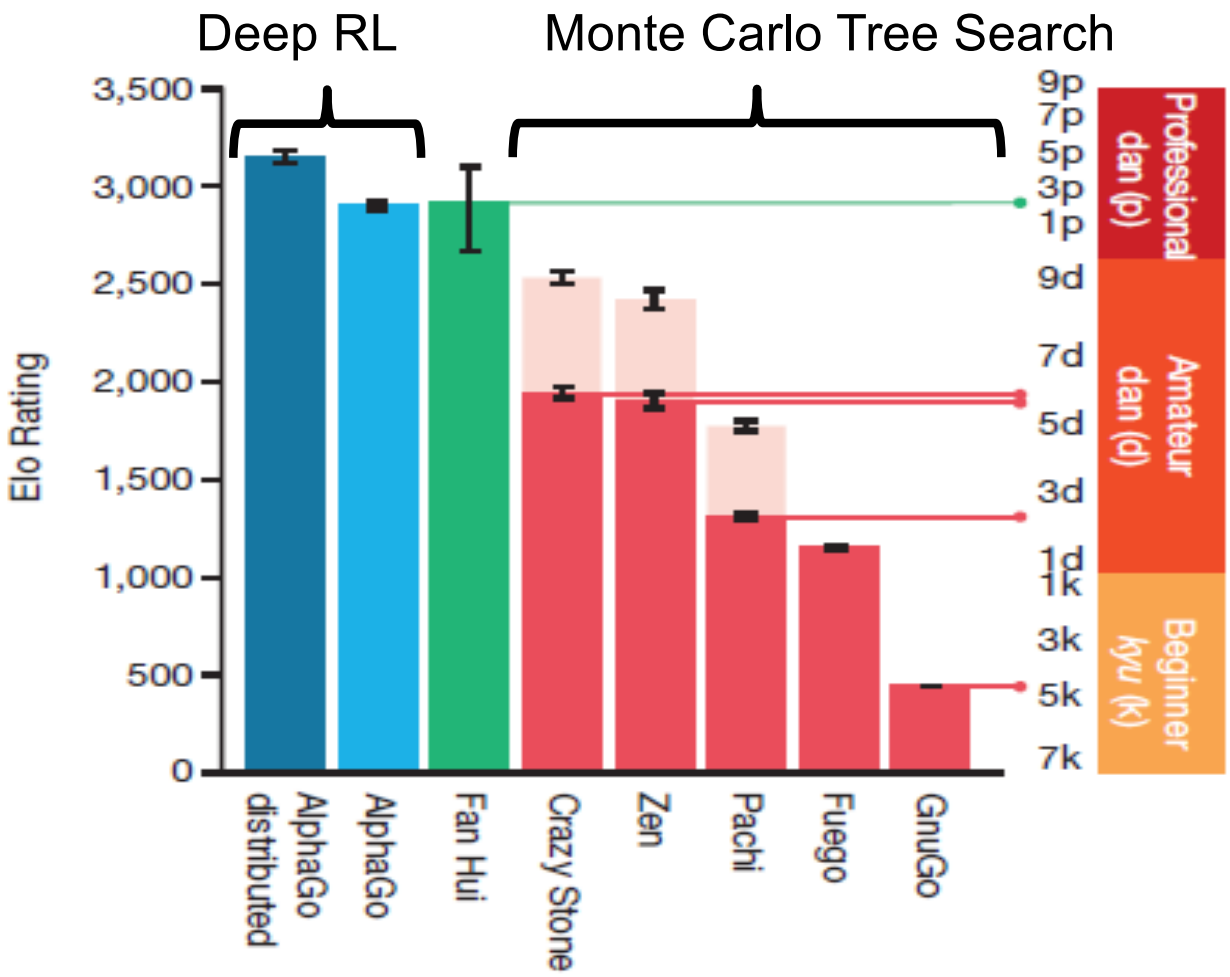
Application: Game of Go

- (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:



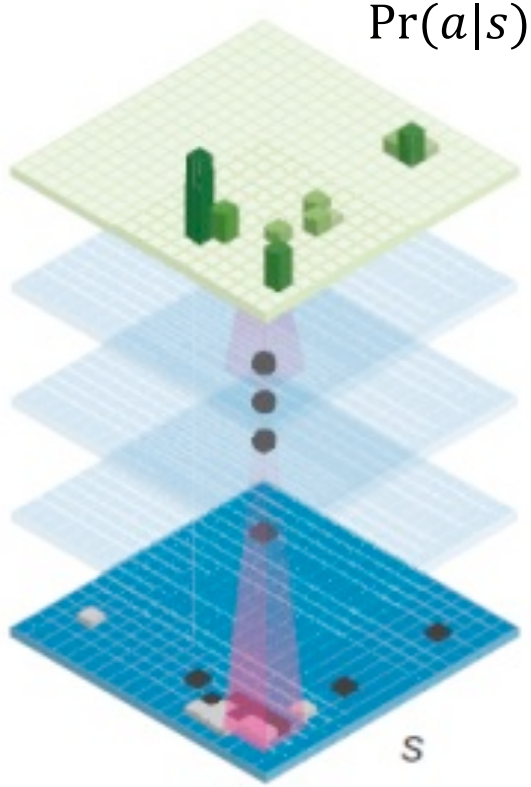
Winning Strategy

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

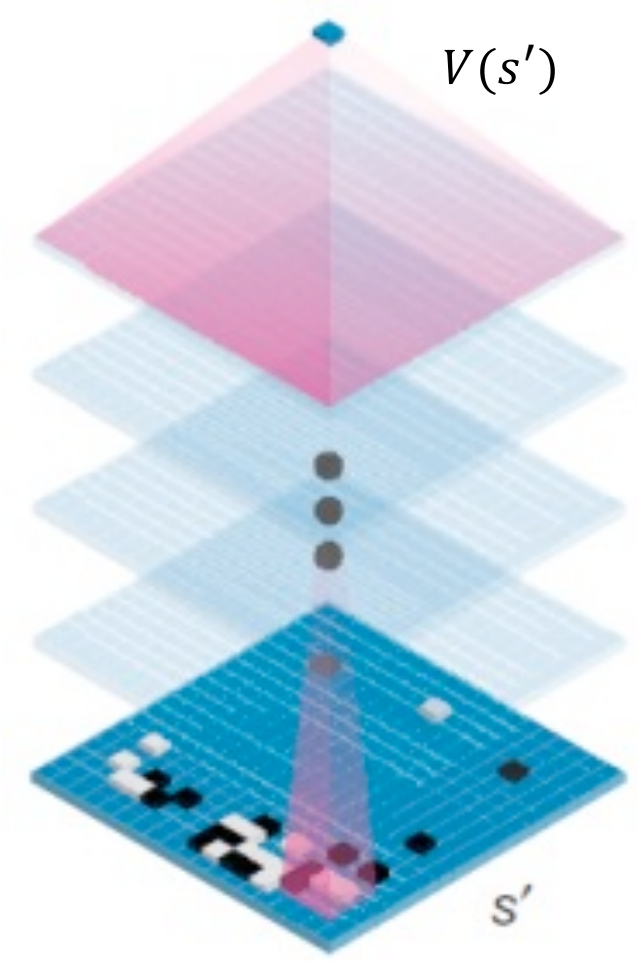
- How can we update a policy network based on reinforcements instead of the optimal action?
- Let $R_t = \sum_i \gamma^i \, r_{t+i}$ be the discounted sum of rewards in a trajectory that starts in s at time t by executing a.
- Gradient: $\nabla \boldsymbol{w} = \frac{\partial \log Pr_{\boldsymbol{w}}(a|S)}{\partial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

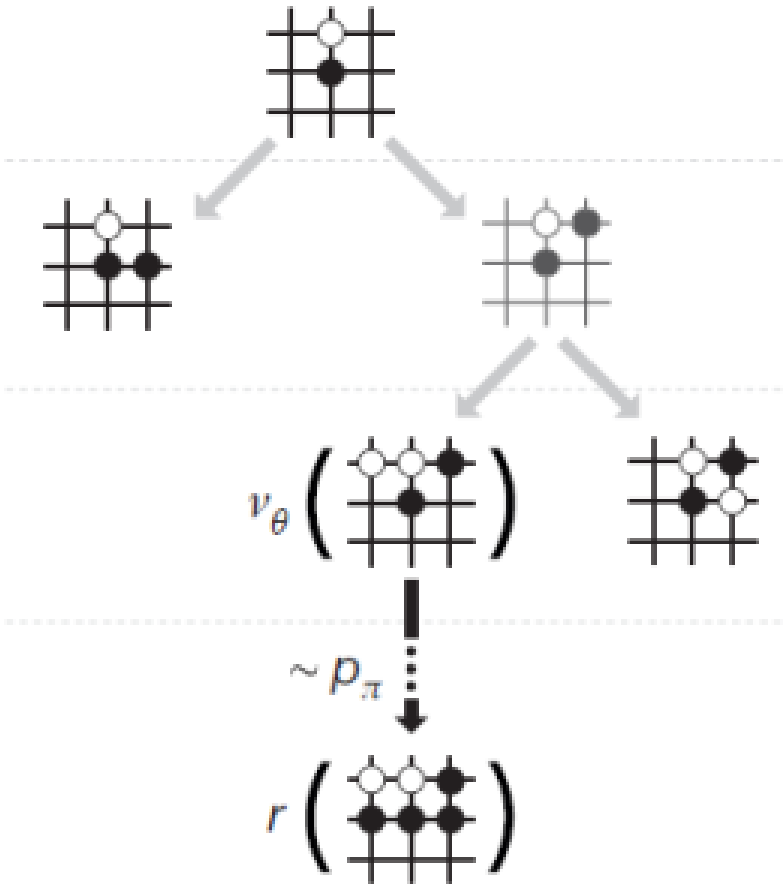
- Let $\boldsymbol{v}$ be the weights of the value network

- 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



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.

Challenges in RL

- Data efficiency
  - Most RL successes: simulated environments

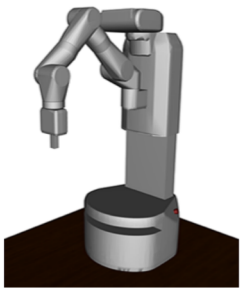
Atari 

MuJoCo 

VizDoom 

Computer Go 

- Robustness
  - How to bridge the “reality gap”



Peng et al. 2018

Robustness

- Safe reinforcement learning
  - Safety constraints
- Imprecise model/simulator
  - Domain adaptation
- Adversarial attacks
  - (Adversarial) partial observability

Safe RL

Important literature on

- Risk-Sensitive RL: change objective
  (Artzner et al., 1999; Borkar 2001; Mihatsch et al., 2002; Delage et al., 2010; Tamar et al., 2012; Chow et al., 2014; Tamar et al., 2015)

  $\max_\pi g(\sum_t \gamma^t r_t)$ where g is a risk measure

- Constrained RL: add constraints
  (Geibel 2006; Achiam et al. 2017; Bathia et al. 2018, Lee et al. 2018)

  $\max_\pi \sum_t \gamma^t E[r_t|s_t, \pi(s_t)] \quad s.t. \ \sum_t \gamma^t E[c_t|s_t, \pi(s_t)] \leq C$

- Many solutions that ensure asymptotic safety
- Open problem: how to ensure safety before convergence

Imprecise Model/Simulator

Domain adaptation

- Randomized training (Peng et al., 2018)

$$\max_\pi E_\mu[\sum_t \gamma^t E[r_t|s_t, \pi(s_t), \mu]]$$

where μ denotes a model

- Adversarial training (Pinto et al., 2017; Tessler et al., 2019)

$$max_\pi \min_\phi \sum_t \gamma^t E[r_t|s, \pi(s), \phi(s)]$$

where ϕ denotes adversary's policy

- Many solutions that improve robustness
- Open problem: performance guarantees

Conclusion

- REINFORCE algorithm
- Robustness:
  - safe RL, imprecise model/simulator, adversarial attacks

- Open Problems:
  - Safe exploration before convergence
  - Performance guarantees in domain adaptation