Lecture 13

Reinforcement Learning-Part 2

Q-Learning

- value iteration (MDP)

$$V(s) \leftarrow \max_{a'} R(s) + \gamma \sum_{s'} \Pr(s'|s,a)\, V(s')$$

- Q-Learning (RL)

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

Important functions

- Policy: $a = \pi(s)$
- Value function: $V(s) \in \mathcal{R}$
- Q-function: $Q(s, a) \in \mathcal{R}$

Q-function Approximation

Let

$$s = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

We need to approximate $Q(s, a)$. The function $Q(s, a)$ could be a linear or a nonlinear function. It can be represented as an approximation given by:

$$Q(s, a) \approx g(\mathbf{x}; \mathbf{w})$$

where $g(\cdot)$ is a function parameterized by $\mathbf{w}$ and $\mathbf{x}$ represents the feature vector.

Gradient Q-learning

- Minimize squared error between Q-value estimate and target
  - Recall: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  - Q-value estimate: $Q_{\boldsymbol{w}}(s,a)$
  - Target: $r + \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')$
- Loss function:

$$Err(\boldsymbol{w}) = \frac{1}{2}[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')]^2$$

- Gradient

$$\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

Gradient Q-learning

Initialize w and $\bar{\boldsymbol{w}}$

Observe the current state s

Loop

Select action a

Receive immediate reward

Observe new state s'

$$\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\bar{\boldsymbol{w}}}(s',a')\right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

$$w \leftarrow w - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$$

$$s \leftarrow s'$$

Instability in Deep Q-Learning

The expression:

$$\left[Q_w(s,a) - r - \gamma \max_{a'} Q_w(s',a')\right] \frac{\partial Q_w(s,a)}{\partial w}$$

may diverge during training, leading to instability in learning.

Solutions to Stabilize Training:

1. **Dual Network Approach:**
   - Use two separate networks: one for $Q_w(s,a)$ and another for $r - \gamma \max_{a'} Q_{\bar{w}}(s',a')$.
2. **Experience Replay:**
   - Store previous experiences and sample from this memory for learning.

Use two networks

- Q-network and Target network should be different.

Experience replay

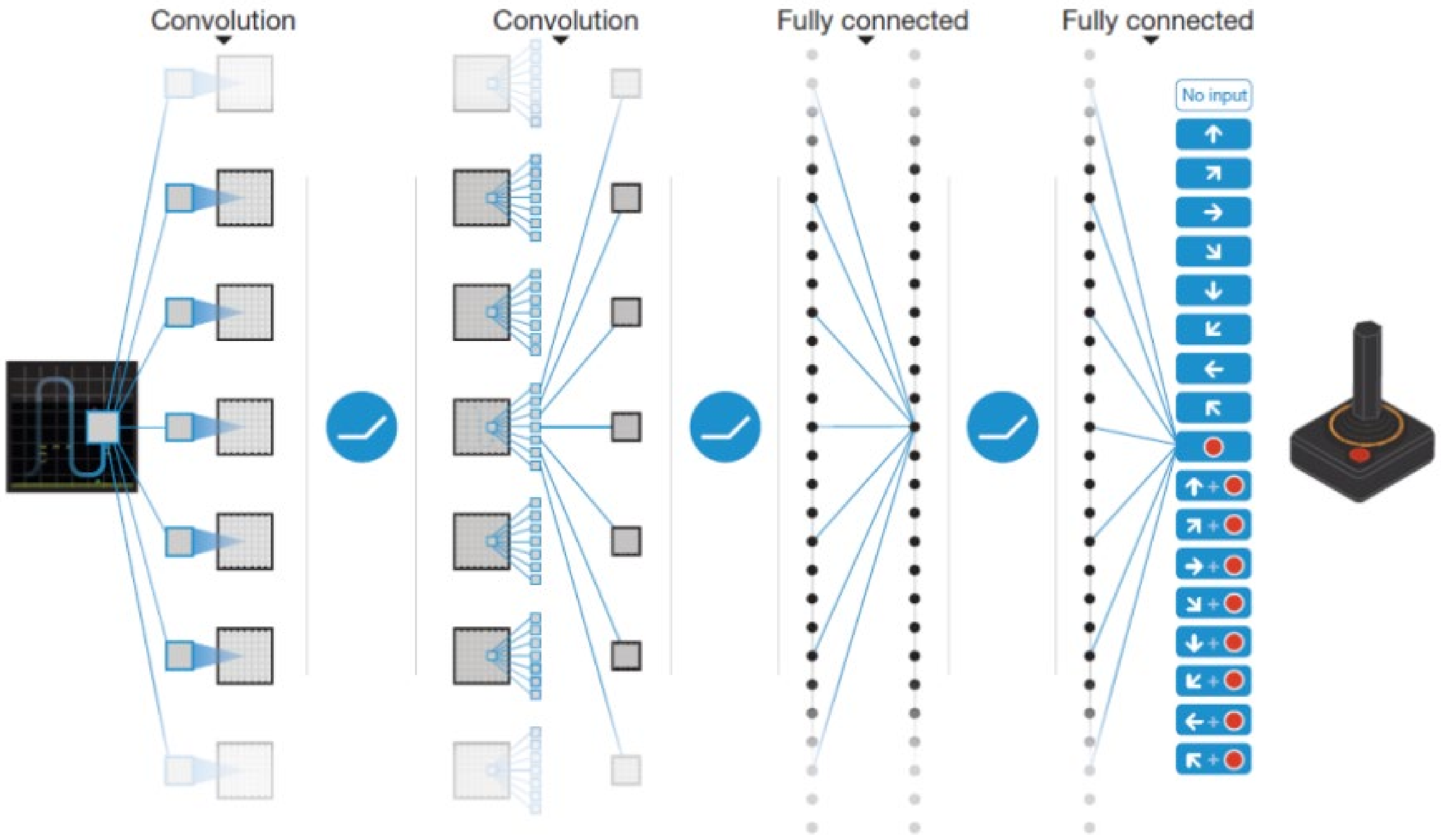
- Store previous experiences (s, a, s', r) into a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning

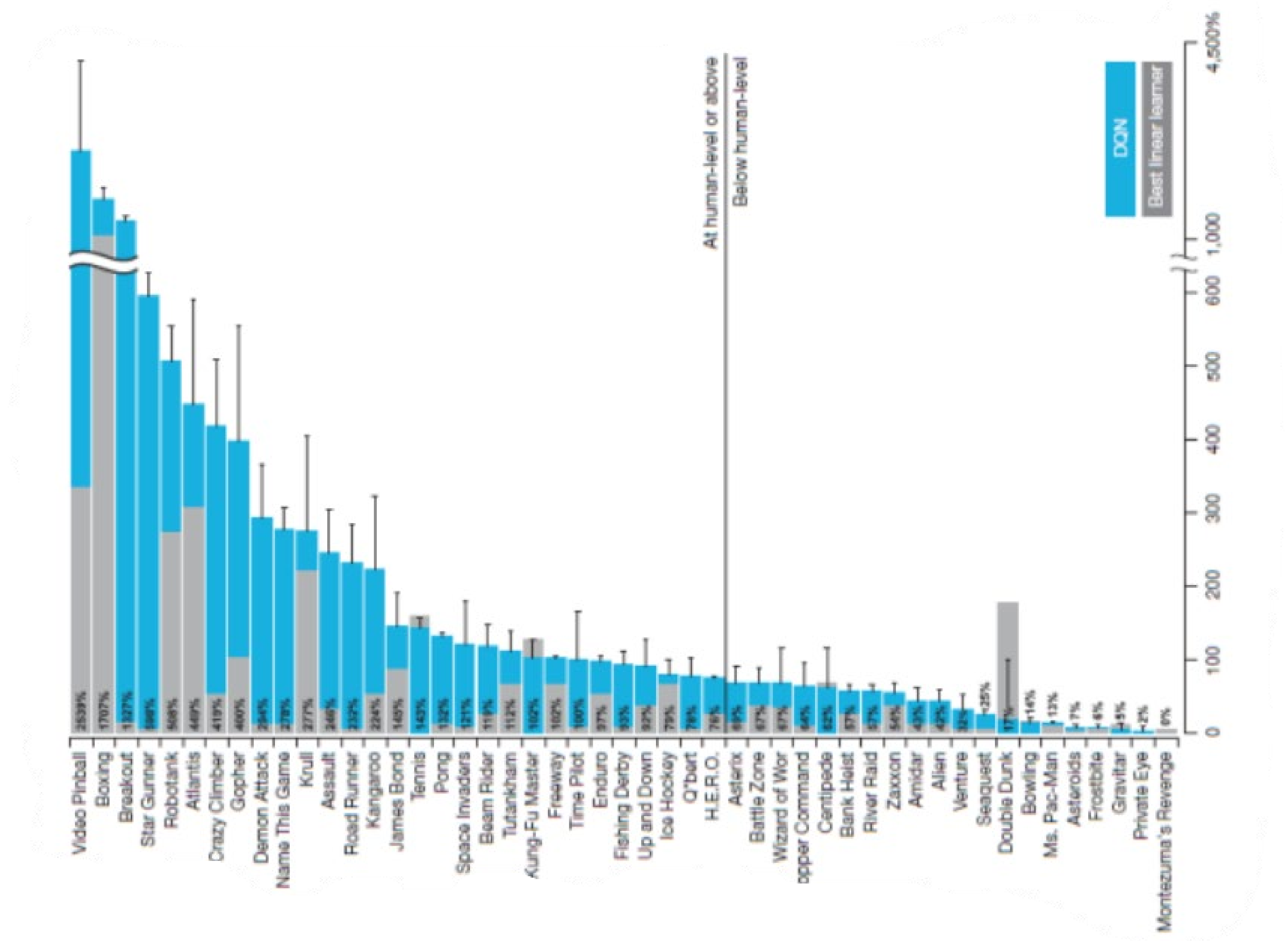
Deep Q-network

- Playing Atari with Deep Reinforcement Learning
  - Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller

- Human-level play in many Atari video games

Deep Q-Network for Atari





DQN
Best linear learner
At human-level or above
Below human-level
Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
0
100
200
300
400
500
600
1,000
4,500%

Policy gradient

- Q-learning
  - Model-free value-based method

- Policy gradient
  - Model-free policy-based method

Deterministic policy vs Stochastic Policy

- Deterministic policy $a = \pi(s)$

- Stochastic policy $\pi_w(a|s) = \Pr(a|s)$

Discrete vs Continuous

Discrete actions

$$\pi_w(a|s) = \frac{\exp(h(s,a;w))}{\sum_{a'} \exp(h(s,a';w))}$$

Supervised Learning

- We want to learn $\pi_w(a|s)$
- We have state-action pairs $\{(s_1, a_1^*), (s_2, a_2^*), \ldots\}$

Supervised Learning

- We want to learn $\pi_w(a|s)$

- We have state-action pairs $\{(s_1, a_1^*), (s_2, a_2^*), \ldots\}$

- Maximize log likelihood of the data

$$w^* = \underset{w}{\operatorname{argmax}} \sum_n \log \pi_w(a_n^*|s_n)$$

$$w_{n+1} \leftarrow w_n + \alpha_n \nabla_w \log \pi_w(a_n^*|s_n)$$

Reinforcement Learning

- We want to learn $\pi_w(a|s)$

- We have state-action-reward triplets
  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots\}$

Reinforcement Learning

- We want to learn $\pi_w(a|s)$

- We have state-action-reward triplets

  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$

- Maximize discounted sum of rewards

$$w^* = \underset{w}{\operatorname{argmax}} \sum_n \gamma^n E_w[r_n | s_n, a_n]$$

Reinforcement Learning

- We want to learn $\pi_w(a|s)$

- We have state-action-reward triplets

  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$

- Maximize discounted sum of rewards

$$w^* = \operatorname*{argmax}_w \sum_n \gamma^n E_w[r_n | s_n, a_n]$$

$$w_{n+1} \leftarrow w_n + \alpha_n \gamma^n G_n \nabla_w \log \pi_w(a_n|s_n)$$

where $G_n = \sum_{t=0}^{\infty} \gamma^t r_{n+t}$

Reinforcement Learning

- We want to learn $\pi_w(a|s)$

- We have state-action-reward triplets
  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots\}$

- Maximize discounted sum of rewards
  $$w^* = \underset{w}{\operatorname{argmax}} \sum_n \gamma^n E_w[r_n | s_n, a_n]$$

$$w_{n+1} \leftarrow w_n + \alpha_n \gamma^n G_n \nabla_w \log \pi_w(a_n | s_n)$$
where $G_n = \sum_{t=0}^{\infty} \gamma^t r_{n+t}$

Reinforcement Learning

- We want to learn $\pi_w(a|s)$

- We have state-action-reward triplets

  $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots\}$

- Maximize discounted sum of rewards

$$w^* = \underset{w}{\operatorname{argmax}} \sum_n \gamma^n E_w[r_n | s_n, a_n]$$

$$w_{n+1} \leftarrow w_n + \alpha_n \nabla_w \log \pi_w(a_n^* | s_n)$$

$$w_{n+1} \leftarrow w_n + \alpha_n \gamma^n G_n \nabla_w \log \pi_w(a_n | s_n)$$

where $G_n = \sum_{t=0}^{\infty} \gamma^t r_{n+t}$

REINFORCE Algorithm

- REINFORCE" stands for "REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility,"

- From a paper by Ronald J. Williams in 1992

REINFORCE Algorithm

- Initialize w
- Loop forever (for each episode)
  - Generate episodes $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$
  - Loop for each step of the episode $n = 0,1, \ldots, T$

    $G_n \leftarrow \sum_{t=0}^{T-n} \gamma^t r_{n+t}$

    Update policy: $w \leftarrow w + \alpha \gamma^n G_n \nabla \log \pi_w(a_n | s_n)$

Return π_w

Policy Gradient

- **Objective**: Maximize the expected return of a stochastic, parameterized policy, π_w.
- **Expected Return**:

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

Where $R(\tau)$ is the total rewrad.

Gradient Ascent Optimization

- **Optimizing the Policy by Gradient Ascent**:

$$w_{k+1} = w_k + \alpha \nabla_w J(\pi_w)$$

- The gradient, $\nabla_w J(\pi_w)$, is the **policy gradient**(Vanilla Policy Gradien).

Useful Facts for Derivation

1. **Probability of a Trajectory**: Given a trajectory $\tau = (s_0, a_0, \ldots, s_{T+1})$ with actions from π_w:

$$P(\tau|w) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_w(a_t|s_t)$$

2. **The Log-Derivative Trick**: The derivative of $\log(u)$ is $\frac{\nabla u}{u}$. By rearrangement $\nabla u = u \nabla log(u)$:

$$\nabla_w P(\tau|w) = P(\tau|w)\nabla_w \log P(\tau|w)$$

Useful Facts for Derivation

3. **Log-Probability of a Trajectory**:

$$\log P(\tau|w) = \log \rho_0(s_0) + \sum_{t=0}^{T} (\log P(s_{t+1}|s_t, a_t) + \log \pi_w(a_t|s_t))$$

4. **Gradients of Environment Functions**: The environment has no dependence on w, so gradients of $\rho_0(s_0)$, $P(s_{t+1}|s_t, a_t)$, and $R(\tau)$ are zero.

Useful Facts for Derivation

- **Grad-Log-Prob of a Trajectory**:
  The gradient of the log-prob of a trajectory is:

$$\nabla_w \log P(\tau|w) = \cancelto{0}{\nabla_w \log \rho_0(s_0)} +$$

$$\sum_{t=0}^{T} \left(\cancelto{0}{\nabla_w \log P(s_{t+1}|s_t, a_t)} + \nabla_w \log \pi_w(a_t|s_t) \right)$$

  Simplifying, we get:

$$\nabla_w \log P(\tau|w) = \sum_{t=0}^{T} \nabla_w \log \pi_w(a_t|s_t)$$

Basic Policy Gradient

$$
\begin{aligned}
\nabla_w J(\pi_w) &= \nabla_w \mathop{\mathrm{E}}_{\tau \sim \pi_w} [R(\tau)] \\
&= \nabla_w \int_\tau P(\tau \mid w) R(\tau) \\
&= \int_\tau \nabla_w P(\tau \mid w) R(\tau) \\
&= \int_\tau P(\tau \mid w) \nabla_w \log P(\tau \mid w) R(\tau) \\
&= \mathop{\mathbf{E}}_{\tau \sim \pi_w} [\nabla_w \log P(\tau \mid w) R(\tau)]
\end{aligned}
$$

$$
\therefore \nabla_w J(\pi_w) = \mathop{\mathbf{E}}_{\tau \sim \pi_w} \left[\sum_{t=0}^{T} \nabla_w \log \pi_w (a_t \mid s_t) R(\tau) \right]
$$

Basic Policy Gradient

- The policy gradient is an expectation, which can be estimated via sample mean.
- Using trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ from the policy π_w, we get:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_w \log \pi_w(a_t|s_t) R(\tau)$$

- $|\mathcal{D}|$ represents the number of trajectories (N in this case).
- This expression is our desired computable form.
- With a policy that allows $\nabla_w \log \pi_w(a|s)$ calculations and by collecting trajectory datasets, we can compute the gradient and update.

Comparing Policy Gradient with REINFORCE

- Both methods aim to optimize the policy to achieve maximum expected rewards.
- Both use gradient ascent to update the policy parameters.
- They differ in the way they handle rewards.

Comparing Policy Gradient with REINFORCE

- **Policy Gradient's update expression:**
  - Gradient of log-policy times Return of the trajectory:

$$R(\tau)\nabla_w \log \pi_w(a_t|s_t)$$

- **REINFORCE's update expression:**
  - Discounted cumulative reward times Gradient of log-policy:

$$\gamma^n G_n \nabla_w \log \pi_w(a_n|s_n)$$

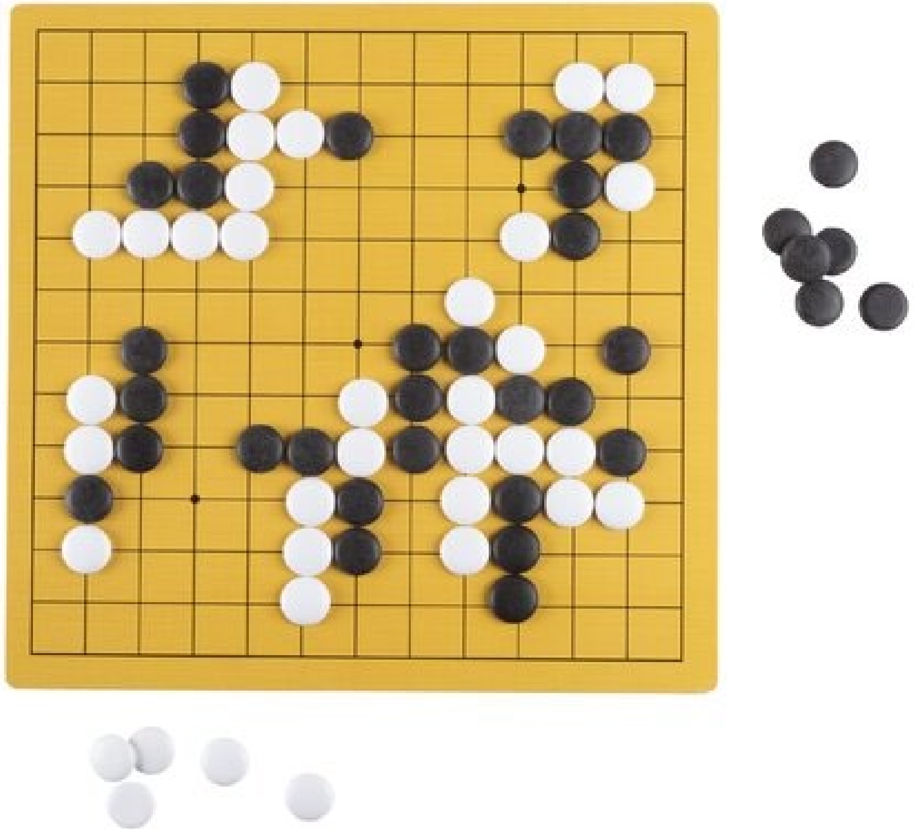
$$G_n = \sum_{t=0}^{\infty} \gamma^t r_{n+t}$$

Comparing Policy Gradient with REINFORCE

- $R(\tau)$: The return for a trajectory in the policy gradient method. It captures the total reward for a sequence of actions.
- G_n: Cumulative discounted reward for the trajectory in REINFORCE. It accounts for the sum of rewards, with future rewards being discounted.
- γ^n: The discount factor in REINFORCE. It diminishes the value of future rewards in a trajectory.
- For $\gamma = 1$, G_n at any time-step n is equivalent to the return $R(\tau)$ from that time-step.

Game of Go

- Players alternate to place a stone on a vacant intersection
- Connected stones that have no adjacent vacant intersection are removed
- Player who controls the largest intersections at the end of the game is the winner.



Game of Go Algorithm

1. **Supervised Learning of Policy Network:** Train the policy network using data from expert players.

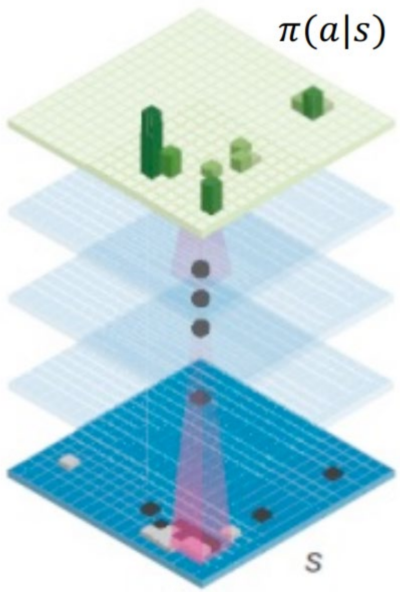
2. **Policy Gradient with Policy Network:** Refine the policy using reinforcement learning to improve strategies beyond the supervised initial training.

3. **Value Gradient with Value Network:** Train a value network to predict the likelihood of winning from a given board state.

4. **Search with Policy and Value Networks:** Utilize both networks to search through possible moves and select the best one, optimizing gameplay.

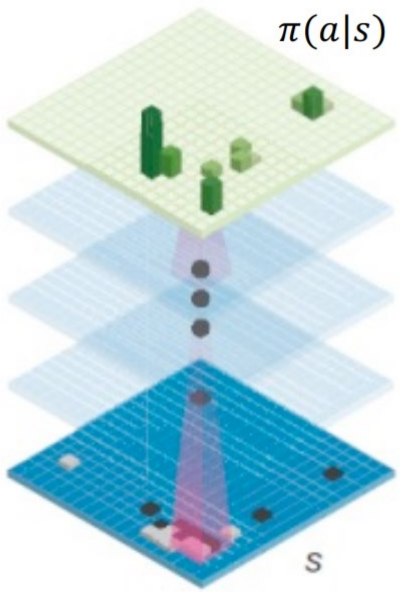
Policy Network

- Policy network: $\pi(a|s)$
  - Input: state s
    - s: board configuration
  - Output: distribution of actions a
    - a: intersection on which the next stone will be place



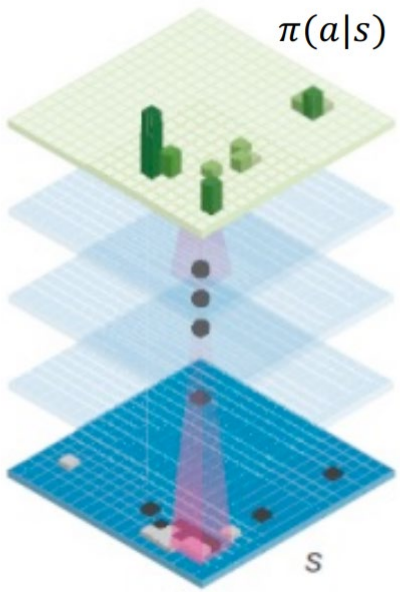
Policy Network

- Train policy network based on 30 million board configurations.



Supervised Learning of the Policy Network

- Train policy network based on 30 million board configurations.



maximize $\log \pi_w(a|s)$

$$\nabla \mathrm{w} = \frac{\partial \log \pi_w(a|s)}{\partial w}$$

$$w \leftarrow w + \alpha \nabla w$$

Policy gradient for the Policy Network

- Play games against its former self.

- For each game $G_n = \begin{cases} 1 & \text{win} \\ -1 & \text{lose} \end{cases}$

Policy gradient for the Policy Network

- Let $G_n = \sum_t \gamma^t r_{n+t}$ be the discounted sum of rewards in a trajectory that starts in s at the time n by executing a.

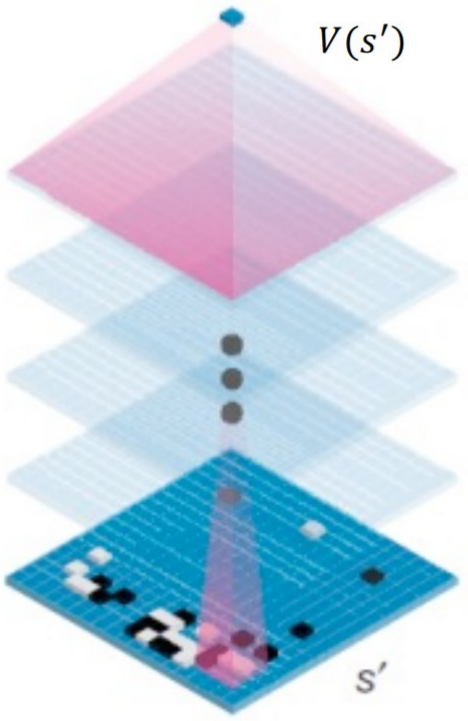
$$\nabla w = \frac{\partial \log \pi_w(a|s)}{\partial w} \gamma^n G_n$$

$$w \leftarrow w + \alpha \nabla w$$

Value Network

- Predict $V(s\quad)$ (i.e., who will win the game)
  - Input: state s
    - s : board configuration
  - Output: expected discounted sum of rewards

  $V(s\quad)$



Gradient Value Learning with Value Networks

- Data: (s, G) where $\mathrm{G} = \begin{cases} 1 & \text{win} \\ -1 & \text{lose} \end{cases}$
- Objective: minimize $\frac{1}{2}(V_w(s) - G)^2$
- Gradient: $\nabla w = \frac{\partial V_w(s)}{\partial w}(V_w(s) - G)$
- Weight update: $w \leftarrow w - \alpha \nabla w$

Monte Carlo Tree Search

- AlphaGo combines policy and value networks into a **Monte Carlo Tree Search (MCTS)** algorithm
  - Node: s
  - Edge: a

Monte Carlo Tree Search

- AlphaGo combines policy and value networks into a **Monte Carlo Tree Search (MCTS)** algorithm
  - Node: s
  - Edge: a