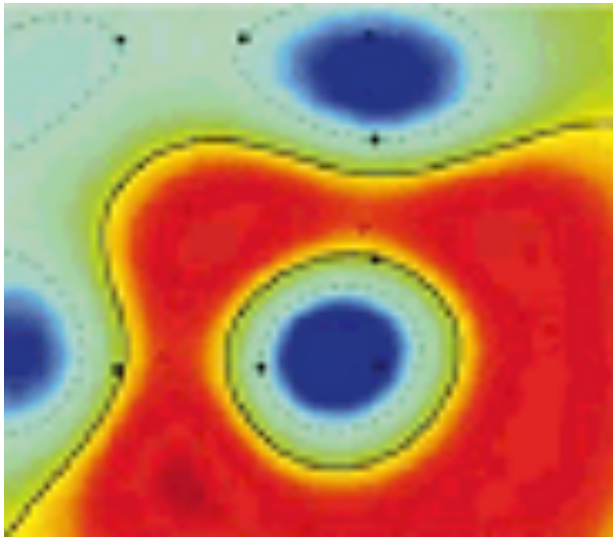
SoSe 2024

Deep Learning 2

Technische Universität Berlin

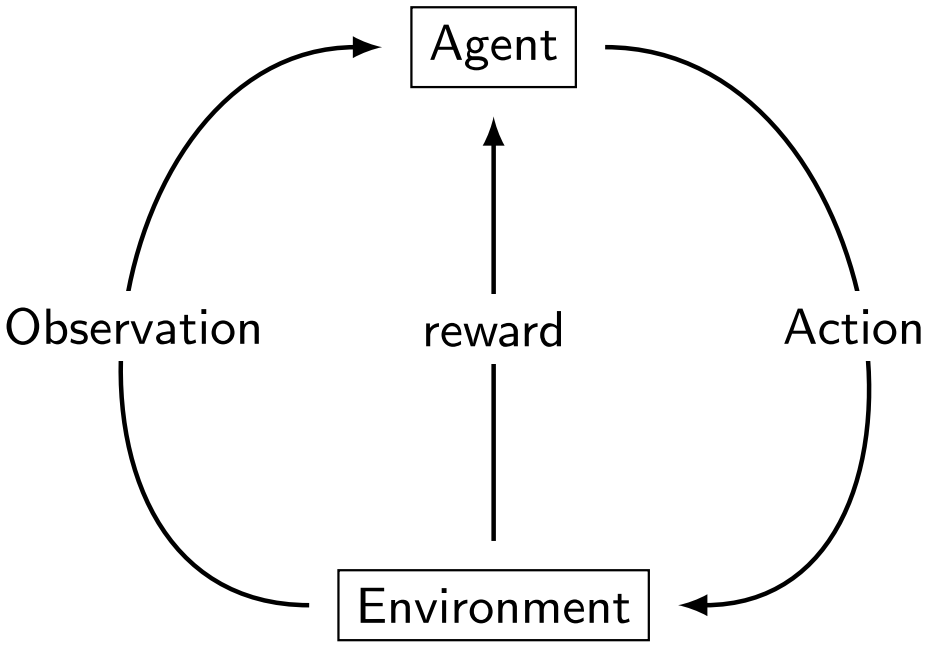


Lecture

Deep Reinforcement Learning

Reinforcement Learning Framework

- Take <u>actions</u> in an <u>environment</u> that <u>maximize reward</u>
  - Environment $\mapsto$ States s' sampled from $T(s'|s,a)$
  - Actions $\mapsto$ a sampled by Policy $\pi(a|s)$
  - Reward $\mapsto$ Feedback from environment $R(s,a)$



Supervised vs Unsupervised vs Reinforcement Learning

- Supervised learning: classification, regression
- Unsupervised learning: density estimation, outlier detection
- Offer metrics which evaluate performance directly

Supervised vs Unsupervised vs Reinforcement Learning

- Supervised learning: classification, regression
- Unsupervised learning: density estimation, outlier detection
- Offer metrics which evaluate performance directly

- Reinforcement learning (RL) is somewhat orthogonal
- Provides a holistic reward signal how well you did in total
- Akin to studying for an exam
  - Grade evaluates your studying behavior
  - Without telling you explicitly what you did good or bad

Supervised vs Unsupervised vs Reinforcement Learning

- Supervised learning: classification, regression
- Unsupervised learning: density estimation, outlier detection
- Offer metrics which evaluate performance directly

- Reinforcement learning (RL) is somewhat orthogonal
- Provides a holistic reward signal how well you did in total
- Akin to studying for an exam
  - Grade evaluates your studying behavior
  - Without telling you explicitly what you did good or bad

The RL challenges

- **Credit assignment problem in sequence of actions**
- **Exploitation vs Exploration**

Notation

Defining the learning problem

State of environment	$s \in \mathcal{S}$
Available actions	$a \in \mathcal{A}(s)$ (dependence on s implicit)
Transition model	$T(s'\|s,a) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R} \geq 0$
Reward function	$R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Notation

Defining the learning problem

State of environment	$s \in \mathcal{S}$
Available actions	$a \in \mathcal{A}(s)$ (dependence on s implicit)
Transition model	$T(s'\|s,a) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R} \geq 0$
Reward function	$R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Our tools to solve it

Policy of agent	$\pi(a\|s) : \mathcal{A} \to \mathbb{R} \geq 0$
Value function	$V(s) : \mathcal{S} \to \mathbb{R}$
Action-Value function	$Q(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
Discount factor	$\gamma \in (0,1]$

Final output of RL algorithm: π

Markov Decision Processes

- History $H_t = \{(s_\tau, a_\tau, r_\tau)\}_{\tau \leq t}$
- Decision Process: $p(r_t, s_{t+1} | H_t)$

Markov Decision Processes

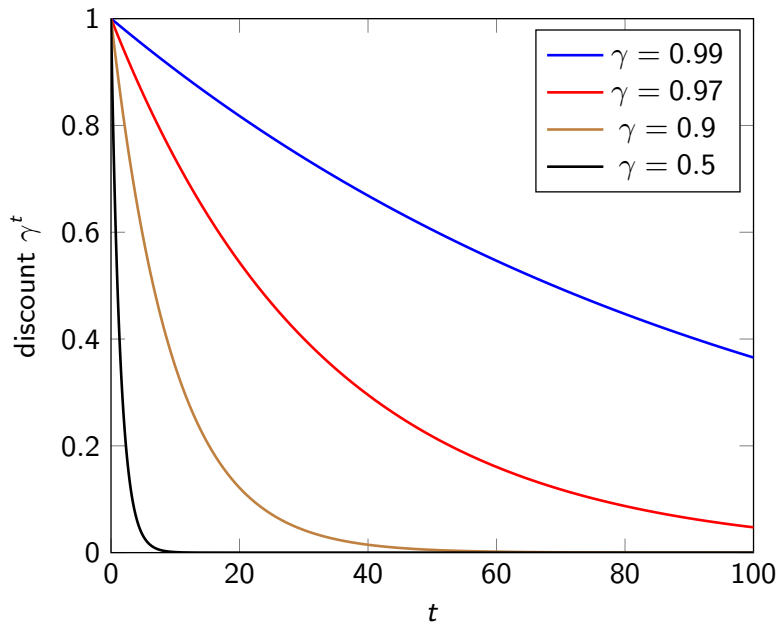
- History $H_t = \{(s_\tau, a_\tau, r_\tau)\}_{\tau \leq t}$
- Decision Process: $p(r_t, s_{t+1} | H_t)$
- *Markov* Decision Process: $p(r_t, s_{t+1} | a_t, s_t) = p(r_t, s_{t+1} | H_t)$
- All relevant information about state and evolution encompassed by current s_t

Value Function V

- Estimation of future rewards following policy π from a state s
- Geometric discounting of future rewards

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s^{(t)}, a^{(t)}) \middle| \begin{array}{l} s^{(0)} = s \\ a^{(t)} \sim \pi(\cdot|s^{(t)}) \\ s^{(t+1)} \sim T(\cdot|s^{(t)}, a^{(t)}) \end{array}\right], \gamma \in [0, 1)$$

$$V^\pi(s) = \mathbb{E}_{\pi,T}\left[\sum_{t=0}^{\infty} \gamma^t R(s^{(t)}, a^{(t)}) | s^{(0)} = s\right]$$



State-action value function Q

- $V^\pi(s)$ only provides state values
  - No information about value of possible actions a
- $Q^\pi(s, a)$ provides value for action a in state s_i

$$Q^\pi(s, a) = \mathbb{E}_{\pi,T}\left[\sum_{t=0}^{\infty} \gamma^t R(s^{(t)}, a^{(t)}) \middle| s^{(0)} = s; a^{(0)} = a\right]$$

State-action value function Q

- $V^\pi(s)$ only provides state values
  - No information about value of possible actions a
- $Q^\pi(s, a)$ provides value for action a in state s_i

$$Q^\pi(s, a) = \mathbb{E}_{\pi,T}\left[\sum_{t=0}^{\infty} \gamma^t R(s^{(t)}, a^{(t)}) \middle| s^{(0)} = s; a^{(0)} = a\right]$$

- More expressive but also more data needed. Note that if a transition model T is available, Q can be computed from V:

$$Q^\pi(s, a) = R(s, a) + \sum_{s'} T(s'|s, a)\gamma V(s')$$

The above equation uses the Bellmann relation introduced in the next slides.

Bellmann equation and dynamic programming

The Bellmann equation is a recursive relationship for value functions:

$$V^{\pi}(s^{(t)}) = E_{\pi,T}\left[R^{(t)} + \gamma \underbrace{R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 R^{(t+3)}}_{V^{\pi}(s^{(t+1)})} \dots\right]$$

$$V^{\pi}(s^{(t)}) = E_{\pi,T}\left[R^{(t)} + \gamma V^{\pi}(s^{(t+1)})\right]$$

To simplify the notation, here we understand that the rewards R are random variables determined by following the policy π under dynamics T.

Bellmann equation and dynamic programming

The Bellmann equation is a recursive relationship for value functions:

$$V^{\pi}(s^{(t)}) = E_{\pi,T}\left[R^{(t)} + \gamma \underbrace{R^{(t+1)} + \gamma^2 R^{(t+2)} + \gamma^3 R^{(t+3)}}_{V^{\pi}(s^{(t+1)})} \dots\right]$$

$$V^{\pi}(s^{(t)}) = E_{\pi,T}\left[R^{(t)} + \gamma V^{\pi}(s^{(t+1)})\right]$$

To simplify the notation, here we understand that the rewards R are random variables determined by following the policy π under dynamics T.

Approximating a value function for a given policy π can be done iteratively via the Bellmann equation:

$$V^{\pi}_{k+1}(s^{(t)}) = E_{\pi,T}\left[R^{(t)} + \gamma V^{\pi}_{k}(s^{(t+1)})\right]$$

$$V^{\pi}_{k+1}(s^{(t)}) = \sum_{a\in\mathcal{A}} \pi(a|s^{(t)}) \sum_{s'\in\mathcal{S}} T(s'|s^{(t)}, a)\left[R(s^{(t)}, a) + \gamma V^{\pi}(s')\right]$$

Monte Carlo methods

- The term "Monte Carlo" is generally used for estimation methods that use random sampling in their process
  - e.g., easy to compute the circle-number π via sampling (not the policy π!)
- Learning from experience
  - No need to know the full dynamics of the model, i.e. $p(r, s'|a, s) \;\; \forall r, s', a, s$
- In RL, we define Monte Carlo methods only for tasks with a terminal state (episodic tasks)
- Assigns state values by averaging over experienced returns after having visited a state

Monte Carlo methods

- Assigns state values by averaging over experienced returns after having visited a state
  - $V(s) = \mathbb{E}[G(s)]$, where $G(s)$ is a random variable specifying the return we get if we have visited state s at least once in an episode. The return G_t is defined as

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

  - E.g. assume we have gathered data from N episodes, resulting in N returns $(G_i)_{i \leq N}$ and N state sequences $(\sigma_i)_{i \leq N}$. Then we assign

$$V(s) \leftarrow \frac{\sum_{i \leq N, s \in \sigma_i} G_i}{|\{G_i \mid i \leq N, s \in \sigma_i\}|}$$

  - Update after just one episode:

$$V(s) \leftarrow V(s) + \alpha \cdot (G_t - V(s))$$

Example: Driving home

- Trying to estimate the time needed to get home after work
- At each step (situation encountered), re-estimate time needed.

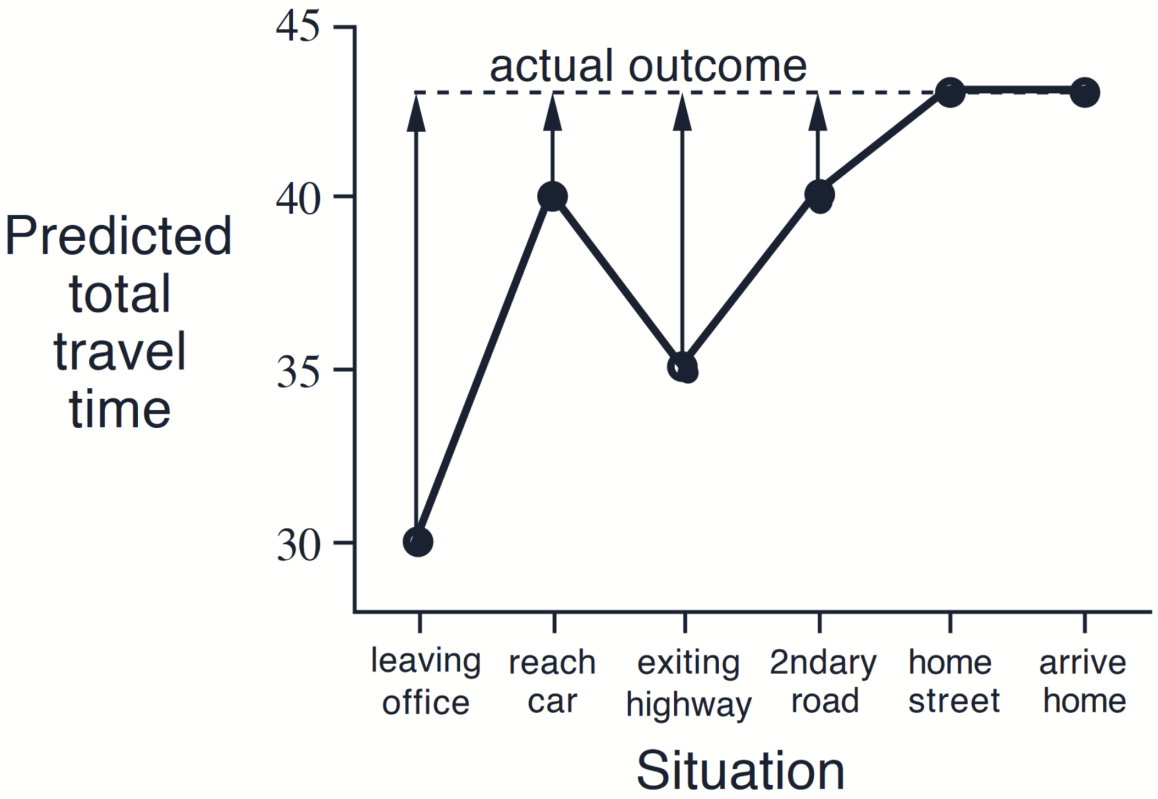


Figure: Value adjustments using Monte Carlo method
cf. "Reinforcement Learning: An introduction" 2nd ed., Figure 6.1, Sutton and Barto 2018

Temporal-Difference (TD) learning

- *The* central aspect of RL
- Like Dynamic Programming, uses recursive update
- Like Monte Carlo, learns by experience
- Update state value function by using reward R_{t+1} of next step (experience) and current estimate of next state's value $V(S_{t+1})$ (bootstrapping)

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \\ &= V(S_t) + \alpha \delta_t \end{aligned}$$

Monte Carlo vs. Temporal-Difference

Monte Carlo update

$$V(s) \leftarrow V(s) + \alpha \cdot (G_t - V(s))$$

Temporal Difference update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

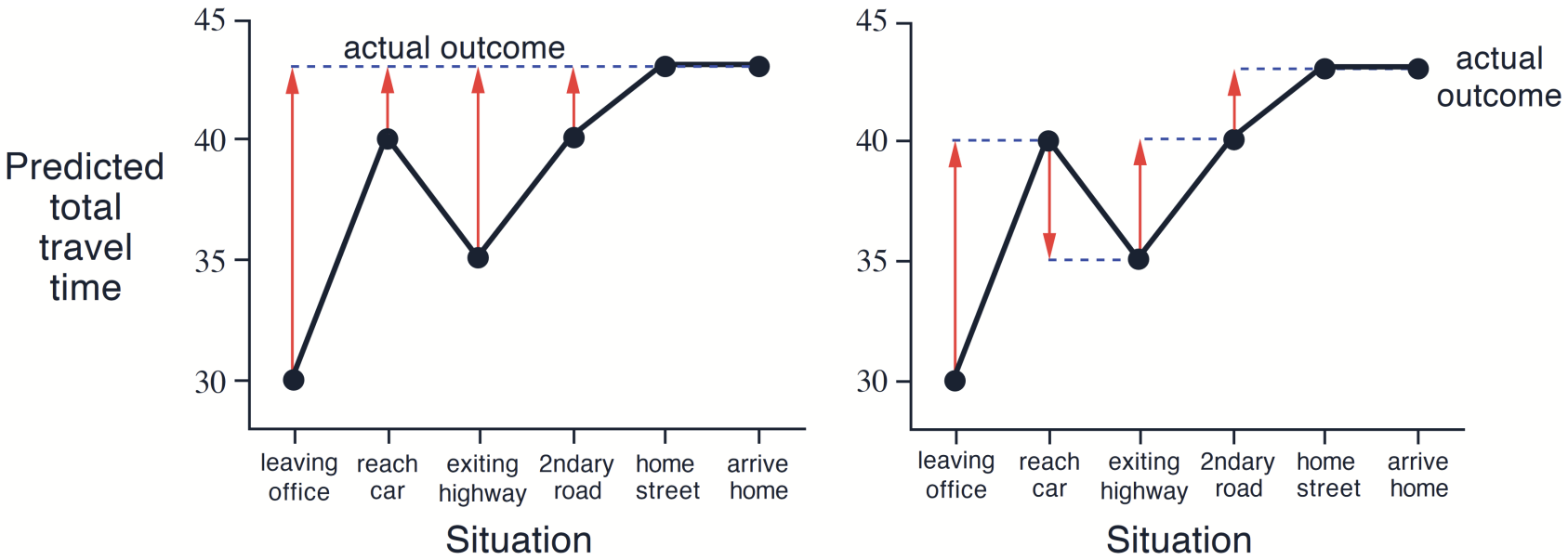


Figure: Comparison of value estimation of MC methods and TD learning
cf. "Reinforcement Learning: An introduction" 2nd ed., Figure 6.1, Sutton and Barto 2018

TD algorithms (on- vs. off-policy)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

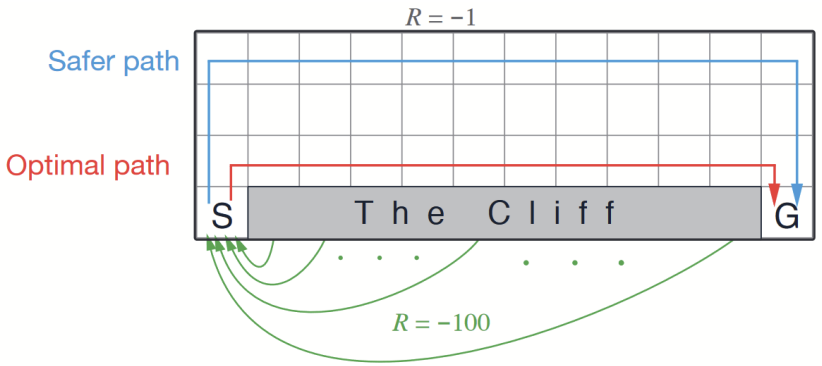
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure: Explaining off- vs. on-policy using examples Sarsa and Q-learning
cf. "Reinforcement Learning: An introduction" 2nd ed., chapter 6.4+6.5, Sutton and Barto 2018

Example: Gridworld cliff walking

- Sarsa learns safer path, because ϵ-greedy strategy of policy is incorporated in learning action value estimates (on-policy) and thus avoids states near the cliff
- Q-learning learns optimal path because the policy is not being considered in estimation of the action-value function (off-policy)
- Sum of rewards in Q-learning lower, because of the use of ϵ-greedy strategy when walking the optimal path, which occasionally (ϵ) causes the agent to "fall off the cliff"



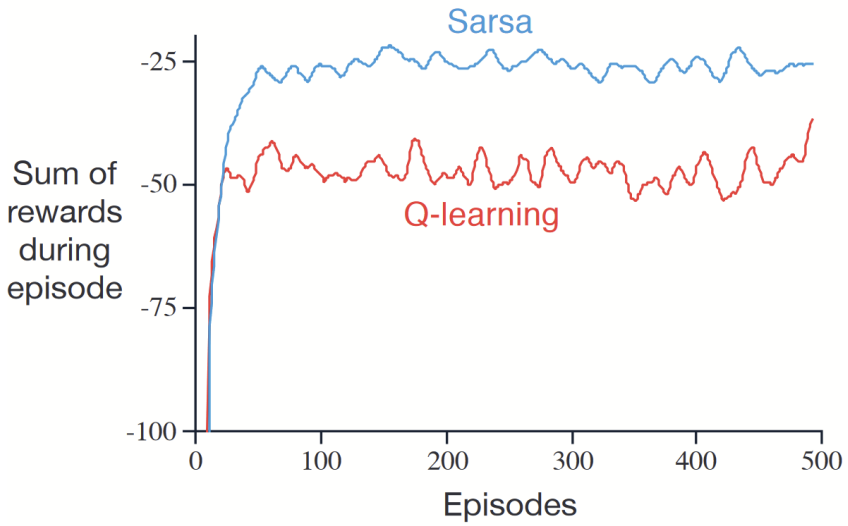


Figure: *cf. "Reinforcement Learning: An introduction" 2nd ed., Example 6.6, Sutton and Barto 2018*

Approximate Reinforcement Learning

- Previous methods worked all tabular
- Not recommendable for large or continuous state-action spaces
  - Board game Go on 19×19 board: $3^{19 \times 19} \approx 10^{172}$
  - Autonomous helicopter control is continuous

Approximate Reinforcement Learning

- Previous methods worked all tabular
- Not recommendable for large or continuous state-action spaces
  - Board game Go on 19×19 board: $3^{19 \times 19} \approx 10^{172}$
  - Autonomous helicopter control is continuous
- Use function approximators (e.g. neural networks) to generalize Value and Q-Value function
- Original DeepMind Atari Agent used Q-Value learning on discrete actions

Approximate Reinforcement Learning

- Previous methods worked all tabular
- Not recommendable for large or continuous state-action spaces
  - Board game Go on 19×19 board: $3^{19 \times 19} \approx 10^{172}$
  - Autonomous helicopter control is continuous
- Use function approximators (e.g. neural networks) to generalize Value and Q-Value function
- Original DeepMind Atari Agent used Q-Value learning on discrete actions

- Uses replay buffer to collect trajectories of agent interaction with environment
- Compute discounted reward for all actions in trajectories
- Train neural network to to predict maximum Q-Value

Q-Value Function Approximation

- Parameterization of Q-Value function with $\boldsymbol{\theta}$: $\widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$
- Minimize MSE between approximation $\widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$ and true $Q^{\pi}(a, s)$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})\right)^2\right]$$

Q-Value Function Approximation

- Parameterization of Q-Value function with $\boldsymbol{\theta}$: $\widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$
- Minimize MSE between approximation $\widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$ and true $Q^{\pi}(a, s)$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})\right)^2\right]$$

- Gradient descent finds the local minimum

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi}\left[\left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})\right)^2\right] \\ &= 2\ \mathbb{E}_{\pi}\left[\left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})\right) \nabla_{\boldsymbol{\theta}} \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})\right]\end{aligned}$$

Q-Value Function Approximation

- Full batch update often unfeasible
- Stochastic gradient descent with update rule instead

$$\Delta\boldsymbol{\theta} = \alpha \left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$$

Q-Value Function Approximation

- Full batch update often unfeasible
- Stochastic gradient descent with update rule instead

$$\Delta\boldsymbol{\theta} = \alpha \left(Q^{\pi}(a, s) - \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \widehat{Q}^{\pi}(a, s; \boldsymbol{\theta})$$

- Target value function $Q^{\pi}(s, a)$ unknown
- $Q^{\pi}(s^{(t)}, a^{(t)}) \approx R(s^{(t)}, a^{(t)}) + \gamma \widehat{Q}^{\pi}(s^{(t+1)}, a^{(t+1)}; \boldsymbol{\theta})$
- Bootstrap your own model as predictor of future rewards

$$\Delta\boldsymbol{\theta} = \alpha \left(R(s^{(t)}, a^{(t)}) + \gamma \widehat{Q}^{\pi}(s^{(t+1)}, a^{(t+1)}; \boldsymbol{\theta}) - \widehat{Q}^{\pi}(s^{(t)}, a^{(t)}; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \widehat{Q}^{\pi}(s^{(t)}, a^{(t)}; \boldsymbol{\theta})$$

$$\Delta\boldsymbol{\theta} = \alpha \delta_t \nabla_{\boldsymbol{\theta}} \widehat{Q}^{\pi}(s^{(t)}, a^{(t)}; \boldsymbol{\theta})$$

Policy Gradient Methods

- Previously determinstic value function approximation with implied policy
- Approximation of policy

$$\pi_{\boldsymbol{\theta}}(s, a) = \mathbb{P}\left[a|s; \boldsymbol{\theta}\right]$$

- Does not learn a value function, but directly a policy

Policy Gradient Methods

- Previously determinstic value function approximation with implied policy
- Approximation of policy

$$\pi_{\boldsymbol{\theta}}(s, a) = \mathbb{P}\left[a|s; \boldsymbol{\theta}\right]$$

- Does not learn a value function, but directly a policy

- More effective in high-dimensional or continuous spaces
- Better convergence
- Can learn stochastic policies

Policy Gradient Methods

- Policy quality measured with objective function $J^{\pi_{\boldsymbol{\theta}}}(s^{(0)})$
- Return measured by expectancy of reward over policy

$$
\begin{aligned}
J^{\pi_{\boldsymbol{\theta}}}(s^{(0)}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)})\right] \\
&= \underbrace{\sum_{a^{(0:T)},} \pi_{\boldsymbol{\theta}}(a^{(0:T)}|s^{(0:T)})}_{\text{controlled by policy}} R(s^{(t)}, a^{(t)})
\end{aligned}
$$

- $\sum_{a^{(0:T)}}$ are all possible actions $a^{(t)}$ at time step t with $t \in \{0, \ldots, T\}$

Policy Gradient Theorem

- Policy gradient

$$\nabla_{\boldsymbol{\theta}} J^{\pi_{\boldsymbol{\theta}}}(s^{(0)}) = \sum_{a^{(0:T)}} \underbrace{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a^{(0:T)}|s^{(0:T)})}_{(\log f(x))' = \frac{f(x)'}{f(x)}} \left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)})\right]$$

$$= \sum_{a^{(0:T)}} \pi_{\boldsymbol{\theta}}(a^{(0:T)}|s^{(0:T)}) \left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)})\right] \nabla_{\boldsymbol{\theta}} \log \left[\pi_{\boldsymbol{\theta}}(a^{(0:T)}|s^{(0:T)})\right]$$

$$= \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)}) \nabla_{\boldsymbol{\theta}} \log \left[\pi_{\boldsymbol{\theta}}(a^{(0:T)}|s^{(0:T)})\right]\right]$$

- Sampling of trajectories with corresponding rewards necessary
- Trajectory sampling inefficient and high-variance
- Alternative derivation at `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/` or chapter 13.2 in "Reinforcement Learning" (Sutton and Barto)

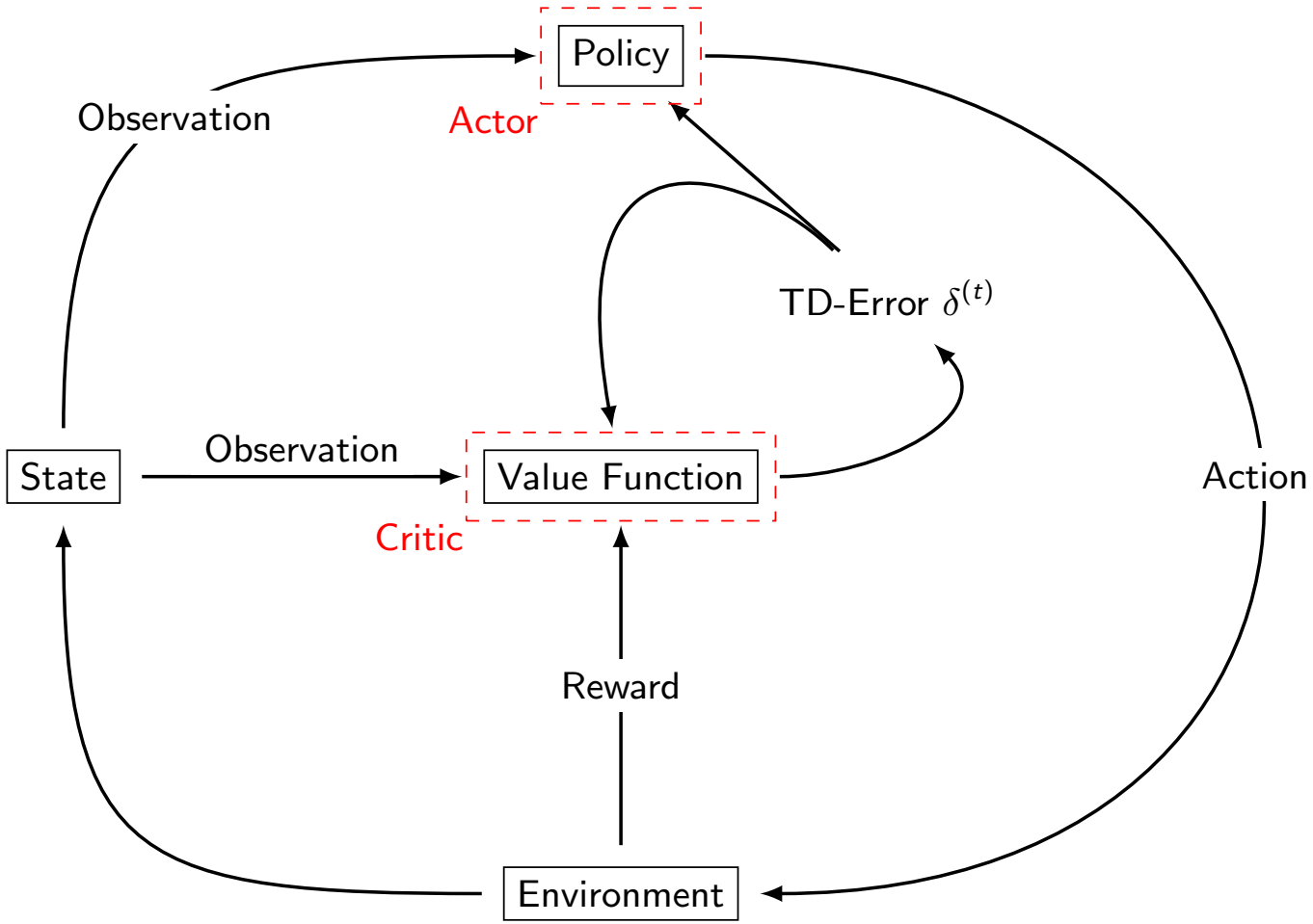
Actor-Critic Methods

- Compatible Function Approximation Theorem:
  Replace trajectory reward with long-term value $Q^{\pi_{\boldsymbol{\theta}}}(s^{(t)}, a^{(t)})$

$$\nabla_{\boldsymbol{\theta}} J^{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)}) \; \nabla_{\boldsymbol{\theta}} \log \left[\pi_{\boldsymbol{\theta}}(a^{(0:T)} | s^{(0:T)}) \right] \right]$$
$$\approx \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[Q^{\pi_{\boldsymbol{\theta}}}(s^{(t)}, a^{(t)}) \nabla_{\boldsymbol{\theta}} \log \left[\pi_{\boldsymbol{\theta}}(a^{(t)} | s^{(t)}) \right] \right]$$

- Actor $\pi_{\boldsymbol{\theta}}$ performs and critic $Q^{\pi_{\boldsymbol{\theta}}}(s, a)$ evaluates
- Critic determines value function and policy update
- Critic reduces variance

Actor-Critic Methods



Advantage Function

- Advantage $A^\pi(a, s) = Q^\pi(a, s) - V^\pi(s)$
- Advantage of discounted future reward $Q^\pi(s, a)$ over value of state $V^\pi(s)$ for specific action a in state s
- Numerically differentiates advantage of actions
- Serves as a 'normalizer' to offer better training signals
- Requires additional networks for Q and V

Trust Region Policy Optimization

- Maximize policy probability $\pi(a|s)$ given advantage $A^{\pi}(a,s) = Q^{\pi}(a,s) - V^{\pi}(s)$

$$\max_{\theta} \quad \mathbb{E}_{\rho_{\pi'}(s)} \left[\mathbb{E}_{\pi_{\theta'}(a|s)} \left[\pi_{\theta}(a|s) \; A^{\pi'}(s,a) \right] \right]$$

  where differentiate between old policy $\pi_{\theta'}$ and current policy π_{θ}
- Old policy $\pi_{\theta'}$ is used to generate training data, but we optimize our current policy π_{θ}

Trust Region Policy Optimization

- Maximize policy probability $\pi(a|s)$ given advantage $A^\pi(a,s) = Q^\pi(a,s) - V^\pi(s)$

$$\max_\theta \quad \mathbb{E}_{\rho_{\pi'}(s)}\left[\mathbb{E}_{\pi_{\theta'}(a|s)}\left[\pi_\theta(a|s)\ A^{\pi'}(s,a)\right]\right]$$

  where differentiate between old policy $\pi_{\theta'}$ and current policy π_θ
- Old policy $\pi_{\theta'}$ is used to generate training data, but we optimize our current policy π_θ
- But next policy shouldn't stray too far from old policy
- Add constrain of KL divergence to probabilistic policies π

$$\max_\theta \quad \mathbb{E}_{\rho_{\pi'}(s)}\left[\mathbb{E}_{\pi_\theta(a|s)}\left[\pi_{\theta'}(a|s)\ A^{\pi'}(s,a)\right]\right]$$
$$\text{s.t. } \mathbb{KL}\left[\ \pi_\theta \ \|\ \pi_{\theta'}\right] \leq \epsilon$$

Proximal Policy Optimization

- Iterates on TRPO to make algorithm easier and robuster
- Improve current stochastic policy π_θ on trajectories generated with old policy $\pi_{\theta'}$
- Ratio of current policy θ to old policy θ': $\pi_\theta(a|s)/\pi_{\theta'}(a|s)$
- Bound policy improvement $\pm\epsilon$ to not overshoot and become unstable

Proximal Policy Optimization

- Iterates on TRPO to make algorithm easier and robuster
- Improve current stochastic policy π_θ on trajectories generated with old policy $\pi_{\theta'}$
- Ratio of current policy θ to old policy θ': $\pi_\theta(a|s)/\pi_{\theta'}(a|s)$
- Bound policy improvement $\pm\epsilon$ to not overshoot and become unstable
- Increase probability of actions with positive advantage
- Gradient *ascent* on probability of advantage improving action

$$A^{\pi_{\theta'}} > 0 : J(\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, 1+\epsilon\right) A^{\pi_{\theta'}}(a,s) \tag{1}$$

Proximal Policy Optimization

- Iterates on TRPO to make algorithm easier and robuster
- Improve current stochastic policy π_θ on trajectories generated with old policy $\pi_{\theta'}$
- Ratio of current policy θ to old policy θ': $\pi_\theta(a|s)/\pi_{\theta'}(a|s)$
- Bound policy improvement $\pm\epsilon$ to not overshoot and become unstable
- Increase probability of actions with positive advantage
- Gradient *ascent* on probability of advantage improving action

$$A^{\pi_{\theta'}} > 0 : J(\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, 1+\epsilon\right) A^{\pi_{\theta'}}(a,s) \tag{1}$$

- Decrease probability of actions which have negative (suboptimal) advantage under policy
- Negative advantage automatically invokes gradient *descent*

$$A^{\pi_{\theta'}} < 0 : J(\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, 1-\epsilon\right) A^{\pi_{\theta'}}(a,s) \tag{2}$$

Soft Actor Critic

- Find optimal policy to achieve goal with entropy of stochastic policy as additional reward

$$J^{\pi_\theta}(s) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} R(s^{(t)}, a^{(t)}) + \alpha H[\pi_\theta(a^{(t)}|s^{(t)})]\right]$$

- Bakes exploration directly into value and Q function with entropy H

$$Q^\pi(s^{(t)}, a^{(t)}) \approx R(s^{(t)}, a^{(t)}) + \gamma\left(\widehat{Q}^\pi(s^{(t+1)}, a^{(t+1)}; \boldsymbol{\theta}) + \alpha H[\pi_\theta(a^{(t+1)}|s^{(t+1)})]\right) \tag{3}$$

$$= R(s^{(t)}, a^{(t)}) + \gamma\left(\widehat{Q}^\pi(s^{(t+1)}, a^{(t+1)}; \boldsymbol{\theta}) - \alpha \log \pi_\theta(a^{(t+1)}|s^{(t+1)})\right) \tag{4}$$

$$V^\pi(s) = \mathbb{E}_{\pi(a|s)}\left[Q^\pi(s, a) + \alpha H(\pi(a|s)\right] \tag{5}$$

- Entropy regularization is propagated through entire training procedure

Offline vs Online Reinforcement Learning

- Online RL algorithms interact freely with environment
- Can create its own training data set of actions, states and rewards
- Usually when environment is simulated as with games or simulations

Offline vs Online Reinforcement Learning

- Online RL algorithms interact freely with environment
- Can create its own training data set of actions, states and rewards
- Usually when environment is simulated as with games or simulations

- Offline RL is restricted to pre-recorded interactions
- Danger of bias as alternative states can not be explored
- Usually encountered in industrial and/or safety-conscious systems

Model-Free vs Model-Based Reinforcement Learning

- Model-Free RL interacts directly with environment
- Usually the case with games and simulations
- Allows direct generation of new transitions

Model-Free vs Model-Based Reinforcement Learning

- Model-Free RL interacts directly with environment
- Usually the case with games and simulations
- Allows direct generation of new transitions

- Model-Based learns additional model for agent to interact with
- Recorded Transitions $\rightarrow$ Model Optimization $\rightarrow$ Agent Optimization
- Agent is subsequently optimized on learned model
- Interesting combination with differentiable simulations

AlphaGo - Go

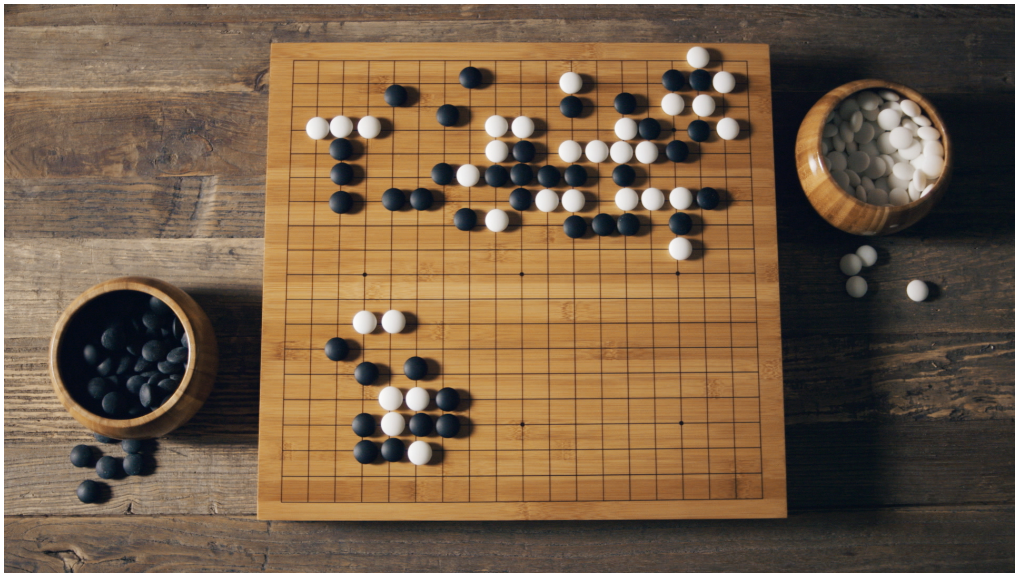
- Ancient board game from Asia on a 19×19 field
- More board configurations than atoms in the universe: $2 \cdot 10^{170}$

AlphaGo - Go

- Ancient board game from Asia on a 19×19 field
- More board configurations than atoms in the universe: $2 \cdot 10^{170}$
- Branching factor $b = 250$ and depth $d = 150$
- Chess: $b = 35$ and $d = 80$

AlphaGo - Go

- Ancient board game from Asia on a 19×19 field
- More board configurations than atoms in the universe: $2 \cdot 10^{170}$
- Branching factor $b = 250$ and depth $d = 150$
- Chess: $b = 35$ and $d = 80$
- Sheer number of positions makes brute force methods unrealistic



Monte Carlo Tree Search

- Go is a perfect information game
- All accesible information is available on the board
- Game tree of all possible board positions unfeasable
- MCTS approximates game tree with random play outs

AlphaGo - Architecture

- Components of AlphaGo
  - Supervised policy network $p_{SL}(a|s)$
  - Reinforced policy network $p_{RL}(a|s)$
  - Value network $v(s)$
  - Fast rollout policy network $p_{FR}(a|s)$

AlphaGo - Architecture

- Components of AlphaGo
  - Supervised policy network $p_{SL}(a|s)$
  - Reinforced policy network $p_{RL}(a|s)$
  - Value network $v(s)$
  - Fast rollout policy network $p_{FR}(a|s)$
- Intialization with supervised learning
- Refinement with reinforcement learning

AlphaGo - Architecture

- Components of AlphaGo
  - Supervised policy network $p_{SL}(a|s)$
  - Reinforced policy network $p_{RL}(a|s)$
  - Value network $v(s)$
  - Fast rollout policy network $p_{FR}(a|s)$
- Intialization with supervised learning
- Refinement with reinforcement learning

- Action from output probability $a_t \sim p(\cdot|s)$
- Value network architecture similar to policy network
- FC and single *tanh*-unit for scalar output
- Predict win $z_T = +1$ or loss $z_T = -1$

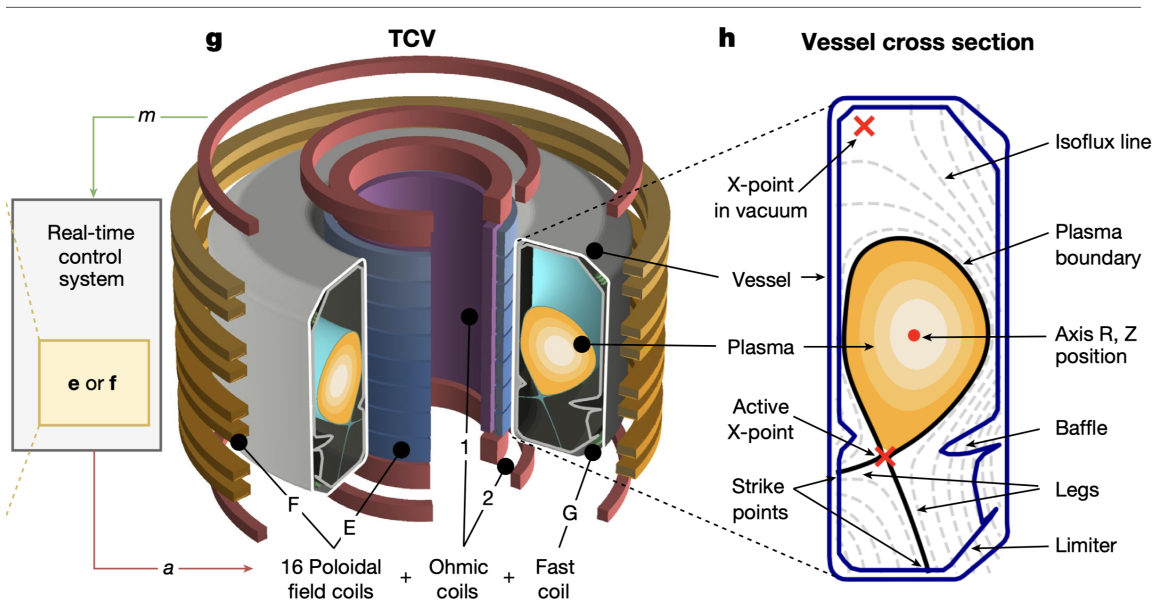
AlphaGo Zero - Motivation

- AlphaGo achieved super-human play strength
  - Supervised learning for pretraining
  - Reinforcement learning for policy network
  - Regression for value network
  - MCTS with policy/value network and rollout policy

AlphaGo Zero - Motivation

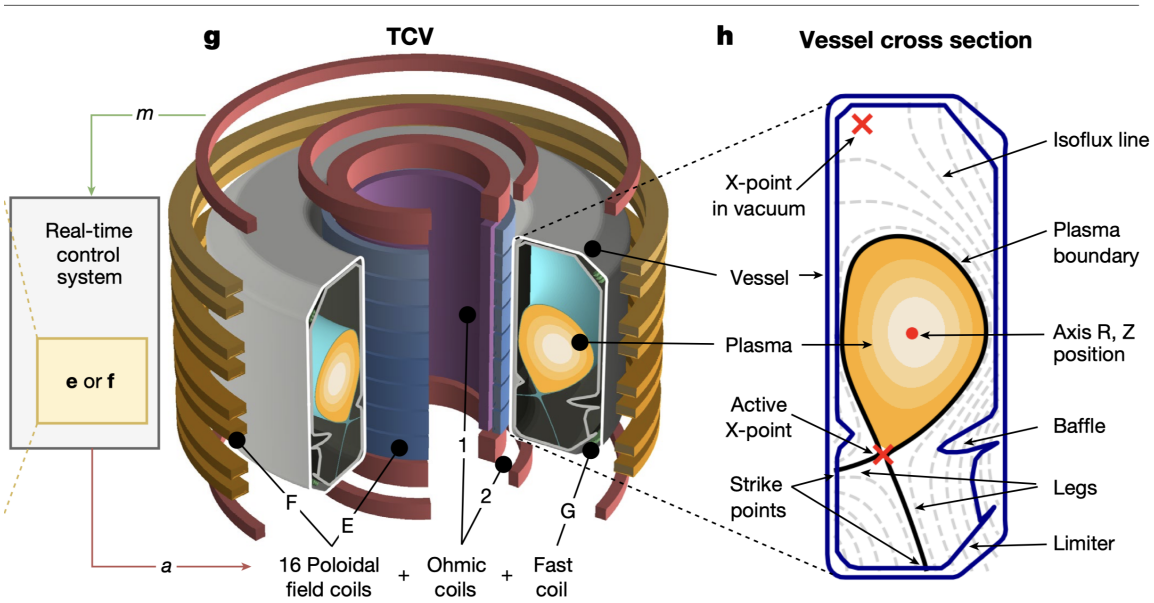
- AlphaGo achieved super-human play strength
  - Supervised learning for pretraining
  - Reinforcement learning for policy network
  - Regression for value network
  - MCTS with policy/value network and rollout policy
- AlphaGo Zero streamlined the learning process without human knowledge
  - Training only through reinforcement learning
  - Multi-headed, single network for policy and evaluation
  - Residual blocks
  - MCTS with single network

Plasma Control in Nuclear Fusion



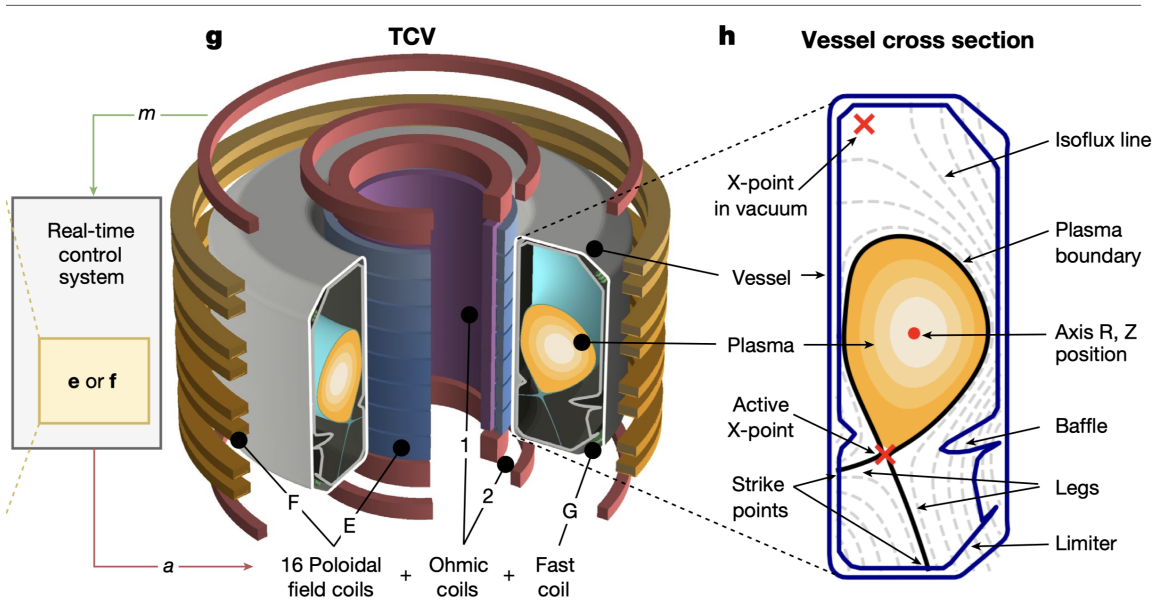
- Plasma in fusion reactors at 150 Mio C
- High energy makes control difficult due to sudden changes
- Simulator used to train Actor-Critic on optimal control of plasma

Plasma Control in Nuclear Fusion



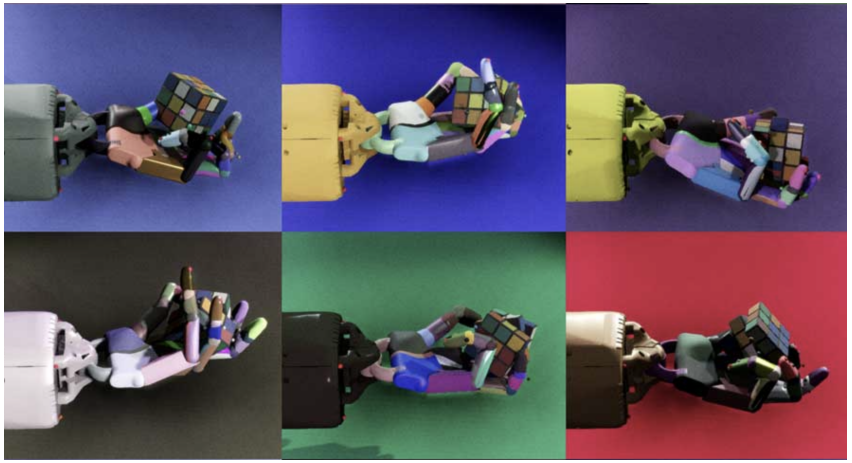
- Plasma in fusion reactors at 150 Mio C
- High energy makes control difficult due to sudden changes
- Simulator used to train Actor-Critic on optimal control of plasma
- Magnetic coil voltage is target control variable
- Complicated, fusion specific reward function

Plasma Control in Nuclear Fusion

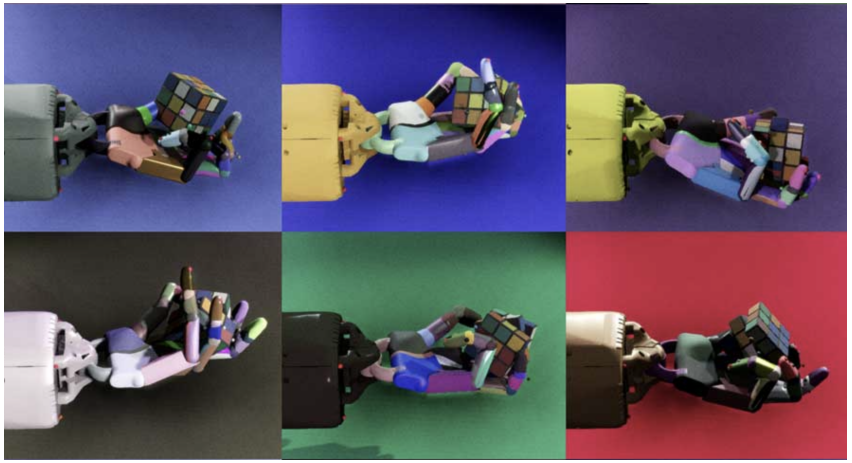


- Plasma in fusion reactors at 150 Mio C
- High energy makes control difficult due to sudden changes
- Simulator used to train Actor-Critic on optimal control of plasma
- Magnetic coil voltage is target control variable
- Complicated, fusion specific reward function
- High frequency control of magnetic coils in Tokamak
- 10 kHz control frequency limits actor to 4×256 MLP
- Larger LSTM critic to offset small actor

OpenAI Robot Control

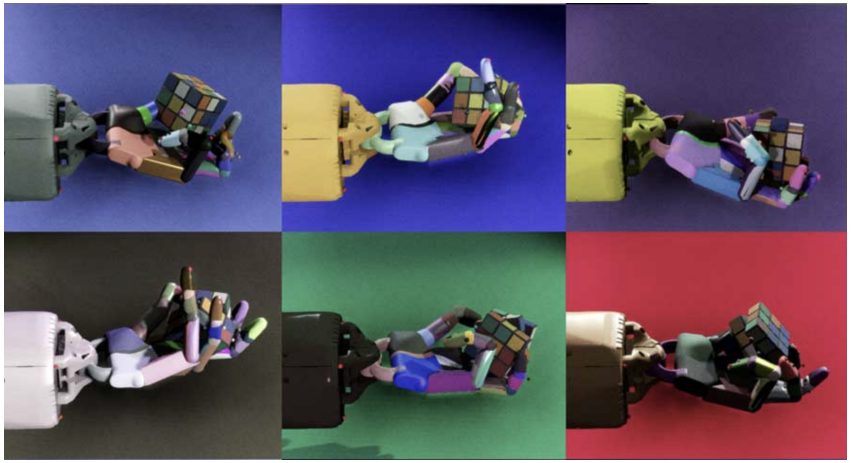


OpenAI Robot Control



- Closing simulation to reality gap
- Perception and dextreous manipulation

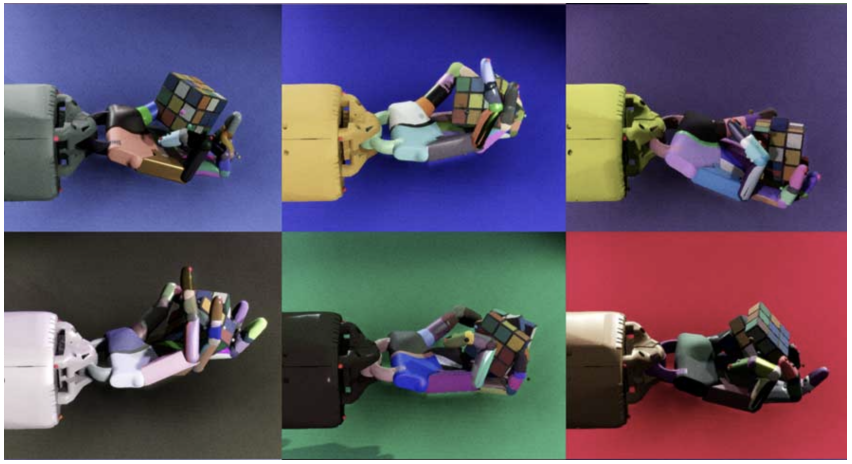
OpenAI Robot Control



- Closing simulation to reality gap
- Perception and dextreous manipulation
  - State estimation CNN
  - Optimal control algorithm to solve rubik cube in silico

- Train policy network to execute analytical solution with hand

OpenAI Robot Control



- Closing simulation to reality gap
- Perception and dextreous manipulation
  - State estimation CNN
  - Optimal control algorithm to solve rubik cube in silico

- Train policy network to execute analytical solution with hand
- Automatic Domain Randomization in simulation
  - size, friction, color, dexterity impairments, ...
- Difficulty/Variation of environment increased with success threshold
- 60% success rate