Reinforcement Learning
Policy Gradient, REINFORCE & Actor-Critic Methods

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

June 30, 2025

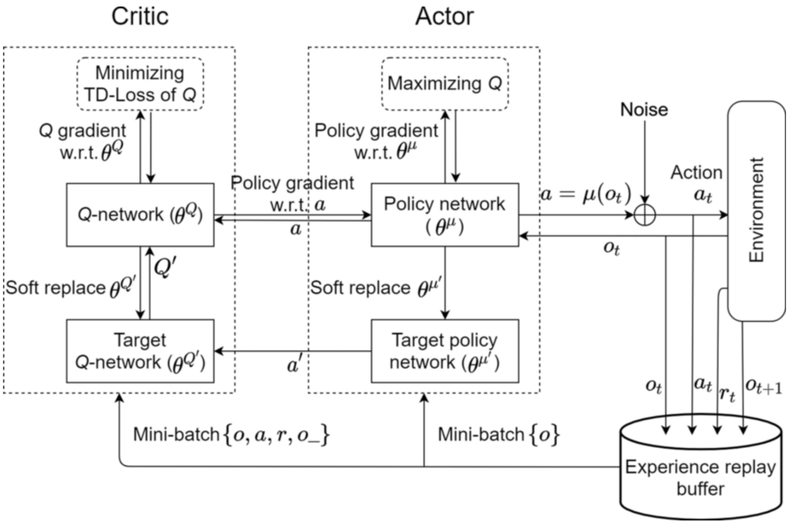

Critic
Actor
Minimizing TD-Loss of Q
Maximizing Q
Q gradient w.r.t. θ^Q
Policy gradient w.r.t. θ^μ
Noise
Environment
Action a_t
Q-network (θ^Q)
Policy gradient w.r.t. a
a
Policy network (θ^μ)
$a = \mu(o_t)$
o_t
Q'
Soft replace $\theta^{Q'}$
Soft replace $\theta^{\mu'}$
Target Q-network ($\theta^{Q'}$)
a'
Target policy network ($\theta^{\mu'}$)
o_t
a_t
r_t
o_{t+1}
Mini-batch $\{o, a, r, o\_\}$
Mini-batch $\{o\}$
Experience replay buffer

Table of Contents

1. Recap
2. Learning Outcomes
3. Motivation for Policy Gradient Methods
4. Policy Gradients
5. REINFORCE
6. Variance Reduction
7. Actor-Critic
8. REINFORCE in action: Recurrent Attention Model (RAM)
9. How to beat the Go world champion - AlphaGo
10. Value-Based and Policy-Based RL
11. Policy Gradient vs. Q-Learning
12. Summary
13. References

Recap

- A Markov Decision Process (MDP) defines the RL problem using:
  - States $\mathcal{S}$
  - Actions $\mathcal{A}$
  - Rewards $\mathcal{R}$
  - Transition probabilities $\mathbb{P}$
  - Discount factor γ
- Markov property: The future depends only on the present state, not the past.
- Agent and environment interact in a loop.
- Policy π decides the agent's actions.

Recap (cont.)

- Value function: Measures how good a state is.
- Q-value function: Measures how good a state-action pair is.
- Bellman equation: Recursively defines value and Q-value functions.
- Q-learning: Updates Q-values to reduce Bellman error.
- Deep Q-Learning: Uses neural networks to approximate Q-values.
- SARSA: On-policy version of Q-learning.

Learning Outcomes

By the end of this session, you will be able to:

- Understand the **limitations of value-based methods** like Q-learning.
- Formally define and derive the **Policy Gradient** objective.
- Implement and interpret the **REINFORCE algorithm**.
- Explain **Actor-Critic architectures** and their benefits.
- Evaluate policy-based methods in different environments.

Motivation for Policy Gradient Methods

- Value-based methods learn Q-values and derive the policy indirectly.
  - Inefficient in continuous or large action spaces
  - Can't represent stochastic policies
  - May lead to high variance and instability

Policy Gradient Methods:

- Learn policy parameters directly to maximize expected return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[R]$$

أكاديمية كاوست
KAUST ACADEMY

Reinforcement Learning: **Policy Gradients**

- **What is the problem with Q-learning?**
- The Q-function can be very complex.
- For example, a robot grasping an object may have a very high-dimensional state space. It can be difficult to learn the exact Q-value for every (state, action) pair.

- **What is the problem with Q-learning?**
- The Q-function can be very complex.
- For example, a robot grasping an object may have a very high-dimensional state space. It can be difficult to learn the exact Q-value for every (state, action) pair.
- However, the policy itself can be much simpler; for instance, just closing the robot's hand.
- Can we learn a policy directly, i.e., find the best policy from a set of possible policies?

Policy Gradients

- Formally, let us define a class of parameterized policies:

$$\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its expected return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

- Formally, let us define a class of parameterized policies:

$$\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its expected return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

- Our goal is to find the optimal policy: $\theta^\star = \arg\max_\theta \mathcal{J}(\theta)$
- How can we achieve this?

Policy Gradients

- Formally, let us define a class of parameterized policies:

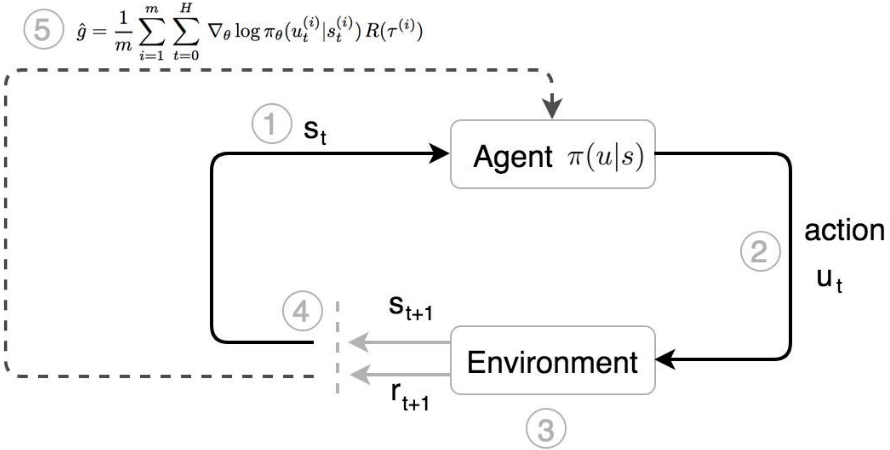
$$\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its expected return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

- Our goal is to find the optimal policy: $\theta^\star = \arg\max_\theta \mathcal{J}(\theta)$
- How can we achieve this?
- **Solution**: Perform gradient ascent on the policy parameters!

أكاديمية كاوست
KAUST ACADEMY



Reinforcement Learning: **REINFORCE**

REINFORCE

- REINFORCE is an elegant algorithm for maximizing the expected return.
- Intuition: trial and error.
- Sample a trajectory τ. If you get a high reward, try to make it more likely; if you get a low reward, try to make it less likely.
- A trajectory is a sequence of states, actions, and rewards: $\tau = (s_0, a_0, r_0, s_1, a_1, \cdots)$.

- Expected reward:

$$\begin{aligned}\mathcal{J}(\theta) &= \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] \\ &= \int_{\tau} r(\tau)p(\tau;\theta)d\tau\end{aligned}$$

REINFORCE

- Expected reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] = \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

- Now let's differentiate this:

$$\nabla_\theta \mathcal{J}(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

$$\text{where} \quad p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Expected reward:

$$\begin{aligned}\mathcal{J}(\theta) &= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\right] \\ &= \int_{\tau} r(\tau)p(\tau;\theta)d\tau\end{aligned}$$

- Now let's differentiate this:

$$\nabla_\theta \mathcal{J}(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

$$\text{where} \quad p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

- But this is **intractable!**

- However, we can use a useful trick:

$$\begin{aligned}\nabla_\theta p(\tau;\theta) &= p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\ &= p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)\end{aligned}$$

- However, we can use a useful trick:

$$\begin{aligned}\nabla_\theta p(\tau;\theta) &= p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\ &= p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)\end{aligned}$$

- Now, if we substitute this back:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &= \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta) d\tau \\ &= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]\end{aligned}$$

- We can estimate this with Monte Carlo sampling.

REINFORCE

- Recall,

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

REINFORCE

- Recall,

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus,

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

REINFORCE

- Recall,

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus,

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

- When differentiating:

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

REINFORCE

- Recall,

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus,

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

- When differentiating:

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- It doesn't depend on the transition probabilities!

- Therefore, when sampling a trajectory τ, we can estimate $\nabla_\theta \mathcal{J}(\theta)$ as:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- Therefore, when sampling a trajectory τ, we can estimate $\nabla_\theta \mathcal{J}(\theta)$ as:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- Interpretation:
  - If $r(\tau)$ is high, increase the probabilities of the actions taken.
  - If $r(\tau)$ is low, decrease the probabilities of the actions taken.

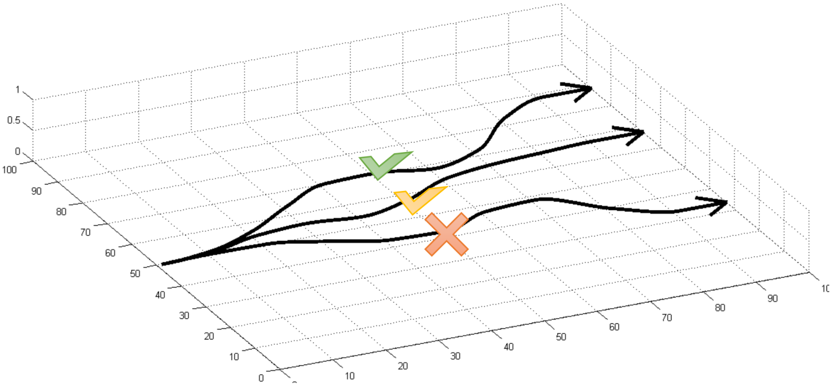
REINFORCE

- Therefore, when sampling a trajectory τ, we can estimate $\nabla_\theta \mathcal{J}(\theta)$ as:

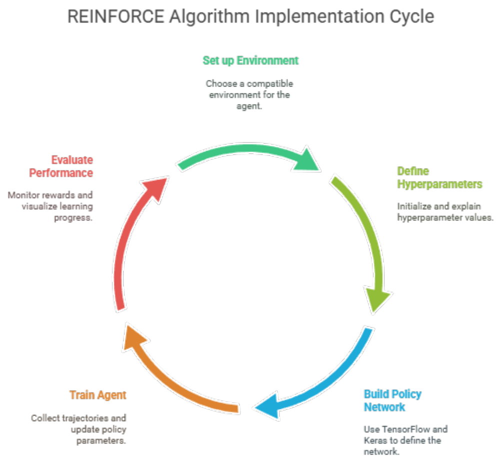
$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- Interpretation:
  - If $r(\tau)$ is high, increase the probabilities of the actions taken.
  - If $r(\tau)$ is low, decrease the probabilities of the actions taken.
- It might seem simplistic to say that if a trajectory is good, then all its actions were good. But in expectation, it averages out!

REINFORCE



REINFORCE Cycle



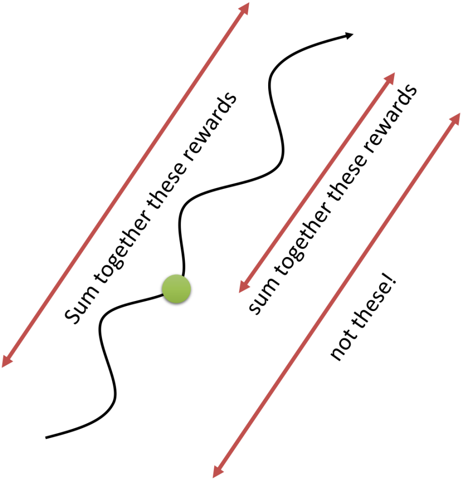
Reinforcement Learning: **Variance Reduction**

- However, there is a problem.
- This approach suffers from high variance because credit assignment is difficult.
- Can we help the estimator?

- However, there is a problem.
- This approach suffers from high variance because credit assignment is difficult.
- Can we help the estimator?
- **First idea:** Increase the probability of an action only by the cumulative future reward from that state:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance Reduction

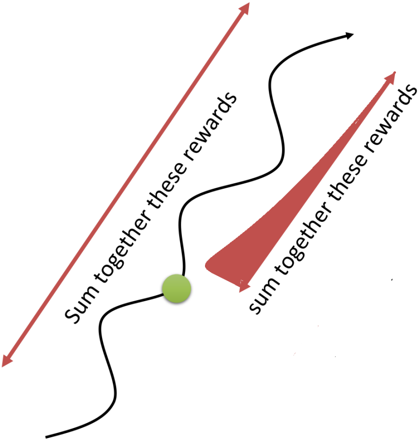


Variance Reduction

- But this still doesn't completely solve the credit assignment problem.
- It can lead to bias due to delayed rewards.

- But this still doesn't completely solve the credit assignment problem.
- It can lead to bias due to delayed rewards.
- **Second idea:** Use a discount factor γ to reduce the effect of delayed rewards:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$


Sum together these rewards
sum together these rewards

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if all rewards are positive, you keep increasing the probabilities of actions.

Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if all rewards are positive, you keep increasing the probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get.

Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if all rewards are positive, you keep increasing the probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get.
- **Idea:** Introduce a baseline function dependent on the state:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if all rewards are positive, you keep increasing the probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get.
- **Idea:** Introduce a baseline function dependent on the state:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- A simple baseline: the moving average of rewards experienced so far from all trajectories.

How to Choose the Baseline?

- Can we choose a better baseline?
- Essentially, we want to increase the probability of an action from a state if this action was better than the **expected value** from that state.

How to Choose the Baseline?

- Can we choose a better baseline?
- Essentially, we want to increase the probability of an action from a state if this action was better than the **expected value** from that state.
- What does this remind you of?

How to Choose the Baseline?

- Can we choose a better baseline?
- Essentially, we want to increase the probability of an action from a state if this action was better than the **expected value** from that state.
- What does this remind you of?
- **Answer:** Q-function and value function!

How to Choose the Baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. In contrast, we are unhappy if it is small.

How to Choose the Baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. In contrast, we are unhappy if it is small.
- The term $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called the **Advantage** and is denoted by $A^\pi(s_t, a_t)$.

How to Choose the Baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. In contrast, we are unhappy if it is small.
- The term $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called the **Advantage** and is denoted by $A^\pi(s_t, a_t)$.
- Using this, we get the estimator:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(Q^\pi(s_t, a_t) - V^\pi(s_t)\right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Reinforcement Learning: **Actor-Critic**

- We do not know the true Q and V functions. Can we learn them?

Actor-Critic

- We do not know the true Q and V functions. Can we learn them?
- Yes, by using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the value or Q-network).

Actor-Critic

- We do not know the true Q and V functions. Can we learn them?
- Yes, by using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the value or Q-network).
- The actor selects actions, while the critic evaluates how good the chosen actions are and provides feedback for improvement.
- The two networks adapt to each other, similar to the training process in GANs.

- We do not know the true Q and V functions. Can we learn them?
- Yes, by using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the value or Q-network).
- The actor selects actions, while the critic evaluates how good the chosen actions are and provides feedback for improvement.
- The two networks adapt to each other, similar to the training process in GANs.
- This approach simplifies the critic's task, as it only needs to estimate values for (state, action) pairs generated by the current policy.
- We can also incorporate Q-learning techniques, such as experience replay.

Actor-Critic

- We do not know the true Q and V functions. Can we learn them?
- Yes, by using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the value or Q-network).
- The actor selects actions, while the critic evaluates how good the chosen actions are and provides feedback for improvement.
- The two networks adapt to each other, similar to the training process in GANs.
- This approach simplifies the critic's task, as it only needs to estimate values for (state, action) pairs generated by the current policy.
- We can also incorporate Q-learning techniques, such as experience replay.
- **Remark:** The advantage function measures how much better an action was compared to the expected value.

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

End for

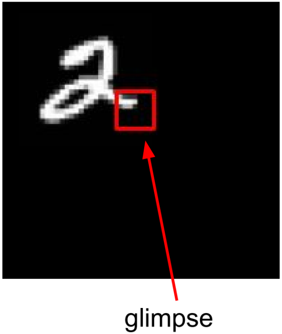
REINFORCE in Action: **Recurrent Attention Model (RAM)**

REINFORCE in Action: Recurrent Attention Model (RAM)

- **Objective:** Image classification
- The model takes a sequence of "glimpses," selectively focusing on regions of the image to predict the class.
  - Inspired by human perception and eye movements
  - Saves computational resources $\Rightarrow$ improves scalability
  - Can ignore clutter or irrelevant parts of the image
- **State:** Glimpses observed so far
- **Action:** (x, y) coordinates (center of the next glimpse) indicating where to look next in the image
- **Reward:** 1 at the final timestep if the image is correctly classified, 0 otherwise

[0] [Mnih et al., 2014]

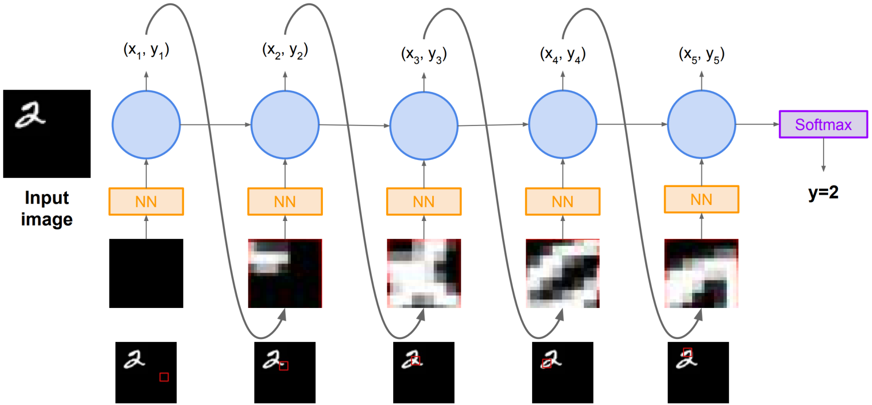
REINFORCE in Action: Recurrent Attention Model (RAM)



- Glimpsing is a non-differentiable operation.
- The policy for selecting glimpse locations is learned using REINFORCE.
- Given the sequence of glimpses observed so far, an RNN models the state and outputs the next action.

[0] [Mnih et al., 2014]

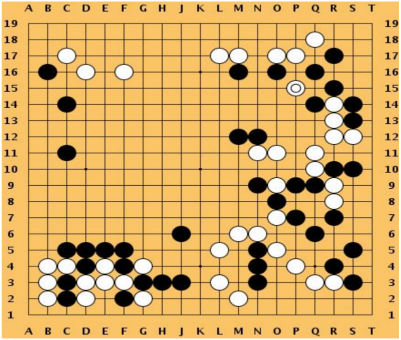
REINFORCE in Action: Recurrent Attention Model (RAM)



[0] [Mnih et al., 2014]

How to Beat the Go World Champion: **AlphaGo**

أكاديمية كاوست
KAUST ACADEMY



- Combination of supervised learning and reinforcement learning
- Integration of traditional methods (Monte Carlo Tree Search) with modern approaches (deep reinforcement learning)

[0] [Silver et al., Nature 2016]

How to Beat the Go World Champion - AlphaGo

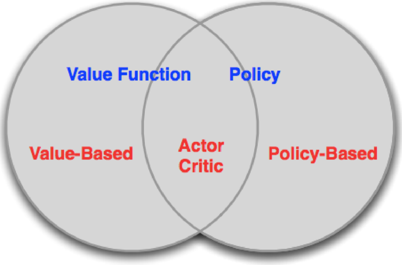
- Featurize the board (stone color, move legality, biases, etc.)
- Initialize the policy network with supervised training on professional Go games, then continue training using policy gradients (self-play from random previous iterations, with +1/-1 reward for winning/losing)
- Learn a value network (critic) to estimate the value of board positions
- Finally, combine the policy and value networks within a Monte Carlo Tree Search algorithm to select actions via lookahead search

[0] [Silver et al., Nature 2016]

Reinforcement Learning: **Value-Based and Policy-Based Methods**

Value-Based and Policy-Based RL

- **Value-Based Methods**
  - Learn a value function
  - Derive policy implicitly (e.g., ϵ-greedy)
- **Policy-Based Methods**
  - Do not learn a value function
  - Learn the policy directly
- **Actor-Critic Methods**
  - Learn both a value function and a policy



Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two fundamentally different representations: policies and value functions.
- Advantage of both methods: no need to model the environment.

Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two fundamentally different representations: policies and value functions.
- Advantage of both methods: no need to model the environment.
- **Policy Gradient: Pros and Cons**
  - **Pros:** Unbiased estimate of the gradient of expected return.
  - Can handle large action spaces (since only one action needs to be sampled).
  - **Cons:** High variance updates (leads to poor sample efficiency).
  - Does not perform credit assignment effectively.

Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two fundamentally different representations: policies and value functions.
- Advantage of both methods: no need to model the environment.
- **Policy Gradient: Pros and Cons**
  - **Pros:** Unbiased estimate of the gradient of expected return.
  - Can handle large action spaces (since only one action needs to be sampled).
  - **Cons:** High variance updates (leads to poor sample efficiency).
  - Does not perform credit assignment effectively.
- **Q-Learning: Pros and Cons**
  - **Pros:** Lower variance updates, more sample efficient.
  - Performs credit assignment.
  - **Cons:** Biased updates due to function approximation.
  - Difficult to handle large action spaces (since the maximum over actions must be computed).

Reinforcement Learning: **Summary**

Comparison of Methods

Method	Policy Type	Gradient Source	Stability	Efficiency
Q-learning	Deterministic	Value gradients	Medium	High
REINFORCE	Stochastic	Monte Carlo returns	Low	Low
Actor-Critic	Stochastic	TD-based advantage	High	High

Limitations

- **REINFORCE:**
  - High variance in gradient estimates.
  - Slow convergence.
- **Actor-Critic:**
  - Sensitive to hyperparameters.
  - Actor and critic updates may interfere.
  - Requires careful tuning and exploration strategies.
  - Can struggle in sparse-reward environments.

Future Directions

- **Trust Region Policy Optimization (TRPO):** Improves stability by constraining policy updates.
- **Proximal Policy Optimization (PPO):** Balances exploration and stability with clipped objective functions.
- **Soft Actor-Critic (SAC):** Uses entropy regularization for improved robustness and exploration.
- **Meta-Reinforcement Learning (Meta-RL):** Enables agents to adapt quickly to new tasks.
- **Multi-agent Actor-Critic:** Facilitates decentralized coordination among multiple agents.

Summary

- It can be hard to learn the exact Q-value for every (state, action) pair in high-dimensional state and action spaces.
- However, we can just learn a policy that maximizes the reward.
- We can use gradient ascent on policy parameters.
- However, this can suffer from high variance. Various strategies exist to tackle this.
- Actor-Critic methods combine Policy Gradients and Q-learning by training both an actor (the policy) and a critic (the Q-network).
- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust.

Reinforcement Learning: **References**

References

[1] Sutton, R. S., & Barto, A. G. (2018).

Reinforcement Learning: An Introduction.

[2] Williams, R. J. (1992).

Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.

Machine Learning Journal.

[3] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015).

Trust Region Policy Optimization.

In *ICML*.

References (cont.)

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017).
Proximal Policy Optimization Algorithms.
arXiv preprint arXiv:1707.06347.

[5] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018).
Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.
In *ICML*.

[6] OpenAI Spinning Up.
https://spinningup.openai.com

[7] Berkeley CS285 Deep RL.
https://rail.eecs.berkeley.edu/deeprlcourse/

[8] Chelsea Finn & Karol Hausman, Stanford CS224R: Deep Reinforcement Learning

[9] Fei-Fei Li, Yunzhu Li & Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision

[10] Jimmy Ba & Bo Wang, UofT CSC413/2516: Neural Networks and Deep Learning

[11] Sergey Levine, Berkeley CS285: Deep Reinforcement Learning

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.