Understanding GRPO

Ernesto Lupercio

May 6, 2025

What Is REINFORCE?

- REINFORCE is a way to teach a computer (or robot) to learn from rewards.
- It tries things, sees what works, and adjusts itself to do better over time.
- Like learning to play a game by trial and error.

How Does It Work?

- The computer sees a **state** s (like the game screen).
- It chooses an **action** a (like jump or move).
- It gets a **reward** R (like a score).
- It uses this info to adjust its brain θ.

What Is a Policy?

- A **policy** tells the computer what to do in each situation.
- Written as $\pi_\theta(a \mid s)$: the chance of taking action a in state s.
- θ: the computer's internal settings (like its brain weights).
- Changing θ changes how it plays.

Learning Rule (Simplified)

- After each move, it adjusts its brain based on how good the result was:

$$\theta \leftarrow \theta + \text{learning rate} \times (\text{action score})$$

- Good actions $\Rightarrow$ increase chance
- Bad actions $\Rightarrow$ decrease chance

Mathematical Form (Optional)

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot R_t$$

- α: learning rate
- ∇_θ: change in settings
- $\log \pi_\theta(a_t \mid s_t)$: how confident it was
- R_t: the reward it got

Why It's Called REINFORCE

- Like a coach saying "Great move! Do that again."
- Or "That was bad. Let's change the plan."
- It reinforces good actions with higher probability.

Everyday Analogy

- Imagine throwing darts at a target.
- You get points based on where you hit.
- You adjust your next throw based on the last result.
- Over time, you learn to aim better.

Summary

- REINFORCE teaches by trying and adjusting.
- It uses feedback (rewards) to get better.
- It increases good decisions and decreases bad ones.
- It's the starting point of many powerful AI systems today.

REINFORCE: Key Formula

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) R_t]$$

- θ: parameters of the policy (like the robot's brain settings)
- $\pi_\theta(a_t|s_t)$: probability of taking action a_t in state s_t
- R_t: total reward after time t
- $\nabla_\theta \log \pi_\theta(a_t|s_t)$: how to change the policy

What Does It Mean?

- Increase the chance of actions that gave good rewards.
- Decrease the chance of actions that led to bad outcomes.
- Learn by trial and error using total reward.

Symbols in Context

- **State (s_t)**: What the agent sees at time t
- **Action (a_t)**: What the agent decides to do
- **Reward (R_t)**: What it gains (or loses) afterward
- **Policy (π_θ)**: Rule the agent follows to choose actions
- **Gradient (∇_θ)**: Tells how to adjust the brain

Pseudocode for REINFORCE

```
Initialize policy parameters theta
Repeat:
Generate an episode: (s_0, a_0, r_1, ..., s_T)
For each time step t in episode:
Compute return R_t = sum of future rewards
Compute gradient: g = grad_theta log(pi_theta(a_t | s_t)) * R_t
Update theta using gradient ascent
Until convergence
```

Actor-Critic

- Teach a robot (or computer) to make good decisions.
- Use rewards to learn what works well.
- Actor-Critic is a smarter way to learn than REINFORCE.

Two Roles: Actor and Critic

- **Actor**: decides what to do (takes actions).
- **Critic**: gives feedback (estimates how good the situation is).
- They work together to learn faster and better.

Key Idea

- Actor learns the **policy** $\pi_\theta(a|s)$: what to do in state s.
- Critic learns the **value** $V_w(s)$: how good is state s?
- Together, they adjust based on feedback.

Mathematical Pieces

- Actor uses the policy gradient:

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) \cdot A(s, a)$$

- Critic estimates the advantage:

$$A(s, a) = r + \gamma V(s') - V(s)$$

- γ: discount factor (between 0 and 1)

How the Critic Helps

- Instead of waiting until the end of the game (like REINFORCE),
- The critic gives feedback immediately using its value estimate.
- This helps the actor make better choices faster.

Learning Together

1. Actor takes action a in state s.
2. Environment gives reward r and next state s'.
3. Critic updates $V(s)$ using:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

4. Actor updates policy based on advantage.

Why Is This Better Than REINFORCE?

- Less randomness (variance)
- Learns from every step, not just full games
- Actor doesn't need to guess — critic helps guide it

Everyday Analogy

- Actor = student trying to answer questions
- Critic = teacher giving hints and feedback
- Together, they help the student (robot) learn faster

Conclusion

- Actor-Critic uses two brains: one to act, one to evaluate.
- It builds on REINFORCE but is more efficient.
- A powerful idea used in many modern AI systems.

TRPO (Trust Region Policy Optimization)

- Imagine training a robot that suddenly forgets everything after one mistake.
- Regular policy gradient can make big, unsafe changes.
- We need a way to say: "Don't change the brain too much in one go."
- That's what TRPO does: it keeps updates **safe and small**.

The TRPO Idea

- TRPO updates the policy, but only if it doesn't stray too far.
- Think of it like driving: you can steer, but not jerk the wheel.
- Mathematically, it uses a constraint:

$$\mathsf{KL}(\pi_{\mathsf{old}} \| \pi_{\mathsf{new}}) \leq \delta$$

- KL divergence measures how different the two policies are.

The TRPO Formula

- TRPO maximizes a **surrogate reward function**:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A(s,a)\right]$$

- This compares the new and old policies.
- But **only allows it** if the KL difference is small.

Why This Matters

- It prevents the robot from making dangerous changes.
- Every update is like a gentle nudge, not a hard shove.
- Result: safer learning, more stable improvement.

Why PPO Was Invented

- TRPO works well, but is complicated to implement.
- PPO keeps the same goal: avoid dangerous updates.
- But it uses a much simpler trick: **clipping**.
- PPO is like saying: "Let the robot learn, but not too fast."

The PPO Idea

- Instead of hard rules (like KL $\leq \delta$), PPO softens it.
- It compares new vs. old action probabilities:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

- If the change is too big, we **clip** it to keep it safe.

The PPO Objective

- PPO tries to maximize:

$$\min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right)$$

- This means:
  - If the update is small, let it happen.
  - If it's too big, clip it so it doesn't go wild.
- This makes PPO **robust and stable**.

Understanding the clip() Function

- PPO uses a **clipping function** to avoid large policy updates.
- Mathematically:

$$\text{clip}(r, 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon & \text{if } r < 1-\epsilon \\ r & \text{if } 1-\epsilon \leq r \leq 1+\epsilon \\ 1+\epsilon & \text{if } r > 1+\epsilon \end{cases}$$

- In words: "Don't let the ratio r go too far from 1."
- Keeps the policy update from being too aggressive.
- Helps maintain stability and trust in learning.

PPO Symbols Explained

- π_θ: policy with parameters θ
- a_t: action taken at time t
- s_t: state observed at time t
- A_t: advantage at time t, how much better the action was than average
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$: how much the new policy favors this action
- ϵ: clip range parameter, e.g., 0.2

Pseudocode for PPO

```
Initialize policy parameters theta and value function parameters
for iteration in range(N):
Collect trajectories using current policy pi_theta
Estimate advantage A_t and returns
for epoch in range(K):
for minibatch in trajectories:
r_t = pi_theta(a_t|s_t) / pi_theta_old(a_t|s_t)
clipped = clip(r_t, 1 - epsilon, 1 + epsilon)
loss = -min(r_t * A_t, clipped * A_t)
Update policy parameters using gradient of loss
Update value function to fit returns
```

Why PPO Works Well

- Easier to implement than TRPO.
- Works with simple gradient descent (no fancy math).
- Learns safely and quickly.
- PPO is used in real-world tasks — from games to finance.

History and Impact of PPO

- **Proposed by OpenAI in 2017**. Aimed to simplify and stabilize policy optimization.
- Built as a lighter alternative to TRPO (Trust Region Policy Optimization).
- Became widely used for its balance of performance and ease of implementation.
- Published in: *"Proximal Policy Optimization Algorithms"*, Schulman et al., 2017.

Spectacular Applications of PPO

- **OpenAI Five** — trained agents to play Dota 2 at a superhuman level.
- **Robotic control** — simulated and real robots for walking, grasping, balancing.
- **Autonomous vehicles** — learning lane-keeping and adaptive driving policies.
- **Finance** — portfolio management, algorithmic trading strategies.
- **Games** — used in Unity ML-Agents Toolkit to train AI in video games.

Group Relative Policy Optimization (GRPO): Origins and Development

- **Introduction of GRPO**: Introduced by DeepSeek in their 2024 paper, *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models* [**?**].
- **Motivation**: Designed to enhance mathematical reasoning in large language models (LLMs) by simplifying the reinforcement learning (RL) process.
- **Key Innovation**: Eliminates the need for a separate value function (critic) by estimating baselines from group scores, reducing computational overhead.
- **Implementation**: First applied in training DeepSeek-R1 and DeepSeek-R1-Zero models, demonstrating improved reasoning capabilities without extensive supervised fine-tuning.

DeepSeek-R1-Zero: Advancing LLM Reasoning with GRPO

- **Model Overview**: DeepSeek-R1-Zero is a large language model trained exclusively using GRPO, without supervised fine-tuning, achieving strong performance in reasoning tasks.
- **Training Efficiency**: Leveraged GRPO to reduce the need for human-annotated data and extensive computational resources, making training more cost-effective.
- **Performance**: Demonstrated competitive results on benchmarks like the American Invitational Mathematics Examination (AIME) and MATH, showcasing GRPO's effectiveness in enhancing LLM reasoning abilities.
- **Impact**: DeepSeek-R1-Zero's success with GRPO has influenced the development of subsequent models and training methodologies in the field of AI.

GRPO: Learning Without a Teacher

Imagine you're teaching a robot to solve math problems.

- You give the robot a math question.
- It tries out k different answers: $a_1, a_2, \ldots, a_k$.
- But there's no teacher around to say which is correct!
- So the robot compares the answers against each other.
- The ones that look better get rewarded, the rest don't.

How Does GRPO Know Which Answer Is Better?

- Even without a teacher, GRPO can compare answers by scoring them using a **reward model**.
- Think of it as an internal judge trained to recognize good answers.
- For example, it might score answers based on:
  - Correctness clues (e.g., algebraic validity)
  - Completeness and clarity
  - Alignment with the question asked
- Then it ranks all answers $a_1, \ldots, a_k$ by these scores.
- The best answers (relative to the rest) get positive rewards $R_i > \bar{R}$.
- GRPO uses these scores to update the robot's brain — favoring better answers.

GRPO: The Update Rule

The robot learns by making better answers more likely.

- Each answer gets a *relative score* — how good it was compared to the rest.
- Define:

$$\theta \leftarrow \theta + \alpha \sum_{i=1}^{k} \nabla_\theta \log \pi_\theta(a_i \mid s) \cdot (R_i - \bar{R})$$

- Where:
  - θ: the robot's brain (model weights)
  - α: learning rate (how fast it learns)
  - $\pi_\theta(a_i \mid s)$: the probability it picked answer a_i
  - R_i: reward for answer a_i
  - $\bar{R}$: average reward of all answers

GRPO: All Symbols Explained

- s: the question or input the robot sees
- a_i: the i-th answer the robot gives
- $\pi_\theta(a_i \mid s)$: the probability assigned to that answer
- R_i: a reward for how good that answer was (relative to others)
- $\bar{R} = \frac{1}{k}\sum_{j=1}^{k} R_j$: the average reward over all answers
- $\nabla_\theta \log \pi_\theta(a_i \mid s)$: tells how to increase or decrease the chance of this answer
- α: how big of a change to make (learning rate)

GRPO Pseudocode

```
for each question s:
Generate k answers a_1, ..., a_k from current policy pi_theta
Score each answer using a reward model: R_1, ..., R_k
Compute average reward: R_bar = average(R_i)
For each i = 1 to k:
Compute log_prob = log pi_theta(a_i | s)
Compute gradient = (R_i - R_bar) * grad_theta log_prob
Accumulate gradient
Update theta using accumulated gradient
```

GRPO: Why It Works

- If an answer is better than average: $R_i > \bar{R} \Rightarrow$ increase chance of that answer.
- If an answer is worse: decrease its chance.
- No need for labeled answers — the robot self-improves.
- It's like playing chess with yourself and always favoring better moves.

DeepSeek and GRPO in Practice

- DeepSeek used GRPO to train a math-solving robot called **DeepSeek-R1-Zero**.
- It didn't get supervised feedback — just learned from its own output quality.
- GRPO made training faster and cheaper — no human labels needed.
- It ended up solving hard math problems like those in international competitions!

Why GRPO is Awesome

- **Self-taught learning**: the robot becomes smarter by comparing its own attempts.
- **No teachers needed**: works even when no labeled data is available.
- **Efficient and powerful**: used in cutting-edge AI research.
- **Good for logic, math, puzzles, and reasoning tasks**.