Study Mode - Reinforce Algorithm in RL

In Reinforcement Learning, what is Reinforce Algorithm in Policy Gradient Based Algorithm?

Before we jump in, let me check one quick thing so I don't overshoot or undershoot:

What's your current level with RL?
For example:

- Have you learned MDPs, value functions, and basic policy gradients already?
- Or would you prefer starting Reinforce from the basics?

(Answer in one line.)

I know the basics ie MDPs, value function, Q learning, Deep Q Learning etc

Great — then let's build REINFORCE on top of what you already know, step-by-step, with guiding questions so *you* make the key connections.

🌱 Step 1: Connect REINFORCE to things you already know

You already know that **in value-based RL**, we try to learn $Q(s, a)$ or $V(s)$ and act by greedily selecting the best action.

Policy gradients flip this:
We **directly optimize the policy** parameters θ by maximizing expected return.

A quick check for you:

❓ Question 1

If our goal is to maximize the expected return $J(\theta)$, what would be the most natural thing to do with respect to θ?
(Think in terms of optimization.)

Just answer in a short phrase.

what do you mean by natural thing? please elaborate on it

Sure — let me make it clearer without giving away the next step.

When you have an objective function in ML (like loss in supervised learning), what do you *usually* do to the parameters?

For example:

- In supervised learning, you have loss $L(\theta)$.
  What operation do you normally apply to θ so the loss decreases?

That same "natural" operation will apply to RL's objective $J(\theta)$.

❓ So here's the revised question

In general ML optimization, how do you adjust parameters to improve an objective?
(One short phrase is enough.)

you mean gradient decent?

Exactly — that's the connection I wanted you to make.
So now we can build REINFORCE cleanly from that idea.

✅ Step 2: From gradient descent → policy gradient

If we want to *maximize* expected return $J(\theta)$, we do:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

So the whole challenge becomes:

How do we estimate $\nabla_\theta J(\theta)$ from experience?

That's where REINFORCE comes in.

🌟 Step 3: What is REINFORCE? (High-level)

Let me guide you to the core idea.

REINFORCE is the **Monte-Carlo Policy Gradient** method that:

- runs full episodes,
- collects returns,
- uses those returns to push up the log-probability of the actions that led to good outcomes,
- and push down those that led to poor outcomes.

It uses the famous formula:

$$\nabla_\theta J(\theta) \approx G_t \, \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Before I expand more, let me check your understanding:

❓ Question 2

Why do you think we use $\log \pi_\theta(a|s)$ instead of $\pi_\theta(a|s)$ directly in the gradient?
(Just give your intuition — even a guess is fine.)

I think the raw policy πθ(aIs) outputs a probability value which is not differentiable and hence we transform it to logprob to make it differentiable

Nice intuition — you're thinking in the right direction. Let me sharpen it a bit so it sticks clearly.

Your idea:

> π(a|s) is a probability, so maybe log makes it differentiable.

It's *actually differentiable already*, but the **log makes the gradient behave in a much more stable way**.

Here's the clean intuition:

✅ Why REINFORCE uses log π(a|s)

Think of π(a|s) as a number between 0 and 1.

If you try to differentiate π(a|s) directly:

- when the probability is small, the gradient becomes tiny
- when the probability is large, gradients can saturate
- it becomes very hard to *scale* the update properly

But if you use:

$$\nabla_\theta \log \pi_\theta(a|s)$$

you get two huge benefits:

1️⃣ Log turns products into sums

Episode probabilities are products:

$$P(\text{trajectory}) = \prod_t \pi(a_t|s_t)$$

Taking log turns them into sums:

$$\log P = \sum_t \log \pi(a_t|s_t)$$

Sums are far easier to differentiate and numerically stable → better gradients.

2️⃣ Log gives the "score function" trick

There is a key identity:

$$\nabla_\theta \log \pi(a|s) = \frac{\nabla_\theta \pi(a|s)}{\pi(a|s)}$$

This ratio creates a strong *sensitivity signal*:

If π(a|s) is small, the gradient becomes **big** → encourages exploration but with direction.

If π(a|s) is big, the gradient becomes **small** → prevents overly aggressive shifts.

3️⃣ It naturally fits probability distributions

If policy is a softmax network, the log-prob nicely cancels terms in gradients, simplifying updates.

Now let's move forward to the central REINFORCE update.

❓ Question 3

REINFORCE multiplies the log-prob gradient with **return** G_t:

$$G_t \, \nabla_\theta \log \pi(a_t|s_t)$$

Why do you think the *return* is used as the “weight” for the gradient?

(Just give your intuition — what does multiplying by G_t achieve?)

because the entire goal of running this optimization is to maximize the expected cumilitive reward?

Exactly — that’s the heart of it.
Let me phrase it in a way that connects directly to how REINFORCE works.

🎯 Why multiply the gradient by return G_t?

Your intuition:

> We want to maximize cumulative reward.

Perfect. Now connect it to the actual mechanism:

If an action leads to high return,

we **increase** its probability:

$$\theta \leftarrow \theta + \alpha \, G_t \nabla_\theta \log \pi(a_t|s_t)$$

If an action leads to low (or negative) return,

we **decrease** its probability:

$$\theta \leftarrow \theta - \alpha \, |G_t| \nabla_\theta \log \pi(a_t|s_t)$$

So G_t acts like a "critic" that says:

> “This action turned out well, do more of it.”
> “This action turned out badly, do less of it.”

That’s why REINFORCE is also called the **Monte Carlo Policy Gradient** — it uses full episode returns directly.

🧩 Now, let's assemble the full algorithm

Here's REINFORCE in words:

1. Run an episode using current policy π_θ.
2. Compute return G_t at each time-step.
3. Update parameters:

$$\theta \leftarrow \theta + \alpha \sum_t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

No value function.

No bootstrapping.

Just pure Monte Carlo estimation.

To solidify, let me ask one last quick check:

❓ Question 4 — Key Concept Check

REINFORCE is known to have **high variance**.

Based on the formula, where do you think that variance comes from?

(Think about what MC returns look like.)

May be from rewards?

You're on the right track — let's make that intuition sharper.

🎯 Why REINFORCE has high variance

Your answer:

> maybe from rewards?

Yes — but specifically **because the returns are full Monte-Carlo returns**.

Meaning:

- G_t depends on *all future rewards* in that episode.

- Episodes can vary wildly depending on randomness, exploration, environment dynamics.
- So the return for the *same* state–action pair might be huge in one episode and terrible in another.

This makes the gradient update:

$$G_t \nabla_\theta \log \pi(a_t|s_t)$$

swing around a lot.

When gradient noise is large → learning becomes unstable and slow.
This is exactly why **baselines**, **advantage functions**, or **Actor–Critic methods** were invented — to reduce that variance.

Quick recap (so it sticks)

Try to summarize the core idea of REINFORCE in **one or two sentences**.
This will help me check if the concept is truly locked in.

> In deep learning the learnable params are weights and biases, so here this theta which we are updating this is basically weight right?

Good question — and yes, you're very close.
Let's make the connection precise.

🎯 What is θ in policy gradient / REINFORCE?

Think of **θ as *all* the learnable parameters of the policy network**.

In a deep learning model, that includes:

- weights of all layers
- biases of all layers
- any parameters inside softmax layers
- any parameters of feature extractors (if end-to-end)

So yes — θ includes what you'd normally call:

- **weights**
- **biases**

and in general **every differentiable parameter of the policy**.

A simple check for you:

If we built a policy as:

```python
pi = nn.Sequential(
    nn.Linear(4, 128),
    nn.ReLU(),
    nn.Linear(128, 2),
    nn.Softmax(dim=-1)
)
```

Then **θ** includes:

- 4×128 weight matrix + 128 bias
- 128×2 weight matrix + 2 bias
- (softmax has no extra parameters)

So:

> θ = all weights + all biases of the policy network.

Quick question for you (to reinforce the idea):

If tomorrow we replace the policy with a **Transformer-based** model,
what would θ represent there?

(Just answer in a short phrase.)

weights only. weights of different embeddings, FCs etc

Perfect — that's exactly the right understanding.

Whether it's:

- a simple MLP
- a CNN policy
- an LSTM / GRU policy
- or a big Transformer

θ = all learnable parameters

(weights *and* biases, though in practice people often just say "weights" informally).

Quick reinforcement check

To see if everything is stitched together, try answering this:

❓ If REINFORCE computes a gradient

$$G_t \, \nabla_\theta \log \pi_\theta(a_t|s_t)$$

what does that gradient *actually* update inside a neural network policy?

(Just describe it in one short line — no equations.)

weights

Exactly — and more precisely:

> It updates **all** learnable parameters (weights + biases) in the policy network in the direction that makes the taken actions more or less likely, proportional to the returns.

That's the entire "learning" part of REINFORCE.

Quick final consolidation

To be sure you have the whole idea clean:

❓ One last small prompt

If I give you a neural network policy, and I say
"apply REINFORCE to update it,"
can you describe **in your own words** the 3–4 steps you'd perform?

Keep it brief — 2–3 lines is enough.