# Search gradients and policy gradients: Evolution and REINFORCE

Janik Euskirchen

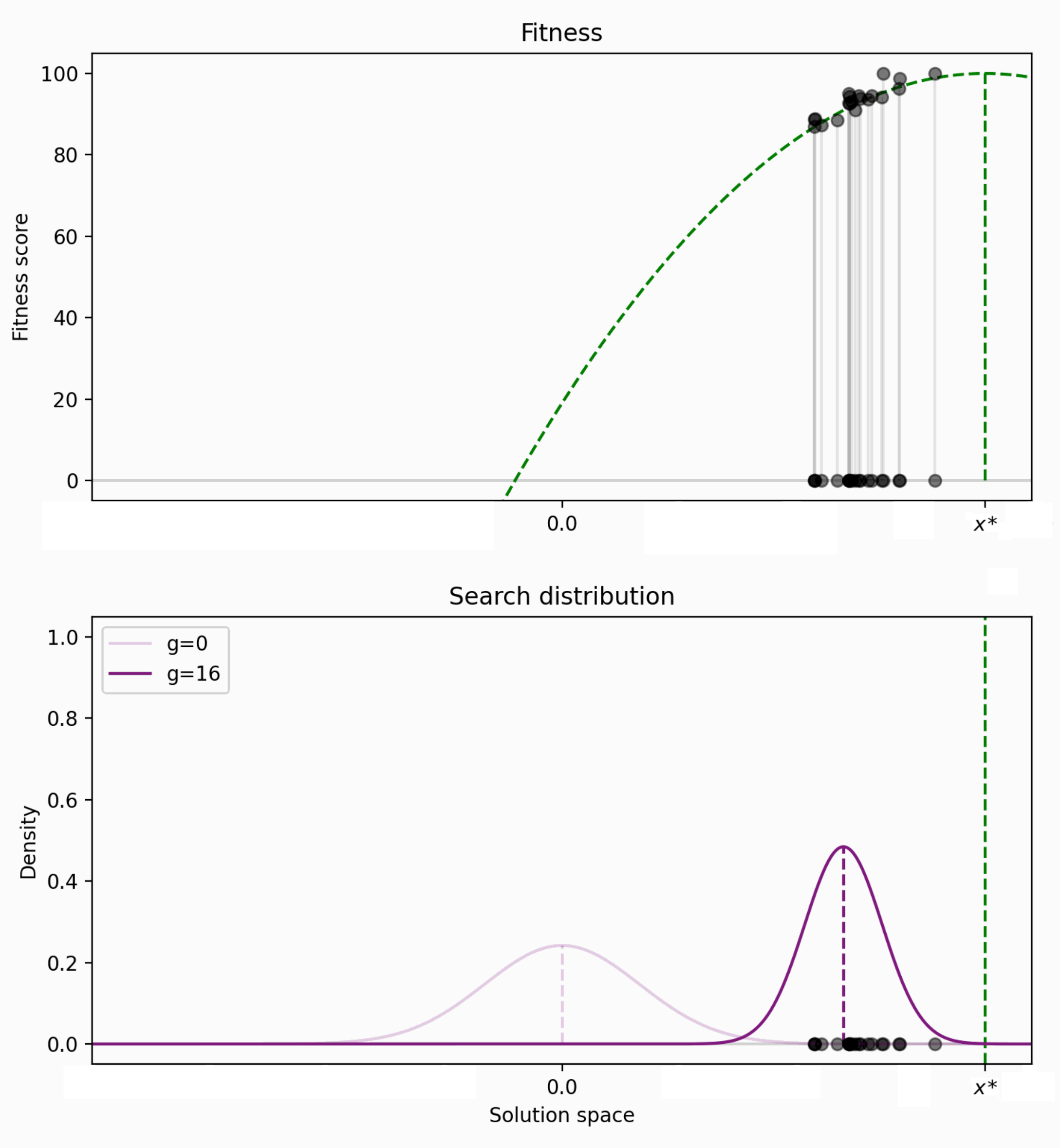

Figure 1: Finding a scalar value that maximizes an unknown, black-box fitness function (visualized as the dashed, green parabola (top)), using the search gradient algorithm. In this example, the fitness evaluation returns a noisy fitness score of the true fitness function. Notice in the top how the fitness values of the samples hint at a gradient towards the right.

# 1. Log-derivative trick

From calculus we get the log-derivative rules:

$$\frac{d}{dx}\log x = \frac{1}{x} \tag{1}$$

$$\frac{d}{dx}\log y = \frac{1}{y}\cdot\frac{dy}{dx} \tag{2}$$

$$= \frac{y'}{y} \tag{3}$$

where in Eq. (2) we simply make use of Eq. (1) and the chain rule.

This means we can rewrite a partial derivative in the following way, which might come in handy, whenever it is easier to compute the gradient of a log-probability than of the probability itself. This is generally the case when the probability is a product, turning the log-probability into a sum.

$$\begin{aligned}\frac{\partial}{\partial\theta}p(x;\theta) &= \frac{p(x;\theta)}{p(x;\theta)}\cdot\frac{\partial}{\partial\theta}p(x;\theta)\\ &= p(x;\theta)\cdot\frac{\frac{\partial}{\partial\theta}p(x;\theta)}{p(x;\theta)}\\ &\overset{(2)}{=} p(x;\theta)\cdot\frac{\partial}{\partial\theta}\log p(x;\theta)\end{aligned}$$

$\frac{\partial}{\partial\theta}\log p(x;\theta)$ is called the **score function**.

## 2. Maximum expected fitness

Assuming we sample a population of candidate solutions $x \sim p(x;\theta)$, then we call $p(x;\theta)$ the **search distribution**, parameterized by $\theta$. We want to find the parameters, $\theta$, that maximize the expected fitness over solutions. If we can compute the gradient of the expected fitness with respect to the parameters, we can perform gradient ascent on it.

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{x\sim p(x;\theta)}\left[f(x)\right]}_{J(\theta)}$$

$$\mathbb{E}_{x\sim p(x;\theta)}\left[f(x)\right] = \int p(x;\theta)\left[f(x)\right]dx$$

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}\left[f(x)\right] = \nabla_\theta \int p(x;\theta)\left[f(x)\right]dx$$

Notice how only $p(x;\theta)$ depends on $\theta$ :

$$= \int f(x)\ \nabla_\theta\, p(x;\theta)\ dx$$

Now we use the log-derivative trick from above:

$$= \int f(x)\ p(x;\theta)\ \nabla_\theta \log p(x;\theta)\ dx$$

We can regroup this to have a term that's weighted by the probability:

$$= \int p(x;\theta)\ \ \left[f(x)\cdot\nabla_\theta \log p(x;\theta)\right]\ dx$$

And this is simply an expectation, so let's write:

$$= \mathbb{E}_{x\sim p(x;\theta)}\left[f(x)\cdot\nabla_\theta \log p(x;\theta)\right]$$

We can estimate this by drawing samples and taking the average:

$$\approx \frac{1}{\lambda}\sum_{k=1}^{\lambda}\left[f(x^{(k)})\cdot\nabla_\theta \log p(x^{(k)};\theta)\right]$$

From this, we can define a simple gradient ascent algorithm called **Search Gradient algorithm** (in reference to the Policy Gradient algorithm).

**Algorithm 1** : Search Gradient algorithm

```
Require: initial parameters $\boldsymbol{\theta}$, fitness function $f$, number of offspring $\lambda$, step size $\alpha$
while not converged do
    for each $k \in \{1, \ldots, \lambda\}$ do
        $\boldsymbol{x}^{(k)} \sim p(\cdot\,;\boldsymbol{\theta})$                              ▷ "Mutation"
        $f^{(k)} \leftarrow f(\boldsymbol{x}^{(k)})$                                   ▷ Evaluation
        $\boldsymbol{g}^{(k)} \leftarrow \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}^{(k)}\,;\boldsymbol{\theta})$          ▷ Log-derivative of search distribution
    $\nabla_{\boldsymbol{\theta}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} f^{(k)} \cdot \boldsymbol{g}^{(k)}$                  ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$                          ▷ "Selection"
return $\boldsymbol{\theta}$                                             ▷ or best sample from final generation
```

## 3. Equivalence of Search Gradient and Policy Gradient

It turns out that this is exactly what Episodic REINFORCE is. And even more generally, the fundamental idea behind the Search Gradient method and Policy Gradient method are equivalent. The Policy Gradients method is simply the application to Markov Decision Processes resulting in some MDP-specific variants. Arguably, REINFORCE (Monte Carlo Policy Gradients) sits right on the border between Search Gradients and Policy Gradients, as you barely require any MDP formalism.

When we try to associate the terminologies, it turns out that trajectories (episodes) in REINFORCE are "solutions" in the Search-Gradient sense. The return (sum of discounted rewards) of a trajectory in REINFORCE is the fitness of a solution. Furthermore, in Episodic REINFORCE, we update the search distribution's parameters, $\theta$, after every episode, which implies a population size $\lambda = 1$.

Let's derive REINFORCE, starting with the Vanilla Search Gradient.

**Algorithm 2** : Search Gradient algorithm ($\lambda = 1$)

```
Require: initial parameters $\boldsymbol{\theta}$, fitness function $f$, number of offspring $\lambda$, step size $\alpha$
while not converged do
    $\boldsymbol{x} \sim p(\cdot\,;\boldsymbol{\theta})$                                  ▷ "Mutation"
    $f \leftarrow f(\boldsymbol{x})$                                        ▷ Evaluation
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}\,;\boldsymbol{\theta})$               ▷ Log-derivative of search distribution
    $\nabla_{\boldsymbol{\theta}} J \leftarrow f \cdot \boldsymbol{g}$                               ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$                          ▷ "Selection"
return $\boldsymbol{\theta}$                                             ▷ or best sample from final generation
```

As noted above, "solutions" $\boldsymbol{x}$ correspond to trajectories, $\tau$.

**Algorithm 3** : Search Gradient algorithm ($\lambda = 1$, $\boldsymbol{x} = \tau$)

```
Require: initial parameters $\boldsymbol{\theta}$, fitness function $f$, step size $\alpha$
while not converged do
    $\tau \sim p(\cdot\,;\boldsymbol{\theta})$                                   ▷ “Mutation”
    $f \leftarrow f(\tau)$
    $g \leftarrow \nabla_{\boldsymbol{\theta}} \log p(\tau\,;\boldsymbol{\theta})$
    $\nabla_{\boldsymbol{\theta}} J \leftarrow f \cdot g$                          ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$        ▷ “Selection”
return $\boldsymbol{\theta}$                                        ▷ or best sample from final generation
```

Here $p(\tau;\boldsymbol{\theta})$ is not simply a Gaussian we can compute for a whole trajectory. We can decompose the trajectory into the individual timesteps, and since only the action components of the trajectory depend on $\theta$, we get the product of action probabilities, $\pi(a \mid s;\, \boldsymbol{\theta})$.

Sampling a trajectory is like mutation, and updating the parameters (reinforcing the desired behaviors) is like selection. Also, at the end of the Search Gradient algorithm, we would probably want to return the best solution from the final generation. But in Policy Gradient, we don't run the algorithm just so we have a single good trajectory at the end. We actually want the parameters and apply them to arbitrary, new trajectories. In other words, the thing we care about is the ”search distribution” itself (the policy).

$$\begin{aligned}
\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \, p(\tau;\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \, p(s_0, a_0, s_1, a_1, \dots;\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \, p(s_0) \prod_{t=1}^{T-1} \pi(a_t \mid s_t;\boldsymbol{\theta}) \cdot p(s_{t+1} \mid s_t, a_t) \\
&= \arg\max_{\boldsymbol{\theta}} \prod_{t=1}^{T-1} \pi(a_t \mid s_t;\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \sum_{t=1}^{T-1} \log \pi(a_t \mid s_t;\boldsymbol{\theta})
\end{aligned}$$

$$\nabla_\theta \log p(\tau;\boldsymbol{\theta}) = \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t;\boldsymbol{\theta})$$

Note that when decomposing the trajectory probability, we can ignore the environment dynamics, as they don't depend on the parameters $\theta$. Crucially, we don't actually need to worry about the probabilities of all the components of the trajectory! We only care about probabilities of an individual action given some state.

In search gradients, we are free to define the form of the particular distribution. In REINFORCE, the probability $p(\tau)$ is defined to be the product of probabilities $\pi(a \mid s; \boldsymbol{\theta})$, which in turn can be represented in various ways.

As noted before, the fitness function is the return of a trajectory:

$$f(\tau) = \sum_{t=1}^{T-1} \gamma^{t-1} r_t$$

**Algorithm 4 :** Episodic REINFORCE

```
Require: initial parameters $\boldsymbol{\theta}$, fitness function $f$, step size $\alpha$
while not converged do
    $\tau \sim p(\cdot\,;\boldsymbol{\theta})$                                   ▷ "Mutation"
    $f \leftarrow \left[\sum_{t=1}^{T-1} \gamma^{t-1} r_t\right]$                ▷ $f(\tau)$
    $g \leftarrow \nabla_{\boldsymbol{\theta}} \log \left[\prod_{t=1}^{T-1} \pi_\theta(a_t \mid s_t)\right]$   ▷ $\nabla_{\boldsymbol{\theta}} \log p(\tau; \boldsymbol{\theta})$
    $\nabla_{\boldsymbol{\theta}} J \leftarrow f \cdot g$                          ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$        ▷ "Selection"
return $\boldsymbol{\theta}$                                  ▷ or best sample from final generation
```

where $\tau_t$ is the trajectory $\tau$ starting at timestep $t$.

**Algorithm 5 :** Episodic REINFORCE

```
Require: initial parameters $\boldsymbol{\theta}$, fitness function $f$, step size $\alpha$
while not converged do
    $\tau \sim p(\cdot\,;\boldsymbol{\theta})$                                   ▷ "Mutation"
    for $t = 1, \ldots, T-1$ do
        $f^{(t)} \leftarrow f(\tau_t)$
        $\boldsymbol{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t\,; \boldsymbol{\theta})$
    $\nabla_{\boldsymbol{\theta}} J \leftarrow \frac{1}{T} \sum_{t=1}^{T-1} f^{(t)} \cdot \boldsymbol{g}^{(t)}$     ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$        ▷ "Selection"
return $\boldsymbol{\theta}$
```

An important design decision we make here that is different from vanilla Search Gradient: we apply the parameter update in each timestep, and we don't use the fitness of the entire trajectory in each timestep, but the fitness of the partial trajectory starting at that timestep:

**Algorithm 6 :** REINFORCE

```
Require: initial parameters $\boldsymbol{\theta}$, reward function $f$, step size $\alpha$
while not converged do
    $\tau \sim p(\cdot\,;\boldsymbol{\theta})$                                        ▷ “Mutation”
    for $t = 1, \dots, T-1$ do
        $f \leftarrow f(\tau_t)$
        $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t\,;\boldsymbol{\theta})$
        $\nabla_{\boldsymbol{\theta}} J \leftarrow f \cdot \boldsymbol{g}$                ▷ Gradient of the expected fitness
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$                     ▷ “Selection”
return $\boldsymbol{\theta}$
```

Or in fewer lines, as you might find it elsewhere:

**Algorithm 7 :** REINFORCE

```
Require: initial parameters $\boldsymbol{\theta}$, reward function $f$, step size $\alpha$
while not converged do
    $\tau \sim p(\cdot\,;\boldsymbol{\theta})$                                        ▷ “Mutation”
    for $t = 1, \dots, T-1$ do
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot f(\tau_t) \cdot \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t\,;\boldsymbol{\theta})$     ▷ “Selection”
return $\boldsymbol{\theta}$
```

### 3.1 Evolving neural network weights

Note that if we apply the Search Gradient algorithm to the weights of a neural network, **w**, rather than the trajectories, we don't need to decompose the trajectory into timesteps. But this comes at the cost of being much less efficient.

**Algorithm 8** : Search Gradient algorithm applied to neural network weights

**Require:** initial parameters $\boldsymbol{\theta}$, fitness function $f$, number of offspring $\lambda$, step size $\alpha$

```
while not converged do
    for each $k \in \{1, \dots, \lambda\}$ do
        $\mathbf{w}^{(k)} \sim p(\cdot\,;\,\boldsymbol{\theta})$                                    ▷ "Mutation"
        $f^{(k)} \leftarrow f(\mathbf{w}^{(k)})$                                         ▷ Evaluation
        $\boldsymbol{g}^{(k)} \leftarrow \nabla_{\boldsymbol{\theta}} \log p(\mathbf{w}^{(k)}\,;\,\boldsymbol{\theta})$    ▷ Log-derivative of search distribution
    $\nabla_{\boldsymbol{\theta}} J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} f^{(k)} \cdot \boldsymbol{g}^{(k)}$          ▷ Gradient of the expected fitness
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \cdot \nabla_{\boldsymbol{\theta}} J$                          ▷ "Selection"
$k^* \leftarrow \arg\max_k f^{(k)}$                                           ▷ best sample from final generation
return $\mathbf{w}^{(k^*)}$
```