Zeroth-Order Optimization

October 9, 2025

What is Zeroth-Order Optimization?

- **First-order optimization** (e.g., gradient descent) uses:

$$\nabla f(x) \quad \text{to update } x \leftarrow x - \eta \nabla f(x)$$

- **Zeroth-order optimization** uses **only function values**:

$$\text{Access to } f(x) \quad \text{but not } \nabla f(x)$$

- Especially useful when:
  - Gradients are unavailable or too expensive to compute
  - Objective is noisy, black-box, or non-differentiable
- Key idea: **Use function values to estimate the gradient**.

Estimating the Gradient with Function Values

- **Gradient estimation via finite differences:**

$$\nabla f(x) \approx \frac{f(x + he_i) - f(x)}{h} \cdot e_i$$

  where e_i is the i-th standard basis vector.

- **Or via random directions:**

$$\nabla f(x) \approx \frac{f(x + hu) - f(x)}{h} \cdot u$$

  where $u \sim \mathcal{N}(0, I)$ or a random unit vector.

- Choose a small $h > 0$ (step size or smoothing parameter).
- This gives a stochastic estimate of the gradient using **only function evaluations**.

Applications:

- Many real-world optimization problems lack gradient information.
- Examples:
  - Hyperparameter tuning
  - Adversarial attacks on neural networks
- Gradient computation costs too much GPU memory.
- Examples:
  - Large Language Model fine tuning.
- Key idea: **Optimize using only function evaluations**.

Application: Hyperparameter Tuning

- **Goal**: Find hyperparameters x (e.g., learning rate, weight decay) that minimize **validation loss** $f(x)$.
- Challenge: No gradient of validation loss w.r.t. hyperparameters.
- Use zeroth-order estimate:

$$\nabla f(x) \approx \frac{f(x + hu) - f(x)}{h} \cdot u$$

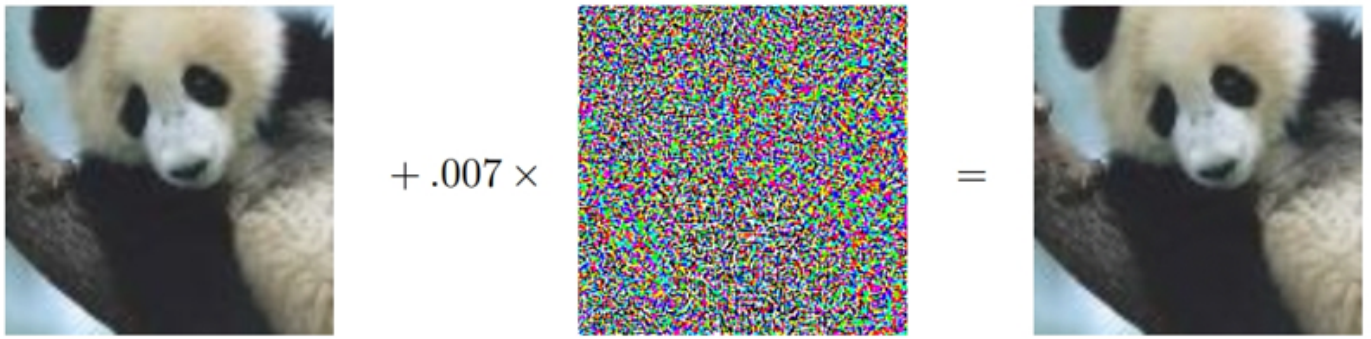
- Example:
  - Sample $u \sim \mathcal{N}(0, I)$, perturb hyperparameters, evaluate validation loss.
  - Update $x \leftarrow x - \eta \cdot$ estimated gradient.
- Note[1]: directly apply this approach usually cannot work. More practical issues should be considered in practice.

[1] Koch, Patrick, et al. "Autotune: A derivative-free optimization framework for hyperparameter tuning." Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2018.

Application: Adversarial Attacks on Neural Networks

- **Goal**: Find small perturbation δ such that $x + \delta$ causes misclassification.
- Objective: Maximize $f(\delta) = \text{loss}(x + \delta, y_{\text{target}})$, where gradient w.r.t. δ may be unavailable (e.g., in black-box models).
- Use zeroth-order estimate:

$$\nabla f(\delta) \approx \frac{f(\delta + hu) - f(\delta)}{h} \cdot u$$



Source: based on material from Ian Goodfellow, 2016.

Application: Fine-tuning Large Language Models[2]

- **Goal**: Update parameters x to minimize loss on task-specific data.
- Computing gradients consumes too much memory for large models.

Example

Consider a two-layer linear neural network. $x \in R^m, y_1 \in R^n, y_2 \in R$.

$$y_1 = \text{Layer1}(x) = W_1 x$$

$$y_2 = \text{Layer2}(y_1) = W_2 y_1$$

- Saved in the Layer2: $\frac{\partial y_2}{\partial y_1} = W_2 \in R^n$
- Saved in the Layer1: $\frac{\partial y_1}{\partial x} = W_1 \in R^{n \times m}$

[2] Malladi, Sadhika, et al. "Fine-tuning language models with just forward passes." Advances in Neural Information Processing Systems 36 (2023): 53038-53075.

Application: Fine-tuning Large Language Models

Example (Continue...)

Backpropagation (first-order): $\frac{\partial y_2}{\partial x} = \frac{\partial y_2}{\partial y_1}\frac{\partial y_1}{\partial x} = W_2 W_1$ (Both $\frac{\partial y_2}{\partial y_1}$ and $\frac{\partial y_1}{\partial x}$ are used.)

- Saved in the Layer2: Perturbed output $(W_2 + \delta_2)y_1 \in R$, Perturbed vector $\delta_2 \in R^n$.
- Saved in the Layer1: Perturbed output $(W_1 + \delta_1)x \in R^n$, Perturbed vector $\delta_1 \in R^{n \times m}$.

Backpropagation (zeroth-order): $\frac{\partial y_2}{\partial x} = \frac{(W_2+\delta_2)y_1 - W_2 W_1 x}{\|\delta_1\|}\delta_1$ (Only δ_1 is used.)

Key idea: When updating the Layer2:

- First-order: both W_2 and W_1 are saved in the memory
- Zeroth-order: only δ_2 are saved in the memory

Application: Fine-tuning Large Language Models

F.5 Memory profiling

We show the detailed numbers of memory profiling results Table 22, which also corresponds to Figure 3. For how we profile the memory usage, please refer to Appendix E.7.

Method	**Zero-shot / MeZO**	**ICL**	**Prefix FT**	**Full-parameter FT**
1.3B	1xA100 (4GB)	1xA100 (6GB)	1xA100 (19GB)	1xA100 (27GB)
2.7B	1xA100 (7GB)	1xA100 (8GB)	1xA100 (29GB)	1xA100 (55GB)
6.7B	1xA100 (14GB)	1xA100 (16GB)	1xA100 (46GB)	2xA100 (156GB)
13B	1xA100 (26GB)	1xA100 (29GB)	2xA100 (158GB)	4xA100 (316GB)
30B	1xA100 (58GB)	1xA100 (62GB)	4xA100 (315GB)	8xA100 (633GB)
66B	2xA100 (128GB)	2xA100 (134GB)	8xA100	16xA100

Table 22: Memory usage on the MultiRC (avg #tokens=400) dataset.

Source: Malladi, Sadhika, et al. "Fine-tuning language models with just forward passes." Advances in Neural Information Processing Systems 36 (2023): 53038-53075.

Theoretical Analysis of Zeroth-Order Gradient Estimator

- By applying the Taylor's theorem to $f(x+v)$:

$$f(x+\mu v) = f(x) + \mu \nabla f(x)^\top v + o(\mu^2)$$

- Solve $\nabla f(x)$:

$$vv^\top \nabla f(x) = \frac{f(x+\mu v) - f(x)}{\mu} + o(\mu)$$

- By choosing appropriate vector v, $\frac{f(x+\mu v)-f(x)}{\mu} v$ can be an unbiased estimator of $\nabla f(x)$ (as μ tends to 0).

Theoretical Analysis of Zeroth-Order Gradient Estimator

- $\frac{f(x+\mu v)-f(x)}{\mu} v$ needs to take two function evaluations.
- $\frac{f(x+\mu v)}{\mu} v$ is still an unbiased estimator of $\nabla f(x)$ (as μ tends to 0). $\frac{f(x+\mu v)}{\mu} \approx \nabla_v f(x)$ (the directional derivative of $f(x)$ along the direction v).
- However, the variance is infinite.