A Lyapunov Analysis of the Lion Optimizer

Qiang Liu

UT Austin

Work with my students: Lizhang Chen, Bo Liu, Kaizhao Liang

The Quest of better Optimizers

- Optimization: The Cornerstone of large AI model training

$$\min_{\theta} L(\theta)$$

  - Stochastic gradient
  - Momentum
  - Adaptive methods: Adam, Adagrad, etc.
  - AdamW is largely the default for LLM pre-training.

- Better optimizer = Money + Time + Performance + Environmental sustainability.

Optimization Background

Gradient descent (SGD):

$$x_{t+1} = x_t - \epsilon \nabla f(x_t).$$

Momentum:

$$\begin{aligned} m_t &= \beta m_{t-1} - (1-\beta)\nabla f(x_t) \\ x_{t+1} &= x_t + \epsilon m_t \end{aligned}$$

Adam(W):

$$\begin{aligned} m_t &= \beta_1 m_{t-1} - (1-\beta_1)\nabla f(x_t) \\ v_t &= \beta_2 v_{t-1} - (1-\beta_2)\nabla f(x_t)^2 \\ \hat{m}_t &= m_t/(1-\beta_1^t), \quad \hat{v}_t = v_t/(1-\beta_1^t) \\ x_{t+1} &= x_t + \epsilon\left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} - \lambda x_t\right). \end{aligned}$$

- Memory/computation cost: AdamW > Momentum > SGD
- Can we find better algorithms than AdamW?

Symbolic Discovery of Optimization Algorithms

Xiangning Chen[1 2 § *] Chen Liang[1 §] Da Huang[1] Esteban Real[1]

Kaiyuan Wang[1] Yao Liu[1 †] Hieu Pham[1] Xuanyi Dong[1] Thang Luong[1]

Cho-Jui Hsieh[2] Yifeng Lu[1] Quoc V. Le[1]

[§]Equal & Core Contribution

[1]Google [2]UCLA

Abstract

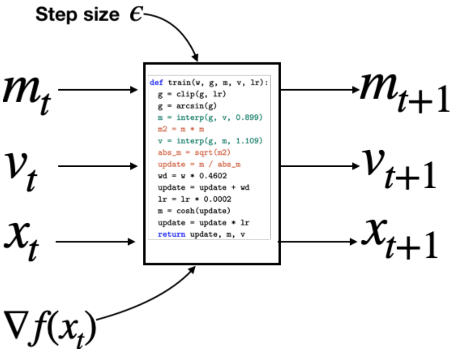
We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (*EvoLved Sign Momentum*). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% *zero-shot* and 91.1% *fine-tuning* accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant. The implementation of Lion is publicly available.[1] Lion is also successfully deployed in production systems such as Google's search ads CTR model.

Symbolic Discovery of Optimization Algorithms

- An algorithm is characterized by a `train` function:

$$x_t, m_t, v_t = \texttt{train}(x_t, m_t, v_t, \nabla f(x_t), \epsilon)$$

- Find good $\texttt{train}(\cdot)$ with evolutionary search, in a predefined program space.



Program 2: An example training loop, where the optimization algorithm that we are searching for is encoded within the `train` function. The main inputs are the weight (`w`), gradient (`g`) and learning rate schedule (`lr`). The main output is the `update` to the weight. `v1` and `v2` are two additional variables for collecting historical information.

```
w = weight_initialize()
v1 = zero_initialize()
v2 = zero_initialize()
for i in range(num_train_steps):
  lr = learning_rate_schedule(i)
  g = compute_gradient(w, get_batch(i))
  update, v1, v2 = train(w, g, v1, v2, lr)
  w = w - update
```

Program 3: Initial program (AdamW). The bias correction and ϵ are omitted for simplicity.

```
def train(w, g, m, v, lr):
  g2 = square(g)
  m = interp(g, m, 0.9)
  v = interp(g2, v, 0.999)
  sqrt_v = sqrt(v)
  update = m / sqrt_v
  wd = w * 0.01
  update = update + wd
  lr = lr * 0.001
  update = update * lr
  return update, m, v
```

Program 4: Discovered program after search, selection and removing redundancies in the raw Program 8. Some variables are renamed for clarity.

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  g = arcsin(g)
  m = interp(g, v, 0.899)
  m2 = m * m
  v = interp(g, m, 1.109)
  abs_m = sqrt(m2)
  update = m / abs_m
  wd = w * 0.4602
  update = update + wd
  lr = lr * 0.0002
  m = cosh(update)
  update = update * lr
  return update, m, v
```

Lion (Evolved Sign Momentum)

Program 8: Raw program of Lion before removing redundent statements.

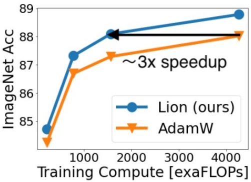
```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  m = clip(m, lr)
  v845 = sqrt(0.6270633339881897)
  v968 = sign(v)
  v968 = v - v
  g = arcsin(g)
  m = interp(g, v, 0.8999999761581421)
  v1 = m * m
  v = interp(g, m, 1.109133005142212)
  v845 = tanh(v845)
  lr = lr * 0.0002171761734643951
  update = m * lr
  v1 = sqrt(v1)
  update = update / v1
  wd = lr * 0.4601978361606598
  v1 = square(v1)
  wd = wd * w
  m = cosh(update)
  lr = tan(1.4572199583053589)
  update = update + wd
  lr = cos(v845)
  return update, m, v
```

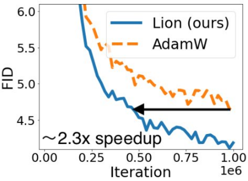
Program 4: Discovered program after search, selection and removing redundancies in the raw Program 8. Some variables are renamed for clarity.

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  g = arcsin(g)
  m = interp(g, v, 0.899)
  m2 = m * m
  v = interp(g, m, 1.109)
  abs_m = sqrt(m2)
  update = m / abs_m
  wd = w * 0.4602
  update = update + wd
  lr = lr * 0.0002
  m = cosh(update)
  update = update * lr
  return update, m, v
```

Lion (Evolved Sign Momentum)

Optimizer	Zero-shot ImageNet	Zero-shot V2	Zero-shot A	Zero-shot R	Zero-shot Sketch	Zero-shot ObjectNet	Fine-tune ImageNet
Adafactor	85.7	80.6	85.6	95.7	76.1	82.3	90.9
Lion	**88.3**	**81.2**	**86.4**	**96.8**	**77.2**	**82.9**	**91.1**






```
def train(weight, gradient, momentum, lr):
  update = interp(gradient, momentum, β1)
  update = sign(update)
  momentum = interp(gradient, momentum, β2)
  weight_decay = weight * λ
  update = update + weight_decay
  update = update * lr
  return update, momentum
```

Sebastian Raschka @rasbt · Mar 9, 2023
Just took the new **Lion optimizer** (arxiv.org/ab
and I am positively surprised.

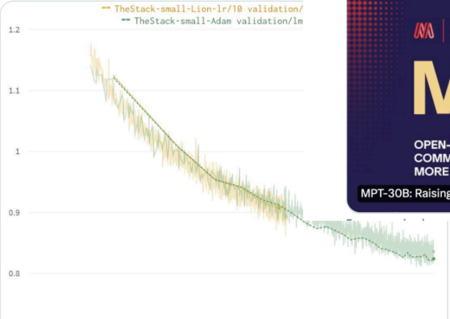
With a bit of tinkering, I got it to perform simila
DistilBERT (never had any luck with SGD on thi

Code here for ref: github.com/rasbt/try-lion...

Maxine @cephaloform · Feb 15, 2023
I trained a 124m param GPT2 model with Google'
found through genetic programming and saw a 3
of steps needed to reach the same loss as Adam
how small this test is)

paper: arxiv.org/pdf/2302.06675...

Vincent Hellendoorn @VHellendoorn · Mar 2, 202
Quick update: after some debugging it now works a
beating Adam starting around 10K steps while runn
memory. Great work @XiangningChen and team! Im
super impactful. Same with @tri_dao's Flash Attenti



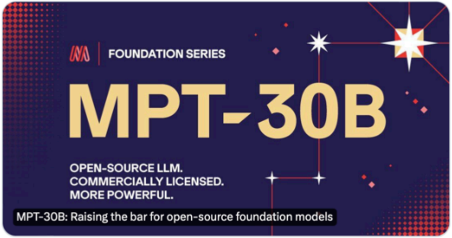
Xiangning Chen @XiangningChen · Jun 12, 2023
Replying to @XiangningChen
"We also train our MPT models with the **Lion optimizer** rather than AdamW, which provides stable update magnitudes and cuts **optimizer** state memory in half."

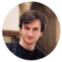
Xiangning Chen @XiangningChen · Jun 12, 2023
I've just been told that MPT-7B is trained by our **Lion optimizer** (arxiv.org/abs/2302.06675).

Lion has also been successfully deployed in production systems such as Google search ads CTR model.

Glad to see that our work has real-world production impact!

Chen Liang @crazydonkey200 · Jun 27, 2023
An open-source 30B language model trained with our Lion optimizer🦁 (arxiv.org/abs/2302.06675) Another proof that Lion is a good replacement for Adam for training large models and "provides stable update magnitudes and cuts optimizer state memory in half".





James Campbell @jam3scampbell · Mar 8, 2023

The acronym abuse in ML is getting ridiculous: Google Brain's new **optimizer**, **Lion**, is short for EvoLved Sign Momentum. What.

2 59 4.6K

Kumail Alhamoud @KumailAlhamoud · Apr 2, 2023

Replying to @SDAIA_KAUST_AI @peter_richtarik and @sameh_abdulah

I think that tweet with the **LION**-like **optimizer** was supposed to be some April fool's joke

Lion Update Rule

$$
\begin{aligned}
m_{t+1} &= \beta_2 m_t - (1-\beta_2)\nabla f(x_t) \\
x_{t+1} &= x_t + \epsilon \left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t \right)
\end{aligned}
$$

Key features:

- Use $\mathrm{sign}[\cdot]$.
- Use linear combination of gradient $\nabla f(x_t)$ and momentum m_t.
- Use weight decay λx_t.
- No need to keep track of v_t, save memory

Lion Update Rule

$$
\begin{aligned}
m_{t+1} &= \beta_2 m_t - (1-\beta_2)\nabla f(x_t) \\
x_{t+1} &= x_t + \epsilon\left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)
\end{aligned}
$$

Unrolled update with default $\beta_1 = 0.9, \beta_2 = 0.99$:

$$
x_{t+1} = (1-\epsilon\lambda)x_t + \mathrm{sign}\left[(10+1)g_t + 0.99 g_{t-1} + \cdots 0.99^k g_{t-k} \cdots\right]
$$

Key features:

- Use $\mathrm{sign}[\cdot]$.
- Use linear combination of gradient $\nabla f(x_t)$ and momentum m_t.
- Use weight decay λx_t.
- No need to keep track of v_t, save memory

Lion Update Rule

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon\left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)$$

Key points:

- Use $\mathrm{sign}[\cdot]$.
  - Generalizes signed SGD and signed Momentum
  - Signed SGD is the steepest descent under ℓ_∞ norm
  - Adam can be viewed as a smoothed signed SGD:

$$m_t = \beta_1 m_{t-1} - (1-\beta_1)\nabla f(x_t)$$
$$v_t = \beta_2 v_{t-1} - (1-\beta_2)\nabla f(x_t)^2$$
$$x_{t+1} = x_t + \epsilon\left(\frac{m_t}{\sqrt{v_t}+\epsilon} - \lambda x_t\right).$$

When $\beta_1 = \beta_2 = \epsilon = \lambda = 0$, *update* $= \frac{m_t}{\sqrt{v_t}} = \mathrm{sign}(\nabla f(x_t))$.

  - Such coordinate-balanced update is crucial for performance in neural network training.

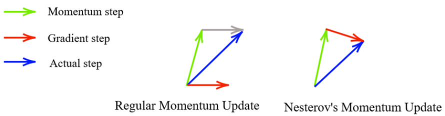
Lion Update Rule

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon\left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)$$

Key points:

- Use linear combination of gradient $\nabla f(x_t)$ and momentum m_t.
    - This is in fact the idea of Nesterov momentum:

$$x_{t+1} = x_t + \epsilon(\beta_1 m_t - (1-\beta_1)\nabla f(x_t)).$$



    - Adding gradient "stabilizes" the momentum update.
    - So we can use more aggressive momentum coefficients (recommended: $\beta_2 = 0.99$, $\beta_1 = 0.9$).

Lion Update Rule

$$
\begin{aligned}
m_{t+1} &= \beta_2 m_t - (1-\beta_2)\nabla f(x_t) \\
x_{t+1} &= x_t + \epsilon \left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)
\end{aligned}
$$

Key points:

- Use "decoupled" weight decay λx_t.
  - Weight decay is applied on update, not on gradient
  - A very useful component of AdamW
  - It is NOT L2 regularization

Should we trust a randomly discovered algorithm?

It turns out it solves a constrained optimization

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon\left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)$$

- It solves, thanks to the weight decay,

$$\min_x f(x) \quad s.t. \quad \|x\|_\infty \leq \frac{1}{\lambda}.$$

- The constraint is enforced very rapidly.

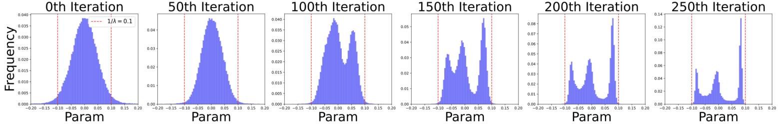


Figure 2: Histograms of the network parameters of ResNet-18 on CIFAR-10 trained by Lion with $\lambda = 10$. The constraint of $\|x\|_\infty \leq 1/\lambda$ (indicated by the red vertical lines) is satisfied within only $\sim$200 steps.

It turns out it solves a constrained optimization

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon \left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)$$

- It solves, thanks to the weight decay,

$$\min_x f(x) \quad s.t. \quad \|x\|_\infty \le \frac{1}{\lambda}.$$

- Why? Intuition:
  - Note that $x_{t+1} = x_t + \epsilon(\mathrm{sign}(u_t) - \lambda x_t)$
  - If $|\lambda x_t| > 1 \ge |\mathrm{sign}(u_t)|$, the decay term would dominate.

It turns out it solves a constrained optimization

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon\left(\mathrm{sign}\left[\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right] - \lambda x_t\right)$$

- It solves, thanks to the weight decay,

$$\min_x f(x) \quad s.t. \quad \|x\|_\infty \leq \frac{1}{\lambda}.$$

- Why? Intuition:
  - Note that $x_{t+1} = x_t + \epsilon(\mathrm{sign}(u_t) - \lambda x_t)$
  - If $|\lambda x_t| > 1 \geq |\mathrm{sign}(u_t)|$, the decay term would dominate.
  - We can show, when $\epsilon\lambda < 1$,

$$dist(x_{t+1}, \mathcal{F}) \leq (1-\epsilon\lambda)\, dist(x_t, \mathcal{F}),$$

where $\mathcal{F} = \{x \colon \|\lambda x\|_\infty \leq 1\}$, under any notion of distance $dist(\cdot)$.

How to show the convergence to optimum?

- Approach: Certify the convergence with a Laypunov function

How to show the convergence to optimum?

- Approach: Certify the convergence with a Laypunov function
- Consider, for example, the standard momentum method:

$$m_{t+1} = \beta_1 m_t - (1-\beta_1)\nabla f(x_t), \qquad x_{t+1} = x_t + \epsilon m_{t+1},$$

  - Continuous limit: the heavy ball dynamics

$$\dot{m}_t = -\nabla f(x_t) - \gamma m_t, \qquad \dot{x}_t = m_t$$

How to show the convergence to optimum?

- Approach: Certify the convergence with a Laypunov function
- Consider, for example, the standard momentum method:

$$m_{t+1} = \beta_1 m_t - (1-\beta_1)\nabla f(x_t), \qquad x_{t+1} = x_t + \epsilon m_{t+1},$$

  - Continuous limit: the heavy ball dynamics

$$\dot{m}_t = -\nabla f(x_t) - \gamma m_t, \qquad \dot{x}_t = m_t$$

- The system monotonically decreases the following Hamiltonian function:

$$H(x,m) = \underbrace{f(x)}_{\text{potential energy}} + \underbrace{\frac{1}{2}\|m\|^2}_{\text{kinetic energy}}.$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}H(x_t,m_t) &= \partial_x H(x_t,m_t)^\top \dot{x}_t + \partial_m H(x_t,m_t)^\top \dot{m}_t \\ &= \nabla f(x_t)^\top \dot{x}_t + \gamma m^\top(-\nabla f(x_t) + \gamma m_t) \\ &= -\gamma\|m_t\|^2 \le 0.\end{aligned}$$

Lion-$\mathcal{K}$: A Generalization

- Let $\mathcal{K}$ be any convex function and $\nabla\mathcal{K}$ its subgradient:

$$\begin{aligned} m_{t+1} &= \beta_2 m_t - (1-\beta_2)\nabla f(x_t) \\ x_{t+1} &= x_t + \epsilon\left(\nabla\mathcal{K}\left(\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\right) - \lambda x_t\right) \end{aligned}$$

When $\mathcal{K}(x) = \|x\|_1$, we take $\nabla\mathcal{K}(x) = \mathrm{sign}(x)$.

- In the continuous time limit, Lion-$\mathcal{K}$ ODE:

$$\dot{m}_t = -\alpha \nabla f(x_t) - \gamma m_t$$
$$\dot{x}_t = \nabla \mathcal{K}(m_t - \varepsilon(\alpha \nabla f(x_t) + \gamma m_t)) - \lambda x_t.$$

- Main result: Lion-$\mathcal{K}$ solves

$$\min_x F(x) := \alpha f(x) + \frac{\gamma}{\lambda} \mathcal{K}^*(\lambda x),$$

$\mathcal{K}^*$ is the convex conjugate of $\mathcal{K}$: $\mathcal{K}^*(x) = \sup_y \left(x^\top y - \mathcal{K}(y)\right)$.

- When $\mathcal{K}^*(x)$ can take infinite values, it is a constrained optimization:

$$\min_x F(x), \qquad s.t. \quad x \in \mathrm{dom}\mathcal{K}^*,$$

where $\mathrm{dom}\mathcal{K}^* = \{x : \mathcal{K}^*(x) < +\infty\}$.

- Example: $\mathcal{K}(x) = \|x\|_1$, then $\mathcal{K}^*(x) = \begin{cases} 0 & \text{if } \|x\|_\infty \leq 1 \\ +\infty & \text{if } \|x\|_\infty > 1. \end{cases}$

- Lion-$\mathcal{K}$ includes a broad family of old and new algorithms

Polyak Momentum [31]	$\mathcal{K}(x) = \|x\|_2^2/2, \gamma\lambda = 0, \varepsilon = 0$
Nesterov Momentum [28]	$\mathcal{K}(x) = \|x\|_2^2/2, \gamma\lambda = 0$
Signed Momentum [5]	$\mathcal{K}(x) = \|x\|_1^2, \varepsilon = 0, \lambda = 0$
Hamiltonian Descent [23]	$\varepsilon = 0, \lambda = 0$
Hamiltonian Descent for Composite Objectives [23]	$\varepsilon = 0, \lambda > 0$
Dual Space Preconditioning [24], Mirror Descent [27]	$\varepsilon\gamma = 1, \lambda = 0$
Signed Gradient Descent [5]	$\mathcal{K}(x) = \|x\|_1, \varepsilon\gamma = 1, \lambda = 0$
Accelerated Mirror Descent [17]	$\gamma = 0, \varepsilon = 0, \lambda > 0$
Frank–Wolfe [11]	$\varepsilon\gamma = 1, \lambda > 0$

Table 1: Lion-$\mathcal{K}$ includes a large family algorithms as special cases. See Section 3.1

Line ID	$\mathcal{K}(x)$	$\nabla\mathcal{K}(x)$	$\min_x f(x) + \mathcal{K}^*(x)$
①	$\|x\|_1$	$\mathrm{sign}(x)$	$\min f(x) \;\; s.t. \;\; \|x\|_\infty \leq 1$
②	$\|x\|_p$	$\frac{\mathrm{sign}(x)\|x\|^{p-1}}{\|x\|_p^{p-1}}$	$\min f(x) \;\; s.t. \;\; \|x\|_q \leq 1$
③	$\sum_i \max(\|x_i\| - e, 0)$	$\mathrm{sign}(x)\mathbb{I}(\|x\| > e)$	$\min f(x) + e\|x\|_1 \;\; s.t. \;\; \|x\|_\infty \leq 1$
④	$\sum_{i \leq i^{cut}} \|x_{(i)}\|$	$\mathrm{sign}(x)\mathbb{I}(\|x\| > \|x_{(i^{cut})}\|)$	$\min f(x) \;\; s.t. \;\; \|x\|_1 \leq i^{cut}, \;\; \|x\|_\infty \leq 1$
⑤	$\sum_i \mathrm{huber}_e(x_i)$	$\mathrm{clip}(x, -e, e)/e$	$\min f(x) + \frac{e}{2}\|x\|_2^2 \;\; s.t. \;\; \|x\|_\infty < 1$

Table 2: Examples of $\mathcal{K}$ and $\nabla\mathcal{K}$, and the optimization problems they solved (we set $\gamma = \lambda = 1$ for simplicity). We assume $x = [x_1, \ldots, x_d] \in \mathbb{R}^d$ and $|x_{(1)}| \geq |x_{(2)}| \geq \cdots$ is a monotonic sorting of the elements of x, and i^{cut} is an integer in $\{1, \ldots, d\}$. The Huber loss is $\mathrm{huber}_e(x_i) = \mathbb{I}(|x_i| \geq e)(|x_i| - \frac{e}{2}) + \mathbb{I}(|x_i| < e)\frac{1}{2e}x_i^2$,

Lyapunov Function of Lion-$\mathcal{K}$

Lion-$\mathcal{K}$ ODE (assume $\varepsilon\gamma \leq 1$):

$$\dot{m}_t = -\alpha \nabla f(x_t) - \gamma m_t$$
$$\dot{x}_t = \nabla \mathcal{K}(m_t - \varepsilon(\alpha \nabla f(x_t) + \gamma m_t)) - \lambda x_t.$$

- **[Phase 1]** If $\mathcal{K}^*(x) = +\infty$ (constraint unsatisfied), we have

$$dist(x_t, \mathrm{dom}\mathcal{K}^*) \leq \exp(-\lambda(t-s)) dist(x_s, \mathrm{dom}\mathcal{K}^*), \qquad \forall 0 \leq s \leq t.$$

Lyapunov Function of Lion-$\mathcal{K}$

Lion-$\mathcal{K}$ ODE (assume $\varepsilon\gamma \leq 1$):

$$\dot{m}_t = -\alpha \nabla f(x_t) - \gamma m_t, \quad \dot{x}_t = \nabla \mathcal{K}(m_t - \varepsilon(\alpha \nabla f(x_t) + \gamma m_t)) - \lambda x_t.$$

Solves:

$$\min_x F(x) := \alpha f(x) + \frac{\gamma}{\lambda} \mathcal{K}^*(x), \qquad s.t. \quad x \in \mathrm{dom}\mathcal{K}^*,$$

- **[Phase 2]** It monotonically decreases the following Lyapunov function ($\frac{\mathrm{d}}{\mathrm{d}t} H(x_t, m_t) \leq 0$):

$$H(x, m) = \underbrace{\alpha f(x) + \frac{\gamma}{\lambda} \mathcal{K}^*(\lambda)}_{\text{“potential” function}} + \underbrace{\frac{1 - \varepsilon\gamma}{1 + \epsilon\lambda} (\mathcal{K}^*(\lambda x) + \mathcal{K}(m) - \lambda m^\top x)}_{\text{“kinetic” energy}}$$

- Fenchel-Young inequality: $(\mathcal{K}^*(\lambda x) + \mathcal{K}(m) - \lambda m^\top x) \geq 0$, equality achieved when $\nabla \mathcal{K}(m) = \lambda x$.

- Minimizing $H(x, m)$ and $F(x)$ are equivalent: $\min_m H(x, m) = F(x)$.

- Lion-$\mathcal{K}$: Discrete time

$$m_{t+1} = \beta_2 m_t - (1-\beta_2)\nabla f(x_t)$$
$$x_{t+1} = x_t + \epsilon\left(\nabla\mathcal{K}\big(\beta_1 m_t - (1-\beta_1)\nabla f(x_t)\big) - \lambda x_{t+1}\right)$$

- **[Phase 1]** Constrained Enforcing:

$$dist(x_{t+1},\ \ \mathrm{dom}\mathcal{K}^*) \le \frac{1}{1+\epsilon\lambda} dist(x_t,\ \ \mathrm{dom}\mathcal{K}^*)$$

- **[Phase 2]** Constrained Optimization:

$$H(x,m) = f(x) + \frac{1}{\lambda}\mathcal{K}^*(x) + \frac{\beta_1}{\epsilon\lambda(1-\beta_1)+(1-\beta_2)}(\mathcal{K}^*(\lambda x) + \mathcal{K}(m) - \lambda x^\top m).$$

Then

$$H(x_{t+1}, m_t) - H(x_t, m_t) \le -\epsilon\Delta_t + \frac{L}{2}\epsilon^2.$$

Hence, $H(x,m)$ decreases when ϵ is small.

Proof: Consider any ODE:

$$\dot{m}_t = U_t(x_t, m_t)$$
$$\dot{x}_t = V_t(x_t, m_t).$$

$$\frac{\mathrm{d}}{\mathrm{d}t} H(x_t, m_t) = \partial_m H(x_t, m_t)^\top U_t(x_t, m_t) + \partial_x H(x_t, m_t)^\top V_t(x_t, m_t).$$

Proof: Consider any ODE:

$$\dot{m}_t = U_t(x_t, m_t)$$
$$\dot{x}_t = V_t(x_t, m_t).$$

$$\frac{\mathrm{d}}{\mathrm{d}t} H(x_t, m_t) = \partial_m H(x_t, m_t)^\top U_t(x_t, m_t) + \partial_x H(x_t, m_t)^\top V_t(x_t, m_t).$$

If we can verify that H and V satisfying the following relation:

$$\partial_m H(x, m) = -b\hat{U}_t(x, m) + cV_t(x, m),$$
$$\partial_x H(x, m) = -a\hat{V}_t(x, m) - cU_t(x, m),$$

where $a, b > 0$, $\hat{V}$ are monotonic transforms of V:

$$\hat{V}_t(x, m)^\top V_t(x, m) \geq 0, \qquad \hat{U}_t(x, m)^\top U_t(x, m) \geq 0.$$

Then

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} H(x_t, m_t) &= (-b\hat{U}_t + cV_t)^\top U_t + (-a\hat{V}_t - cU_t)^\top V_t \\ &= -a\hat{V}_t{}^\top V_t - b\hat{U}_t{}^\top U_t \leq 0.\end{aligned}$$

Key: The cross term $U_t^\top V_t$ is canceled.

Thoughts

- Initially, we did not believe it was a right algorithm, and tried hard to find "more theoretically principled" variants.
- Surprising that a machine-discovered algorithms yields such an intriguing mathematical structure.
- Potential directions:
  - Better search programs:
    - A good investment because new efficient algorithms and save computation in the future.
  - Improving and Extending Lion:
    - Example: in ongoing works, we are developing distributed Lion, leveraging the $\mathrm{sign}(\cdot)$ to develop distributed optimization that only requires to communicate random bits.

Thank You!