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:

\[ m_t = \beta m_{t-1} - (1 - \beta) \nabla f(x_t) \]
\[ x_{t+1} = x_t + \epsilon m_t \]

Adam(W):

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

• Memory/computation cost: AdamW > Momentum > SGD
• Can we find better algorithms than AdamW?
Symbolic Discovery of Optimization Algorithms

Xiangning Chen\textsuperscript{1, 2} $\S$ * Chen Liang\textsuperscript{1} $\S$ Da Huang\textsuperscript{1} Esteban Real\textsuperscript{1}

Kaiyuan Wang\textsuperscript{1} Yao Liu\textsuperscript{1} Hieu Pham\textsuperscript{1} Xuanyi Dong\textsuperscript{1} Thang Luong\textsuperscript{1}

Cho-Jui Hsieh\textsuperscript{2} Yifeng Lu\textsuperscript{1} Quoc V. Le\textsuperscript{1}

$\S$Equal & Core Contribution

\textsuperscript{1}Google  \textsuperscript{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 (\textit{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.\textsuperscript{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:

  \[ x_t, m_t, v_t = \text{train}(x_t, m_t, v_t, \nabla f(x_t), \epsilon) \]

- Find good train(·) 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.

\[
\begin{align*}
w &= \text{weight\_initialize}() \\
v1 &= \text{zero\_initialize}() \\
v2 &= \text{zero\_initialize}() \\
\text{for} \ i \ \text{in} \ \text{range}(\text{num\_train\_steps}): & \\
 & \quad \text{lr} = \text{learning\_rate\_schedule}(i) \\
 & \quad g = \text{compute\_gradient}(w, \text{get\_batch}(i)) \\
 & \quad \text{update, } v1, v2 = \text{train}(w, g, v1, v2, lr) \\
 & \quad w = w - \text{update}
\end{align*}
\]

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

\[
\begin{align*}
\text{def} \ \text{train}(w, g, m, v, lr): \\
& \quad g2 = \text{square}(g) \\
& \quad m = \text{interp}(g, m, 0.9) \\
& \quad v = \text{interp}(g2, v, 0.999) \\
& \quad \text{sqrt\_v} = \text{sqrt}(v) \\
& \quad \text{update} = m / \text{sqrt\_v} \\
& \quad wd = w * 0.01 \\
& \quad \text{update} = \text{update} + wd \\
& \quad lr = lr * 0.001 \\
& \quad \text{update} = \text{update} * lr \\
& \text{return} \ \text{update}, m, v
\end{align*}
\]

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

\[
\begin{align*}
\text{def} \ \text{train}(w, g, m, v, lr): \\
& \quad g = \text{clip}(g, lr) \\
& \quad g = \text{arcsin}(g) \\
& \quad m = \text{interp}(g, v, 0.899) \\
& \quad m2 = m * m \\
& \quad v = \text{interp}(g, m, 1.109) \\
& \quad \text{abs\_m} = \text{sqrt}(m2) \\
& \quad \text{update} = m / \text{abs\_m} \\
& \quad wd = w * 0.4602 \\
& \quad \text{update} = \text{update} + wd \\
& \quad lr = lr * 0.0002 \\
& \quad m = \text{cosh}(\text{update}) \\
& \quad \text{update} = \text{update} * lr \\
& \text{return} \ \text{update}, m, v
\end{align*}
\]
Lion (Evolved Sign Momentum)

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

```python
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.

```python
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)

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>ImageNet</th>
<th>V2</th>
<th>Zero-shot</th>
<th>Sketch</th>
<th>ObjectNet</th>
<th>Fine-tune ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaFactor</td>
<td>85.7</td>
<td>80.6</td>
<td>85.6</td>
<td>95.7</td>
<td>76.1</td>
<td>82.3</td>
</tr>
<tr>
<td>Lion</td>
<td>88.3</td>
<td>81.2</td>
<td>86.4</td>
<td>96.8</td>
<td>77.2</td>
<td>82.9</td>
</tr>
</tbody>
</table>

```
def train(weight, gradient, momentum, lr):
    update = interp(gradient, momentum, \beta_1)
    update = sign(update)
    momentum = interp(gradient, momentum, \beta_2)
    weight_decay = weight * \lambda
    update = update + weight_decay
    update = update * lr
    return update, momentum
```
Sebastian Raschka @rasbt · Mar 9, 2023

Just took the new Lion optimizer ([arxiv.org/abs](https://arxiv.org/abs) and I am positively surprised.

With a bit of tinkering, I got it to perform similarly to DistilBERT (never had any luck with SGD on this).


Maxine @cephaloform · Feb 15, 2023

I trained a 124m param GPT2 model with Google's [Genetic Programming](https://arxiv.org/pdf/2302.06675) and saw a 3x improvement in the number of steps needed to reach the same loss as Adam how small this test is)


Vincent Hellendoorn @VHellendoorn · Mar 2, 2023

Quick update: after some debugging it now works a beating Adam starting around 10K steps while running on memory. Great work @XiangningChen and team! Im super impactful. Same with @tridao's Flash Attenti

[arxiv.org/abs/2302.06675](https://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".

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](https://arxiv.org/abs/2302.06675)).

Lion has also been successfully deployed in production search engines and Sarah 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](https://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.

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

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

Key features:

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

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

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

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

Key features:

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

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

Key points:
- Use \( \text{sign}[\cdot] \).
- Generalizes signed SGD and signed Momentum
- Signed SGD is the steepest descent under \( \ell_\infty \) 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 \( \frac{m_t}{\sqrt{v_t}} = \text{sign}(\nabla f(x_t)) \).
- Such coordinate-balanced update is crucial for performance in neural network training.
Lion Update Rule

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

Key points:

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

\[ x_{t+1} = x_t + \epsilon \left( \beta_1 m_t - (1 - \beta_1) \nabla f(x_t) \right) \]

- 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

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

Key points:
- Use “decoupled” weight decay $\lambda x_t$.
- 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

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

• 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

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

- 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 \( x_{t+1} = x_t + \epsilon (\text{sign}(u_t) - \lambda x_t) \)
  - If \( |\lambda x_t| > 1 \geq |\text{sign}(u_t)| \), the decay term would dominate.
It turns out it solves a constrained optimization

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

- 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 \( x_{t+1} = x_t + \epsilon (\text{sign}(u_t) - \lambda x_t) \)
  - If \( |\lambda x_t| > 1 \geq |\text{sign}(u_t)| \), the decay term would dominate.
  - We can show, when \( \epsilon \lambda < 1 \),

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

where \( \mathcal{F} = \{x : \|\lambda x\|_\infty \leq 1\} \), under any notion of distance \( \text{dist}(\cdot) \).
How to show the convergence to optimum?

- Approach: Certify the convergence with a Lyapunov function

\[
\begin{align*}
    m_{t+1} &= \beta_1 m_t - (1 - \beta_1) \nabla f(x_t), \\
    x_{t+1} &= x_t + \epsilon m_{t+1},
\end{align*}
\]

- Continuous limit: the heavy ball dynamics

\[
\begin{align*}
    \dot{m}_t &= -\nabla f(x_t) - \gamma m_t, \\
    \dot{x}_t &= m_t,
\end{align*}
\]

The system monotonically decreases the following Hamiltonian function:

\[
H(x, m) = f(x) + \frac{1}{2} \|m\|^2.
\]

\[
\frac{d}{dt} H(x_t, m_t) = \nabla f(x_t)^\top \dot{x}_t + \nabla m_t^\top (\dot{m}_t - \gamma m_t) = -\gamma \|m_t\|^2 \leq 0.
\]
How to show the convergence to optimum?

• Approach: Certify the convergence with a Lyapunov function
• Consider, for example, the standard momentum method:

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

• Continuous limit: the heavy ball dynamics

$$\dot{m}_t = -\nabla f(x_t) - \gamma m_t, \quad \dot{x}_t = m_t$$
How to show the convergence to optimum?

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

\[ m_{t+1} = \beta_1 m_t - (1 - \beta_1) \nabla f(x_t), \quad x_{t+1} = x_t + \epsilon m_{t+1}, \]

• Continuous limit: the heavy ball dynamics

\[ \dot{m}_t = -\nabla f(x_t) - \gamma m_t, \quad \dot{x}_t = m_t \]

• The system monotonically decreases the following Hamiltonian function:

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

\[ \frac{d}{dt} H(x_t, m_t) = \partial_x H(x_t, m_t)^\top \dot{x}_t + \partial_m H(x_t, m_t)^\top \dot{m}_t 
\]
\[ = \nabla f(x_t)^\top \dot{x}_t + \gamma m^\top (\nabla f(x_t) + \gamma m_t) 
\]
\[ = -\gamma \| m_t \|^2 \leq 0. \]
Let $K$ be any convex function and $\nabla K$ its subgradient:

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

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

**Lion-K: A Generalization**
• 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 $\mathcal{K}$: $\mathcal{K}^*(x) = \sup_y (x^\top y - \mathcal{K}(y))$.

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

\[
\min_{x} F(x), \quad s.t. \quad x \in \text{dom}\mathcal{K}^*,
\]

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

• 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(\mathcal{K}(x))</th>
<th>(\nabla \mathcal{K}(x))</th>
<th>(\min_x f(x) + \mathcal{K}^*(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polyak Momentum [31]</td>
<td>(\mathcal{K}(x) = |x|_2^2 / 2, \gamma\lambda = 0, \varepsilon = 0)</td>
<td>(\text{sign}(x))</td>
<td>(\min f(x) \text{ s.t. } |x|_\infty \leq 1)</td>
</tr>
<tr>
<td>Nesterov Momentum [28]</td>
<td>(\mathcal{K}(x) = |x|_2^2 / 2, \gamma\lambda = 0)</td>
<td>(\text{sign}(x))</td>
<td>(\min f(x) \text{ s.t. } |x|_\infty \leq 1)</td>
</tr>
<tr>
<td>Signed Momentum [5]</td>
<td>(\mathcal{K}(x) = |x|_1, \varepsilon = 0, \lambda = 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Hamiltonian Descent [23]</td>
<td>(\varepsilon = 0, \lambda = 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Hamiltonian Descent for Composite Objectives</td>
<td>(\varepsilon = 0, \lambda &gt; 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Dual Space Preconditioning [24], Mirror Descent</td>
<td>(\varepsilon\gamma = 1, \lambda = 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Signed Gradient Descent [5]</td>
<td>(\mathcal{K}(x) = |x|_1, \varepsilon\gamma = 1, \lambda = 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Accelerated Mirror Descent [17]</td>
<td>(\gamma = 0, \varepsilon = 0, \lambda &gt; 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>Frank–Wolfe [11]</td>
<td>(\varepsilon\gamma = 1, \lambda &gt; 0)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Line ID</th>
<th>(\mathcal{K}(x))</th>
<th>(\nabla \mathcal{K}(x))</th>
<th>(\min_x f(x) + \mathcal{K}^*(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(|x|_1)</td>
<td>(\text{sign}(x))</td>
<td>(\min f(x) \text{ s.t. } |x|_\infty \leq 1)</td>
</tr>
<tr>
<td>2</td>
<td>(|x|_p)</td>
<td>(\text{sign}(x)</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>(\sum_{i} \max(</td>
<td>x_i</td>
<td>- e, 0))</td>
</tr>
<tr>
<td>4</td>
<td>(\sum_{i \leq i_{\text{cut}}}</td>
<td>x_i</td>
<td>)</td>
</tr>
<tr>
<td>5</td>
<td>(\sum_{i} \text{huber}_e(x_i))</td>
<td>(\text{clip}(x, -e, e) / e)</td>
<td>(\min f(x) + |x|<em>2^2 \text{ s.t. } |x|</em>\infty &lt; 1)</td>
</tr>
</tbody>
</table>

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 \(i_{\text{cut}}\) is an integer in \(\{1, \ldots, d\}\). The Huber loss is \(\text{huber}_e(x_i) = I(|x_i| \geq e)(|x_i| - \frac{e}{2}) + I(|x_i| < e)\frac{1}{2e}x_i^2\),
Lyapunov Function of Lion-$\mathcal{K}$

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

\[
\begin{align*}
\dot{m}_t &= -\alpha \nabla f(x_t) - \gamma m_t \\
\dot{x}_t &= \nabla \mathcal{K}(m_t - \varepsilon (\alpha \nabla f(x_t) + \gamma m_t)) - \lambda x_t.
\end{align*}
\]

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

\[
dist(x_t, \text{dom}\mathcal{K}^*) \leq \exp(-\lambda (t - s)) dist(x_s, \text{dom}\mathcal{K}^*), \quad \forall 0 \leq s \leq t.
\]
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:

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

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

\[
H(x, m) = \alpha f(x) + \frac{\gamma}{\lambda} \mathcal{K}^*(\lambda) + \frac{1 - \varepsilon \gamma}{1 + \varepsilon \lambda} (\mathcal{K}^*(\lambda x) + \mathcal{K}(m) - \lambda m^\top x)
\]

\[
\begin{align*}
\text{“potential” function} & \quad \text{“kinetic” energy}
\end{align*}
\]

• 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

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

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

\[
dist(x_{t+1}, \text{dom}\mathcal{K}^*) \leq \frac{1}{1 + \epsilon \lambda} dist(x_t, \text{dom}\mathcal{K}^*)
\]

• **[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

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

Hence, $H(x, m)$ decreases when $\epsilon$ is small.
Proof: Consider any ODE:

\[
\begin{align*}
\dot{m}_t &= U_t(x_t, m_t) \\
\dot{x}_t &= V_t(x_t, m_t).
\end{align*}
\]

\[
\frac{d}{dt} H(x_t, m_t) = \partial_m H(x_t, m_t) \top U_t(x_t, m_t) + \partial_x H(x_t, m_t) \top V_t(x_t, m_t).
\]
**Proof:** Consider any ODE:

\[
\begin{align*}
\dot{m}_t &= U_t(x_t, m_t) \\
\dot{x}_t &= V_t(x_t, m_t).
\end{align*}
\]

\[
\frac{d}{dt} H(x_t, m_t) = \partial_m H(x_t, m_t) \top U_t(x_t, m_t) + \partial_x H(x_t, m_t) \top V_t(x_t, m_t).
\]

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

\[
\begin{align*}
\partial_m H(x, m) &= -b \hat{U}_t(x, m) + c V_t(x, m), \\
\partial_x H(x, m) &= -a \hat{V}_t(x, m) - c U_t(x, m),
\end{align*}
\]

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

\[
\begin{align*}
\hat{V}_t(x, m) \top V_t(x, m) &\geq 0, \\
\hat{U}_t(x, m) \top U_t(x, m) &\geq 0.
\end{align*}
\]

Then

\[
\frac{d}{dt} H(x_t, m_t) = (-b \hat{U}_t + c V_t) \top U_t + (-a \hat{V}_t - c U_t) \top V_t
\]

\[
= -a \hat{V}_t \top V_t - b \hat{U}_t \top U_t \leq 0.
\]

**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 sign(·) to develop distributed optimization that only requires to communicate random bits.

Thank You!