Training Deep Learning Models with Norm-Constrained LMOs

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu,
Antonio Silveti-Falls†, Volkan Cevher

CRM Workshop - Montreal - May 28th, 2025

†CentraleSupélec, Université de Paris-Saclay, inria

A question...

Can we design an optimization algorithm which respects the natural geometry of neural networks?

A question...

Can we design an optimization algorithm which respects the natural geometry of neural networks?

(in such a way that we guarantee effective learning across different model scales)

Motivation

What has been done so far?

Stochastic Gradient Descent (SGD):

Input: $x^0 \in \mathcal{X}$, step sizes $\{\gamma\}$,
horizon $n \in \mathbb{N}^*$
for $k = 0, 1, \ldots, n-1$ **do**
  Sample ξ_k
  $g^k = \nabla f(x^k, \xi_k)$
  $x^{k+1} = x^k - \gamma g^k$
Output: x^n

- SGD uses a Euclidean geometry:

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma} \|x - x^k\|_2^2$$

In the beginning, there was SGD

Stochastic Gradient Descent (SGD):

Input: $x^0 \in \mathcal{X}$, step sizes $\{\gamma\}$, horizon $n \in \mathbb{N}^*$

for $k = 0, 1, \ldots, n-1$ **do**

  Sample ξ_k

  $g^k = \nabla f(x^k, \xi_k)$

  $x^{k+1} = x^k - \gamma g^k$

Output: x^n

- SGD uses a Euclidean geometry:

$$x^{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^d} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma} \|x - x^k\|_2^2$$

Two major improvements:

1. *On-the-fly adaptation*: Methods that adapt during training (AdaGrad, RMSprop, Adam, AdamW)
2. *A priori adaptation*: Methods designed with problem-specific geometry in mind (Bregman methods, Riemannian optimization, μP parameterizations, etc)

Stochastic Gradient Descent (SGD):

Input: $x^0 \in \mathcal{X}$, step sizes $\{\gamma\}$,
horizon $n \in \mathbb{N}^*$
for $k = 0, 1, \ldots, n-1$ **do**
  Sample ξ_k
  $g^k = \nabla f(x^k, \xi_k)$
  $x^{k+1} = x^k - \gamma g^k$
Output: x^n

- SGD uses a Euclidean geometry:

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma} \|x - x^k\|_2^2$$

Two major improvements:

1. *On-the-fly adaptation*: Methods that adapt during training (AdaGrad, RMSprop, Adam, AdamW)
2. *A priori adaptation*: Methods designed with problem-specific geometry in mind (Bregman methods, Riemannian optimization, μP parameterizations, etc)

Adam: on-the-fly adaptation

Adam:

Input: $x^0 \in \mathcal{X}$, step size γ, $\epsilon > 0$, momentum β_1, β_2, horizon $n \in \mathbb{N}^*$

for $k = 0, 1, \ldots, n-1$ **do**

Sample ξ_k

$g^k = \nabla f(x^k, \xi_k)$

$m^k = \beta_1 m^{k-1} + (1-\beta_1) g^k$

$v^k = \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2$

$\hat{m}^k = \frac{m^k}{1-\beta_1^k}$

$\hat{v}^k = \frac{v^k}{1-\beta_2^k}$

$x^{k+1} = x^k - \frac{\gamma}{\sqrt{\hat{v}^k}+\epsilon} \odot \hat{m}^k$

Output: x^n

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma* University of Amsterdam, OpenAI dpkingma@openai.com

Jimmy Lei Ba* University of Toronto jimmy@psi.utoronto.ca

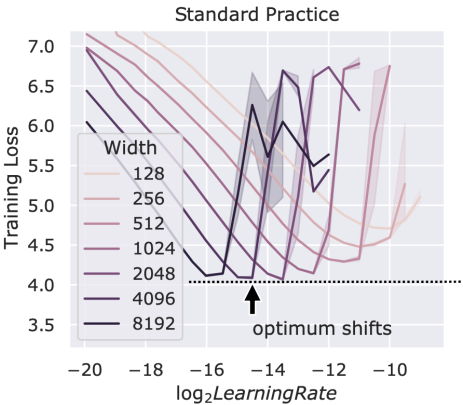
- Uses a Mahalanobis geometry:

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle \hat{m}^k, x - x^k \rangle + \frac{1}{2\gamma} \|x - x^k\|^2_{2,H_k}$$

with $\|x\|_{2,H_k} := \sqrt{\langle x, H_k x \rangle}$ and $H_k \approx \operatorname{diag}(\hat{v}^k + \varepsilon^2)$

- Adagrad, RMSProp, Adam, AdamW, etc adapt using a Mahalanobis norm.
- tl;dr: coordinate-wise adaptive step size using 2nd moment + momentum.

- Adam is "unaware" of the architecture, its dimensionality, matrix structure, etc.



(Figure from Yang et al)

- Because it's on-the-fly, Adam takes more memory when we scale our network (we have to keep track of + store the moments).

Failure of Adam to learn features as width scales

With standard parametrization (intialization + learning rate), we get stuck in the "lazy" regime if we scale width.

On Lazy Training in Differentiable Programming

Lénaïc Chizat
CNRS, Université Paris-Sud
Orsay, France
lenaic.chizat@u-psud.fr

Edouard Oyallon
CentraleSupelec, INRIA
Gif-sur-Yvette, France
edouard.oyallon@centralesupelec.fr

Francis Bach
INRIA, ENS, PSL Research University
Paris, France
francis.bach@inria.fr

Feature Learning in Infinite-Width Neural Networks

Greg Yang
Microsoft Research AI
gregyang@microsoft.com

Edward J. Hu*
Microsoft Azure AI
edwardhu@microsoft.com

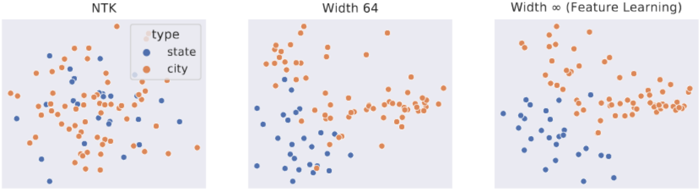


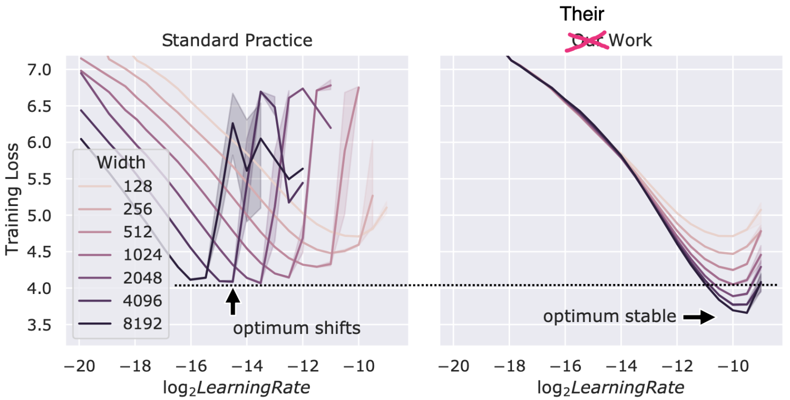
Figure 1: PCA of Word2Vec embeddings of top US cities and states, for NTK, width-64, and width-∞ feature learning networks (Definition 5.1). NTK embeddings are essentially random, while cities and states get naturally separated in embedding space as width increases in the feature learning regime.

(Figure from Tensor Programs IV paper by Yang et al.)

A priori adaptation via μP

μP: a certain layerwise step size and initialization that is scaled by dimensions to ensure

- the correct scaling behavior as the width goes to infinity (feature learning),
- (byproduct) Adam/SGD has hyperparameter transfer for the global step size.



μP is architecture aware (different scaling depending on dimensions)

$\rightarrow$ **this is a form of a priori adaptation.**

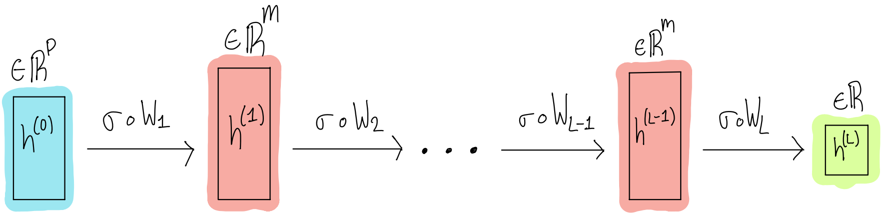
Plan

- Motivation
- **Feature Learning**
- Noneuclidean Optimization
- (unconstrained) Stochastic Conditional Gradient
- A natural geometry for neural networks
- Empirical Results
- Theoretical Results

We consider an L-layer fully-connected neural network with input $a \in \mathbb{R}^p$ and output $b \in \mathbb{R}$:

$$h^{(0)} = a \qquad h^{(\ell)}(h^{(\ell-1)}) = \sigma\left(\begin{bmatrix} \quad \mathbf{W}_\ell \quad \end{bmatrix}\begin{bmatrix} h^{(\ell-1)} \end{bmatrix}\right), \qquad b = h_x(a) = h^{(L)}(h^{(L-1)}(\ldots)).$$

- $x := [W_1, W_2, \ldots, W_L]$, $W_1 \in \mathbb{R}^{m \times p}$, $W_L \in \mathbb{R}^{1 \times m}$, $W_\ell \in \mathbb{R}^{m \times m}$ $\forall \ell \in \{2, \ldots, L-1\}$
- m is the *width* of the network



Feature Learning Desiderata

Definition (Feature Learning)

Let $\Delta h^{(\ell)}$ denote the feature change after one iteration of training, for the ℓ^{th} layer. We are in the feature learning regime if the following properties hold:

1. $\|h^{(\ell)}\|_{\text{RMS}} = \Theta(1), \quad \forall \ell \in [L]$ (stable forward pass),
2. $\|\Delta h^{(\ell)}\|_{\text{RMS}} = \Theta(1), \quad \forall \ell \in [L]$ (bounded, nontrivial feature update),

where the RMS norm is defined as $\|\cdot\|_{\text{RMS}} := \frac{1}{\sqrt{m}}\|\cdot\|_2$

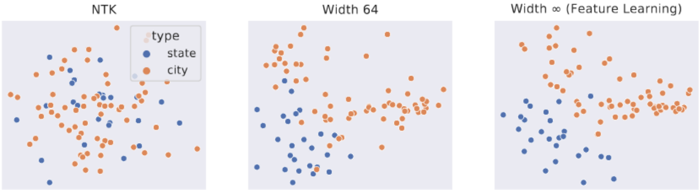


Figure 1: PCA of Word2Vec embeddings of top US cities and states, for NTK, width-64, and width-∞ feature learning networks (Definition 5.1). NTK embeddings are essentially random, while cities and states get naturally separated in embedding space as width increases in the feature learning regime.

Spectral Conditions for Feature Learning

Definition (Spectral Feature Learning (Yang et al 2023))

Given an L-layer NN, consider applying an update ΔW_ℓ to the weight matrix W_ℓ. If the spectral norms of the weights and the weight updates satisfy the following $\forall \ell \in \{2, \ldots, L-1\}$,

$$
\begin{aligned}
\|W_1\|_{\mathrm{RMS}_p \to \mathrm{RMS}_m} &= \Theta(1) \qquad & \|\Delta W_1\|_{\mathrm{RMS}_p \to \mathrm{RMS}_m} &= \Theta(1) \\
\|W_\ell\|_{\mathrm{RMS}_m \to \mathrm{RMS}_m} &= \Theta(1) \qquad & \|\Delta W_\ell\|_{\mathrm{RMS}_m \to \mathrm{RMS}_m} &= \Theta(1) \\
\|W_L\|_{\mathrm{RMS}_m \to \mathrm{RMS}_1} &= \Theta(1) \qquad & \|\Delta W_L\|_{\mathrm{RMS}_m \to \mathrm{RMS}_1} &= \Theta(1)
\end{aligned}
$$

then we have *feature-learning*.

- This spectral condition ensures that $\|h^{(\ell)}\|_{\mathrm{RMS}} = \Theta(1)$ and $\|\Delta h^{(\ell)}\|_{\mathrm{RMS}} = \Theta(1)$.

Spectral Conditions for Feature Learning

Definition (Spectral Feature Learning (Yang et al 2023))

Given an L-layer NN, consider applying an update ΔW_ℓ to the weight matrix W_ℓ. If the spectral norms of the weights and the weight updates satisfy the following $\forall \ell \in \{2, \ldots, L-1\}$,

$$\|W_1\|_{\text{op}} = \Theta\left(\sqrt{\frac{m}{p}}\right) \qquad \|\Delta W_1\|_{\text{op}} = \Theta\left(\sqrt{\frac{m}{p}}\right)$$
$$\|W_\ell\|_{\text{op}} = \Theta(1) \qquad \|\Delta W_\ell\|_{\text{op}} = \Theta(1)$$
$$\|W_L\|_{\text{op}} = \Theta\left(\sqrt{\frac{1}{m}}\right) \qquad \|\Delta W_L\|_{\text{op}} = \Theta\left(\sqrt{\frac{1}{m}}\right)$$

then we have *feature-learning*.

- This spectral condition ensures that $\|h^{(\ell)}\|_{\text{RMS}} = \Theta(1)$ and $\|\Delta h^{(\ell)}\|_{\text{RMS}} = \Theta(1)$.

Spectral Conditions for Feature Learning

Definition (Spectral Feature Learning (Yang et al 2023))

Given an L-layer NN, consider applying an update ΔW_ℓ to the weight matrix W_ℓ. If the spectral norms of the weights and the weight updates satisfy the following $\forall \ell \in \{2, \ldots, L-1\}$,

$$\begin{aligned} \|W_1\|_{\text{op}} &= \Theta\left(\sqrt{\frac{m}{p}}\right) & \|\Delta W_1\|_{\text{op}} &= \Theta\left(\sqrt{\frac{m}{p}}\right) \\ \|W_\ell\|_{\text{op}} &= \Theta(1) & \|\Delta W_\ell\|_{\text{op}} &= \Theta(1) \\ \|W_L\|_{\text{op}} &= \Theta\left(\sqrt{\frac{1}{m}}\right) & \|\Delta W_L\|_{\text{op}} &= \Theta\left(\sqrt{\frac{1}{m}}\right) \end{aligned}$$

then we have *feature-learning*.

- This spectral condition ensures that $\|h^{(\ell)}\|_{\text{RMS}} = \Theta(1)$ and $\|\Delta h^{(\ell)}\|_{\text{RMS}} = \Theta(1)$.
- This can be extended to rectangular matrices by requiring $\|\cdot\|_{\text{op}}$ scales like $\Theta\left(\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}\right)$.

$\implies$ **we need to control scaled operator norms layer-by-layer in the network to ensure feature learning as we scale width.**

An architecture-aware norm for neural networks

1. Cook up a noneuclidean norm based on the layerwise scaled operator norms.[1]

2. Build our optimization algorithm to around this norm (a priori adaptation).

[1] Similar idea proposed in Large et al. Modular Norm (2024) for deriving a layer-wise learning rate for SGD.

Plan

- Motivation
- Feature Learning
- **Noneuclidean Optimization**
- (unconstrained) Stochastic Conditional Gradient
- A natural geometry for neural networks
- Empirical Results
- Theoretical Results

Steepest Descent via Dual Norms

The update of SGD can be written

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma_k} \|x - x^k\|_2^2.$$

The update of SGD can be written

$$x^{k+1} = \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k\rangle + \frac{1}{2\gamma_k}\|x - x^k\|_2^2.$$

What if we change the norm?

The update of Steepest Descent can be written

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma_k} \|x - x^k\|^2.$$

What if we change the norm?

The update of Steepest Descent can be written

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma_k} \|x - x^k\|^2.$$

What if we change the norm?

This update has a closed-form solution using the dual norm $\|\cdot\|_*$,

$$x^{k+1} = x^k + \gamma_k \|g^k\|_* \operatorname{lmo}(g^k)$$

where lmo is the *linear minimization oracle*:

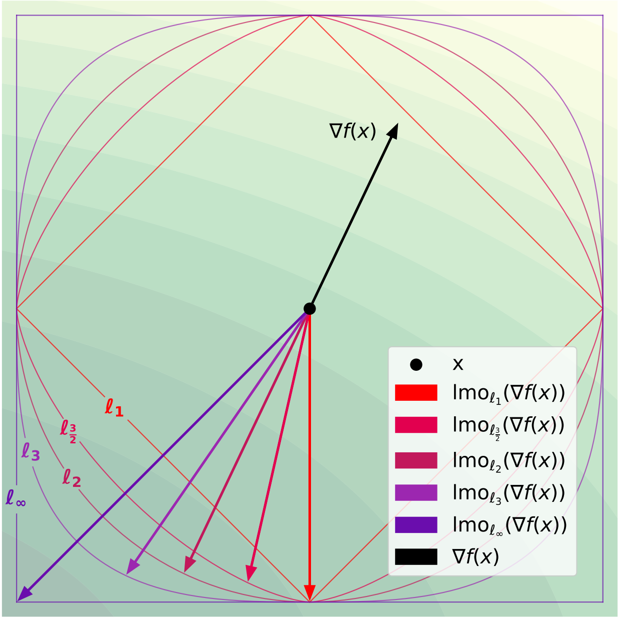
$$\operatorname{lmo}(g^k) \in \underset{s \in \mathcal{D}}{\operatorname{argmin}} \langle g^k, s \rangle = -\partial \|g^k\|_*$$

and $\mathcal{D}$ is the unit-ball for the norm $\|\cdot\|$.

Given a norm $\|\cdot\|$, the associated *linear minimization oracle* (lmo) gives back a direction least aligned with its input,

$$\text{lmo}(g) \in \underset{\{s:\ \|s\| \leq 1\}}{\text{argmin}} \langle g, s \rangle.$$

- The output of the lmo is always on the boundary of the ball.
- The lmo for the scaled ball is the scaled lmo for the unit ball.



Linear Minimization Oracles (lmo) for Norm Balls

If $\mathcal{D}$ is the unit-ball associated to a norm $\|\cdot\|$,
then $\mathrm{lmo}_{\mathcal{D}}(g) = -\partial\|g\|_*$ where $\|\cdot\|_*$ is the *dual norm*.

Primal Norm	**Linear Minimization Oracle (lmo)**
ℓ_2	$\mathrm{lmo}(g) = -\frac{g}{\|g\|_2}$
Dual Norm	**Steepest Descent ($-\|g\|_* \, \mathrm{lmo}(g)$)**
$\|\cdot\|_* = \|\cdot\|_2$	$-\|g\|_2 \left(-\frac{g}{\|g\|_2}\right) = g$

Steepest Descent in ℓ^2-norm recovers gradient descent/SGD.

Linear Minimization Oracles (lmo) for Norm Balls

If $\mathcal{D}$ is the unit-ball associated to a norm $\|\cdot\|$,
then $\mathrm{lmo}_{\mathcal{D}}(g) = -\partial\|g\|_*$ where $\|\cdot\|_*$ is the *dual norm*.

Primal Norm	**Linear Minimization Oracle (lmo)**
ℓ_∞	$\mathrm{lmo}(g) = -\mathrm{sign}(g)$
Dual Norm	**Steepest Descent ($-\|g\|_* \,\mathrm{lmo}(g)$)**
$\|\cdot\|_* = \|\cdot\|_1$	$-\|g\|_1\,(-\mathrm{sign}(g)) = \left(\sum_i g_i\right) \mathrm{sign}(g)$

Steepest Descent in ℓ^∞-norm recovers sign descent (up to step size).

Linear Minimization Oracles (lmo) for Norm Balls
If $\mathcal{D}$ is the unit-ball associated to a norm $\|\cdot\|$,
then $\mathrm{lmo}_{\mathcal{D}}(g) = -\partial\|g\|_*$ where $\|\cdot\|_*$ is the *dual norm*.

Primal Norm	**Linear Minimization Oracle (lmo)**
$\ell_2 \to \ell_2$ Operator Norm $\|\cdot\|_{\mathrm{op}}$	$\mathrm{lmo}(g) = -UV^T$ where $g = U\Sigma V^T$ (reduced SVD)
Dual Norm	**Steepest Descent ($-\|g\|_* \,\mathrm{lmo}(g)$)**
$\|\cdot\|_* = \|\cdot\|_{\mathrm{Nuc}}$	$-\|g\|_{\mathrm{Nuc}}\left(-UV^T\right) = \left(\sum_i \sigma_i(g)\right)\left(UV^T\right)$

Steepest Descent in $\|\cdot\|_{\mathrm{op}}$ recovers spectral descent/Muon (up to step size)

(we can compute this without SVD, just using matrix multiplication[2])

[2]Note: it requires more than one matrix multiplication to compute this.

Instead of Steepest Descent

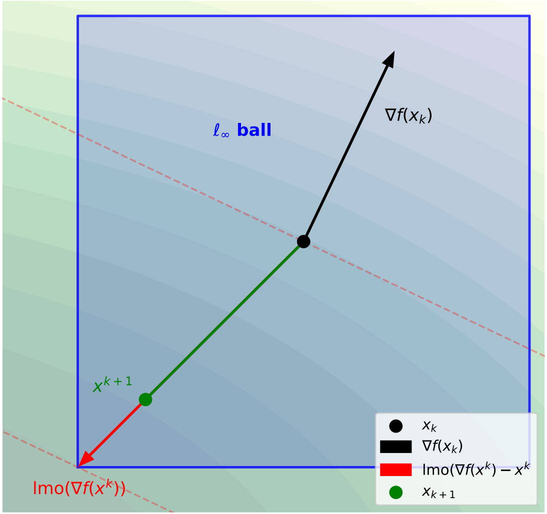
$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \frac{1}{2\gamma_k} \|x - x^k\|^2$$

which scales the update by $\|g^k\|_*$, we can directly use

$$x^{k+1} = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \langle g^k, x - x^k \rangle + \iota_{\gamma_k \mathcal{D}}(x - x^k)$$

to get

$$x^{k+1} = x^k + \gamma_k \mathrm{lmo}_{\mathcal{D}}(g^k).$$



The conditional gradient algorithm (also known as the Frank-Wolfe algorithm) solves constrained optimization problems:

$$\min_{x \in \mathcal{D}} f(x)$$

Conditional Gradient (CG):

Input: $x_0 \in \mathcal{D}$, step sizes $\{\gamma_k\}$ where $\gamma_k \in [0, 1]$, horizon $n \in \mathbb{N}^*$

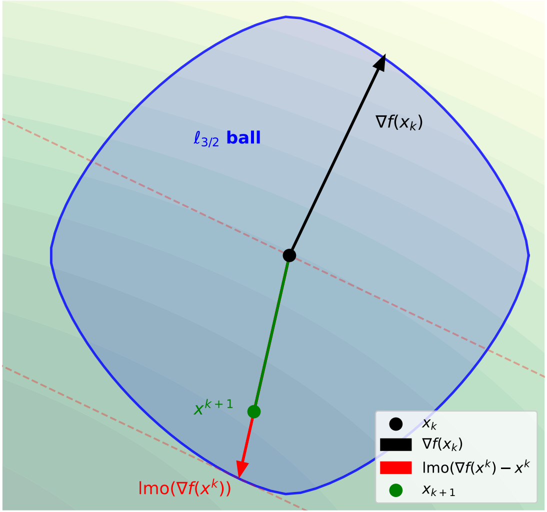
for $k = 0, 1, \ldots, n-1$ **do**

$s^k = \text{lmo}(\nabla f(x^k))$

$v^k = s^k - x^k$

$x^{k+1} = x_k + \gamma_k v^k$

Output: x^n



The conditional gradient algorithm (also known as the Frank-Wolfe algorithm) solves constrained optimization problems:

$$\min_{x \in \mathcal{D}} f(x)$$

Conditional Gradient (CG):

Input: $x_0 \in \mathcal{D}$, step sizes $\{\gamma_k\}$ where $\gamma_k \in [0, 1]$, horizon $n \in \mathbb{N}^*$

for $k = 0, 1, \ldots, n-1$ **do**

$s^k = \text{lmo}(\nabla f(x^k))$

$v^k = s^k - x^k$

$x^{k+1} = x_k + \gamma_k v^k$

Output: x^n

Plan

- Motivation
- Feature Learning
- Noneuclidean Optimization
- **(unconstrained) Stochastic Conditional Gradient**
- A natural geometry for neural networks
- Empirical Results
- Theoretical Results

(Unconstrained) Stochastic Conditional Gradient (uSCG/SCG):

Input: $x^1 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 1, 2, \ldots n - 1$ **do**

Sample ξ_k

$g^k = \nabla f(x^k, \xi_k)$

$d^k = (1 - \alpha_k)d^{k-1} + \alpha_k g^k$

$s^k = \operatorname{lmo}(d^k)$

$$v^k = \begin{cases} s^k & \text{uSCG} \\ s^k - x^k & \text{SCG} \end{cases}$$

$x^{k+1} = x^k + \gamma_k v^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

(Unconstrained) Stochastic Conditional Gradient (uSCG/SCG):

Input: $x^1 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 1, 2, \ldots n-1$ **do**

Sample ξ_k

$g^k = \nabla f(x^k, \xi_k)$

$d^k = (1-\alpha_k)d^{k-1} + \alpha_k g^k$

$s^k = \mathrm{lmo}(d^k)$

$$v^k = \begin{cases} s^k & \text{uSCG} \\ s^k - x^k & \text{SCG} \end{cases}$$

$x^{k+1} = x^k + \gamma_k v^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

- Momentum reduces variance in stochastic setting.

(Unconstrained) Stochastic Conditional Gradient (uSCG/SCG):

Input: $x^1 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 1, 2, \ldots n-1$ **do**

Sample ξ_k
$g^k = \nabla f(x^k, \xi_k)$
$d^k = (1 - \alpha_k) d^{k-1} + \alpha_k g^k$
$s^k = \text{lmo}(d^k)$
$$v^k = \begin{cases} s^k & \text{uSCG} \\ s^k - x^k & \text{SCG} \end{cases}$$
$x^{k+1} = x^k + \gamma_k v^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

- Momentum reduces variance in stochastic setting.
- The direction s^k has fixed norm of our choosing.

(Unconstrained) Stochastic Conditional Gradient (uSCG/SCG):

Input: $x^1 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 1, 2, \ldots n-1$ **do**

Sample ξ_k
$g^k = \nabla f(x^k, \xi_k)$
$d^k = (1 - \alpha_k) d^{k-1} + \alpha_k g^k$
$s^k = \operatorname{lmo}(d^k)$

$$v^k = \begin{cases} s^k & \text{uSCG} \\ s^k - x^k & \text{SCG} \end{cases}$$

$x^{k+1} = x^k + \gamma_k v^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

- Momentum reduces variance in stochastic setting.
- The direction s^k has fixed norm of our choosing.
- SCG is “just” uSCG with weight decay.

(Unconstrained) Stochastic Conditional Gradient (uSCG/SCG):

Input: $x^1 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 1, 2, \ldots n-1$ **do**

Sample ξ_k
$g^k = \nabla f(x^k, \xi_k)$
$d^k = (1 - \alpha_k) d^{k-1} + \alpha_k g^k$
$s^k = \mathrm{lmo}(d^k)$

$$v^k = \begin{cases} s^k & \text{uSCG} \\ s^k - x^k & \text{SCG} \end{cases}$$

$x^{k+1} = x^k + \gamma_k v^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

- Momentum reduces variance in stochastic setting.
- The direction s^k has fixed norm of our choosing.
- SCG is "just" uSCG with weight decay.
- uSCG solves the problem $\min_{x \in \mathbb{R}^d} f(x)$ while SCG solves the problem $\min_{x \in \mathcal{D}} f(x)$ where $\mathcal{D}$ is a norm ball.

Weight Decay and SCG

Deep learning community argues that Weight Decay should not simply be seen as Tikhonov regularization (Hutter et al.).

$$\text{GD with weight decay (decoupled):} \quad x^{k+1} = (1-\lambda)x^k - \gamma \nabla f(x^k)$$

$$\text{GD on Tikhonov problem (coupled):} \quad x^{k+1} = x^k - \gamma(\nabla f(x^k) + \lambda x^k)$$

However, these really are equivalent up to a rescaling/renaming of constants (but decoupled is known to work "better").

Weight Decay and SCG

Deep learning community argues that Weight Decay should not simply be seen as Tikhonov regularization (Hutter et al.).

$$\text{GD with weight decay (decoupled):} \quad x^{k+1} = (1-\lambda)x^k - \gamma\nabla f(x^k)$$
$$\text{GD on Tikhonov problem (coupled):} \quad x^{k+1} = x^k - \gamma(\nabla f(x^k) + \lambda x^k)$$

However, these really are equivalent up to a rescaling/renaming of constants (but decoupled is known to work "better").
In a noneuclidean setting, this point is *critical* because the lmo is nonlinear.

$$\begin{aligned}\text{uSCG + weight decay} \to \text{SCG:} \quad x^{k+1} &= (1-\lambda)x^k - \gamma\,\mathrm{lmo}(\nabla f(x^k)) \\ &= (1-\lambda)x^k - \lambda\,\mathrm{lmo}_{\gamma/\lambda}(\nabla f(x^k))\end{aligned}$$
$$\text{uSCG on Tikhonov problem:} \quad x^{k+1} = x^k - \gamma\,\mathrm{lmo}(\nabla f(x^k) + \lambda x^k)$$

Weight Decay and SCG

Deep learning community argues that Weight Decay should not simply be seen as Tikhonov regularization (Hutter et al.).

$$\text{GD with weight decay (decoupled):}\quad x^{k+1} = (1-\lambda)x^k - \gamma\nabla f(x^k)$$
$$\text{GD on Tikhonov problem (coupled):}\quad x^{k+1} = x^k - \gamma(\nabla f(x^k) + \lambda x^k)$$

However, these really are equivalent up to a rescaling/renaming of constants (but decoupled is known to work "better").
In a noneuclidean setting, this point is *critical* because the lmo is nonlinear.

$$\begin{aligned}\text{uSCG + weight decay} \to \text{SCG:}\quad x^{k+1} &= (1-\lambda)x^k - \gamma\,\mathrm{lmo}(\nabla f(x^k))\\ &= (1-\lambda)x^k - \lambda\,\underset{\gamma/\lambda}{\mathrm{lmo}}(\nabla f(x^k))\end{aligned}$$
$$\text{uSCG on Tikhonov problem:}\quad x^{k+1} = x^k - \gamma\,\mathrm{lmo}(\nabla f(x^k) + \lambda x^k)$$

The "correct" interpretation of Weight Decay in this context is that it transforms your unconstrained optimizer into a constrained optimizer, with implicit radii that are dictated by the chosen combination of step size γ and Weight Decay λ!

Plan

- Motivation
- Feature Learning
- Noneuclidean Optimization
- (unconstrained) Stochastic Conditional Gradient
- **A natural geometry for neural networks**
- Empirical Results
- Theoretical Results

Picking a Norm and Initializations

- If we can specify a norm $\|\cdot\|_{\alpha_\ell}$ for the input space and a norm $\|\cdot\|_{\beta_\ell}$ for the output spaces of each layer of our network, then this induces an operator norm for each layer.

Picking a Norm and Initializations

- If we can specify a norm $\|\cdot\|_{\alpha_\ell}$ for the input space and a norm $\|\cdot\|_{\beta_\ell}$ for the output spaces of each layer of our network, then this induces an operator norm for each layer.
- We can specify a norm for the whole set of parameters by taking

$$\|x\| = \max_{\ell \in [L]} \{\|W_\ell\|_{\alpha_\ell \to \beta_\ell}\}$$

Picking a Norm and Initializations

- If we can specify a norm $\|\cdot\|_{\alpha_\ell}$ for the input space and a norm $\|\cdot\|_{\beta_\ell}$ for the output spaces of each layer of our network, then this induces an operator norm for each layer.
- We can specify a norm for the whole set of parameters by taking

$$\|x\| = \max_{\ell \in [L]} \{\|W_\ell\|_{\alpha_\ell \to \beta_\ell}\}$$

- Spectral feature learning suggests taking the RMS norm on the input and output spaces of intermediary layers.
  $\to$ leads to a scaled $\ell^2 \to \ell^2$ operator norm $\|\cdot\|_{\mathrm{op}}$ on weight matrices

$$\|W\|_{\mathrm{RMS}\to\mathrm{RMS}} = \sqrt{\frac{d_{\mathrm{in}}}{d_{\mathrm{out}}}}\|W\|_{\mathrm{op}}.$$

  The lmo associated to the ball for this norm is given by the scaled matrix sign

$$\mathrm{lmo}(g) = -\sqrt{\frac{d_{\mathrm{out}}}{d_{\mathrm{in}}}}UV^T, \quad g = U\Sigma V^T.$$

Picking a Norm and Initializations

- If we can specify a norm $\|\cdot\|_{\alpha_\ell}$ for the input space and a norm $\|\cdot\|_{\beta_\ell}$ for the output spaces of each layer of our network, then this induces an operator norm for each layer.
- We can specify a norm for the whole set of parameters by taking

$$\|x\| = \max_{\ell \in [L]} \{\|W_\ell\|_{\alpha_\ell \to \beta_\ell}\}$$

- Spectral feature learning suggests taking the RMS norm on the input and output spaces of intermediary layers.
  $\rightarrow$ leads to a scaled $\ell^2 \rightarrow \ell^2$ operator norm $\|\cdot\|_{\mathrm{op}}$ on weight matrices

$$\|W\|_{\mathrm{RMS}\to\mathrm{RMS}} = \sqrt{\frac{d_{\mathrm{in}}}{d_{\mathrm{out}}}}\|W\|_{\mathrm{op}}.$$

  The lmo associated to the ball for this norm is given by the scaled matrix sign

$$\mathrm{lmo}(g) = -\sqrt{\frac{d_{\mathrm{out}}}{d_{\mathrm{in}}}}UV^T, \quad g = U\Sigma V^T.$$

The first and final layers require more thought!

Norms for input layers

The operator norm chosen for the initial layer differs from the intermediary layers, depending on the task (NLP, images, etc).

vs

Large Language Models (LLMs), such as GPT-3 and GPT-4, utilize a process called tokenization. Tokenization involves breaking down text into smaller units, known as tokens, which the model can process and understand. These tokens can range from individual characters to entire words or even larger chunks, depending on the model. For GPT-3 and GPT-4, a Byte Pair Encoding (BPE) tokenizer is used. BPE is a subword tokenization technique that allows the model to dynamically build a vocabulary during training, efficiently representing common words and word fragments. Although the core tokenization process remains similar across different versions of these models, the specific implementation can vary based on the model's architecture and training objectives.

For image domains, we use the RMS norm which gives the scaled operator norm for the initial layer.

For language tasks, the input z is usually a 1-hot encoded vector so

$$\|z\|_2 = \|z\|_1 = \|z\|_\infty = 1$$

identically. This means

$$\|W_1\|_{2\to\mathrm{RMS}} = \|W_1\|_{1\to\mathrm{RMS}} = \|W_1\|_{\infty\to\mathrm{RMS}}$$

on this restricted domain.

Parameter	W_1 (1-hot encoded input)		
Norm	$2 \to \mathrm{RMS}$	$1 \to \mathrm{RMS}$	$1 \to \infty$
LMO	$-\sqrt{d_{\mathrm{out}}}\,UV^\top$	$\mathrm{col}_j(W_1) \mapsto -\sqrt{d_{\mathrm{out}}}\,\frac{\mathrm{col}_j(W_1)}{\|\mathrm{col}_j(W_1)\|_2}$	$-\mathrm{sign}(W_1)$
Init.	Semi-orthogonal	Column-wise normalized Gaussian	Random sign

Norms for output layers

- We have no restriction to bound the output in RMS norm; instead we consider bounding the maximal entry using L_∞.

Norms for output layers

- We have no restriction to bound the output in RMS norm; instead we consider bounding the maximal entry using L_∞.
- We can bound $\|W\|_{\mathrm{RMS}\to\infty} \leq \frac{1}{d_{\mathrm{in}}}\|W\|_{1\to\infty}$ which gives us a scaled sign lmo for the last layer.

Norms for output layers

- We have no restriction to bound the output in RMS norm; instead we consider bounding the maximal entry using L_∞.
- We can bound $\|W\|_{\mathrm{RMS}\to\infty} \leq \frac{1}{d_{\mathrm{in}}}\|W\|_{1\to\infty}$ which gives us a scaled sign lmo for the last layer.

Parameter	W_L		
Norm	$\mathrm{RMS} \to \mathrm{RMS}$	$\mathrm{RMS} \to \infty$	$1 \to \infty$
LMO	$-\sqrt{d_{\mathrm{out}}/d_{\mathrm{in}}}\,UV^\top$	$\mathrm{row}_i(W_L) \mapsto -\frac{1}{\sqrt{d_{\mathrm{in}}}}\frac{\mathrm{row}_i(W_L)}{\|\mathrm{row}_i(W_L)\|_2}$	$-\frac{1}{d_{\mathrm{in}}}\mathrm{sign}(W_L)$
Init.	Semi-orthogonal	Row-wise normalized Gaussian	Random sign

We recommend the following norms (First layer $\rightarrow$ Intermediary layers $\rightarrow$ Last layer):

- image domains: Spectral $\rightarrow$ Spectral $\rightarrow$ Sign
- 1-hot input: ColNorm or Sign $\rightarrow$ Spectral $\rightarrow$ Sign

We recommend the following norms (First layer → Intermediary layers → Last layer):

- image domains: Spectral → Spectral → Sign
- 1-hot input: ColNorm or Sign → Spectral → Sign

We refer to the instantiation of uSCG and SCG using operator norms as UNCONSTRAINED SCION and SCION respectively, which stands for

Stochastic **C**onditional grad**I**ent with **O**perator **N**orms
Scion

Plan

- Motivation
- Feature Learning
- Noneuclidean Optimization
- (unconstrained) Stochastic Conditional Gradient
- A natural geometry for neural networks
- **Empirical Results**
- Theoretical Results

3B NanoGPT Training

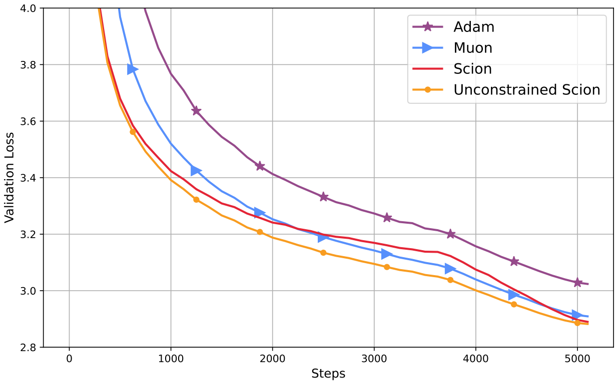


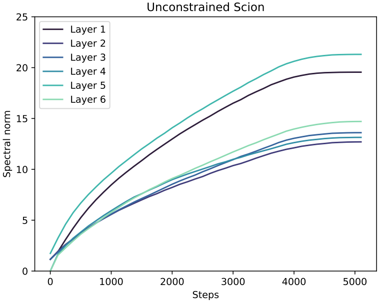
Table 5. Validation loss on a 3B parameter GPT model.

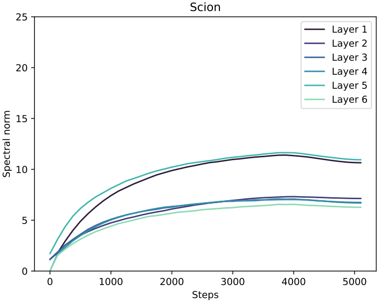
Adam	Muon	UNCONSTRAINED SCION	SCION
3.024	2.909	**2.882**	2.890

(Sign→Spectral→Sign)

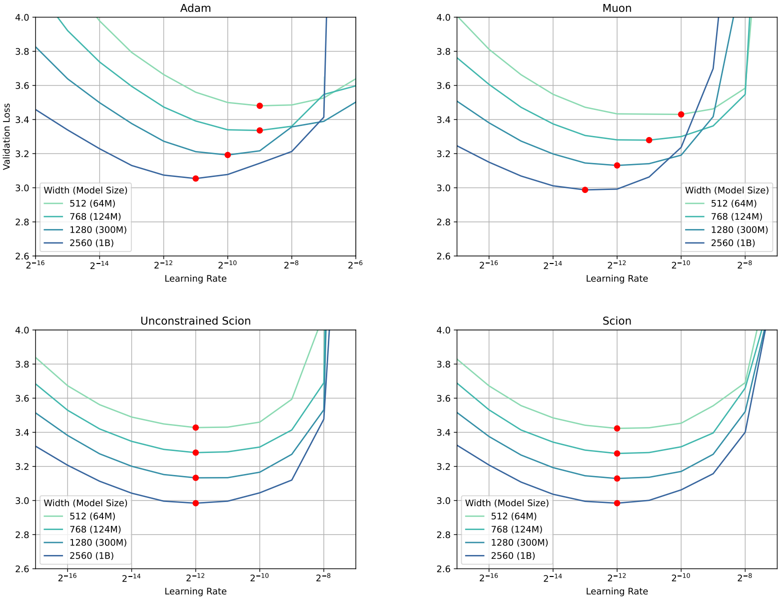
Let ρ be the radius of the set $\mathcal{D}$ that is used to define lmo. Both uSCG and SCG provide control over the norm of the output $\bar{x}^n$:

- SCG Guarantees $\|\bar{x}^n\| \leq \rho$
- uSCG Guarantees $\|\bar{x}^n\| \leq \rho \sum_{k=0}^{n-1} \gamma_k$

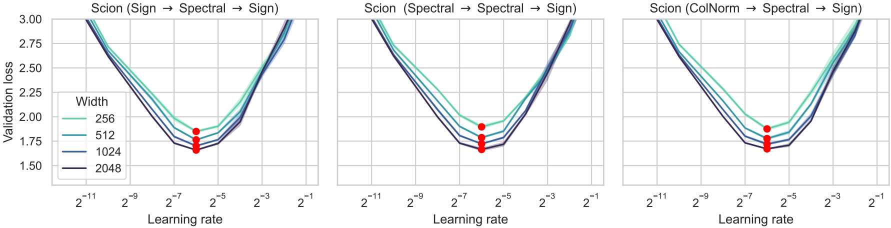




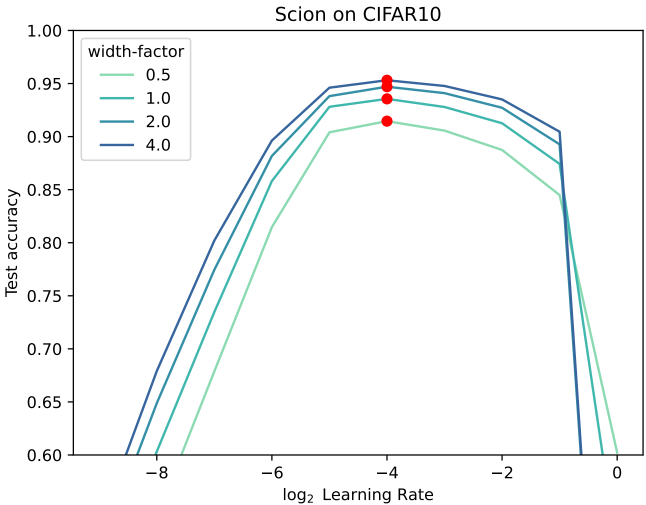
Hyperparameter Transfer: GPT Training



Shallow (3 layers) GPT on Shakespeare dataset.



All 3 admit hyperparameter transfer.



Optimal step size transfer across width in a convolutional NN trained to classify with CIFAR10.

Plan

- Motivation
- Feature Learning
- Noneuclidean Optimization
- (unconstrained) Stochastic Conditional Gradient
- A natural geometry for neural networks
- Empirical Results
- **Theoretical Results**

To analyze the algorithm, we consider the class of problems

$$\min_{x \in \mathcal{X}} f(x) := \mathbb{E}_\xi[f(x, \xi)]$$

where

- $\mathcal{X}$ is either $\mathbb{R}^d$ (unconstrained) or $\mathcal{D}$ (constrained), with

$$\mathcal{D} := \{x \colon \|x\| \leq \rho\}.$$

- $\mathbb{E}_\xi[f(\cdot, \xi)]$ is Lipschitz-smooth with respect to some norm.
- We have access to a stochastic first-order oracle $\nabla f(\cdot, \xi)$ which is unbiased

$$\mathbb{E}_\xi[\nabla f(\cdot, \xi)] = \nabla f(\cdot)$$

  and has bounded variance

$$\mathbb{E}_\xi[\|\nabla f(\cdot, \xi) - \nabla f(\cdot)\|_2^2] \leq \sigma^2.$$

Convergence Results for fixed α

Let ρ be the radius of the set $\mathcal{D}$ used in the lmo.

Theorem (Convergence rate for uSCG with constant α)

Let $n \in \mathbb{N}^*$ and let $\bar{x}^n$ be the output of uSCG with $\alpha \in (0,1)$ and constant step size $\gamma = \frac{1}{\sqrt{n}}$. Then,

$$\mathbb{E}[\|\nabla f(\bar{x}^n)\|_*] \leq O\left(\frac{L\rho}{\sqrt{n}} + \sigma\right)$$

Theorem (Convergence rate for SCG with constant α)

Let $n \in \mathbb{N}^*$ and let $\bar{x}^n$ be the output of SCG with $\alpha \in (0,1)$ and constant step size $\gamma = \frac{1}{\sqrt{n}}$. Then, for all $u \in \mathcal{D}$,

$$\mathbb{E}[\langle \nabla f(\bar{x}^n), \bar{x}^n - u\rangle] \leq O\left(\frac{L\rho^2}{\sqrt{n}} + \sigma\right)$$

$\Longrightarrow$ convergence to a noise-dominated region induced by σ.

Let ρ be the radius of the set $\mathcal{D}$ used in the lmo.

Theorem (Convergence rate for uSCG with vanishing α_k)

Let $n \in \mathbb{N}^*$ and let $\bar{x}^n$ be the output of uSCG with $\alpha_k = 1/\sqrt{k}$ and constant step size $\gamma = \frac{3}{4n^{3/4}}$. Then,

$$\mathbb{E}[\|\nabla f(\bar{x}^n)\|_*] \leq O\left(\frac{1}{n^{1/4}} + \frac{L\rho}{n^{3/4}}\right)$$

Theorem (Convergence rate for SCG with vanishing α_k)

Let $n \in \mathbb{N}^*$ and let $\bar{x}^n$ be the output of SCG with $\alpha_k = 1/\sqrt{k}$ and constant step size $\gamma = \frac{3}{4n^{3/4}}$. Then, for all $u \in \mathcal{D}$,

$$\mathbb{E}[\langle \nabla f(\bar{x}^n), \bar{x}^n - u\rangle] \leq O\left(\frac{1}{n^{1/4}} + \frac{L\rho^2}{n^{3/4}}\right)$$

$\Longrightarrow$ "convergence to a first-order critical point" for either the unconstrained (uSCG) or the constrained (SCG) problem.

Algorithm	α	**Norm**	$\text{lmo}(d)$ **Formula**
Normalized SGD	1	Euclidean $\|\cdot\|_2$	$-\frac{d}{\|d\|_2}$
Normalized SGD with momentum	$]0, 1]$	Euclidean $\|\cdot\|_2$	$-\frac{d}{\|d\|_2}$
SignSGD	1	Max-norm $\|\cdot\|_\infty$	$-\text{sign}(d)$
Signum (SignSGD with momentum)	$]0, 1]$	Max-norm $\|\cdot\|_\infty$	$-\text{sign}(d)$
Muon*	$]0, 1]$	$\ell^2 \to \ell^2$ operator-norm $\|\cdot\|_{\text{op}}$	$-UV^T$, $d = U\Sigma V^T$

Our framework generalizes these algorithms through norm selection and momentum parameter.

- Muon blogpost: Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein (Dec. 2024)
- Kimi Moonshot AI: many (Feb. 2025)
- PSGD: Omead Pooladzandi and Xi-Lin Li (Feb. 2024)

Also MARS (related via STORM estimator of gradient) that will be talked about.

arXiv:2502.07529, also at ICML 2025 (Spotlight)

arXiv > cs > arXiv:2502.07529

Search...

Help | A

Computer Science > Machine Learning

[Submitted on 11 Feb 2025]

Training Deep Learning Models with Norm-Constrained LMOs

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, Volkan Cevher

In this work, we study optimization methods that leverage the linear minimization oracle (LMO) over a norm-ball. We propose a new stochastic family of algorithms that uses the LMO to adapt to the geometry of the problem and, perhaps surprisingly, show that they can be applied to unconstrained problems. The resulting update rule unifies several existing optimization methods under a single framework. Furthermore, we propose an explicit choice of norm for deep architectures, which, as a side benefit, leads to the transferability of hyperparameters across model sizes. Experimentally, we demonstrate significant speedups on nanoGPT training without any reliance on Adam. The proposed method is memory-efficient, requiring only one set of model weights and one set of gradients, which can be stored in half-precision.

Subjects: **Machine Learning (cs.LG)**; Optimization and Control (math.OC)

Cite as: arXiv:2502.07529 **[cs.LG]**

(or arXiv:2502.07529v1 **[cs.LG]** for this version)

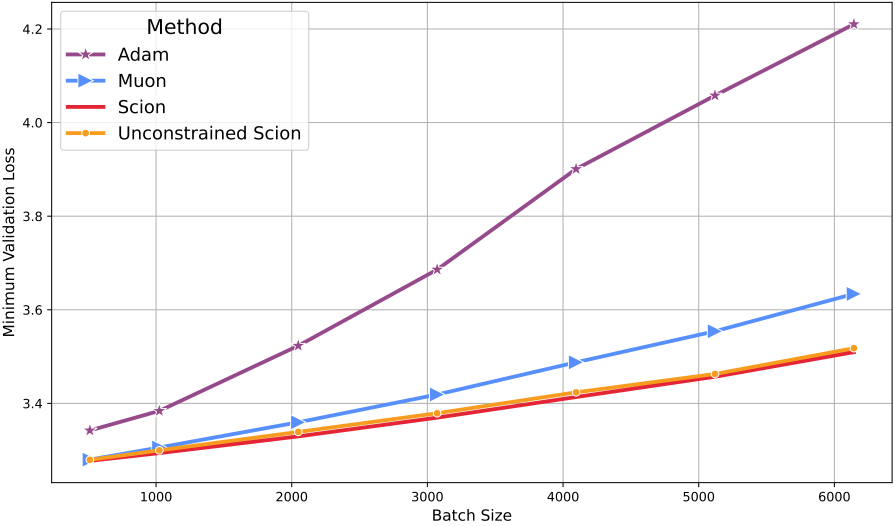
https://doi.org/10.48550/arXiv.2502.07529

github: https://github.com/LIONS-EPFL/scion

Extension to (L_0, L_1) smoothness with clipping to appear this week; we are working on more extensions.

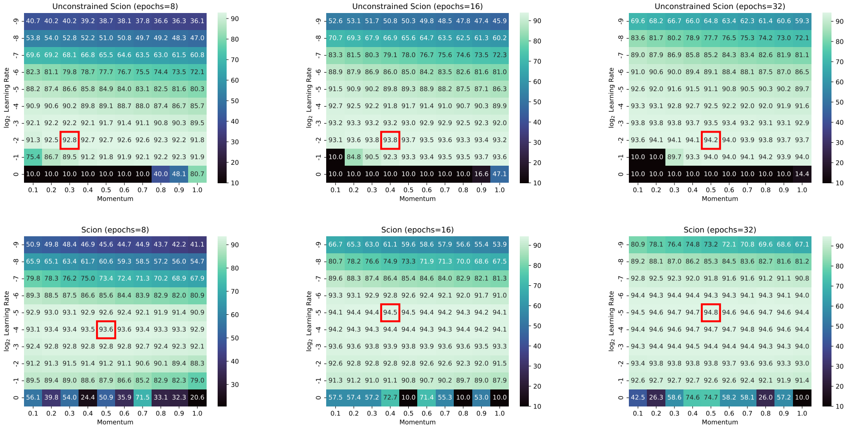
Effect of batch size

Batchsize sensitivity on NanoGPT (124M).



Scion is less sensitive to batch increases (for a fixed token budget).

Tuning the momentum on CIFAR10



Averaged **LMO** direction**N**al **D**escent (ALMOND):

Input: $x^0 \in \mathcal{D}$, step sizes $\{\gamma_k\}$, momentum $\{\alpha_k\}$, horizon $n \in \mathbb{N}$
Initialize $d^0 = 0$
for $k = 0, 1, 2, \ldots n-1$ **do**

$g^k = \nabla f(x^k, \xi_k)$

$d^k = (1-\alpha_k)d^{k-1} + \alpha_k \mathrm{lmo}(g^k)$

$x^{k+1} = x^k + \gamma_k d^k$

Output: $\bar{x}^n$ selected uniformly at random among all iterates.

Not competitive empirically. Theoretically, can only show convergence to a noise dominated region.

In the case where $x = [W_1, \ldots, W_L]$ and we want to assign a norm $\|\cdot\|_{\{\ell\}}$ to each W_ℓ for $\ell \in [L]$, we can take the *max*-norm,

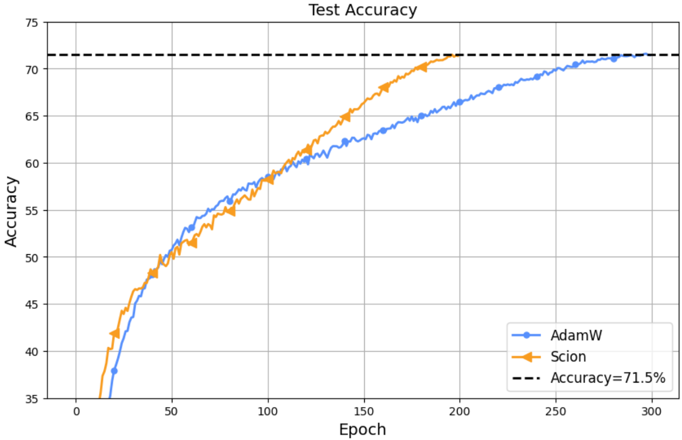
$$\|x\| := \max \left\{ \|W_1\|_{\{1\}}, \ldots, \|W_L\|_{\{L\}} \right\}$$

so that lmo(g) with respect to this norm is *separable* across the parameters g_ℓ:

$$\operatorname{lmo}(g) = \operatorname{lmo}([g_1, \ldots, g_L]) = [\underset{\{1\}}{\operatorname{lmo}}(g_1), \ldots, \underset{\{L\}}{\operatorname{lmo}}(g_L)]$$

with each $\operatorname{lmo}_{\{\ell\}}$ corresponding to the lmo over the ball induced by the norm $\|\cdot\|_{\{\ell\}}$.

Image Transformers (DeiT-Base)



Much more sample-efficient!