

# Training Large Language Models (LLMs)

Prof. Volkan Cevher  
[volkan.cevher@epfl.ch](mailto:volkan.cevher@epfl.ch)

## *Lecture 2: Optimization*

Laboratory for Information and Inference Systems (LIONS)  
École Polytechnique Fédérale de Lausanne (EPFL)

EE-628 (Spring 2025)

**lions@epfl**



# License Information for Training LLMs Slides

- ▶ This work is released under a [Creative Commons License](#) with the following terms:
- ▶ **Attribution**
  - ▶ The licensor permits others to copy, distribute, display, and perform the work. In return, licensees must give the original authors credit.
- ▶ **Non-Commercial**
  - ▶ The licensor permits others to copy, distribute, display, and perform the work. In return, licensees may not use the work for commercial purposes – unless they get the licensor's permission.
- ▶ **Share Alike**
  - ▶ The licensor permits others to distribute derivative works only under a license identical to the one that governs the licensor's work.
- ▶ [Full Text of the License](#)

# Outline

- ▶ Scaling laws
- ▶ Feature learning
- ▶ Tensor programs &  $\mu$ P initialization and  $\mu$ -transfer (hyperparameter transfer)
- ▶ Optimization algorithms
- ▶ Systems optimizations

# Large language models (LLMs): A story of scaling

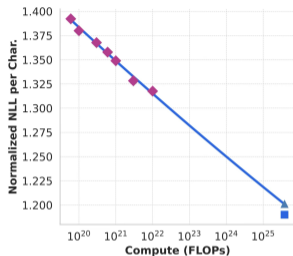


Figure: Scaling law forecast for ARC (Llama 3)

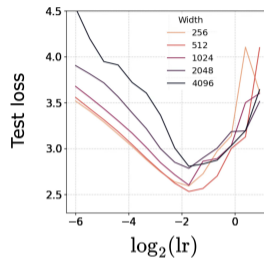


Figure: Naive scaling of Mamba worsens the performance

- More compute through larger models, more training data, or longer training improves performance.
- Is “scale” all we need?
  - ▶ Need guidance on the adjustments of models, training procedures, and hyperparameters when scaling up.
  - ▶ Need *scaling rules* that allow predictability and optimality!

## On predictability: Scaling laws

### Definition (Neural scaling law [48])

Neural scaling laws describe how neural network performance changes as key factors are scaled up or down.

**Remarks:** ○ In general, neural network (pre)training can be characterized by four factors:

1. Size of the model ( $N$ ): number of parameters
2. Size of the training dataset ( $D$ ): number of samples or tokens
3. Compute ( $C$ ): measured in FLOPs (FLoating-point OPerations)
4. Test loss after training ( $L$ ): generalization performance

## Scale and performance

- Increasing compute, dataset and model size improves performance, particularly in language models.

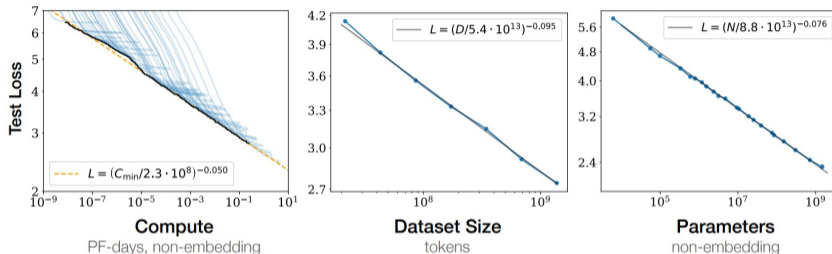


Figure: Neural scaling laws for language modeling [60].

- Remarks:**
- Language modeling performance improves smoothly as we “scale.”
  - For optimal performance all three factors must be scaled up in tandem.
  - Test performance has a power-law relationship with each individual factor.
  - These are empirical curve fits rather than scaling “theory.”

## Kaplan scaling laws [60]: Power law relationships

- Test loss exhibits a power law relationship with available resources.

### Scaling Laws [60]

1. For models with a limited number of parameters, trained to convergence on sufficiently large datasets:

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \sim 0.076, \quad N_c \sim 8.8 \times 10^{13} \text{ (parameters)}$$

2. For large models trained with a limited dataset with early stopping:

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad \alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13} \text{ (input samples)}$$

3. When training with a limited amount of compute, a sufficiently large model, and a sufficiently small batch size (making optimal use of compute):

$$L(C_{\min}) = \left(\frac{C_{\min}^c}{C_{\min}}\right)^{\alpha_C^{\min}}, \quad \alpha_C^{\min} \sim 0.050, \quad C_{\min}^c \sim 2.3 \times 10^8 \text{ (PF-days)}$$

**Remarks**▷ [60] estimates the constants through extensive empirical analysis on language models.

- Scaling laws are widely observed: speech [49], image classification [123], reinforcement learning [101].

## Upshot of a scaling law: The compute allocation decision

### A compute allocation scheme proposed by [60]

Given a fixed budget of compute  $C$ , the model size  $N$ , the batch size  $B$ , and the training steps  $S$  should scale as  $N \propto C^{0.71}$ ,  $B \propto C^{0.24}$ , and  $S \propto C^{0.03}$ , respectively.

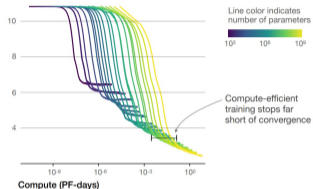


Figure: Efficient training should stop before convergence.

- Observations:**
- Most of the compute should be invested on model size.
  - The scheme suggests that the training steps should almost be held fixed when scaling.
  - Large models should be under-trained, if trained efficiently.
  - The allocation scheme serves as a scaling rule to guide model training.

## Chinchilla scaling law and its allocation scheme

### Chinchilla Scaling Law [50]

Hoffman et al. [50] propose the following approach combining model size and data size:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta},$$

where  $E$  is the irrecoverable error and  $A, B, \alpha, \beta$  are estimated constants.

- Remarks:**
- The functional form of  $L(N, D)$  proposed by [60] is  $L(N, D) = \left[ \left( \frac{N_c}{N} \right)^{\frac{\alpha N}{\alpha D}} + \frac{D_c}{D} \right]^{\alpha D}$ .
  - This rule recovers the first two scalings on page 7 when we take  $D$  ( $N$ , resp.) to infinity.

### Chinchilla Allocation Scheme

When scaling up compute  $C$ , the model size  $N$  and dataset size  $D$  should increase at the same rate.

- Remarks:**
- This is drastically different from what Kaplan's model projects [60]!
  - Hoffmann et al. [50] validated this result through three different numerical procedures.
  - Hoffmann et al. [50] did not account for batch size scaling like Kaplan et al. [60] did.

## On the existence of a critical batch-size for training

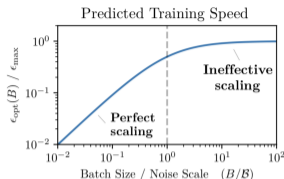


Figure: The scaling of optimal learning rate with respect to the batch size saturates [80].

- Large batch sizes are desirable, as higher parallelism leads to shorter serial training time [120].
- The speedup gained from large batch sizes saturates as the batch size increases [80].
  - ▶  $\epsilon_{\max}$ : step size maximizing loss drop given the true gradient.
  - ▶  $\epsilon_{\text{opt}}(B)$ : step size maximizing *expected* loss drop given the noisy gradient estimated from a  $B$ -sized batch.
  - ▶  $\mathcal{B}$  (see [120]): a quantity that is correlated with noise-to-signal ratio of gradient estimation.
  - ▶  $B = \mathcal{B}$ : optimal batch size; the gain in training speed (step size) significantly decreases if  $B > \mathcal{B}$ .
  - ▶ The above figure was derived for SGD, but the insight may be (rigorously) extrapolated for ADAM [72].
- Unfortunately, a larger batch size often lead to worse generalization [62].

## Scale and performance: a detour for emergent capabilities

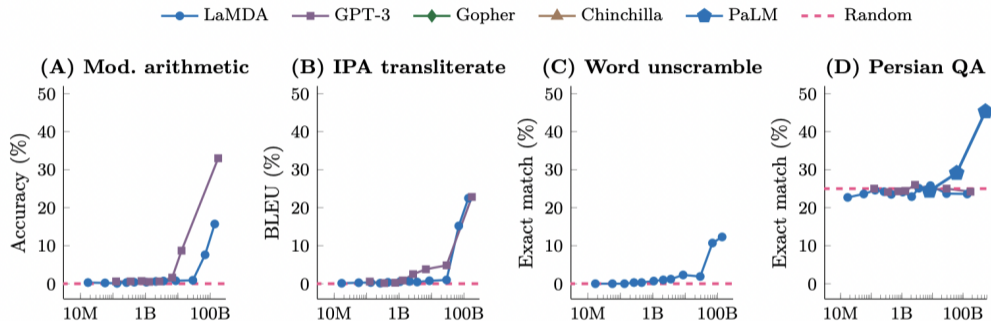


Figure: Some LLM capabilities emerge abruptly with scale [110].

- Discontinuity manifested in emergence may be an artifact of the used metrics being discontinuous [102].

## The story of scale continues beyond pre-training

- There is increasing interest in post-training scaling as well as test-time scaling.

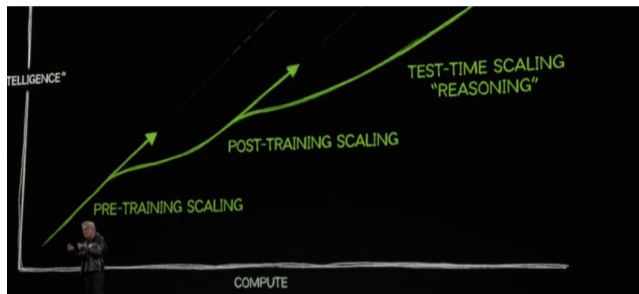


Figure: NVIDIA CEO Jensen Huang discusses AI scaling laws during CES 2025 [source].

- **Post-training:** finetuning [125], quantization [13], pruning [16], distillation [12], etc.
- **Test-time:** thinking time [87], dynamic model adjustment [33], etc.

## Scaling laws (i.e., curve fits) vs. Scaling theory

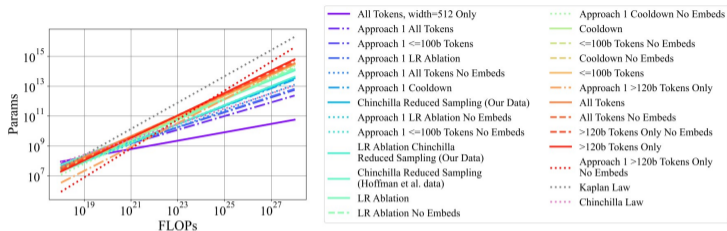


Figure: McLeish et al. [81] show that scaling law slope is sensitive to many experimental designs

- o Kaplan et al. [60] and Hoffmann et al. [50] extracted different scaling laws, but neither of them is wrong.
- o Tweaking seemingly innocuous experimental design changes the scaling law slope [81].

### Remarks:

- o Some notable settings that affect scaling laws are as follows:
  - ▶ parameter counting rule (i.e., include or not include embedding parameters)
  - ▶ width and depth ratio of the model
  - ▶ learning rate scheduler
  - ▶ data points sampled for numerical fits

## Towards scaling theory beyond curve fitting

- Desiderata for the theory and methodology:
  - ▶ Predictability: Project with certainty regarding how performance improves with scale.
  - ▶ Optimality: Provide theoretical guidance on the training of highly performant models.
- Our methodology will be based on the following:
  - ▶ Investigate the impact of taking width and/or depth to infinity (see the next slide for examples).
  - ▶ Focus on intuitive objectives, such as training stability and effectiveness of updates.
  - ▶ Develop algorithms that exploit and adapt to the structures in the training formulations.

## Limits taken in deep learning theories

- Infinite width, finite depth:

- ▶ Neural network Gaussian processes correspondence: Neal et al. [88], Daniely et al. [20], Lee et al. [67], Matthews et al. [79], Yang et al. [115], Novak et al. [93].
- ▶ Neural tangent kernel (NTK) limits under different architectures: Arora et al. [4], Du et al. [21], Litwin-Kumar et al. [73], Alemohammad et al. [1], Hron et al. [51], Huang et al. [53].
- ▶ Mean-field/ $\mu P$  limits: Chizat et al. [17], Mei et al. [82], Rotskoff et al. [100], Sirignano et al. [105], Araújo et al. [3], Fang et al. [25], Yang et al. [117].

- Infinite width, then infinite depth:

- ▶ Poole et al. [97], Pennington et al. [94], Chen et al. [15], Hanin [39], Hanin and Rolnick [41], Hayou et al. [44], Hayou [43], Hayou and Yang [45], Yang et al. [118].

- Infinite width and infinite depth:

- ▶ Hanin and Nica [40], Hu and Huang [52], Li et al. [71], Noci et al. [90], Noci et al. [91], Bordelon et al. [11].

## Setting the stage with the key ingredient: NN architectures

◦ We consider an  $L$ -layer fully-connected neural network with input  $\mathbf{a} \in \mathbb{R}^p$  and output  $b \in \mathbb{R}^1$ :

$$h^{(0)}(\mathbf{a}) = \mathbf{a},$$

$$h^{(l)}(h^{(l-1)}) = \sigma \left( \underbrace{\begin{bmatrix} \mathbf{X}_l \end{bmatrix} \begin{bmatrix} h^{(l-1)} \end{bmatrix}}_{\text{pre-activation } g^l} \right), \quad (L\text{-Layer NN})$$

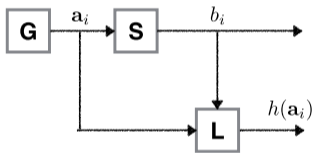
$$b = h_{\mathbf{x}}(\mathbf{a}) = h^{(L)}(h^{(L-1)}(\dots)) = \frac{1}{\alpha} \sigma \left( \mathbf{X}_L h^{(L-1)}(\dots) \right), \quad \mathbf{x} := [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_L].$$

- ▶ Architecture:  $m$  is the width and  $\alpha$  is the output scaling factor.
- ▶ Parameters:  $\mathbf{X}_1 \in \mathbb{R}^{m \times p}$ ,  $\mathbf{X}_L \in \mathbb{R}^{1 \times m}$ ,  $\mathbf{X}_l \in \mathbb{R}^{m \times m}$  for  $l = 2, 3, \dots, L - 1$  (weights).
- ▶ Initialization:  $\mathbf{X}_1 \sim \mathcal{N}(0, \beta_1^2)$ ,  $\mathbf{X}_L \sim \mathcal{N}(0, \beta_L^2)$ ,  $\mathbf{X}_l \sim \mathcal{N}(0, \beta^2)$  for  $l = 2, 3, \dots, L - 1$  (weights).
- ▶ Activation function ReLU:  $\sigma(\cdot) = \max(\cdot, 0) : \mathbb{R} \rightarrow \mathbb{R}$ .
- ▶ Without loss of generality, we will avoid the bias variables in the sequel.

## Another key ingredient: Loss function

### Definition (Loss function)

A **loss function**  $\mathcal{L} : \mathcal{B} \times \mathcal{B} \rightarrow \mathbb{R}$  on a set is a function that satisfies some or all properties of a metric. We use loss functions in statistical learning to measure the data fidelity  $\mathcal{L}(h(\mathbf{a}), b)$ .



### Definition (Metric)

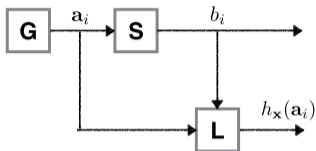
Let  $\mathcal{B}$  be a set. A function  $d(\cdot, \cdot) : \mathcal{B} \times \mathcal{B} \rightarrow \mathbb{R}$  is a metric if  $\forall b_1, b_2, b_3 \in \mathcal{B}$  :

- (a)  $d(b_1, b_2) \geq 0$  for all  $b_1$  and  $b_2$  (*nonnegativity*)
- (b)  $d(b_1, b_2) = 0$  if and only if  $b_1 = b_2$  (*definiteness*)
- (c)  $d(b_1, b_2) = d(b_2, b_1)$  (*symmetry*)
- (d)  $d(b_1, b_2) \leq d(b_1, b_3) + d(b_3, b_2)$  (*triangle inequality*)

### Remarks:

- A **pseudo-metric** satisfies (a), (c) and (d) but not necessarily (b).
- **Norms** induce **metrics** while **pseudo-norms** induce **pseudo-metrics**.
- A **divergence** satisfies (a) and (b) but not necessarily (c) or (d)

## Empirical risk minimization and beyond



### Definition (Empirical Risk Minimization (ERM))

Let  $h_{\mathbf{x}} : \mathbb{R}^p \rightarrow \mathbb{R}$  be a model with parameters  $\mathbf{x}$  and let  $\{(\mathbf{a}_i, b_i)\}_{i=1}^n$  be samples with  $b_i \in \{-1, 1\}$  and  $\mathbf{a}_i \in \mathbb{R}^p$ . The ERM problem reads

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) \right\},$$

where  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i)$  is the loss on the sample  $(\mathbf{a}_i, b_i)$ .

### Some frequently used loss functions

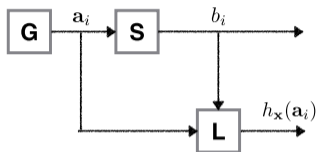
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = \log(1 + \exp(-b_i h_{\mathbf{x}}(\mathbf{a}_i)))$
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = (b_i - h_{\mathbf{x}}(\mathbf{a}_i))^2$
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = \max(0, 1 - b_i h_{\mathbf{x}}(\mathbf{a}_i))$

*Logistic loss.*

*Squared error.*

*Hinge loss.*

## Empirical risk minimization and beyond



### Definition (Empirical Risk Minimization (ERM))

Let  $h_{\mathbf{x}} : \mathbb{R}^p \rightarrow \mathbb{R}$  be a model with parameters  $\mathbf{x}$  and let  $\{(\mathbf{a}_i, b_i)\}_{i=1}^n$  be samples with  $b_i \in \{-1, 1\}$  and  $\mathbf{a}_i \in \mathbb{R}^p$ . The ERM problem reads

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) \right\},$$

where  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i)$  is the loss on the sample  $(\mathbf{a}_i, b_i)$ .

### Other objectives beyond ERM

- ▶  $\min_{\mathbf{x}} \left\{ \frac{1}{n} \sum_{i=1}^n \left[ \max_{\delta: \|\delta\|_{\infty} \leq \epsilon} \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i + \delta), b_i) \right] \right\}$  Adversarial training [54].
- ▶  $\min_{\mathbf{x}} \left\{ \frac{1}{n} \sum_{i=1}^n \left[ \max_{\delta: \|\delta\|_2 \leq \epsilon} \mathcal{L}(h_{\mathbf{x} + \delta}(\mathbf{a}_i), b_i) \right] \right\}$   $\epsilon$ -stability training [9],  
Sharpness-aware minimization [28].
- ▶  $\min_{\mathbf{x}} \max_{b^c \in [C]} \frac{1}{n_c} \sum_{i=1}^{n_c} \left[ \max_{\delta: \|\delta\| \leq \epsilon} \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i + \delta), b^c) \right]$  Class fairness [95].

**Remark:** ◦ In the sequel, we will focus on the minimization problems.

# Gradient descent

## Definition (Gradient Descent (GD))

Starting from  $\mathbf{x}^0 \in \text{dom}(f)$ , update  $\{\mathbf{x}^k\}_{k \geq 0}$  as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k) = \mathbf{x}^k + \alpha_k \mathbf{p}^k.$$

Notice that  $\mathbf{p}^k := -\nabla f(\mathbf{x}^k)$  is the steepest descent (anti-gradient) search direction.

- Remarks:**
- We denote the parameter update at iteration  $k$  as  $\Delta \mathbf{x}^k = \mathbf{x}^{k+1} - \mathbf{x}^k = -\alpha_k \nabla f(\mathbf{x}^k)$ .
  - GD approximates the solution of  $\frac{d\mathbf{x}}{dt} = -\nabla f(\mathbf{x})$ , i.e., the limit when the learning rate  $\alpha_k \rightarrow 0$ .
    - ▶ The continuous limit is also known as Gradient Flow.
  - We need an initialization  $\mathbf{x}^0$  to start GD!

## Initialization in deep ReLU NNs

- Initialization:  $\mathbf{X}_1 \sim \mathcal{N}(0, \beta_1^2)$ ,  $\mathbf{X}_L \sim \mathcal{N}(0, \beta_L^2)$ ,  $\mathbf{X}_l \sim \mathcal{N}(0, \beta^2)$  for  $l = 2, 3, \dots, L - 1$  (weights).

Table: Summary of common weight initializations in NN training

Initialization	$\beta_1^2$ (Input Layer)	$\beta^2$ (Hidden Layers)	$\beta_L^2$ (Output Layer)	$\alpha$ (Output Scaling Factor)
LeCun [66]	$\frac{1}{p}$	$\frac{1}{m}$	$\frac{1}{m}$	1
He [47]	$\frac{2}{p}$	$\frac{2}{m}$	$\frac{2}{m}$	1
NTK [2]	$\frac{2}{m}$	$\frac{2}{m}$	1	1
E et al. [23]	1	1	See Remark	1
Xavier [35]	$\frac{2}{m+p}$	$\frac{1}{m}$	$\frac{2}{m+1}$	1
Mean-field [83]	1	1	1	$m$
$\mu P$ [117]	1	$\frac{1}{m}$	$\frac{1}{m^2}$	1

- Remark:**
- Weight initialization influences the training dynamics, affecting convergence and generalization.
  - The initialization proposed by E et al. [23] for  $\mathbf{X}_L$  follows a Rademacher distribution
    - E et al. [23] samples the last layer  $\pm\beta_c$  with equal probability for a fixed  $\beta_c$ .

# Phase diagram of initialization methods in two-layer ReLU networks

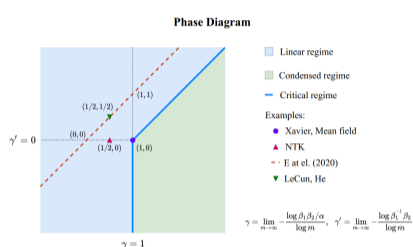


Figure: Phase diagram of two-layer ReLU networks in the infinite-width limit, illustrating different training regimes [77].

- Cyan: Linear (lazy) regime with minimal feature learning; the network behaves like a kernel method.
- Green: Condensed (non-lazy) regime where feature learning is achieved.
- Blue (boundary): Critical regime balancing feature learning and linearization.

**Remark:** ○ Standard parameterizations (e.g., LeCun, He, NTK) follow lazy training in infinite width limit.

## Lazy training

### Definition (Lazy-training (or Linear) regime [77])

Define an  $L$ -layer fully-connected ReLU NN via ( $L$ -Layer NN). Let  $\mathbf{x}(t) := [\mathbf{X}_1(t), \mathbf{X}_2(t), \dots, \mathbf{X}_L(t)]$  represent the weights of network at training time  $t$ . As the width  $m \rightarrow \infty$ , if the following condition holds

$$\sup_{t \in [0, +\infty)} \frac{\|\mathbf{X}_l(t) - \mathbf{X}_l(0)\|_2}{\|\mathbf{X}_l(0)\|_2} \rightarrow 0, \quad \forall l \in [L],$$

then the NN training dynamics fall into the lazy-training regime.

- Remarks:**
- [18] identifies the lazy training behavior for  $m \rightarrow \infty$ .
  - In the lazy training, NN parameters stay close to initialization during the training.
  - The gradient flow of the NN effectively follows the linearization of the NN in lazy training.
  - We also refer to the regime with this behavior as the linear regime.
  - Lazy training has been extensively studied both empirically and theoretically [4, 56, 68].
  - Standard initializations (e.g., LeCun, He, NTK) lead to the lazy regime in the infinite-width limit.

## Critical regime

### Definition (Critical regime [77])

Define an  $L$ -layer fully-connected ReLU NN via ( $L$ -Layer NN). Let  $\mathbf{x}(t) := [\mathbf{X}_1(t), \mathbf{X}_2(t), \dots, \mathbf{X}_L(t)]$  represent the weights of network at training time  $t$ . As the width  $m \rightarrow \infty$ , if the following condition holds

$$\sup_{t \in [0, +\infty)} \frac{\|\mathbf{X}_l(t) - \mathbf{X}_l(0)\|_2}{\|\mathbf{X}_l(0)\|_2} \rightarrow \Theta(1), \quad \forall l \in [L],$$

then the NN training dynamic falls into the critical regime.

- Remarks:**
- Critical regime is included in the “feature learning” regime.
  - In the critical regime, parameters deviate from initialization but not excessively, avoiding instability.
  - Mean field and Xavier initializations follow the critical regime in the infinite-width limit.
  - The supremum above can tend to infinity for the infinite-width limit.
    - ▶ It typically coincides with an initialization with an extremely small variance [112]
    - ▶ The training dynamics are quite complex in this setting [30].

## Feature learning: definition

- How should the weight updates interact during training for “good performance”?

### Definition (Feature Learning)

Let  $\Delta h^{(l)}$  denote the feature change after one algorithmic update for  $l$ -th layer. We are in the feature learning regime if the following properties hold:

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

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

- Remarks:**
- Feature learning primarily concerns the early training phase rather than behavior local convergence.
  - NTK satisfies stable forward passes but have minimal feature updates (i.e., lazy).
  - If the feature updates  $\|\Delta h^{(l)}\|$  are too large, training becomes unstable.
  - Without a lower bound on  $\|\Delta h^{(l)}\|$ , features may never appear.
  - The mean field and  $\mu\text{P}$  initializations are useful for feature learning.

## A sufficient condition for feature learning

### Definition (Spectral condition [119])

Given an  $L$ -layer NN as defined before, consider applying a gradient update  $\Delta \mathbf{X}_l$  to the weight matrix  $\mathbf{X}_l$ . If the spectral norms of the weights and the weight updates satisfy the following  $\forall 2 < l < L$ ,

$$\begin{aligned}\|\mathbf{X}_1\|_{S_\infty} &= \Theta\left(\sqrt{\frac{m}{p}}\right) & \|\Delta \mathbf{X}_1\|_{S_\infty} &= \Theta\left(\sqrt{\frac{m}{p}}\right), \\ \|\mathbf{X}_l\|_{S_\infty} &= \Theta(1) & \|\Delta \mathbf{X}_l\|_{S_\infty} &= \Theta(1), \\ \|\mathbf{X}_L\|_{S_\infty} &= \Theta\left(\sqrt{\frac{1}{m}}\right) & \|\Delta \mathbf{X}_L\|_{S_\infty} &= \Theta\left(\sqrt{\frac{1}{m}}\right).\end{aligned}$$

then, we have feature learning. The  $S_\infty$ -norm is known as the Schatten infinity norm or the spectral norm.

- Remarks:**
- Spectral condition ensures  $\|h^{(l)}\|_{\text{RMS}} = \Theta(1)$  and  $\|\Delta h^{(l)}\|_{\text{RMS}} = \Theta(1)$
  - There is a general version of the spectral condition
    - ▶ It requires that  $\|\mathbf{X}\|_{S_\infty}$  and their updates  $\|\Delta \mathbf{X}\|_{S_\infty}$  scale with  $\Theta\left(\sqrt{\frac{n_{\text{in}}}{n_{\text{out}}}}\right)$
    - ▶  $n_{\text{out}}$  and  $n_{\text{in}}$  denote the input and output dimensions of  $\mathbf{X}$ .
  - Spectral condition ultimately relies on the tensor programming framework in the sequel.

## Tensor program (TP)

- Tensor program is a general framework for expressing NN computations (i.e., forward/backward pass).

### Main takeaway [116]

If a set of pre-activations  $g^1, \dots, g^L$  of a network satisfies certain initialization assumptions, under the TP framework, we have that

$$\frac{1}{m} \sum_{i=1}^m \psi(g_i^1, \dots, g_i^L) \rightarrow \text{a finite computable scalar,}$$

for any “well-behaved” map  $\psi$  a.s. as  $m \rightarrow \infty$ . This map needs to be chosen to avoid pathological cases.

### Remarks:

- Here,  $1, \dots, L$  typically denote depth; in other networks, they may represent, e.g., convolution channels.
- The “assumptions” ensure pre-activations and gradients are component-wise iid. at **any time** during training.
- TP characterizes NN dynamics in infinite-width limit by tracking the distributions of activations.
  - ▶ You can visualize coordinate checks on pre-activations ( [▶ See example for the SCION algorithm](#) ).
- TP can compute Gaussian processes, analyze NTK, and study feature learning in the infinite-width limit.
- We need the TP framework to analyze  $\mu P$  initialization in the sequel.

## $\mu$ P: motivation & key results

- Infinite-width networks in NTK regime can not learn features.
- Real-world networks benefit from evolving features.
- We start with GD/SGD scaling rules parameterizations of learning rate and variance of initialization.

### **Yet another initialization?**

- $\mu$ P (Maximal Update Parameterization) [117] enables effective feature learning even in the infinite-width limit.
- $\mu$ Transfer [114] leverages  $\mu$ P for efficient hyperparameter tuning across scales.

## Parameterization & scaling in neural networks

- Standard parameterization with  $\mathcal{O}(\frac{1}{m})$  learning rate and with infinite width, leads to the linear regime.
- Standard parameterization with a constant-order learning rate, results in unstable training.
- The scaling of weights and activations at initialization determines whether training leads to feature learning.
- A proper choice of parameterization is required to maintain both stability and feature learning.

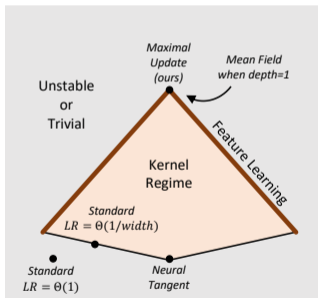


Figure: Illustration of scaling effects on network behavior [117].

## Implementation of $\mu\text{P}$

### abc-parameterization [117]

- Initialize the weights of each layer as  $\mathbf{X}_l = m^{-a_l} \mathbf{W}_l$ , where  $\mathbf{W}_l$  is the actual trainable parameter.
- Initialize each entry of  $\mathbf{W}_l$  such that  $[\mathbf{W}_l]_{ij} \sim \mathcal{N}(0, m^{-2b_l})$ .
- Set the GD learning rate to  $\eta m^{-c}$ , where  $\eta$  is a width-independent constant.
- The Maximal Update Parametrization ( $\mu\text{P}$ ), for an  $L$ -hidden-layer MLP, is defined by the following [117]:

$$a_l = \begin{cases} -\frac{1}{2} & \text{for } l = 1, \\ 0 & \text{for } 2 \leq l < L, \\ \frac{1}{2} & \text{for } l = L, \end{cases} \quad b_l = \frac{1}{2} \quad \forall l \in [L], \quad c = 0.$$

- ▶  $\mu\text{P}$  updates of every parameter/layer have a non-trivial effect on the output of the network.
- ▶  $\mu\text{P}$  uniquely enables maximal feature learning in the infinite-width limit within abc-parameterizations.
- ▶  $\mu\text{P}$  satisfies spectral condition and achieves feature learning.

## An illustration of feature learning with $\mu\text{P}$ initialization

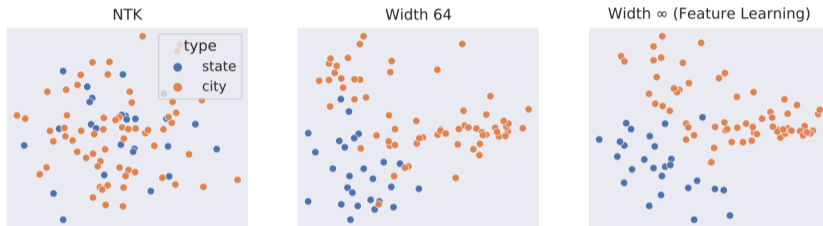


Figure: PCA of Word2Vec embeddings for NTK, width-64, and infinite-width feature-learning networks. NTK embeddings remain random, while increasing width in the feature-learning regime naturally separates cities and states [117].

- o NTK limits feature learning, while  $\mu\text{P}$  enables evolving features and structured embeddings as width increases.

## $\mu$ Transfer: Zero-shot hyperparameter transfer with $\mu$ P

- **Question:** What can we do based on  $\mu$ P's feature learning properties and its scaling behaviors?
- Standard parameterizations (using He/LeCun initialization) do not share similar optimal hyperparameters.
- $\mu$ P for different widths share similar feature learning dynamics. How about optimal hyperparameters?
- Experimental results show that  $\mu$ P for networks with different widths have similar optimal learning rates.

## $\mu$ -transfer: Hyperparameter transferability and empirical results

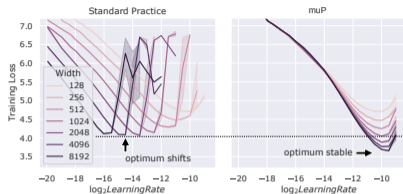


Figure: Loss of Transformers trained with Adam and  $\mu$ P against learning rate for different network widths [114].

- Remarks:**
- Under  $\mu$ P, the optimal learning rate remains largely unchanged when scaling network width.
  - $\mu$ -transfer works for fixed batch-size.
  - Theoretical foundations for  $\mu$ -transfer relate to the edge of stability and sharpness [92].
  - $\mu$ P can be extended to more complex network architectures, algorithms and training objectives.
    - ▶ ResNet [10].
    - ▶ State-space models (SSM) [108].
    - ▶ Sharpness-aware minimization (SAM) [37].

## $\mu\mathbf{P}$ and $\mu$ -transfer for depth and ResNet hyperparameter transfer [10]

- Direct hyperparameter transfer across depths is difficult, as original  $\mu\mathbf{P}$  causes inconsistent training dynamics.
- Residual connections are needed for depth-wise hyperparameter transfer.
- Scaling residual branches by  $1/\sqrt{L}$  ensures feature learning and transfer across both depth and width.

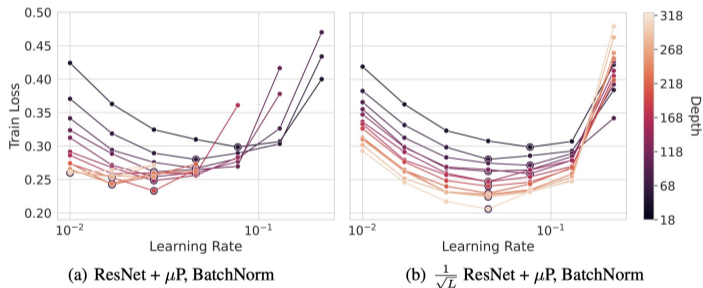


Figure: Impact of  $1/\sqrt{L}$  scaling on ResNet18 training loss (CIFAR-10, 20 epochs). This scaling enables hyperparameter transfer when the depth is scaled [10].

## Recall: State space models (SSM)

- A state-space model is a framework using state variables to describe a system's dynamics over time.
- The Selective and Structured State Space Model (S6) enhances SSM through two key aspects:
  - ▶ Incorporating a selective attention mechanism to focus on key parts of the input sequence.
  - ▶ Using a structured state-space model to capture the temporal dynamics of continuous data.

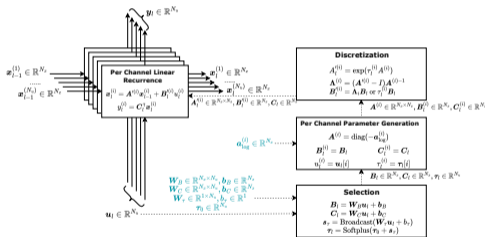


Figure: Illustration of Mamba S6 Layer: Selection, discretization, and per-channel linear recurrence enable efficient computation with adaptive weights. Trainable parameters are shown in blue [109].

**Warning:** The notation used here differs from the notations introduced for MLPs.

## $\mu$ P and $\mu$ -transfer for SSMs

- Original  $\mu$ P is not applicable to SSMs, as SSMs cannot be directly analyzed using the TP framework.
- Spectral conditions effective in MLPs and Transformers do not guarantee feature learning in SSMs.

### Conditions for non-trivial feature updates in a S6 Mamba Layer. [109]

The updates in a S6 Mamba recurrent layer  $\mathbf{y}_{1:L} = f_{\text{mamba}}(\mathbf{u}_{1:L})$  evolve non-trivially in the infinite-width limit under the following conditions:

$$\sigma_B = \Theta\left(\sqrt{\frac{N_x}{N_u}}\right), \quad \sigma_C = \Theta\left(\frac{1}{\sqrt{N_x N_u}}\right), \quad \eta_a = \Theta(N_u), \quad \eta_B = \Theta\left(\frac{N_x}{\sqrt{N_u}}\right), \quad \eta_C = \Theta\left(\frac{1}{N_x \sqrt{N_u}}\right).$$

#### Remarks:

- $\sigma$  represents the variance of initialization, while  $\eta$  denotes the learning rate for different trainable parameters.
- Experiments show this condition enables hyperparameter transfer to larger SSMs.

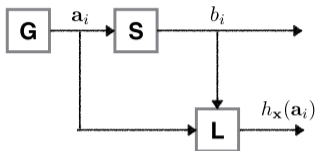
## Feature learning and $\mu\mathbf{P}$ : conclusion

- Key takeaways for feature learning:
  - ▶ Feature learning requires activations and their updates to be properly scaled ( $\Theta(1)$ ) during training.
  - ▶ Spectral condition ensures feature learning via spectral norms of weights and updates.
- Key takeaways for  $\mu\mathbf{P}$  and  $\mu$ -transfer:
  - ▶  $\mu\mathbf{P}$  redefines parameterization achieves maximal feature learning in infinite-width networks.
  - ▶  $\mu$ -**transfer** achieve zero -shot hyperparameter transfer across different network scales.

## Hyperparameter transfer: more...

- Can hyperparameter transfer be achieved from perspectives other than parameterization?
- Yes! Algorithm!

## Recall: Empirical risk minimization and beyond



### Definition (Empirical Risk Minimization (ERM))

Let  $h_{\mathbf{x}} : \mathbb{R}^p \rightarrow \mathbb{R}$  be a model with parameters  $\mathbf{x}$  and let  $\{(\mathbf{a}_i, b_i)\}_{i=1}^n$  be samples with  $b_i \in \{-1, 1\}$  and  $\mathbf{a}_i \in \mathbb{R}^p$ . The ERM problem reads

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) \right\},$$

where  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i)$  is the loss on the sample  $(\mathbf{a}_i, b_i)$ .

### Some frequently used loss functions

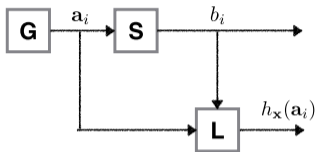
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = \log(1 + \exp(-b_i h_{\mathbf{x}}(\mathbf{a}_i)))$
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = (b_i - h_{\mathbf{x}}(\mathbf{a}_i))^2$
- ▶  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) = \max(0, 1 - b_i h_{\mathbf{x}}(\mathbf{a}_i))$

*Logistic loss.*

*Squared error.*

*Hinge loss.*

## Recall: Empirical risk minimization and beyond



### Definition (Empirical Risk Minimization (ERM))

Let  $h_{\mathbf{x}} : \mathbb{R}^p \rightarrow \mathbb{R}$  be a model with parameters  $\mathbf{x}$  and let  $\{(\mathbf{a}_i, b_i)\}_{i=1}^n$  be samples with  $b_i \in \{-1, 1\}$  and  $\mathbf{a}_i \in \mathbb{R}^p$ . The ERM problem reads

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i) \right\},$$

where  $\mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i), b_i)$  is the loss on the sample  $(\mathbf{a}_i, b_i)$ .

### Other objectives beyond ERM

- ▶  $\min_{\mathbf{x}} \left\{ \frac{1}{n} \sum_{i=1}^n \left[ \max_{\delta: \|\delta\|_{\infty} \leq \epsilon} \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i + \delta), b_i) \right] \right\}$  Adversarial training [54].
- ▶  $\min_{\mathbf{x}} \left\{ \frac{1}{n} \sum_{i=1}^n \left[ \max_{\delta: \|\delta\|_2 \leq \epsilon} \mathcal{L}(h_{\mathbf{x} + \delta}(\mathbf{a}_i), b_i) \right] \right\}$   $\epsilon$ -stability training [9],  
Sharpness-aware minimization [28].
- ▶  $\min_{\mathbf{x}} \max_{b^c \in [C]} \frac{1}{n_c} \sum_{i=1}^{n_c} \left[ \max_{\delta: \|\delta\| \leq \epsilon} \mathcal{L}(h_{\mathbf{x}}(\mathbf{a}_i + \delta), b^c) \right]$  Class fairness [95].

# Stochastic gradient descent (SGD) method and its generalizations

- Focus on the following master algorithmic template:

$$\mathbf{x}^{k+1} \in \arg \min_{\mathbf{x} \in \mathcal{X}} \alpha_k \langle \mathbf{d}^k, \mathbf{x} \rangle + r_k(\mathbf{x}) \quad (1)$$

- ▶  $\alpha_k > 0$  is the step size
- ▶  $r_k$  is a convex regularizer ensuring tractability
- ▶  $\mathbf{d}^k$  is the update, defined in the dual (gradient) space, which we call “dual” feedback
- A classic example is the Euclidean regularizer  $\frac{1}{2} \|\mathbf{x} - \mathbf{x}^k\|_2^2$ , which recovers (stochastic) gradient descent:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{d}^k \quad (\text{SGD})$$

- ▶ The dual feedback  $\mathbf{d}^k$  can be the stochastic gradient  $\mathbf{g}^k$ , the full gradient  $\nabla f(\mathbf{x}^k)$ , and more...
- ▶ When it is the full gradient, then there is an optimal step-size  $\alpha_k = 1/\mathbb{L}$ 
  - ▶  $\mathbb{L}$  is the Lipschitz constant of the objective  $f(\mathbf{x})$
- ▶  $\mathbb{E} \mathbf{g}^k = \nabla f(\mathbf{x})$  is often an unbiased estimator with a bounded variance  $\sigma^2$
- ▶ It is possible to handle biased estimators [85]

**Remark:** ○ Mirror descent uses the Bregman regularizer

## An equivalent characterization of smoothness

### Lemma

Let  $f$  be a continuously differentiable function. We say  $f$  is  $\mathbb{L}$ -smooth, when  $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_* \leq \mathbb{L}\|\mathbf{x} - \mathbf{y}\|$ .

$$f \text{ is } \mathbb{L}\text{-smooth} \implies f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\mathbb{L}}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

**Proof:**

○ By Taylor's theorem:

$$f(\mathbf{y}) = f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle d\tau.$$

Therefore,

$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle &\leq \int_0^1 \|\nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x})\|_* \cdot \|\mathbf{y} - \mathbf{x}\| d\tau \\ &\leq \mathbb{L}\|\mathbf{y} - \mathbf{x}\|^2 \int_0^1 \tau d\tau = \frac{\mathbb{L}}{2} \|\mathbf{y} - \mathbf{x}\|^2 \end{aligned}$$

## An equivalent characterization of smoothness

### Lemma

Let  $f$  be a continuously differentiable function. We say  $f$  is  $\mathbb{L}$ -smooth, when  $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_* \leq \mathbb{L}\|\mathbf{x} - \mathbf{y}\|$ .

$$f \text{ is } \mathbb{L}\text{-smooth} \implies f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\mathbb{L}}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

**Proof:**

○ By Taylor's theorem:

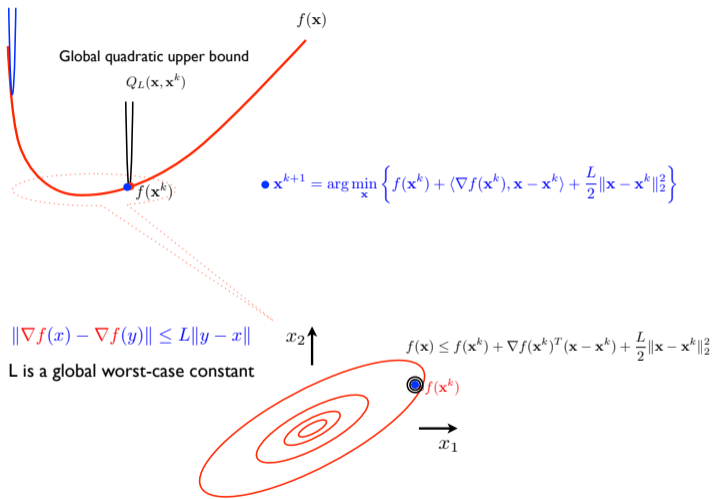
$$f(\mathbf{y}) = f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \int_0^1 \langle \nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle d\tau.$$

Therefore,

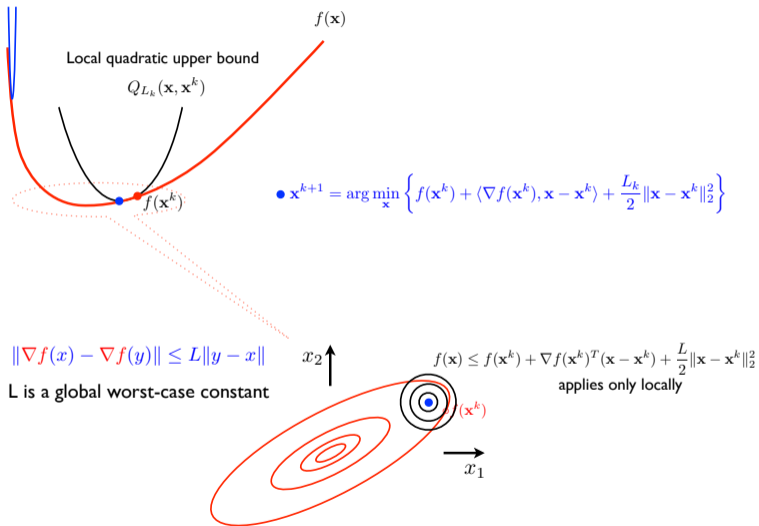
$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle &\leq \int_0^1 \|\nabla f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x})\|_* \cdot \|\mathbf{y} - \mathbf{x}\| d\tau \\ &\leq \mathbb{L}\|\mathbf{y} - \mathbf{x}\|^2 \int_0^1 \tau d\tau = \frac{\mathbb{L}}{2} \|\mathbf{y} - \mathbf{x}\|^2 \end{aligned}$$

**Question:** ○ When the norm is the  $\ell_2$ -norm, what do we get when we minimize this upper bound at  $\mathbf{x}^k$ ?

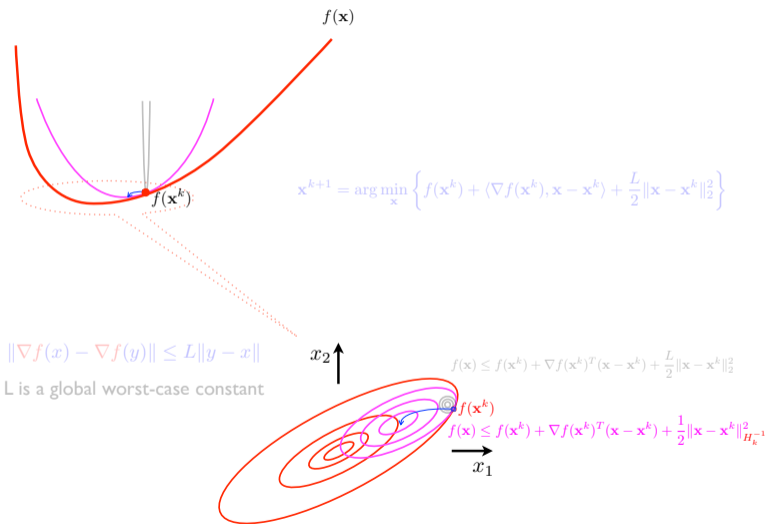
# How can we better adapt to the local geometry?



# How can we better adapt to the local geometry?



# How can we better adapt to the local geometry?



## Preconditioned stochastic gradient descent

- Classical optimization methods treat the problem as a black box.
- If we can accommodate for the geometry, we can gain efficiency
- We can capture geometry with a preconditioned norm associated with the Mahalanobis inner product:

$$\|\mathbf{x}\|_{\mathbf{H}^{-1}} = \sqrt{\langle \mathbf{x}, \mathbf{H}\mathbf{x} \rangle},$$

where  $\mathbf{H}$  is a positive definite matrix.

- With this tool, we can define the preconditioned stochastic gradient descent (PSGD) method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k (\mathbf{H}^k)^{-1/2} \mathbf{g}^k. \quad (\text{PSGD})$$

- Adaptive methods make use of the information from **stochastic gradients and their norms** to compute  $\mathbf{H}$ .

## Adaptive gradient methods

### Intuition

Adaptive gradient methods adapt locally by setting  $\mathbf{H}_k$  as a function of past **stochastic** gradient information.

- Roughly speaking,  $\mathbf{H}_k = \text{function}(\mathbf{g}^1, \mathbf{g}^2, \dots, \mathbf{g}^k)$
- Some well-known examples:

### AdaGrad [22]

$$\mathbf{H}_k = \sum_{t=1}^k \mathbf{g}^t \mathbf{g}^{t\top}$$

### RmsProp [106]

$$\mathbf{H}_k = \beta \mathbf{H}_{k-1} + (1 - \beta) \text{diag}(\mathbf{g}^k)^2$$

### ADAM [63]

$$\begin{aligned} \hat{\mathbf{H}}_k &= \beta \hat{\mathbf{H}}_{k-1} + (1 - \beta) \text{diag}(\mathbf{g}^k)^2 \\ \mathbf{H}_k &= \hat{\mathbf{H}}_k / (1 - \beta^k) \end{aligned}$$

## AdaGrad - Adaptive gradient method with $\mathbf{H}_k = \lambda_k \mathbf{I}$

- If  $\mathbf{H}_k = \lambda_k \mathbf{I}$ , it becomes stochastic gradient descent method with adaptive step-size  $\frac{\alpha_k}{\lambda_k}$ .

### How step-size adapts?

If the stochastic gradient  $\|\mathbf{g}^k\|$  is large/small  $\rightarrow$  AdaGrad adjusts step-size  $\alpha_k/\lambda_k$  smaller/larger

#### Adaptive gradient descent (AdaGrad with $\mathbf{H}_k = \lambda_k \mathbf{I}$ ) [70]

1. Set  $Q^0 = 0$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\begin{cases} Q^k &= Q^{k-1} + \|\mathbf{g}^k\|^2 \\ \mathbf{H}_k &= Q^k \mathbf{I} \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha_k \mathbf{H}_k^{-1/2} \mathbf{g}^k \end{cases}$$

## AdaGrad - Adaptive gradient method with $\mathbf{H}_k = \lambda_k \mathbf{I}$

- If  $\mathbf{H}_k = \lambda_k \mathbf{I}$ , it becomes stochastic gradient descent method with adaptive step-size  $\frac{\alpha_k}{\lambda_k}$ .

### How step-size adapts?

If the stochastic gradient  $\|\mathbf{g}^k\|$  is large/small  $\rightarrow$  AdaGrad adjusts step-size  $\alpha_k/\lambda_k$  smaller/larger

#### Adaptive gradient descent (AdaGrad with $\mathbf{H}_k = \lambda_k \mathbf{I}$ ) [70]

1. Set  $Q^0 = 0$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\begin{cases} Q^k &= Q^{k-1} + \|\mathbf{g}^k\|^2 \\ \mathbf{H}_k &= Q^k \mathbf{I} \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha_k \mathbf{H}_k^{-1/2} \mathbf{g}^k \end{cases}$$

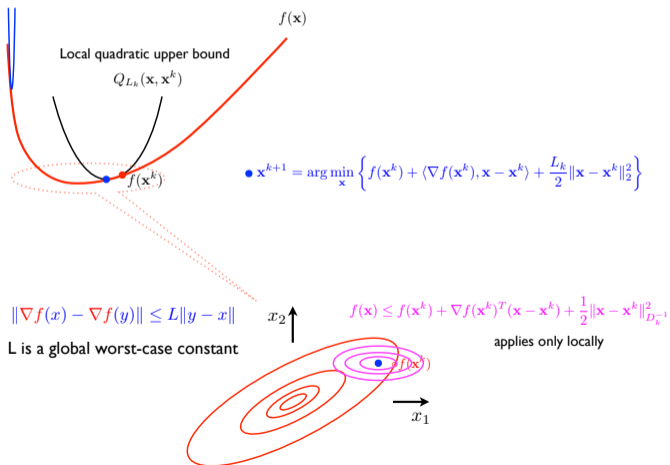
### Adaptation through first-order information

- ▶ When  $H_k = \lambda_k I$ , AdaGrad estimates local geometry through stochastic gradient norms.
- ▶ Akin to estimating a local quadratic upper bound (majorization / minimization) using gradient history.

# AdaGrad - Adaptive gradient method with $\mathbf{H}_k = \mathbf{D}_k$

## Adaptation strategy with a positive diagonal matrix $\mathbf{D}_k$

Adaptive step-size + coordinate-wise extension = adaptive step-size for each coordinate



## AdaGrad - Adaptive gradient method with $\mathbf{H}_k = \mathbf{D}_k$

◦ Suppose  $\mathbf{H}_k$  is diagonal,

$$\mathbf{H}_k := \begin{bmatrix} \lambda_{k,1} & & 0 \\ & \ddots & \\ 0 & & \lambda_{k,d} \end{bmatrix},$$

◦ For each coordinate  $i$ , we have different step-size  $\frac{\alpha_k}{\lambda_{k,i}}$  is the step-size.

### Adaptive gradient descent (AdaGrad with $\mathbf{H}_k = \mathbf{D}_k$ )

1. Set  $\mathbf{Q}^0 = 0$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\begin{cases} \mathbf{Q}^k &= \mathbf{Q}^{k-1} + \text{diag}(\mathbf{g}^k)^2 \\ \mathbf{H}_k &= \mathbf{Q}^k \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha_k \mathbf{H}_k^{-1/2} \mathbf{g}^k \end{cases}$$

### Adaptation across each coordinate

- ▶ When  $\mathbf{H}_k = \mathbf{D}_k$ , we adapt across each coordinate individually.
- ▶ Essentially, we have a finer treatment of the function we want to optimize.

## RMSProp - Adaptive gradient method with $\mathbf{H}_k = \mathbf{D}_k$

### What could be improved over AdaGrad?

1. Stochastic gradients have equal weights in step size.
2. Consider a *steep* function, flat around minimum  $\rightarrow$  slow convergence at flat region.

#### AdaGrad with $\mathbf{H}_k = \mathbf{D}_k$

1. Set  $\mathbf{Q}_0 = 0$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\begin{cases} \mathbf{Q}^k &= \mathbf{Q}^{k-1} + \text{diag}(\mathbf{g}^k)^2 \\ \mathbf{H}_k &= \mathbf{Q}^k \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha_k \mathbf{H}_k^{-1/2} \mathbf{g}^k \end{cases}$$

#### RMSProp

1. Set  $\mathbf{Q}_0 = 0$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\begin{cases} \mathbf{Q}^k &= \beta \mathbf{Q}^{k-1} + (1 - \beta) \text{diag}(\mathbf{g}^k)^2 \\ \mathbf{H}_k &= \mathbf{Q}^k \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha_k \mathbf{H}_k^{-1/2} \mathbf{g}^k \end{cases}$$

- o RMSProp uses weighted averaging with constant  $\beta$
- o Recent gradients have greater importance

# ADAM - Adaptive moment estimation

## Over-simplified idea of ADAM

RMSProp + 2nd order moment estimation = ADAM

ADAM	
<b>Input.</b>	Step size $\alpha$ , exponential decay rates $\beta_1, \beta_2 \in [0, 1)$
<b>1.</b>	Set $\mathbf{m}_0, \mathbf{v}_0 = 0$
<b>2.</b>	For $k = 0, 1, \dots$ , iterate
$\left\{ \begin{array}{l} \mathbf{g}_k \\ \mathbf{m}_k \\ \mathbf{v}_k \\ \hat{\mathbf{m}}_k \\ \hat{\mathbf{v}}_k \\ \mathbf{H}_k \\ \mathbf{x}^{k+1} \end{array} \right.$	$\begin{array}{l} = \nabla f(\mathbf{x}^{k-1}) \\ = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1) \mathbf{g}_k \leftarrow \text{1st order estimate} \\ = \beta_2 \mathbf{v}_{k-1} + (1 - \beta_2) \mathbf{g}_k^2 \leftarrow \text{2nd order estimate} \\ = \mathbf{m}_k / (1 - \beta_1^k) \leftarrow \text{Bias correction} \\ = \mathbf{v}_k / (1 - \beta_2^k) \leftarrow \text{Bias correction} \\ = \hat{\mathbf{v}}_k + \epsilon \\ = \mathbf{x}^k - \alpha \mathbf{H}_k^{-1/2} \hat{\mathbf{m}}_k \end{array}$
<b>Output :</b>	$\mathbf{x}^k$

(Every vector operation is an element-wise operation)

## Leveraging problem structure

- We could leverage the prior knowledge of problem structure (architecture and input domain).
- Hard-code such information into the regularizer  $r_k$ .
- Using this regularizer, we will develop in the sequel
  1. Steepest descent methods
  2. Conditional Gradient methods

## Sharp descent: Steepest descent in a chosen norm

- To generalize SGD beyond the Euclidean case, we can simply regularize with a **non-Euclidean norm**.

### Stochastic sharp descent (SAD)

By defining the sharp-operator  $[\mathbf{d}]^\sharp \in \arg \min_{\mathbf{x} \in \mathcal{X}} \{ \langle \mathbf{d}, \mathbf{x} \rangle + \frac{1}{2} \|\mathbf{x}\|^2 \}$  [89], the (stochastic) sharp descent method (SAD) in a normed space is introduced in:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha [\mathbf{d}^k]^\sharp. \quad (\text{SAD})$$

#### Remarks:

- When  $f$  is  $\mathbb{L}$ -smooth, we can use  $\alpha < \frac{2}{\mathbb{L}}$  with the deterministic gradients.
- Here are some examples:
  - ▶ The  $\ell_\infty$ -norm leads to a sign-based update

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \|\mathbf{d}^k\|_1 \text{sign}(\mathbf{d}^k).$$

- ▶ The spectral norm, matrix analogue to the  $\ell_\infty$ -norm, gives stochastic spectral descent [14]:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \|\sigma^k\|_1 U^k V^k, \quad (\text{SSD})$$

where  $\mathbf{d}^k = U^k \text{diag}(\sigma^k) V^k$  is the reduced SVD of  $\mathbf{d}^k$ .

- ▶ The Muon algorithm drops the norm scaling in SSD and combines with Adam [59].

## A relative of the sharp-operator

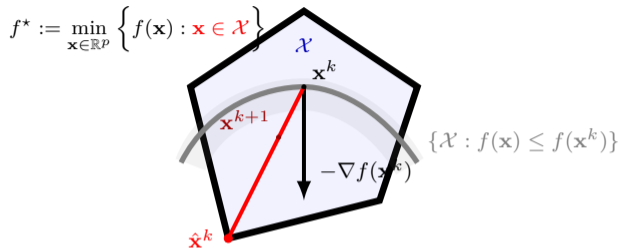
### Definition (Linear minimization oracle)

Let  $\mathcal{X}$  be a convex, closed and bounded set. Then, the linear minimization oracle of  $\mathcal{X}$  ( $\text{lmo}_{\mathcal{X}}$ ) returns a vector  $\hat{\mathbf{x}}$  such that

$$\text{lmo}_{\mathcal{X}}(\mathbf{d}) := \hat{\mathbf{x}} \in \arg \min_{\mathbf{x} \in \mathcal{X}} \mathbf{d}^T \mathbf{x} \quad (2)$$

- Remarks:**
- $\text{lmo}_{\mathcal{X}}$  returns an extreme point of  $\mathcal{X}$  when it is a polyhedral.
  - $\text{lmo}_{\mathcal{X}}$  is not single valued, note  $\in$  in the definition.
  - $\mathbf{d}^{\#} = -c\|\mathbf{d}\|_* \text{lmo}_{\mathcal{X}/c}(\mathbf{d})$  when  $\mathcal{X} := \{\mathbf{x} \mid \|\mathbf{x}\| \leq c\}$ .
  - Without loss of generality, we can consider unit norm constraints.

# The unsung hero in deep learning: The conditional gradient method



## Conditional gradient method (CGM)

1. Choose  $\mathbf{x}^0 \in \mathcal{X}$ .
2. For  $k = 0, 1, \dots$  perform:

$$\mathbf{x}^{k+1} := (1 - \alpha_k)\mathbf{x}^k + \alpha_k \text{lmo}_{\mathcal{X}}(\nabla f(\mathbf{x}^k))$$

where  $\alpha_k := \frac{2}{k+2}$  is typically used when  $f$  is convex.

- Remarks:**
- The step-size  $\alpha$  looks like weight decay but is not! More on this later.
  - Without the “weight decay,” the algorithm looks like steepest descent in a chosen norm...
    - ▶ without the dual norm scaling (recall  $\mathbf{d}^\sharp = -c\|\mathbf{d}\|_* \text{lmo}_{\mathcal{X}/c}(\mathbf{d})$  when  $\mathcal{X} := \{\mathbf{x} \mid \|\mathbf{x}\| \leq c\}$ )!

## Stochastic conditional gradient methods

### Stochastic conditional gradient (SCG) [96, 86]

$$\mathbf{x}^{k+1} = (1 - \alpha_k)\mathbf{x}^k + \alpha_k \text{lmo}(\mathbf{d}^k).$$

### Unconstrained stochastic conditional gradient (uSCG) [96]

Instead of the convex combination in SCG (or “weight decay”), we consider an unconstrained version:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \text{lmo}(\mathbf{d}^k) = \mathbf{x}^k - \alpha \mathbf{d}^\# / \|\mathbf{d}\|_*.$$

#### Remarks:

- The stepsize will no longer be required to be  $\alpha < 2/L$  nor  $\alpha \in (0, 1)$ .
- uSCG is SAD when we drop the dual norm (in the same manner as Muon).
- uSGD with the  $\ell_2$ -norm is the normalized SGD:  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \frac{\mathbf{d}^k}{\|\mathbf{d}^k\|_2}$ .
- uSGD with the  $\ell_\infty$ -norm leads to signSGD update [8]:  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \text{sign}(\mathbf{d}^k)$ .
- uSGD with the  $S_\infty$ -norm is  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k U^k V^k$  (i.e., SSD or partial Muon) [14, 59].

## To weight decay, or not?

### Tikhonov regularization [107]

$$\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\mu}{2} \|\mathbf{x}\|_2^2$$

- Remarks:**
- Updating with the gradient gives:  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k (\nabla f(\mathbf{x}^k) + \mu \mathbf{x}^k)$ .
  - This results in the ubiquitous weight decay update:  $\mathbf{x}^{k+1} = (1 - \mu \alpha_k) \mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)$ .
  - [76] decouples weight decay and objective function  $\mathbf{x}^{k+1} = (1 - \mu \alpha_k) \mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)$ .
  - [76] argues that this update rule results in preconditioned regularizers.
  - In contrast, updating with the lmo gives:  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \text{lmo} (\nabla f(\mathbf{x}^k) + \mu \mathbf{x}^k)$ .
  - lmo is not additive:  $\text{lmo} (\nabla f(\mathbf{x}^k) + \mu \mathbf{x}^k) \neq \text{lmo} (\nabla f(\mathbf{x}^k)) + \text{lmo} (\mu \mathbf{x}^k)$
  - lmo is not linear:  $\text{lmo}(\mathbf{x}^k) \neq \mathbf{x}^k$ .
  - SCG provides strict norm control:  $\|\mathbf{x}^k\| \leq c, \forall k$ ; when  $\mathbf{x}^{k+1} = (1 - \alpha_k) \mathbf{x}^k + \alpha_k c \text{lmo}_{\mathcal{X}/c}(\mathbf{d}^k)$ .

## Further nuances

### uSCG with weight decay

Weight decay combined with uSCG can be written as

$$\mathbf{x}^{k+1} = (1 - \mu\alpha_k)\mathbf{x}^k + \alpha_k \text{lmo}(\mathbf{d}^k)$$

with weight decay parameter  $\mu \in (0, 1]$ . This is exactly SCG when  $\mu = 1$ .

- Remarks:**
- The above update does not correspond to minimization with Tikhonov regularization.
  - Weight decay in this case changes the objective, the form of which is unclear!

### Stochastic case via momentum

In the stochastic setting, lmo and the sharp-operator may be biased even if  $\nabla f(\cdot, \xi)$  is unbiased, when lmo or the sharp-operator is non-linear.

$$\mathbf{d}^k = (1 - \beta_k)\mathbf{d}^{k-1} + \beta_k \nabla f(\mathbf{x}^k, \xi_k),$$

where  $\beta_k \in (0, 1]$ .

- Remark:**
- There are several ways to select  $\mathbf{d}_k$  and we present above only a basic one (cf., [121]).

## Choice of norm constraint

- Choose the appropriate norm from the operator norm perspective [64].
- Consider a linear MLP with layers defined as:

$$g^{(1)}(\mathbf{a}) = \mathbf{X}_1 \mathbf{a}, \quad g^{(\ell)}(g^{(\ell-1)}) = \mathbf{X}_\ell g^{(\ell-1)}, \quad \forall \ell \in \{2, \dots, L\}$$

where  $\mathbf{X}_\ell \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$  and the loss function is  $\mathcal{L}(g^{(L)}, b)$ .

- Feature learning states that the typical element of the vectors should be of order  $\Theta(1)$ .

### Norm bounds

To control hidden activation  $g^{(\ell)}(\mathbf{z})$  growth with the input  $\mathbf{z} \in \mathbb{R}^d$ , we enforce one of these norm bounds:

- ▶ Average Entry Bounded:  $\frac{1}{d_{\text{out}}} \|g^{(\ell)}(\mathbf{z})\|_1 \leq 1$
- ▶ Typical Entry Bounded:  $\|g^{(\ell)}(\mathbf{z})\|_{\text{RMS}} \leq 1$ , where  $\|\mathbf{z}\|_{\text{RMS}} = \frac{1}{\sqrt{d}} \|\mathbf{z}\|_2$
- ▶ Max Entry Bounded:  $\|g^{(\ell)}(\mathbf{z})\|_\infty \leq 1$

## Choice of norm constraint

- Control operator norm constraints on weights  $\{\mathbf{X}_\ell\}$  can ensure bounded activations  $g^{(\ell)}(\mathbf{z})$ .

### Definition (Operator norm)

The operator norm quantifies how much a matrix  $\mathbf{X}$  stretches inputs between normed spaces.

$$\|\mathbf{X}\|_{\alpha \rightarrow \beta} := \max_{\mathbf{z} \in \mathbb{R}^d, \mathbf{z} \neq 0} \frac{\|\mathbf{X}\mathbf{z}\|_\beta}{\|\mathbf{z}\|_\alpha} = \sup_{\|\mathbf{z}\|_\alpha \leq 1} \|\mathbf{X}\mathbf{z}\|_\beta.$$

- Remarks:**
- If  $\|\mathbf{z}\|_\alpha \leq 1$ , then the output  $\|\mathbf{X}\mathbf{z}\|_\beta$  is bounded when  $\|\mathbf{X}\|_{\alpha \rightarrow \beta}$  is bounded.
  - Operator norm transformations:
    - If  $\|\mathbf{z}\|_\beta \leq \rho \|\mathbf{z}\|_c$  for all  $\mathbf{z}$ , then  $\|\mathbf{X}\|_{\alpha \rightarrow \beta} \leq \rho \|\mathbf{X}\|_{\alpha \rightarrow c}$ .
    - If  $\|\mathbf{z}\|_\alpha \geq \frac{1}{\rho} \|\mathbf{z}\|_c$  for all  $\mathbf{z}$ , then  $\|\mathbf{X}\|_{\alpha \rightarrow \beta} \leq \rho \|\mathbf{X}\|_{c \rightarrow \beta}$ .
  - Vector norm relations:

$$\|\mathbf{z}\|_\infty \leq \|\mathbf{z}\|_2 \leq \|\mathbf{z}\|_1 \leq \sqrt{d} \|\mathbf{z}\|_2 \leq d \|\mathbf{z}\|_\infty.$$

## Layerwise norm selection

- Different layers may require different norms [7].

### Operator norm choices for the MLP [96]

- ▶ Initial layer:  $\|\mathbf{X}_1\|_{\alpha_1 \rightarrow \text{RMS}} \leq 1$ .
- ▶ Intermediary layers:  $\|\mathbf{X}_\ell\|_{\text{RMS} \rightarrow \text{RMS}} \leq 1, \forall \ell \in \{2, \dots, L-1\}$ .
- ▶ Last layer:  $\|\mathbf{X}_L\|_{\text{RMS} \rightarrow \beta_L} \leq 1$ .

- Remarks:**
- The operator norm for the intermediary layers  $\|\cdot\|_{\text{RMS} \rightarrow \text{RMS}}$  is a scaled spectral norm:

$$\|\mathbf{X}\|_{\text{RMS} \rightarrow \text{RMS}} = \sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}} \|\mathbf{X}\|_{\mathcal{S}_\infty}.$$

- For the intermediary layers,  $\text{Im}_0$  is set to  $\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} UV^\top$ .
- The choice of input norm  $\alpha_1$  and output norm  $\beta_L$  depends on the application.

## Layerwise norm selection

### Input layer with language data [96]

Input is typically 1-hot encoded, meaning  $\|\mathbf{a}\|_\infty = \|\mathbf{a}\|_2 = \|\mathbf{a}\|_1 = 1$ .

- ▶ Equivalent operator norms:  $\|\mathbf{X}_1\|_{\infty \rightarrow \text{RMS}} = \|\mathbf{X}_1\|_{2 \rightarrow \text{RMS}} = \|\mathbf{X}_1\|_{1 \rightarrow \text{RMS}}$ .
- ▶ lmo can be computed exactly through  $\|\cdot\|_{1 \rightarrow \text{RMS}}$ :  $\text{col}_i(\mathbf{X}_1) \mapsto \sqrt{d_{\text{out}}} \frac{\text{col}_i(\mathbf{X}_1)}{\|\text{col}_i(\mathbf{X}_1)\|_2}$ ;  
or alternatively  $\|\cdot\|_{2 \rightarrow \text{RMS}}$  provides a more aggressive update scaling factor:  $\sqrt{d_{\text{out}}/d_{\text{in}}} UV^\top$ .

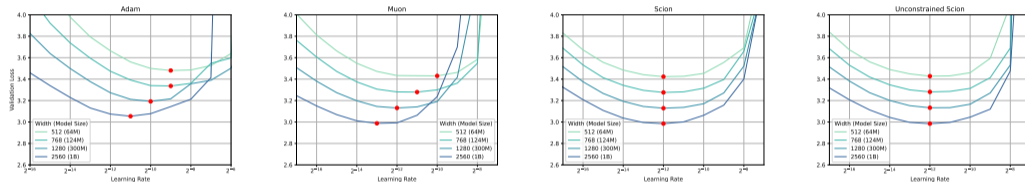
### Output layer [96]

We have no restriction to bound the output in  $\ell_{\text{RMS}}$  and can instead bound the maximal entry using  $\ell_\infty$ .

- ▶ lmo is computed through  $\|\mathbf{X}_L\|_{\text{RMS} \rightarrow \infty}$ :  $\frac{1}{\sqrt{d_{\text{in}}}} \frac{\text{row}_j(\mathbf{X}_L)}{\|\text{row}_j(\mathbf{X}_L)\|_2}$
- ▶ Alternatively, we can bound  $\|\mathbf{X}_L\|_{\text{RMS} \rightarrow \infty} \leq \frac{1}{d_{\text{in}}} \|\mathbf{X}_L\|_{1 \rightarrow \infty}$ , then lmo is set to  $\frac{1}{d_{\text{in}}} \text{sign}(\mathbf{X}_L)$ .

## Empirical results

- We define uSCG and SCG with operator norms as UNCONSTRAINED SCION and SCION, respectively.
- We build on the Muon codebase [58]:



### Remarks:

- SCION and UNCONSTRAINED SCION maintain optimal stepsize across model widths.
- Even when the Muon is tuned on the largest model size it achieves a validation loss of 2.988
- In comparison, UNCONSTRAINED SCION obtains 2.984.
- These methods eliminate the need for Adam in Muon's implementation.
- SCION shows advantages in long-term runs [74].

## Illustration of norm control

- Spectral norm is constrained in SCION.
- This is key in improving generalization (see Lecture 9 of EE-556).

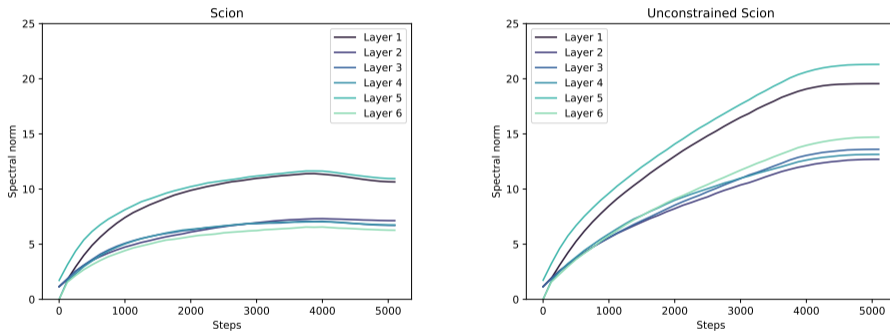


Figure: Spectral norm of weight matrices on NanoGPT (124M). The linear stepsize decay starts at iteration 3650.

## Illustration of norm control

- Spectral norm is constrained in SCION.
- This is key in improving generalization (see Lecture 9 of EE-556).

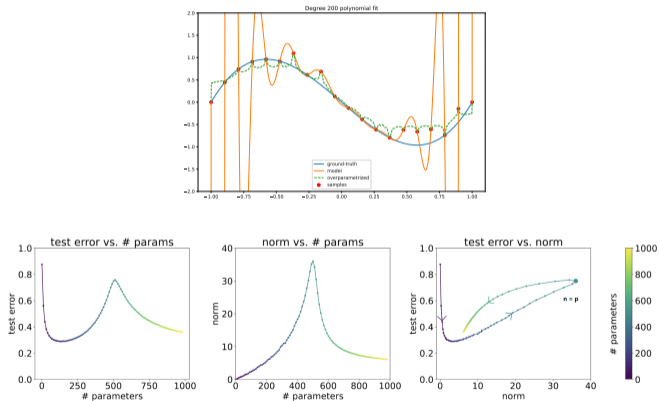
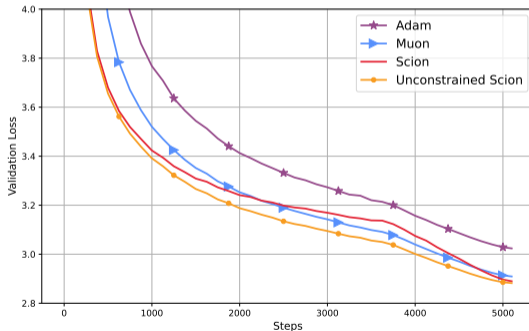


Figure: Cévher overfitting: You can achieve the same loss but the one that has the least norm is often better.

## NanoGPT $\mu$ -transfer

- Scale up to a 3B parameter model using the 124M proxy model's optimal setup.

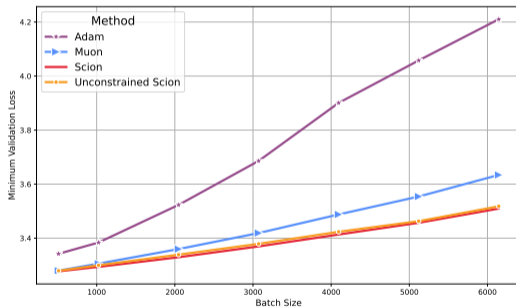
Adam	Muon	UNCONSTRAINED SCION	SCION
3.024	2.909	<b>2.882</b>	2.890



**Remark:**  $\circ$  SCION- like algorithms show advantages in long-term runs [74].

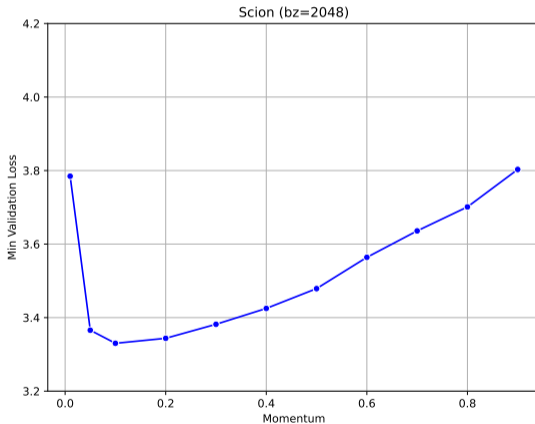
## On the effect of the batch size

- Fix the total tokens and sweep batch sizes, adjusting steps accordingly.
- (UNCONSTRAINED) SCION maintains lower validation loss as batch size increases, outperforming baselines.



## The sweet spot for the momentum parameter

- While the relationship is complicated, it is possible to derive an optimal momentum parameter [96].



## Learning rate scheduling

- Choosing a sequence of step sizes for an algorithm  $\{\alpha_k\}_{k=1}^T$  is a core problem in optimization.

### Definition

Learning rate scheduling incorporates two elements:

1. A **baseline** learning rate  $\alpha$ , which can be determined adaptively;
2. A **schedule** baseline multiplier, which has a predetermined sequence of values  $\{s_k\}_{k=1}^T$ .

- Example:**
- How can we incorporate a learning rate schedule into gradient descent (GD)?

#### GD with step size scheduling

1. Set  $\mathbf{x}^0 \in \text{dom}(f)$ .
2. For  $k = 0, 1, \dots$ , iterate

$$\left\{ \begin{array}{l} \mathbf{x}^{k+1} = \mathbf{x}^k - \alpha s_k \nabla f(\mathbf{x}^k) \end{array} \right.$$

## Basic forms

- The choice of schedule is empirically motivated.
- Basic examples include linear, exponential and step-ladder-like decreasing sequences [6].

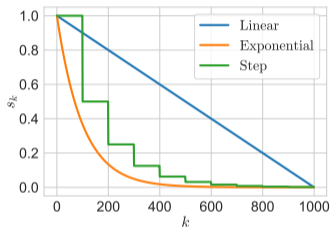


Figure: Example of common schedule (sub)sequences.

## Warm up

[36, 65] note how better results for computer vision models can be achieved by adding a *warm up* phase where the step-size scheduler initially increases the step-size multiplier from a small value to a large one.

## Additional considerations

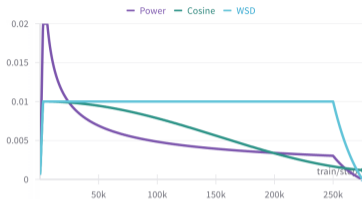


Figure: Example Warm-up+Runtime+Cool-down schedules vs. cosine annealing schedule. Credit: [103].

### Fusion of ideas

[78, 124] motivate *three-phase* schedules with results for residual neural networks and vision transformers:

1. *Warm-up* schedule with increasing step size.
2. *Runtime* schedule with constant or decreasing step size.
3. *Cool down* or *Ramp down* or *Decay* phase of sharp learning rate decrease.

#### Remarks:

- [122] theoretically motivates fast linear decay at the end of schedules.
- [55] describes the benefits of non-linear step size cool-down for language models.

## From algorithm to real implementation

- Algorithms are crucial for deep learning.
- System implementation is also essential for scalable deep learning.
  - ▶ Data parallelism
  - ▶ Model parallelism
  - ▶ Pipeline parallelism
- System constraints can easily eliminate a theoretically optimal algorithm from being applied.

## Data parallelism

- Partition training data into batches, and deploy the entire model on each GPU.
- Compute the gradient of each GPU and aggregate them for the update.
- Cannot train large models that exceed GPU device memory.

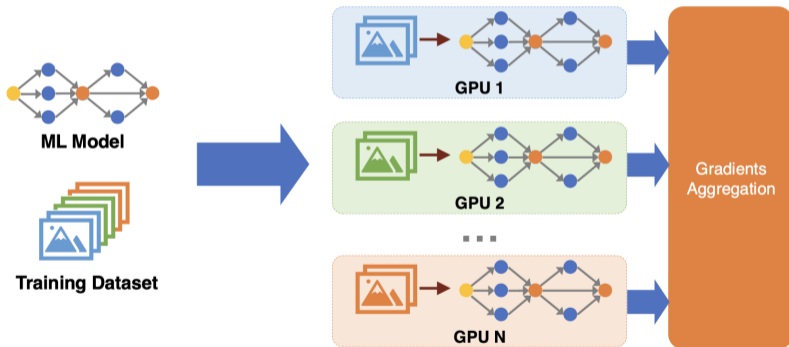


Figure: Data parallelism [26]

## Model parallelism

- Split a model into multiple parts and deploy them on multiple GPUs.
- It is challenging to split a model correctly.

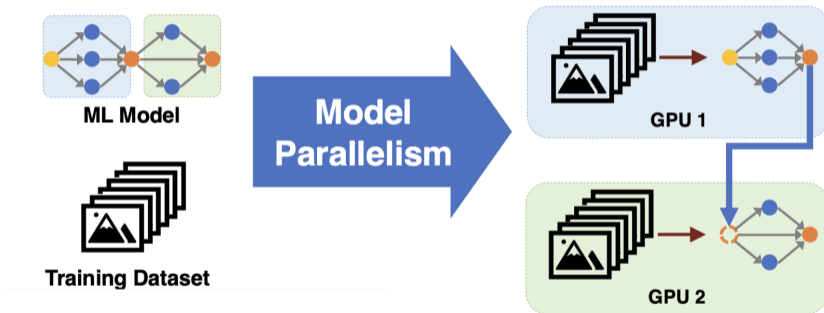


Figure: Model parallelism [26]

# Tensor parallelism

- Tensor parallelism is a type of model parallelism.
- Parameters/gradients can be partitioned within a layer.

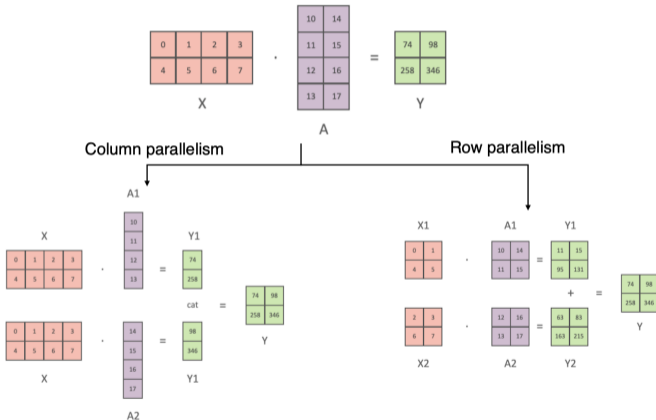


Figure: Tensor parallelism [27]

## Tensor parallelism: example

- Update an MLP without GPU synchronization until the end.
- Multihead attention layer is inherently parallel, simplifying parallelization.

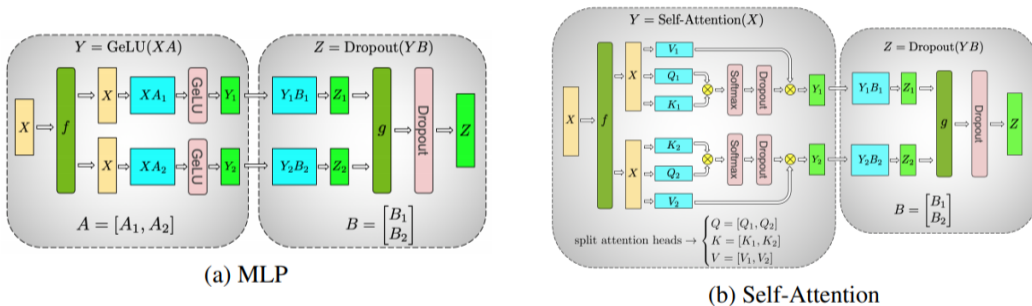
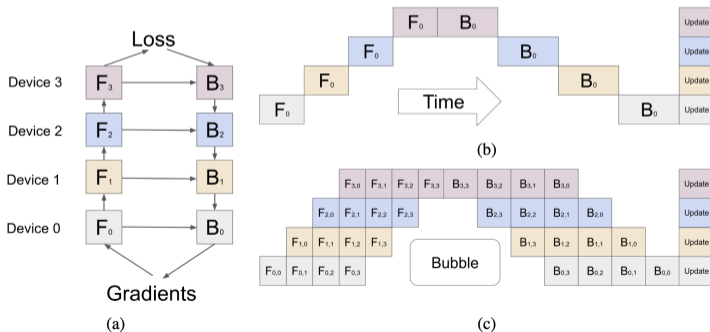


Figure: Blocks of Transformer with Model Parallelism [104]

## Pipeline parallelism

- Naive model parallelism under-utilizes compute (fig b).
- Pipeline parallelism splits mini-batches into micro-batches for parallel execution (fig c).
- Must store activations of all micro-batches before backpropagation.
- Idle time called “bubbles” reduces efficiency.
- Methods to improve efficiency or reduce memory: PipeDream [42], DAPPLE [24], PipeMare [113].



## Parallelism comparison

Table: Comparison of parallelism strategies in deep learning [26]

Parallelism	Pros	Cons
Data	<ul style="list-style-type: none"><li>○ Massively parallelizable</li><li>○ No communication during forward/backward</li></ul>	<ul style="list-style-type: none"><li>○ Do not support out-of-memory models</li><li>○ Poor scaling for large models</li></ul>
Model	<ul style="list-style-type: none"><li>○ Support large models</li><li>○ Efficient for large parameter counts</li></ul>	<ul style="list-style-type: none"><li>○ Limited parallelizability</li><li>○ Need intermediate result transfers</li></ul>
Pipeline	<ul style="list-style-type: none"><li>○ Support large-batch training</li><li>○ Efficient for deep models</li></ul>	<ul style="list-style-type: none"><li>○ Bubbles reduce utilization</li></ul>



## Wrap up!

- ▶ Lecture 3 about Data next Thursday!

## Towards training with neural networks

- What do we have at hand?
  1. The optimization objective  $f(\mathbf{x})$  from multi-layer, multi-class, convolutions, transformers, etc.
  2. First-order gradient via backpropagation  $\nabla f(\mathbf{x})$
- Barriers to training of neural networks:
  1. Curse-of-dimensionality → first-order methods
  2. Non-convexity → stochasticity + momentum
  3. Ill-conditioning → adaptive gradient methods

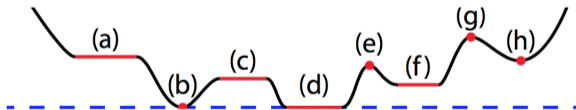


Figure: A non-convex function. (a) and (c) are plateaus, (b) and (d) are global minima, (f) and (h) are local minima, (e) and (g) are local maxima. [38]

## Stochastic Gradient Descent (SGD) and some key variants

### Vanilla (Minibatch) SGD

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

**1. For**  $k = 0, 1, \dots$ :

obtain the (minibatch) stochastic gradient  $g^k$

update  $x^{k+1} \leftarrow x^k - \gamma_k g^k$

## Stochastic Gradient Descent (SGD) and some key variants

### Vanilla (Minibatch) SGD

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

**1. For**  $k = 0, 1, \dots$ :

obtain the (minibatch) stochastic gradient  $g^k$

update  $x^{k+1} \leftarrow x^k - \gamma_k g^k$

### Perturbed Stochastic Gradient Descent [31]

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

**1. For**  $k = 0, 1, \dots$ :

sample noise  $\xi$  uniformly from unit sphere

update  $x^{k+1} \leftarrow x^k - \alpha_k (g^k + \xi)$

# Stochastic Gradient Descent (SGD) and some key variants

## Vanilla (Minibatch) SGD

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

1. For  $k = 0, 1, \dots$ :  
obtain the (minibatch) stochastic gradient  $g^k$   
update  $x^{k+1} \leftarrow x^k - \gamma_k g^k$

## Perturbed Stochastic Gradient Descent [31]

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

1. For  $k = 0, 1, \dots$ :  
sample noise  $\xi$  uniformly from unit sphere  
update  $x^{k+1} \leftarrow x^k - \alpha_k (g^k + \xi)$

## \*Stochastic Gradient Langevin Dynamics [111]

**Input:** Stochastic gradient oracle  $g$ , initial point  $x^0$ , step size  $\alpha_k$

1. For  $k = 0, 1, \dots$ :  
sample noise  $\xi$  standard Gaussian  
update  $x^{k+1} \leftarrow x^k - \alpha_k g^k + \sqrt{2\alpha_k} \xi$

## Basic questions:

1. Does SGD converge with probability 1?
2. Does SGD avoid non-minimum points with probability 1?
3. How fast does SGD converge to local minimizers?

## Critical points

### Recall (Classification of critical points)

Let  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be twice differentiable and let  $\bar{x}$  be a critical point. Let  $\{\lambda_i\}_{i=1}^d$  be the eigenvalues of the hessian  $\nabla^2 f(\bar{x})$ , then

- ▶  $\lambda_i > 0$  for all  $i \Rightarrow \bar{x}$  is a local minimum
- ▶  $\lambda_i < 0$  for all  $i \Rightarrow \bar{x}$  is a local maximum
- ▶  $\lambda_i > 0, \lambda_j < 0$  for some  $i, j$  and  $\lambda_i \neq 0$  for all  $i \Rightarrow \bar{x}$  is a saddle point
- ▶ Other cases  $\Rightarrow$  inconclusive

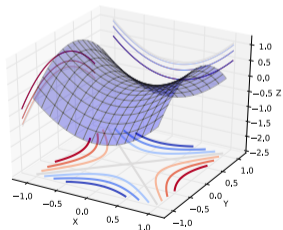


Figure: Minmax saddle ( $\lambda_i \neq 0$  for all  $i$ )

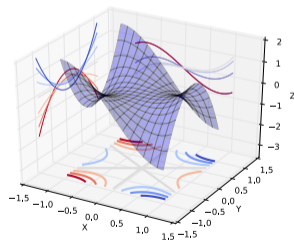


Figure: Monkey saddle ( $\lambda_i = 0$  for some  $i$ )

## The strict saddle property

### Definition (Strict saddle)

A twice differentiable function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is  $(\alpha, \beta, \epsilon, \delta)$ -strict saddle if for any point  $x$  at least one of the following is true

1.  $\|\nabla f(x)\| \geq \epsilon$ .
2.  $\lambda_{\min}(\nabla^2 f(x)) \leq -\beta$ .
3. There is a local minimum  $x^*$  such that  $\|x - x^*\| \leq \delta$  and the function  $f$  restricted to a  $2\delta$  neighborhood of  $x^*$  is  $\alpha$  strongly convex.

### (Informal)

For any point whose gradient is small, it is either close to a local minimum, or is a saddle point (or local maximum) with a significant negative eigenvalue.

## Q1: Does SGD converge?

- SGD converges to the critical points of  $f$  as  $k \rightarrow \infty$ .
  1. GD converges from any initialization with constant step-size and full gradients
  2. With probability 1, (P)SGD does not converge with constant step-size  $\gamma$  [5, 99]
  3. With probability 1, SGD converges with vanishing step-size if  $\mathbf{x}^k$  is bounded with probability 1 [75, 5]

Boundedness is not required (Theorem 1 of [84])

Assume Lipschitzness, **sublevel regularity**,  $\mathbb{E}\|\mathbf{g}\|^q \leq \sigma^q$  and  $\sum_k \gamma_k^{1+q/2} < \infty$  ( $q \geq 2$ ). Then,  $\mathbf{x}^k$  converges with probability 1.

## Q2: Does SGD avoid saddle points?

o SGD avoids strict saddles ( $\lambda_{\min}(\nabla^2 f(\mathbf{x}^*)) < 0$ )

1. GD avoids strict saddles from almost all initializations [69]

2. With probability  $1 - \zeta$ , PSGD with constant  $\gamma$  escapes strict saddles after  $\Omega(\log(1/\zeta)/\gamma^2)$  iterations [32]

▶ However, SGD does not converge with constant  $\gamma$

▶ We cannot take  $\zeta = 0$

### SGD avoids traps almost surely (Theorem 3 of [84])

Assume bounded uniformly exciting noise and  $\gamma_k = \mathcal{O}\left(\frac{1}{k^\kappa}\right)$  for  $\kappa \in (0, 1]$ . Then, SGD avoids strict saddles from any initial condition with probability 1.

### Remark

However, there are LIONS™ hidden in the tall grass: converging to sharp minima or even local *maxima* and other undesirable behaviours are unfortunately possible [126]...

### Q3: How fast does SGD converge to local minimizers?

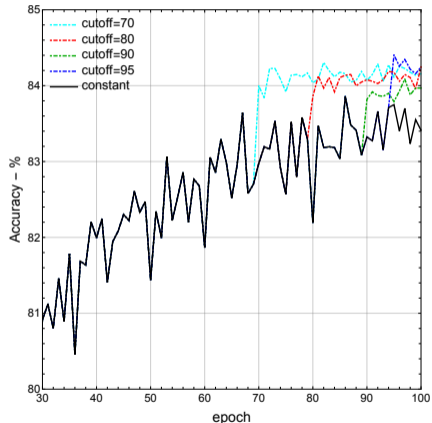
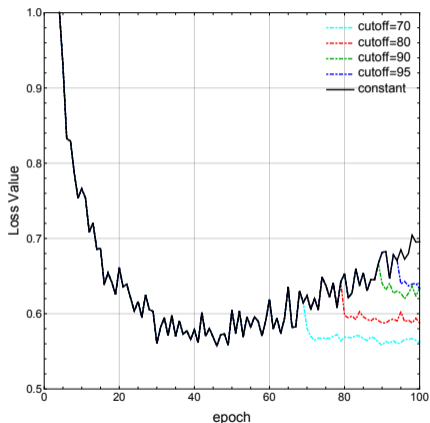
- SGD remains close to Hurwicz minimizers (i.e.,  $\mathbf{x}^* : \lambda_{\min}(\nabla^2 f(\mathbf{x}^*)) > 0$ )
  1. SGD with constant  $\gamma$  can obtain objective value  $\epsilon$ -close to a Hurwicz minimizer in  $\mathcal{O}(1/\epsilon^2)$ -iterations [32, 34]
    - ▶ However, SGD does not converge with constant  $\gamma$
    - ▶ Need averaging which is problematic in non-convex optimization

Using a vanishing step-size helps! (Theorem 4 of [84])

Using  $\gamma_k = \mathcal{O}\left(\frac{1}{k}\right)$ , SGD enjoys a  $\mathcal{O}\left(\frac{1}{k}\right)$  convergence rate in objective value.

## Using $1/k$ step-size decrease helps in practice

- ResNet training at different cool-down cut-offs



## Frank-Wolfe's approach - I

$$f^* := \min_{\mathbf{x} \in \mathcal{R}^p} \left\{ f(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \right\},$$

### Conditional gradient method (CGM, see [57] for review)

A plausible strategy which dates back to 1956 [29]. At iteration  $k$ :

1. Consider the linear approximation of  $f$  at  $\mathbf{x}^k$

$$\phi_k(\mathbf{x}) := f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)^T (\mathbf{x} - \mathbf{x}^k)$$

2. Minimize this approximation within constraint set

$$\hat{\mathbf{x}}^k \in \min_{\mathbf{x} \in \mathcal{X}} \phi_k(\mathbf{x}) = \min_{\mathbf{x} \in \mathcal{X}} \nabla f(\mathbf{x}^k)^T \mathbf{x}$$

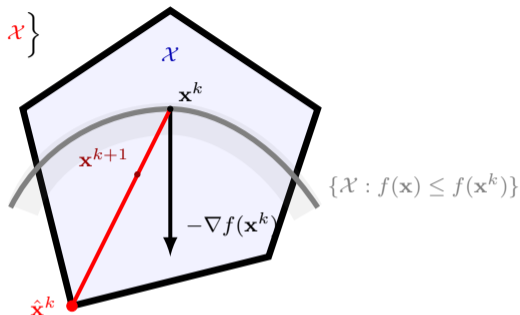
3. Take a step towards  $\hat{\mathbf{x}}^k$  with step-size  $\alpha_k \in [0, 1]$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k (\hat{\mathbf{x}}^k - \mathbf{x}^k)$$

- $\mathbf{x}^{k+1}$  is feasible since it is convex combination of two other feasible points.

## Frank-Wolfe's approach - II

$$f^* := \min_{\mathbf{x} \in \mathbb{R}^p} \left\{ f(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \right\}$$



### Conditional gradient method (CGM)

1. Choose  $\mathbf{x}^0 \in \mathcal{X}$ .
2. For  $k = 0, 1, \dots$  perform:

$$\begin{cases} \hat{\mathbf{x}}^k & := \arg \min_{\mathbf{x} \in \mathcal{X}} \nabla f(\mathbf{x}^k)^T \mathbf{x} \\ \mathbf{x}^{k+1} & := (1 - \alpha_k) \mathbf{x}^k + \alpha \hat{\mathbf{x}}^k, \end{cases}$$

where  $\alpha_k := \frac{2}{k+2}$ .

## Layerwise norm selection

### Input layer with image data

The input  $\mathbf{a}$  is rescaled to  $[-1, 1]$ , ensuring  $\|\mathbf{a}\|_{\text{RMS}} \leq 1$ .

- ▶ The appropriate operator norm is  $\|\mathbf{X}_1\|_{\text{RMS} \rightarrow \text{RMS}} = \sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}} \|\mathbf{X}_1\|_{\mathcal{S}_\infty}$ .
- ▶ lmo is set to  $\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} UV^\top$ .

## Summary of norm choices

Table: Example lmo choices for 1-hot encoded inputs.

Parameter	$W_1$ (image domain)	$W_1$ (1-hot encoded input)		
<b>Norm</b>	RMS $\rightarrow$ RMS	2 $\rightarrow$ RMS	1 $\rightarrow$ RMS	1 $\rightarrow$ $\infty$
<b>LMO</b>	$\max(1, \sqrt{d_{\text{out}}/d_{\text{in}}})UV^\top$	$\sqrt{d_{\text{out}}}UV^\top$	$\text{col}_i(W_1) \mapsto \sqrt{d_{\text{out}}} \frac{\text{col}_i(W_1)}{\ \text{col}_i(W_1)\ _2}$	$\text{sign}(W_L)$
<b>Init.</b>	Semi-orthogonal	Semi-orthogonal	Column-wise normalized Gaussian	Random sign

Table: The choice of lmo can be different between layers and can depend on the assumptions on the input. For simplicity we overload notation and write the reduced SVD as  $W_\ell = U \text{diag}(\sigma) V^\top \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$  for all  $\ell \in [L]$ .

Parameter	$\{W_\ell\}_{\ell \in [2, \dots, L-1]}$	$W_L$		
<b>Norm</b>	RMS $\rightarrow$ RMS	RMS $\rightarrow$ RMS	RMS $\rightarrow$ $\infty$	1 $\rightarrow$ $\infty$
<b>LMO</b>	$\sqrt{d_{\text{out}}/d_{\text{in}}}UV^\top$	$\sqrt{d_{\text{out}}/d_{\text{in}}}UV^\top$	$\text{row}_j(W_L) \mapsto \frac{1}{\sqrt{d_{\text{in}}}} \frac{\text{row}_j(W_L)}{\ \text{row}_j(W_L)\ _2}$	$\frac{1}{d_{\text{in}}} \text{sign}(W_L)$
<b>Init.</b>	Semi-orthogonal	Semi-orthogonal	Row-wise normalized Gaussian	Random sign

## Scion additional experiments

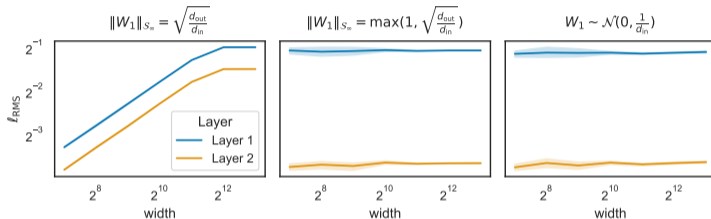
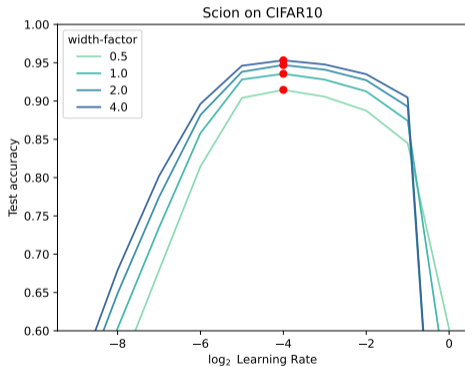


Figure: Coordinate check at initialization: Preactivations vary with spectral scaling  $\sqrt{\frac{d_{out}}{d_{in}}}$  when  $d_{in} > d_{out}$ .

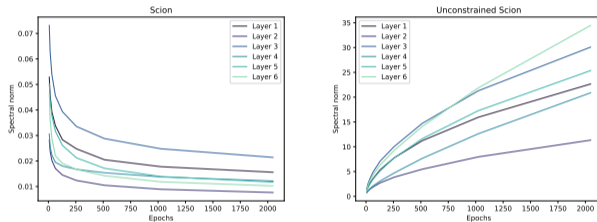
▶ Go back to page 26

## Scion additional experiments

We additionally test SCION on a CNN on the CIFAR10: (Spectral  $\rightarrow$  Spectral  $\rightarrow$  Sign)



## Scion additional experiments



**Figure:** Spectral norm of weight matrices on CIFAR10, while sweeping over total number of epochs. The spectral norm grows empirically as  $\sqrt{n}$  for UNCONSTRAINED SCION with a fixed stepsize  $\gamma$ , whereas the norm (provably) stays bounded for SCION. Due to the spectral norm control, one can expect the weight decay in SCION to be particularly useful for numerical stability in long runs in low-precision.

## Special instantiations

**Table:** Special instantiations of uSCG and SSD according to different choices of norm. The reduced SVD is given as  $d = U \text{diag}(\sigma) V^\top$ .

	Method	$\alpha_k$	Norm	$-\text{Imo}(d) / [d]^\#$	Reference
Steepest descent	SGD	1	Euclidean $\ \cdot\ _2$	$d$	
	SGD with momentum	$[0, 1]$	Euclidean $\ \cdot\ _2$	$d$	
	PSGD (AdaGrad, RMSProp)	1	Euclidean $\ \cdot\ _{2,H}$	$(H)^{-1/2}d$	
	Norm-scaled sign	1	Max-norm $\ \cdot\ _\infty$	$\ d\ _1 \text{sign}(d)$	[61]
	Norm-scaled sign with momentum	$[0, 1]$	Max-norm $\ \cdot\ _\infty$	$\ d\ _1 \text{sign}(d)$	
	Spectral descent	1	Spectral $\ \cdot\ _{S_\infty}$	$\ \sigma\ _1 UV^\top$	[14]
	Spectral descent with momentum	$[0, 1]$	Spectral $\ \cdot\ _{S_\infty}$	$\ \sigma\ _1 UV^\top$	
Conditional Gradient	Normalized SGD	1	Euclidean $\ \cdot\ _2$	$\frac{d}{\ d\ _2}$	[46]
	Normalized SGD with momentum	$[0, 1]$	Euclidean $\ \cdot\ _2$	$-\frac{d}{\ d\ _2}$	[19]
	SignSGD	1	Max-norm $\ \cdot\ _\infty$	$\text{sign}(s)$	[8, Thm. 1]
	Signum	$[0, 1]$	Max-norm $\ \cdot\ _\infty$	$\text{sign}(s)$	[8, Thm. 3]
	Muon <sup>1</sup>	$[0, 1]$	Spectral $\ \cdot\ _{S_\infty}$	$UV^\top$	[59]

<sup>1</sup> With non-Nesterov based momentum.

## References I

- [1] Saeed Alemohammad, Greg Yang, Quynh Nguyen, Jaehoon Lee, and Samuel S. Schoenholz. The recurrent neural tangent kernel. In *Advances in Neural Information Processing Systems*, 2020. (Cited on page 15.)
- [2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019. (Cited on page 21.)
- [3] Adolfo Araújo, Luís Pereira, Natesh Thamrongrat, Francesco Tilli, and Romain Veltz. A mean-field limit for certain deep neural networks. *arXiv preprint arXiv:1906.00193*, 2019. (Cited on page 15.)
- [4] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019. (Cited on pages 15 and 23.)
- [5] Michel Benaïm. Dynamics of stochastic approximation algorithms. In Jacques Azéma, Michel Émery, Michel Ledoux, and Marc Yor, editors, *Séminaire de Probabilités XXXIII*, volume 1709 of *Lecture Notes in Mathematics*, pages 1–68. Springer Berlin Heidelberg, 1999. (Cited on page 91.)
- [6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, September 2012. (Cited on page 73.)
- [7] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. *arXiv preprint arXiv:2410.21265*, 2024. (Cited on page 64.)

## References II

- [8] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. `signsgd`: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018. (Cited on pages 59 and 102.)
- [9] Ilija Bogunovic, Jonathan Scarlett, Stefanie Jegelka, and Volkan Cevher. Adversarially robust optimization with gaussian processes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5765–5775, 2018. (Cited on pages 18, 19, 39, and 40.)
- [10] Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit, 2023. (Cited on pages 33 and 34.)
- [11] Brendan Bordelon and Cengiz Pehlevan. The spectrum of deep learning: A statistical physics perspective. *arXiv preprint arXiv:2401.01234*, 2024. (Cited on page 15.)
- [12] Dan Busbridge, Amitis Shidani, Floris Weers, Jason Ramapuram, Etai Littwin, and Russ Webb. Distillation scaling laws, 2025. (Cited on page 12.)
- [13] Zeyu Cao, Cheng Zhang, Pedro Gimenes, Jianqiao Lu, Jianyi Cheng, and Yiren Zhao. Scaling laws for mixed quantization in large language models, 2024. (Cited on page 12.)
- [14] David Carlson, Volkan Cevher, and Lawrence Carin. Stochastic spectral descent for restricted boltzmann machines. In *Artificial Intelligence and Statistics*, pages 111–119. PMLR, 2015. (Cited on pages 56, 59, and 102.)

## References III

- [15] Minmin Chen, Jeffrey Pennington, Samuel S Schoenholz, Jascha Sohl-Dickstein, and Surya Ganguli. Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, pages 873–882, 2018. (Cited on page 15.)
- [16] Xiaodong Chen, Yuxuan Hu, Xiaokang Zhang, Yanling Wang, Cuiping Li, Hong Chen, and Jing Zhang.  $P^2$  law: Scaling law for post-training after model pruning, 2024. (Cited on page 12.)
- [17] Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in neural information processing systems*, pages 3036–3046, 2018. (Cited on page 15.)
- [18] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems*, 32, 2019. (Cited on page 23.)
- [19] Ashok Cutkosky and Harsh Mehta. Momentum improves normalized SGD. In *International conference on machine learning*, pages 2260–2268. PMLR, 2020. (Cited on page 102.)
- [20] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances in Neural Information Processing Systems*, pages 2253–2261, 2016. (Cited on page 15.)
- [21] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685, 2019. (Cited on page 15.)

## References IV

- [22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159, July 2011. (Cited on page 48.)
- [23] Weinan E, Chao Ma, and Lei Wu. A comparative analysis of optimization and generalization properties of two-layer neural network and random feature models under gradient descent dynamics. *Science China Mathematics*, 2020. (Cited on page 21.)
- [24] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021. (Cited on page 80.)
- [25] Zhen Fang, Tiejong Zeng, and Zhouchen Lin. Modeling deep neural networks with differential equations: Mean-field theory of residual networks. *arXiv preprint arXiv:2001.01704*, 2020. (Cited on page 15.)
- [26] Kayvon Fatahalian. Parallel and distributed deep learning training. Lecture Notes, Carnegie Mellon University, 2020. (Cited on pages 76, 77, and 81.)
- [27] Samuel Foreman. High-performance computing workshop 2024 slides, 2024. (Cited on page 78.)
- [28] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. (Cited on pages 18, 19, 39, and 40.)
- [29] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Res. Logis. Quart.*, 3:95–110, 1956. (Cited on page 95.)

## References V

- [30] Spencer Frei, Niladri S Chatterji, and Peter Bartlett. Benign overfitting without linearity: Neural network classifiers trained by gradient descent for noisy linear data. In *Conference on Learning Theory*, pages 2668–2703. PMLR, 2022. (Cited on page 24.)
- [31] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015. (Cited on pages 85, 86, and 87.)
- [32] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points — Online stochastic gradient for tensor decomposition. In *COLT '15: Proceedings of the 28th Annual Conference on Learning Theory*, 2015. (Cited on pages 92 and 93.)
- [33] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatle, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. (Cited on page 12.)
- [34] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013. (Cited on page 93.)
- [35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. (Cited on page 21.)

## References VI

- [36] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, April 2018. (Cited on page 73.)
- [37] Moritz Haas, Jin Xu, Volkan Cevher, and Leena Chennuru Vankadara.  $\mu\mathbf{P}^2$ : Effective sharpness aware minimization requires layerwise perturbation scaling. 2024. (Cited on page 33.)
- [38] Benjamin D Haeffele and René Vidal. Global optimality in neural network training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7331–7339, 2017. (Cited on page 84.)
- [39] Boris Hanin. Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2596–2604, 2019. (Cited on page 15.)
- [40] Boris Hanin and Mihai Nica. Finite depth and width corrections to the neural tangent kernel. *arXiv preprint arXiv:1909.05989*, 2019. (Cited on page 15.)
- [41] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, pages 571–581, 2018. (Cited on page 15.)
- [42] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018. (Cited on page 80.)
- [43] Soufiane Hayou. Mean-field behaviour of neural tangent kernel for deep residual networks with polynomial width. *arXiv preprint arXiv:2301.10763*, 2023. (Cited on page 15.)

## References VII

- [44] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. *arXiv preprint arXiv:1902.06853*, 2019. (Cited on page 15.)
- [45] Soufiane Hayou and Greg Yang. Infinite-depth limit of neural networks. *arXiv preprint arXiv:2302.07292*, 2023. (Cited on page 15.)
- [46] Elad Hazan, Kfir Levy, and Shai Shalev-Shwartz. Beyond convexity: Stochastic quasi-convex optimization. *Advances in neural information processing systems*, 28, 2015. (Cited on page 102.)
- [47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. (Cited on page 21.)
- [48] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. (Cited on page 5.)
- [49] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017. (Cited on page 7.)

## References VIII

- [50] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. (Cited on pages 9 and 13.)
- [51] Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: NNGP and NTK for deep attention networks. In *International Conference on Machine Learning*, 2020. (Cited on page 15.)
- [52] Wei Hu and Dehua Huang. Universality laws for high-dimensional learning: A kernel perspective. *arXiv preprint arXiv:2105.13361*, 2021. (Cited on page 15.)
- [53] Minshuo Huang, Yuan Yao, Xiaojing Yuan, Yangyang Guo, Tuo Chen, Guillermo Sapiro, Tom Goldstein, Raman Arora, and Furong Zhao. Deep networks with exact polynomial spectral bias. In *Advances in Neural Information Processing Systems*, 2020. (Cited on page 15.)
- [54] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015. (Cited on pages 18, 19, 39, and 40.)
- [55] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations, May 2024. (Cited on page 74.)

## References IX

- [56] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018. (Cited on page 23.)
- [57] M. Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. *JMLR W&CP*, 28(1):427–435, 2013. (Cited on page 95.)
- [58] Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. (Cited on page 66.)
- [59] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cecista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. (Cited on pages 56, 59, and 102.)
- [60] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. (Cited on pages 6, 7, 8, 9, and 13.)
- [61] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014. (Cited on page 102.)

## References X

- [62] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. (Cited on page 10.)
- [63] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 48.)
- [64] Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. *arXiv preprint arXiv:2405.14813*, 2024. (Cited on page 62.)
- [65] Guillaume Leclerc and Aleksander Madry. The Two Regimes of Deep Network Training, February 2020. *arXiv:2002.10376 [cs, stat]*. (Cited on page 73.)
- [66] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012. (Cited on page 21.)
- [67] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. (Cited on page 15.)
- [68] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, 2019. (Cited on page 23.)

## References XI

- [69] Jason D. Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. First-order methods almost always avoid strict saddle points. *Mathematical Programming*, 176(1):311–337, February 2019. (Cited on page 92.)
- [70] Kfir Levy. Online to offline conversions, universality and adaptive minibatch sizes. In *Advances in Neural Information Processing Systems*, pages 1613–1622, 2017. (Cited on pages 49 and 50.)
- [71] Qianxiao Li and Yuzhe Wei. Statistical mechanics of neural networks: A mathematical approach. *arXiv preprint arXiv:2106.05001*, 2021. (Cited on page 15.)
- [72] Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyang Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, Yangyu Tao, Bin Cui, and Di Wang. Surge phenomenon in optimal learning rate and batch size scaling, 2024. (Cited on page 10.)
- [73] Adam Litwin-Kumar, Jaehoon Lee, Jeffrey Pennington, Samuel S. Schoenholz, and Jascha Sohl-Dickstein. Neural tangent kernel eigenvalues accurately predict generalization. *arXiv preprint arXiv:1906.04345*, 2020. (Cited on page 15.)
- [74] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training, 2025. (Cited on pages 66 and 69.)

## References XII

- [75] Lennart Ljung. Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, 22(4):551–575, August 1977. (Cited on page 91.)
- [76] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. (Cited on page 60.)
- [77] Tao Luo, Zhi-Qin John Xu, Zheng Ma, and Yaoyu Zhang. Phase diagram for two-layer relu neural networks at infinite-width limit. *Journal of Machine Learning Research*, 2021. (Cited on pages 22, 23, and 24.)
- [78] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the Limits of Weakly Supervised Pretraining, May 2018. (Cited on page 74.)
- [79] Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, and Richard E. Turner. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018. (Cited on page 15.)
- [80] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training, 2018. (Cited on page 10.)
- [81] Sean McLeish, John Kirchenbauer, David Yu Miller, Siddharth Singh, Abhinav Bhatele, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Gemstones: A model suite for multi-faceted scaling laws, 2025. (Cited on page 13.)

## References XIII

- [82] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018. (Cited on page 15.)
- [83] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layers neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 2018. (Cited on page 21.)
- [84] Panayotis Mertikopoulos, Nadav Hallak, Ali Kavis, and Volkan Cevher. On the almost sure convergence of stochastic gradient descent in non-convex problems. pages 1117–1128, 2020. (Cited on pages 91, 92, and 93.)
- [85] Panayotis Mertikopoulos, Ya-Ping Hsieh, and Volkan Cevher. A unified stochastic approximation framework for learning in games. *Mathematical Programming*, 203(1):559–609, 2024. (Cited on page 41.)
- [86] Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. Stochastic conditional gradient methods: From convex minimization to submodular maximization. *arXiv preprint arXiv:1804.09554*, 2018. (Cited on page 59.)
- [87] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. (Cited on page 12.)
- [88] Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996. (Cited on page 15.)
- [89] Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 103:127–152, 2005. (Cited on page 56.)

## References XIV

- [90] Lorenzo Noci, Sebastian Goldt, Florent Krzakala, and Lenka Zdeborová. Precise high-dimensional asymptotics for learning with random features and kernel models. *arXiv preprint arXiv:2106.06515*, 2021. (Cited on page 15.)
- [91] Lorenzo Noci, Sebastian Goldt, Florent Krzakala, and Lenka Zdeborová. Convergence rates for neural networks with random features. *arXiv preprint arXiv:2301.12345*, 2023. (Cited on page 15.)
- [92] Lorenzo Noci, Alexandru Meterez, Thomas Hofmann, and Antonio Orvieto. Why do learning rates transfer? reconciling optimization and scaling limits for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. (Cited on page 33.)
- [93] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019. (Cited on page 15.)
- [94] Jeffrey Pennington, Samuel S Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Advances in Neural Information Processing Systems*, pages 4785–4795, 2017. (Cited on page 15.)
- [95] Thomas Pethick, Grigorios G Chrysos, and Volkan Cevher. Revisiting adversarial training for the worst-performing class. *Transactions on Machine Learning Research*, 2023. (Cited on pages 18, 19, 39, and 40.)
- [96] Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained lmos, 2025. (Cited on pages 59, 64, 65, and 71.)

## References XV

- [97] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *arXiv preprint arXiv:1606.05340*, 2016. (Cited on page 15.)
- [98] Microsoft Research. Deepspeed: Extreme-scale model training for everyone, 2020. Accessed: 2025-02-24. (Cited on page 82.)
- [99] Grégory Roth and William H. Sandholm. Stochastic approximations with constant step size and differential inclusions. *SIAM Journal on Control and Optimization*, 51(1):525–555, 2013. (Cited on page 91.)
- [100] Grant M Rotskoff and Eric Vanden-Eijnden. Neural networks as interacting particle systems: Asymptotic convexity of the loss landscape and universal scaling of the approximation error. *arXiv preprint arXiv:1805.00915*, 2018. (Cited on page 15.)
- [101] Oleh Rybkin, Michal Nauman, Preston Fu, Charlie Snell, Pieter Abbeel, Sergey Levine, and Aviral Kumar. Value-based deep rl scales predictably, 2025. (Cited on page 7.)
- [102] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? In *NeurIPS*, 2023. (Cited on page 11.)
- [103] Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D. Cox, and Rameswar Panda. Power Scheduler: A Batch Size and Token Number Agnostic Learning Rate Scheduler, September 2024. (Cited on page 74.)

## References XVI

- [104] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-Lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. (Cited on page 79.)
- [105] Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A central limit theorem. *arXiv preprint arXiv:1808.09372*, 2018. (Cited on page 15.)
- [106] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. (Cited on page 48.)
- [107] Andrey Nikolayevich Tikhonov et al. On the stability of inverse problems. In *Dokl. akad. nauk sssr*, volume 39, pages 195–198, 1943. (Cited on page 60.)
- [108] Leena Chennuru Vankadara, Jin Xu, Moritz Haas, and Volkan Cevher. On feature learning in structured state space models. 2024. (Cited on page 33.)
- [109] Leena Chennuru Vankadara, Jin Xu, Moritz Haas, and Volkan Cevher. On feature learning in structured state space models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. (Cited on pages 35 and 36.)
- [110] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. (Cited on page 11.)

## References XVII

- [111] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011. (Cited on pages 85, 86, and 87.)
- [112] Weihang Xu and Simon S. Du. Over-parameterization exponentially slows down gradient descent for learning a single neuron, 2023. (Cited on page 24.)
- [113] Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems*, 3:269–296, 2021. (Cited on page 80.)
- [114] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, pages 17084–17097, 2021. (Cited on pages 28 and 33.)
- [115] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019. (Cited on page 15.)
- [116] Greg Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes, 2021. (Cited on page 27.)

## References XVIII

- [117] Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pages 11727–11737. PMLR, 2021. (Cited on pages 15, 21, 28, 29, 30, and 31.)
- [118] Greg Yang, Adam Littwin-Kumar, Jeffrey Pennington, Samuel S Schoenholz, and Jascha Sohl-Dickstein. Tensor programs iv: Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2401.00010*, 2024. (Cited on page 15.)
- [119] Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023. (Cited on page 26.)
- [120] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019. (Cited on page 10.)
- [121] Alp Yurtsever, Suvrit Sra, and Volkan Cevher. Conditional gradient methods via stochastic path-integrated differential estimator. In *International Conference on Machine Learning*, pages 7282–7291. PMLR, 2019. (Cited on page 61.)
- [122] Moslem Zamani and François Glineur. Exact convergence rate of the last iterate in subgradient methods, July 2023. (Cited on page 74.)
- [123] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022. (Cited on page 7.)

## References XIX

- [124] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling Vision Transformers, June 2022. (Cited on page 74.)
- [125] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method, 2024. (Cited on page 12.)
- [126] Liu Ziyin, Botao Li, James B Simon, and Masahito Ueda. SGD can converge to local maxima. In *International Conference on Learning Representations, 2022*. (Cited on page 92.)