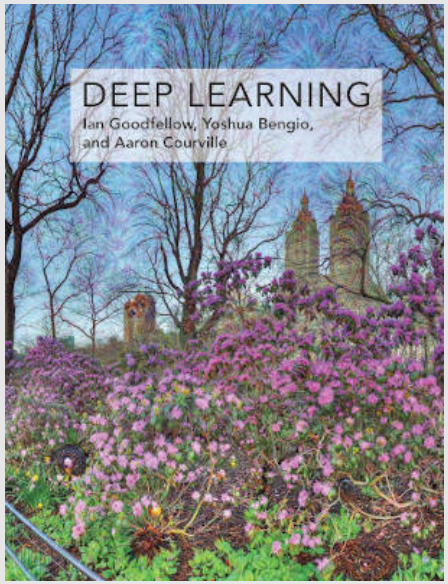
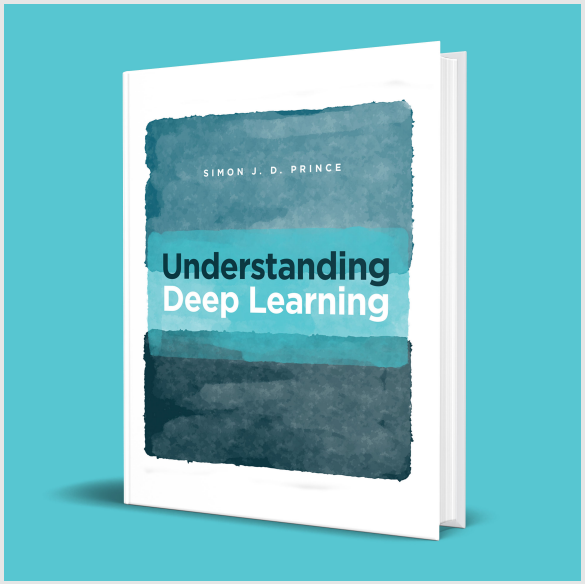
An introduction to optimization for deep learning

Edouard Pauwels

CIMI Thematic school: Optimization & algorithms for high-dimensional machine learning and inference (October 2024)

Motivation, content, and acknowledgements

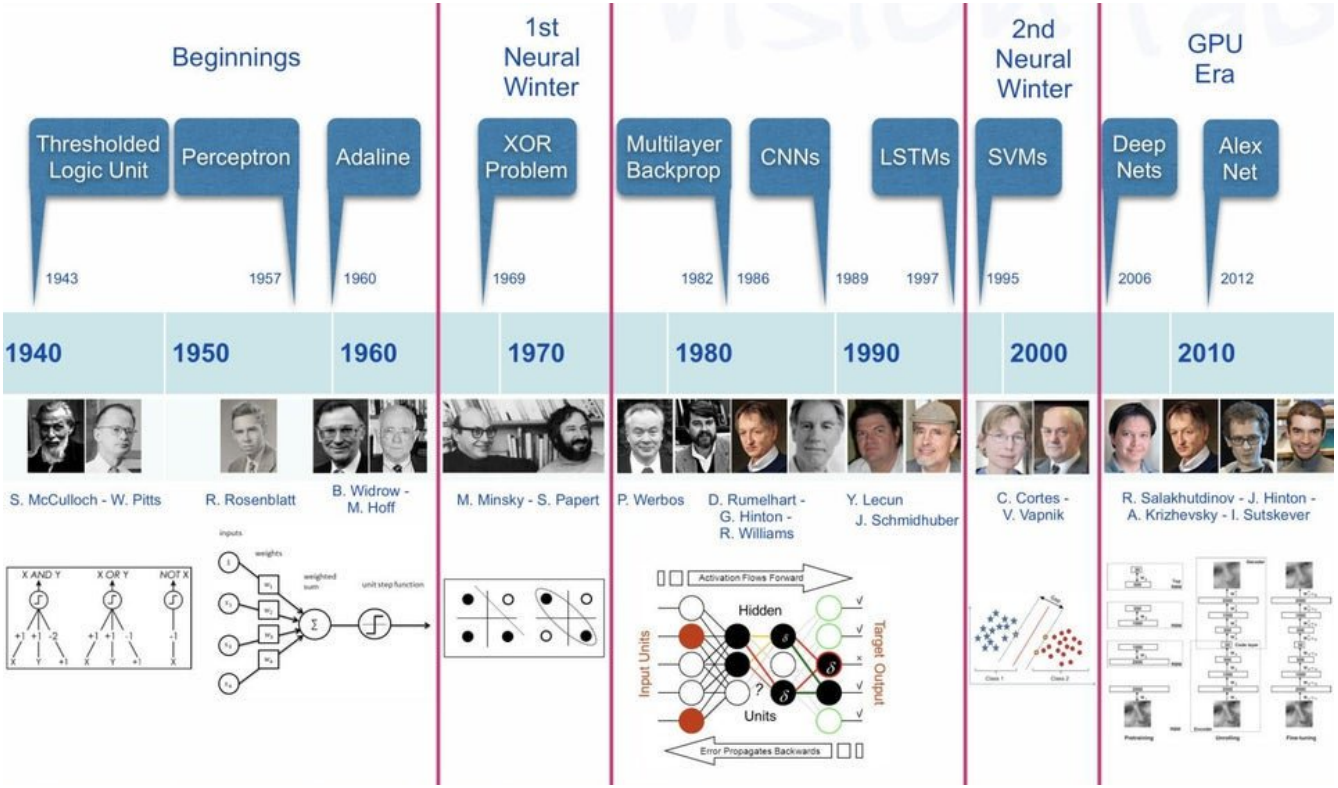




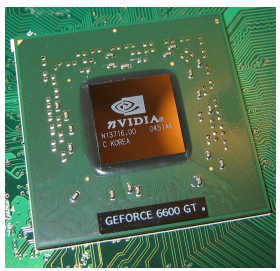
- Deep learning = machine learning with a specific class of models
- Deep network training reduces to an optimization problem
- Two important structural specificities: large sum, compositional model.
- Specific context, with its own goals and difficulties, dedicated algorithms.

Many collaborators: Bolte, Boustany, Castera, Févotte, Le, Le, Glaudin, Rios-Zertuche, Silvetti, Traoré, Vaiter ...

Why do we have AI/ML now?



- Research
- Computer hardware, large amount of data, open source softwares.
- A culture of competition, benchmarking, sharing (Donoho)



IMAGENET

TensorFlow

What is behind AI practically?

Plan

Plan

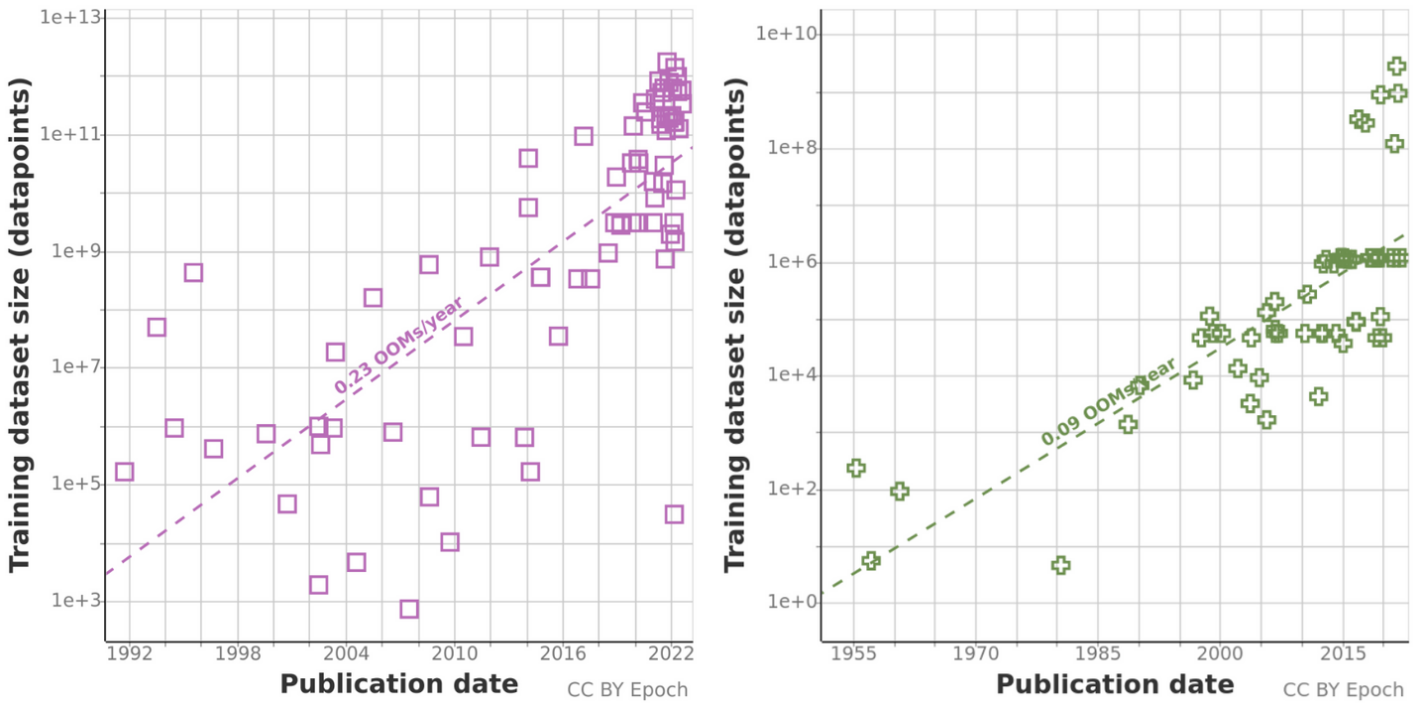
Basic presentation: supervised learning, reduction to an optimization problem

Predict y from x with n training examples: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

Parameterized model, p # model parameters: $F : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$.

Solve $F(x) \simeq y$

$$\min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad := \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2 \quad = \quad \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$



Villalobos and Ho (2022). Trends in Training Dataset Sizes. Online

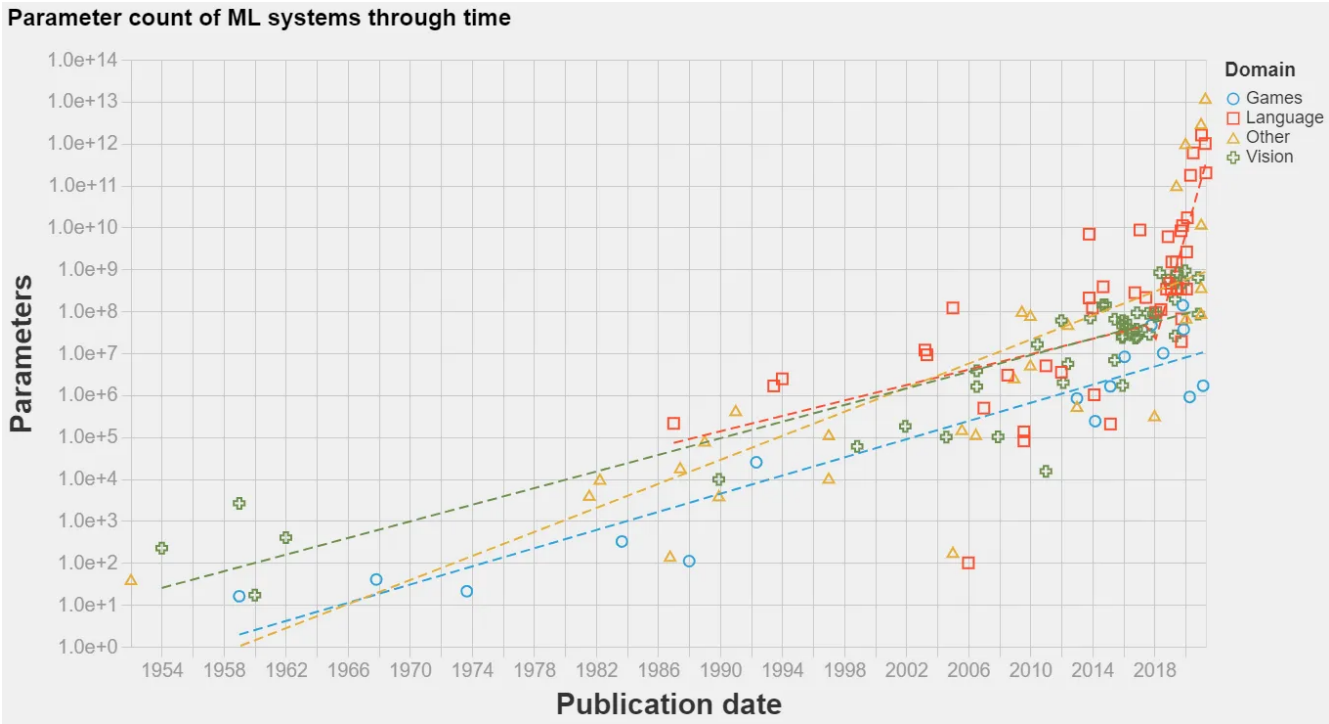
Basic presentation: supervised learning, reduction to an optimization problem

Predict y from x with n training examples: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

Parameterized model, p # model parameters: $F : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$.

Solve $F(x) \simeq y$

$$\min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad := \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2 \quad = \quad \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$



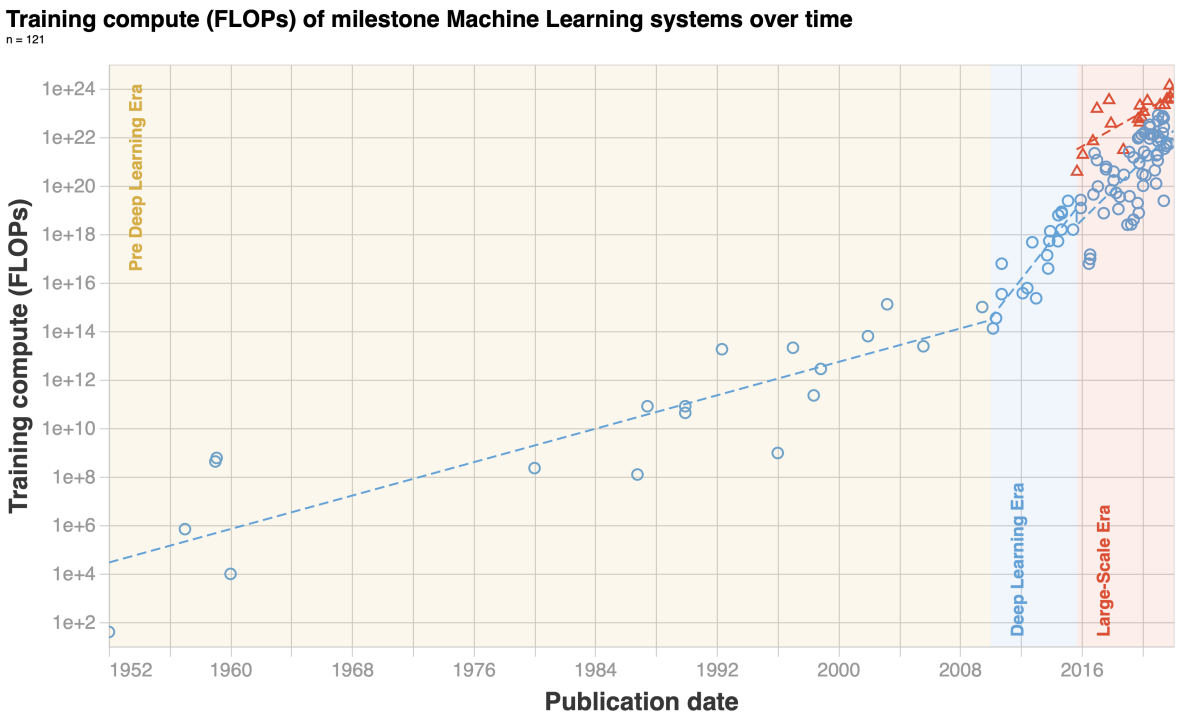
Sevilla *et.al.* (2021). Parameter Counts in Machine Learning. Online.

Predict y from x with n training examples: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

Parameterized model, p # model parameters: $F : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$.

Solve $F(x) \simeq y$

$$\min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad := \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2 \quad = \quad \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$



Sevilla *et.al.* (2022). Compute trends across three eras of machine learning. IEEE IJCNN.

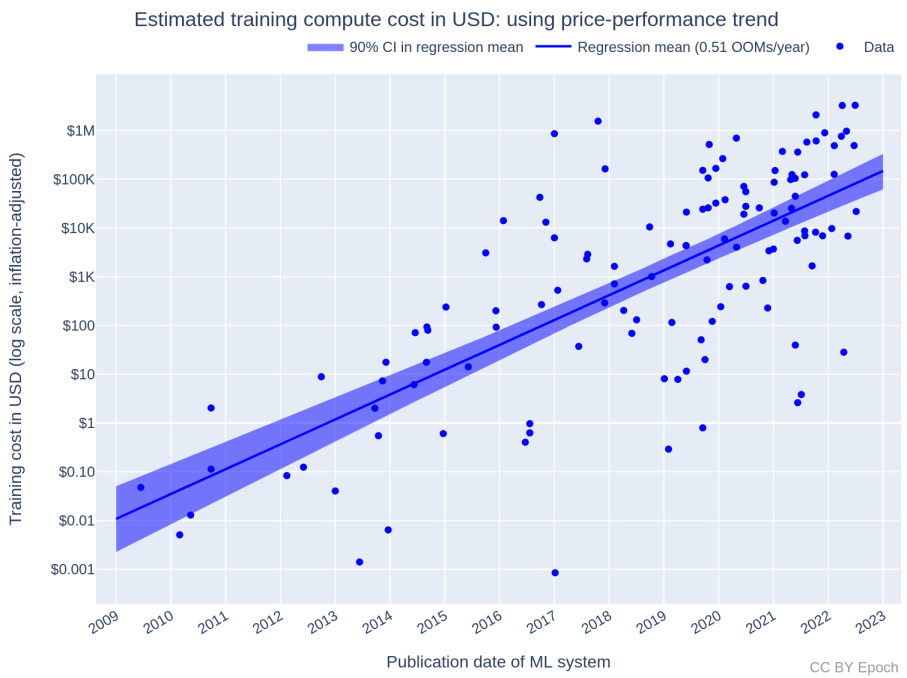
Basic presentation: supervised learning, reduction to an optimization problem

Predict y from x with n training examples: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

Parameterized model, p # model parameters: $F : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$.

Solve $F(x) \simeq y$

$$\min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad := \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2 \quad = \quad \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$



Ben Cottier (2023). Trends in the dollar training cost of machine learning systems. Online.

Extensions: self supervision, multiple models

Self supervision: n unlabeled training examples: $(x_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$.

"Create" an instrumental learning task.

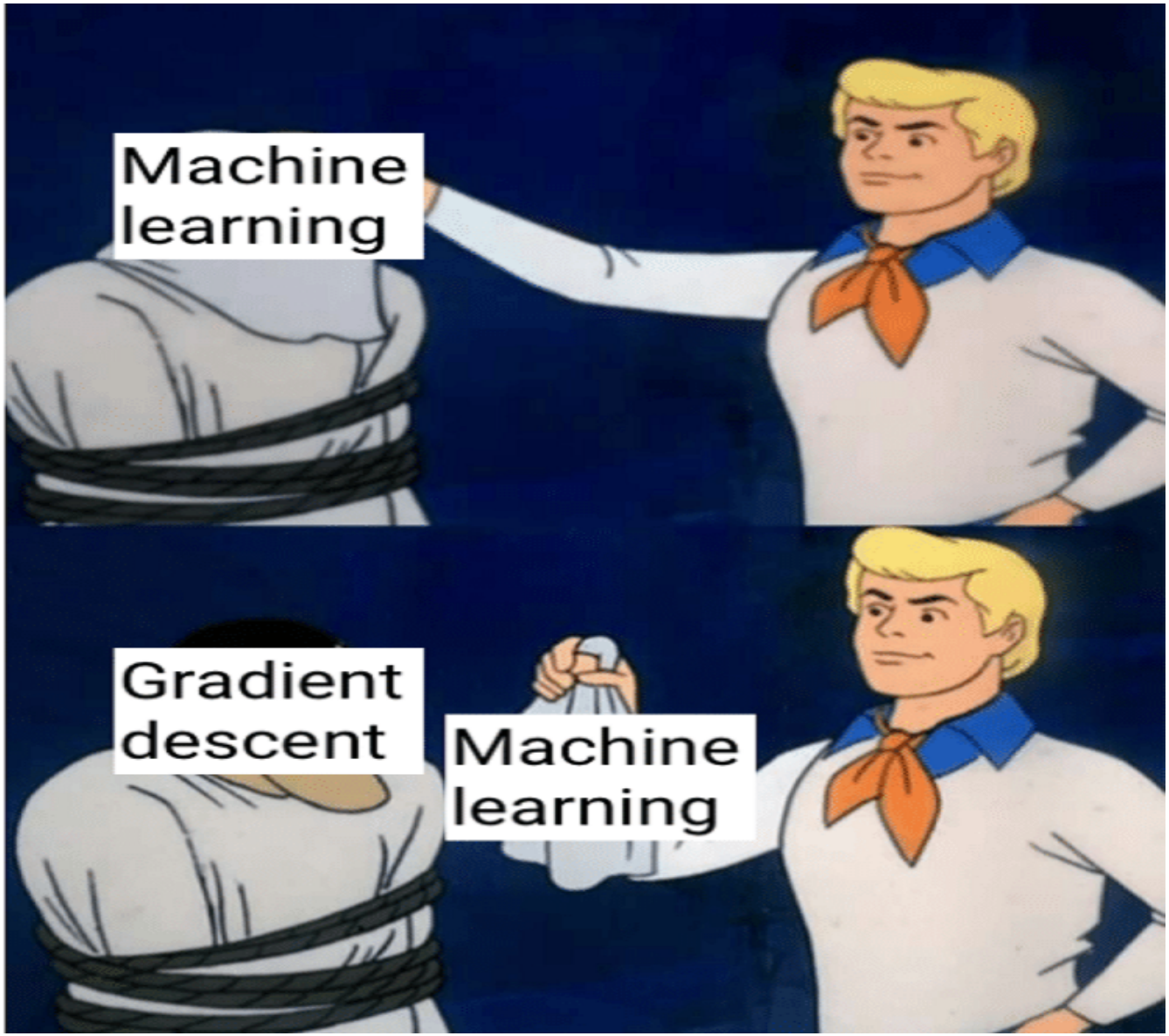
- Predict the next word in a sentence (large language models).
- Remove noise from a noisy image (Plug and play, diffusion models).
- Recognize similar images, rotation, translation, contrast, zoom (Vision transformers).

"Foundational models", "generative AI", structured data.

Multiple intracting networks: for example

- Distillation: use the output of a model as input for another one.
- Generative adversarial networks.
- Mixture of experts.
- Multimodality: *e.g.* text + image.

Mathematical formulation: based on minimization, optimality.

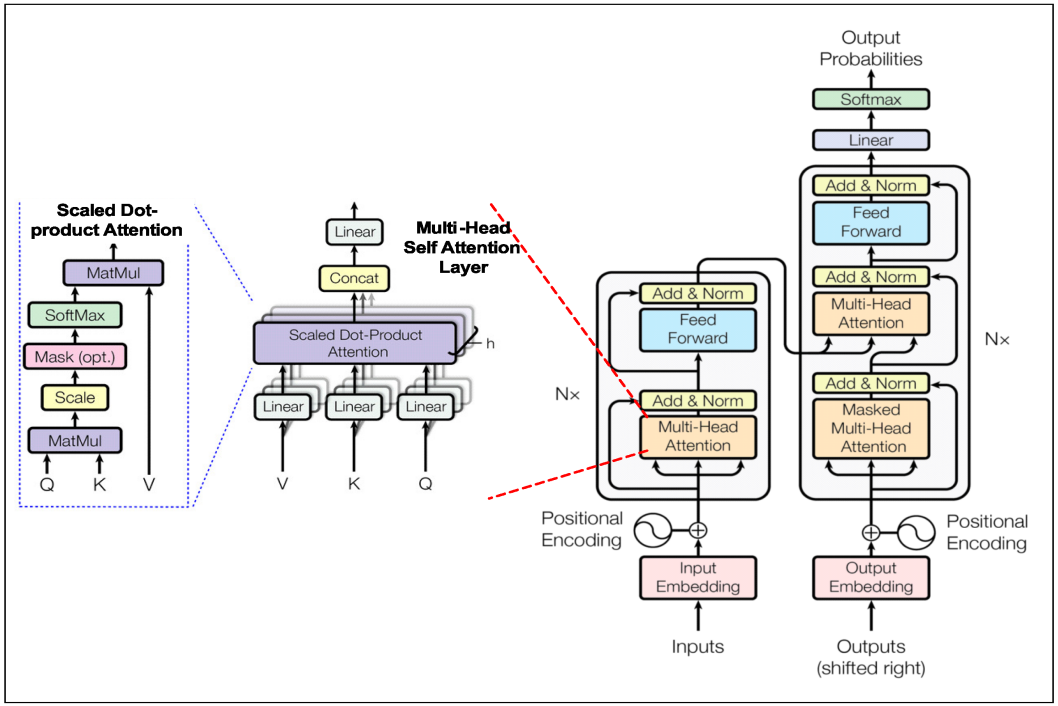

Machine learning
Gradient descent
Machine learning

- Adadelta (> 2010)
- Adagrad (> 2010)
- Adam (> 2010)
- AdamW (> 2010)
- Adamax (> 2010)
- Ftrl (> 2010)
- Nadam (> 2010)
- RMSprop (> 2010)
- SGD (1951)

- Adadelta (> 2010)
- Adagrad (> 2010)
- Adam (> 2010)
- AdamW (> 2010)
- SparseAdam (> 2010)
- Adamax (> 2010)
- Averaged SGD (90's)
- LBFGS (70's)
- RMSprop (> 2010)
- Rprop, signs (90's)
- SGD (1951)

$$\min_{\theta \in \mathbb{R}^p} J(\theta) := \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$

Derivative of J_i?



Vaswani, Ashish, *et. al.* Attention is all you need. (2017). ($\sim$ 70000 citations)

Derivatives in AI/ML:

- Numbers are extremely large.
- Cannot compute by hand.

Backpropagation, autodiff: differentiating programs

```
from jax import *

def fun(x):
    return x * x

gradFun = jax.grad(fun)
gradFun(3.14)
```

```
DeviceArray(6.28, dtype=float32, weak_type=True)
```

$$\min_{\theta \in \mathbb{R}^p} \quad \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$

- **Neural network training:** optimization.
- **Large dimension (p):** gradient algorithms.
- **Large sum (n):** Stochastic subsampling approximation.
- **Compositional structure:** algorithmic differentiation.

Profusion of numerical tools: democratized the usage of these models. Goes beyond neural nets (differentiable programming).

History: algorithmic differentiation + subsampling for neural network training since at least the 80s. Explosion of scales $\sim$ 2010s.

Neural network recipe

First step: define the problem and solution method

- Choose a dataset.
- Define a parametric model.
- Define the modalities of training.

Described by a numerical program.

Train: transparent and automatic execution

- Compile training modalities.
- Optimize using available numerical solvers.
- Automatized calculus of derivatives.

Tensorflow, Pytorch, Jax:
$\sim$ transparent, efficient and flexible implementation of this recipe.

Plan

- $f : \mathbb{R}^p \to \mathbb{R}^m$ differentiable expressed as a "program" ($\sim$ composition).
  g_i "elementary", appropriate size, differentiable.

$$f = g_L \circ g_{L-1} \ldots \circ g_1$$

- autodiff : efficient algorithm implementing the chain rule.

$$\boxed{\operatorname{Jac} f(x_0) = \operatorname{Jac} g_L(x_{L-1}) \times \operatorname{Jac} g_{L-1}(x_{L-2}) \times \ldots \times \operatorname{Jac} g_1(x_0)}$$

Straight line program
Input: $x_0 \in \mathbb{R}^p$.
1: **for** $k = 1, 2, \ldots, L$ **do**
2: $\quad x_k = g_i(x_{k-1})$
3: **end for**
Return: $x_L \in \mathbb{R}^m$.

Forward differentiation: $F_i = g_i \circ \ldots \circ g_1$, $F_L = f$,

$$F_{i+1} = g_{i+1} \circ F_i$$

Backward differentiation: $H_j = g_L \circ \ldots \circ g_j$, $H_1 = f$,

$$H_j = H_{j+1} \circ g_j$$

$$\operatorname{Jac} F_{i+1}(x_0) = \operatorname{Jac} g_{i+1}(x_i) \times \operatorname{Jac} F_i(x_0)$$
$$\operatorname{Jac} H_j(x_{j-1}) = \operatorname{Jac} H_{j+1}(x_j) \times \operatorname{Jac} g_j(x_{j-1})$$

Time and space complexity

Forward derivative computation

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0			
Memory 2				

Take away: memory footprint independent of the length of the composition

Forward derivative computation

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0			
Memory 2	$\mathrm{Jac}\, g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$		
Memory 2	$\mathrm{Jac}\, g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_1 = g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_1 = g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_1(x_0)$	$\mathrm{Jac}\, g_2(x_1) \times \mathrm{Jac}\, g_1(x_0)$		

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_1 = g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_1 = g_1(x_0)$	$x_2 = g_2(x_1)$		
Memory 2	$\mathrm{Jac}\, g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_2 = g_2 \circ g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_2 = g_2 \circ g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_2 \circ g_1(x_0)$	$\mathrm{Jac}\, g_3(x_2) \times \mathrm{Jac}\, g_2 \circ g_1(x_0)$		

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_2 = g_2 \circ g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_3 \circ g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_2 = g_2 \circ g_1(x_0)$	$x_3 = g_3(x_2)$		
Memory 2	$\mathrm{Jac}\, g_3 \circ g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	$x_3 = g_3 \circ g_2 \circ g_1(x_0)$			
Memory 2	$\mathrm{Jac}\, g_3 \circ g_2 \circ g_1(x_0)$			

Take away: memory footprint independent of the length of the composition

Backward derivative computation

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0			
Memory 2				

Take away: memory footprint depends on the length of the composition

Backward derivative computation

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$		
Memory 2				

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$	$x_2 = g_2(x_1)$	
Memory 2				

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$	$x_2 = g_2(x_1)$	$x_3 = g_3(x_2)$
Memory 2				

Take away: memory footprint depends on the length of the composition

Backward derivative computation

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$	$x_2 = g_2(x_1)$	$x_3 = g_3(x_2)$
Memory 2			$\mathrm{Jac}\, g_3(x_2)$	

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$		$x_3 = g_3(x_2)$
Memory 2			$\mathrm{Jac}\, g_3(x_2)$	

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0	$x_1 = g_1(x_0)$		$x_3 = g_3(x_2)$
Memory 2		$\mathrm{Jac}\, g_3 \circ g_2(x_1)$	$\mathrm{Jac}\, g_3(x_2)$	

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0			$x_3 = g_3(x_2)$
Memory 2		$\mathrm{Jac}\, g_3 \circ g_2(x_1)$		

Take away: memory footprint depends on the length of the composition

$$g_3 \circ g_2 \circ g_1$$

Memory 1	x_0			$x_3 = g_3(x_2)$
Memory 2	$\mathrm{Jac}\, g_3 \circ g_2 \circ g_1(x_0)$	$\mathrm{Jac}\, g_3 \circ g_2(x_1)$		

Take away: memory footprint depends on the length of the composition

Intuition: $g\colon \mathbb{R}^p \to \mathbb{R}^p$, "elementary".
Similar computational cost: $g(x)$, $\operatorname{Jac} g(x)v$, $\operatorname{Jac} g(x)^T v$ for $x, v \in \mathbb{R}^p$.

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \xrightarrow[g]{} \begin{pmatrix} f(x_1) \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \qquad \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_p \end{pmatrix} \xrightarrow[\operatorname{Jac} g(x)]{} \begin{pmatrix} f'(x_1)v_1 \\ v_2 \\ \vdots \\ v_p \end{pmatrix} \qquad \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{pmatrix} \xrightarrow[\operatorname{Jac} g(x)^T]{} \begin{pmatrix} f'(x_1)u_1 \\ u_2 \\ \vdots \\ u_p \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \xrightarrow[g]{} \begin{pmatrix} x_1x_2 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \qquad \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_p \end{pmatrix} \xrightarrow[\operatorname{Jac} g(x)]{} \begin{pmatrix} x_2v_1 + x_1v_2 \\ v_2 \\ \vdots \\ v_p \end{pmatrix} \qquad \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{pmatrix} \xrightarrow[\operatorname{Jac} g(x)^T]{} \begin{pmatrix} x_2u_1 \\ x_1u_1 + u_2 \\ \vdots \\ u_p \end{pmatrix}$$

Basic time complexity estimates:
Function evaluation $\sim$ Jacobian vector product (JVP) $\sim$ Vector Jacobian product (VJP).
Jacobian Jacobians product $\sim p\times$ Function evaluation.

Forward versus backward

$f : \mathbb{R}^p \to \mathbb{R}$ differentiable expressed $\sim$ composition, $\ell \colon \mathbb{R}^p \to \mathbb{R}$.

$$f = \ell \circ g_L \circ g_{L-1} \ldots \circ g_1$$

Chain rule: $\operatorname{Jac} f(x_0) = \nabla f(x_0)^T$

$$\nabla f(x_0)^T = \operatorname{Jac} \ell(x_L) \nabla \ell(x_L)^T \times \operatorname{Jac} g_L(x_{L-1}) \times \operatorname{Jac} g_{L-1}(x_{L-2}) \times \ldots \times \operatorname{Jac} g_1(x_0)$$

Forward AD: $\left(\nabla \ell^T \times \left(\operatorname{Jac} g_L \times \left(\operatorname{Jac} g_{L-1} \times \left(\ldots (\operatorname{Jac} g_2 \times \operatorname{Jac} g_1) \ldots \right)\right)\right)\right)$
Backward AD: $\left(\left(\ldots \left(\left(\nabla \ell^T \times \operatorname{Jac} g_L\right) \times \operatorname{Jac} g_{L-1}\right) \times \ldots \operatorname{Jac} g_2\right) \times \operatorname{Jac} g_1\right)$

Straight line program
Input: $x_0 \in \mathbb{R}^p$,
$g_k \colon \mathbb{R}^p \to \mathbb{R}^p$.
1: **for** $k = 1, 2, \ldots, L$ **do**
2: $x_k = g_k(x_{k-1})$
3: **end for**
Return: $\ell(x_L) \in \mathbb{R}$.

Backward and forward algorithmic differentiation (AD)

$L \geq p$	prog. eval	fwd AD	bwd AD
Computation time	L	$O(Lp)$	$O(L)$
Memory footprint	p	$O(p^2)$	$O(Lp)$

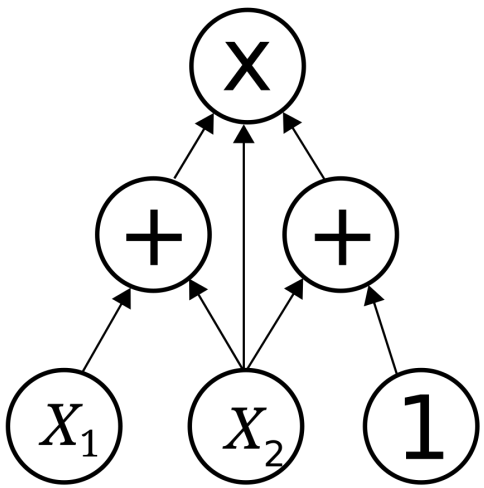
Remark: If $f \colon \mathbb{R} \to \mathbb{R}^p$, or for a single directional derivative, reversed situation.

Baur-Strassen Theorem:

$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$ for rational functions,

instead of the naive $\text{Computing cost } (f, \nabla f) \leq p \text{ Computing cost } (f)$

Computing cost: arithmetic circuit complexity $\sim$ minimal umber of operations



Proof: backward algorithmic differentiation.

Time complexity of derivative computation:

Baur-Strassen Theorem:

$$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$$

Example: $(a, b, c, d, e, f) \mapsto abcdef$. Multiply p numbers.

Time complexity of derivative computation:

Baur-Strassen Theorem:

$$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$$

Example: $(a, b, c, d, e, f) \mapsto abcdef$. Multiply p numbers.

ab	abc	$abcd$	$abcde$	$abcdef$	$p-1$ multiplications

Baur-Strassen Theorem:

$$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$$

Example: $(a, b, c, d, e, f) \mapsto abcdef$. Multiply p numbers.

ab	abc	$abcd$	$abcde$	$abcdef$	$p-1$ multiplications
$bcdef$	$cdef$	def	ef		$p-2$ multiplications

Baur-Strassen Theorem:

$$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$$

Example: $(a, b, c, d, e, f) \mapsto abcdef$. Multiply p numbers.

ab	abc	$abcd$	$abcde$	$abcdef$	$p-1$ multiplications
$bcdef$	$cdef$	def	ef		$p-2$ multiplications
	$acdef$	$abdef$	$abcef$	$abcdf$	$p-2$ multiplications

Baur-Strassen Theorem:

$$\boxed{\text{Computing cost } (f, \nabla f) \leq 5 \text{ Computing cost } (f)}$$

Example: $(a, b, c, d, e, f) \mapsto abcdef$. Multiply p numbers.

ab	abc	$abcd$	$abcde$	$abcdef$	$p-1$ multiplications
$bcdef$	$cdef$	def	ef		$p-2$ multiplications
	$acdef$	$abdef$	$abcef$	$abcdf$	$p-2$ multiplications

Differentiable programming (DP) framework:
numerical program, **fixed library** of basic operations
$\rightarrow$ automatized Jacobians chaining
$g_L \circ \ldots \circ g_1 \sim$ numerical program $\rightarrow$ $\mathrm{Jac}\, g_L \circ \ldots \circ g_1 \sim$ numerical program.

Specification for elementary g_i:
function evaluation + vector-Jacobian / Jacobian-vector product.

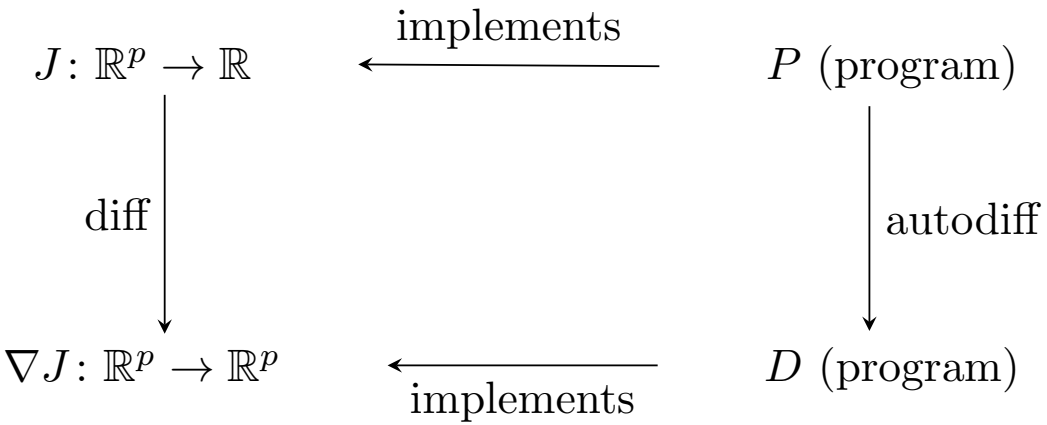
```
from jax import *

def fun(x):
    return x * x

gradFun = jax.grad(fun)
gradFun(3.14)
```

```
DeviceArray(6.28, dtype=float32, weak_type=True)
```

Supervised learning: θ model parameters, (x_i, y_i) input-output training data, $g_1, \ldots, g_L$ elementary computational operations, from a **fixed library**

$$\min_{\theta} \quad \frac{1}{n} \sum_{i=1}^{n} \left\| y_i - g_L \left(g_{L-1} \left(\ldots g_1(x_i, \theta) \ldots, \ldots \right), \theta \right) \right\|^2$$

Algorithmic differentiation: takeways



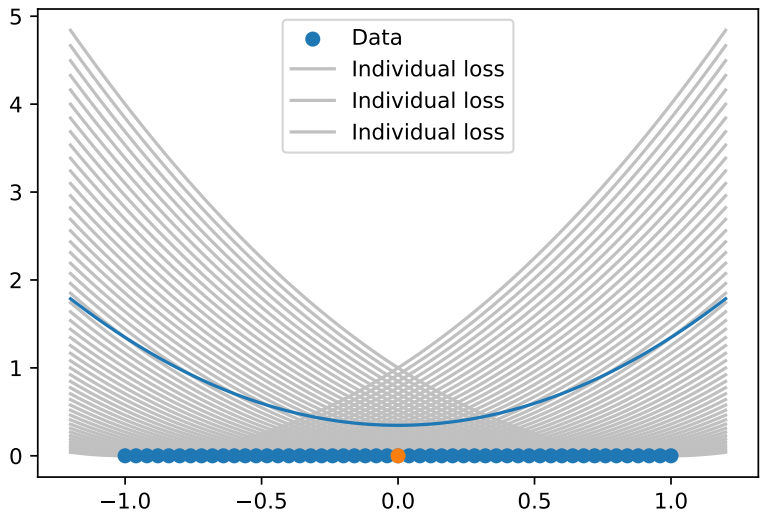
- backprop is backward algorithmic differentiation.
- Automatized implementation of the chain rule of Jacobians.
- What one can program, one can differentiate.
- Complexity of backprop / complexity of program evaluation:
  - Time: bounded
  - Memory: linear in program size
- Forward mode is favorable in some cases.
- Custom elementary g: needs evaluation + VJP / JVP.

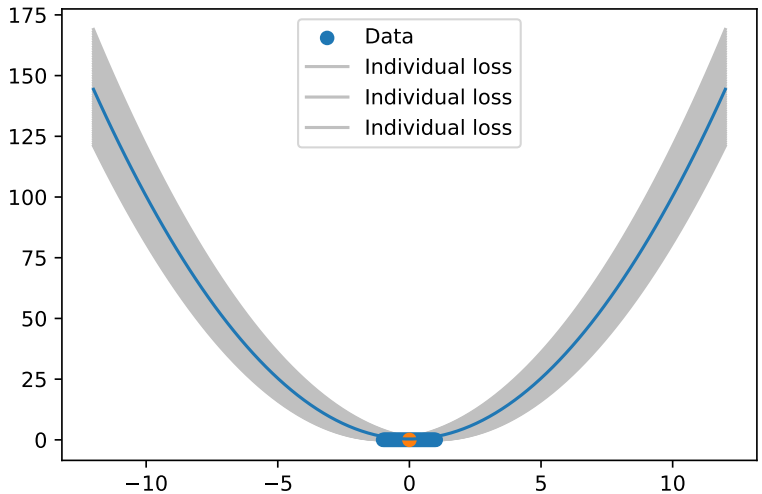
History: Precursors (50s/60s), many independant occurences (70s / 80s), tightly connected to neural networks since the 80s.

Plan

$$\min_{\theta} \quad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Main idea: Present exemples online, in sequence, not all at once.
At the heart of neural network training litterature since its very beginning.





Typical algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k).$

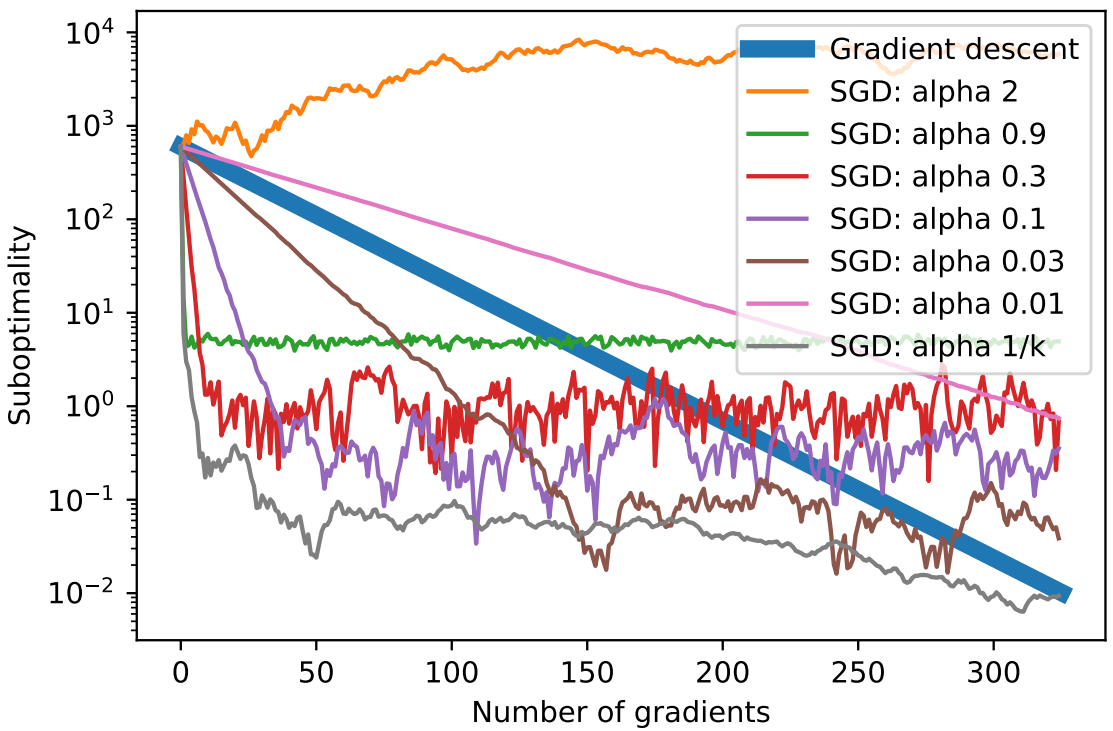
- $\alpha_k > 0.$
- $I_k \in \{1, \ldots, n\}.$

Subsampling gradient method: example

$$\min_{\theta} \quad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Typical algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k).$

- $\alpha_k > 0$.
- $I_k \in \{1, \ldots, n\}$, independent, identically distributed, uniform.



$$\min_{\theta} \qquad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Vocabullary:

- **Epoch:** compute n gradients among $(J_i)_{i=1}^n$, independant of how chosen.
- **Examples:**
  - Gradient algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J(\theta_k)$, one step = one epoch.
  - Stochastic gradient algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k)$, n steps = one epoch.
- **In practice:**
  - $(I_k)_{k\in\mathbb{N}}$ are not i.i.d, sampling without replacement, random permutation.
  - Average gradient over a few terms, minibatch.

Nonconvexity:

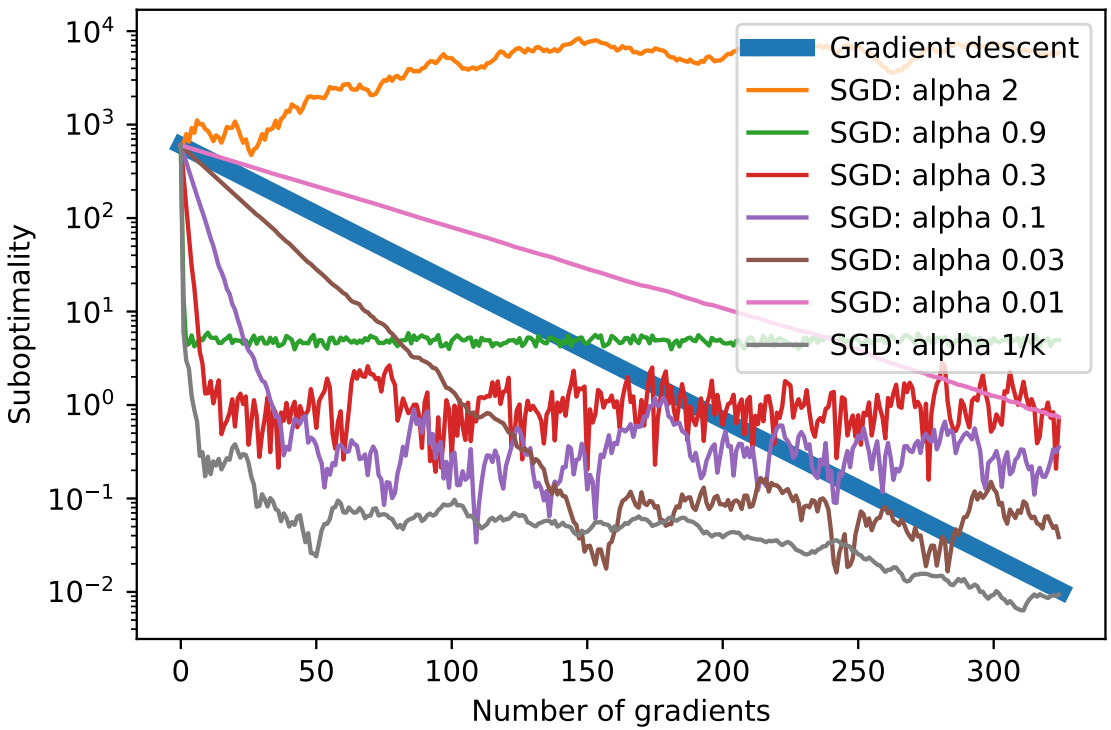
- (Matrix) multiplication is nonconvex
- Composition does not preserve convexity
- J cannot be assumed to be convex.

$$\min_{\theta} \quad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Typical algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k).$

- $\alpha_k > 0$, $I_k \in \{1, \ldots, n\}$, independent, identically distributed, uniform.

Does it optimize?



Lemma: Assume that $f : \mathbb{R}^p \to \mathbb{R}$ has, L Lipschitz gradient, then for any $x, y \in \mathbb{R}^d$,

$$|f(y) - f(x) - \langle \nabla f(x), y - x \rangle| \leq \frac{L}{2} \|y - x\|^2$$

Setting $\gamma(t) = x + t(y - x)$

$$\begin{aligned}
& |f(y) - f(x) - \langle \nabla f(x), y - x \rangle| \\
= & \left| \int_0^1 \frac{d}{dt} f(\gamma(t)) - \langle \nabla f(x), y - x \rangle \, dt \right| \\
= & \left| \int_0^1 \langle \nabla f(\gamma(t)), \dot{\gamma}(t) \rangle - \langle \nabla f(x), y - x \rangle \, dt \right| \\
= & \left| \int_0^1 \langle \nabla f(x + t(y - x)) - \nabla f(x), y - x \rangle \, dt \right| \\
\leq & \int_0^1 \|\nabla f(x + t(y - x)) - \nabla f(x)\| \cdot \|y - x\| dt \\
\leq & \int_0^1 tL \|y - x\|^2 dt = \frac{L}{2} \|y - x\|^2
\end{aligned}$$

$J = \frac{1}{n}\sum_{i=1}^{n} J_i$, each J_i has L Lipschitz gradient, fix an input θ.

For any $i \in \{1, \ldots, n\}$, for I uniform on $\{1, \ldots, n\}$, $\mathbb{E}_I[\nabla J_I(\theta)] = \nabla J(\theta)$,

$$
\begin{aligned}
J(\theta - \alpha \nabla J_i(\theta)) &\leq J(\theta) - \alpha \left\langle \nabla J(\theta), \nabla J_i(\theta) \right\rangle + \frac{L\alpha^2}{2} \|\nabla J_i(\theta)\|^2 \\
\mathbb{E}_I[J(\theta - \alpha \nabla J_I(\theta))] &\leq J(\theta) - \alpha \|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2} \mathbb{E}_I[\|\nabla J_I(\theta)\|^2] \\
&= J(\theta) - \alpha \|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2} \left(\|\nabla J(\theta)\|^2 + \mathbb{E}_I[\|\nabla J_I(\theta) - \nabla J(\theta)\|^2] \right) \\
&= J(\theta) - \alpha \left(1 - \frac{L\alpha}{2}\right) \|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2} \mathbb{E}_I[\|\nabla J_I(\theta) - \nabla J(\theta)\|^2] \\
&\leq J(\theta) - \frac{\alpha}{2} \|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2} \sigma^2(\theta)
\end{aligned}
$$

where $\alpha \leq 1/L$, $\sigma^2(\theta) = \mathbb{E}_I[\|\nabla J_I(\theta) - \nabla J(\theta)\|^2]$

$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$, each J_i has L Lipschitz gradient, $\alpha \leq 1/L$, fix an input θ.

I uniform on $\{1, \ldots, n\}$, $\sigma^2(\theta) = \mathbb{E}_I[\|\nabla J_I(\theta) - \nabla J(\theta)\|^2]$

$$\mathbb{E}_I[J(\theta - \alpha \nabla J_I(\theta))] \leq J(\theta) - \frac{\alpha}{2}\|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2}\sigma^2(\theta)$$

- In expectation. θ is fixed (not random).
- Relative magnitude of $\|\nabla J(\theta)\|^2$ and $\sigma^2(\theta)$.
- α balance two antagonist objectives

$J(\theta) = \frac{1}{n}\sum_{i=1}^n J_i(\theta)$, each J_i has L Lipschitz gradient.

First analysis: Suppose $\sigma(\theta) \leq \sigma$ and $J(\theta) \geq J^*$, for all $\theta \in \mathbb{R}^p$.
$(I_k)_{k\in\mathbb{N}}$ iid unif., $0 < \alpha \leq 1/L$, $\theta_0 \in \mathbb{R}^p$, $\theta_{k+1} = \theta_k - \alpha\nabla J_{I_k}(\theta_k)$

Set for all $k \geq 1$, $\mathbb{E}_k = \mathbb{E}_{I_0,\ldots,I_{k-1}}$,

$$\min_{i=0\ldots k} \mathbb{E}_k\left[\|\nabla J(\theta_i)\|^2\right] \leq \frac{2J(\theta_0) - J^*}{(k+1)\alpha} + L\alpha\sigma^2.$$

If $k+1 \geq \frac{4L(J(\theta_0)-J^*)}{\sigma^2}$ and $\alpha = 2\sqrt{\frac{J(\theta_0)-J^*}{(k+1)L\sigma^2}} \leq \frac{1}{L}$, the l.h.s. is $2\sqrt{\frac{(J(\theta_0)-J^*)L\sigma^2}{k+1}}$.

I_k independent of $I_0, \ldots, I_{k-1}$. θ_k random, independent of $I_k, I_{k+1}, \ldots$.

$$\mathbb{E}_{I_k}[J(\theta_k - \alpha\nabla J_{I_k}(\theta_k))] \leq J(\theta_k) - \frac{\alpha}{2}\|\nabla J(\theta_k)\|^2 + \frac{L\alpha^2}{2}\sigma^2$$

$$\mathbb{E}_{k+1}[J(\theta_{k+1})] \leq \mathbb{E}_k[J(\theta_k)] - \frac{\alpha}{2}\mathbb{E}_k\left[|\nabla J(\theta_k)\|^2\right] + \frac{L\alpha^2}{2}\sigma^2$$

$$\frac{\alpha}{2}\min_{i=0\ldots k} \mathbb{E}_k\left[\|\nabla J(\theta_k)\|^2\right] \leq \frac{\alpha}{2(k+1)}\sum_{i=0}^k \mathbb{E}_i\left[\|\nabla J(\theta_i)\|^2\right]$$

$$\leq \frac{J(\theta_0) - \mathbb{E}_{k+1}[J(\theta_{k+1})]}{k+1} + \frac{L\alpha^2\sigma^2}{2} \leq \frac{J(\theta_0) - J^*}{k+1} + \frac{L\alpha^2\sigma^2}{2}$$

$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$, each J_i has L Lipschitz gradient.

"Convergence in quadratic mean": Suppose $\sigma(\theta) \leq \sigma$ and $J(\theta) \geq J^*$, for all $\theta \in \mathbb{R}^p$.
$(I_k)_{k\in\mathbb{N}}$ iid unif., $\quad 0 < \alpha \leq 1/L, \quad \theta_0 \in \mathbb{R}^p, \quad \theta_{k+1} = \theta_k - \alpha \nabla J_{I_k}(\theta_k)$

For $k \in \mathbb{N}$ large enough, and $\alpha > 0$ small enough, SGD finds approximate critical points, in quadratic mean.

$$\mathbb{E}_{I_k}[J(\theta_k - \alpha \nabla J_{I_k}(\theta_k))] \leq J(\theta_k) - \frac{\alpha}{2}\|\nabla J(\theta_k)\|^2 + \frac{L\alpha^2}{2}\sigma^2$$

Almost sure convergence: Robbins-Monro (1951), Robbins-Sigmund (1971).

$(\alpha_k)_{k\in\mathbb{N}}$ positive:
$\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k)$

Suppose: $\sum_{k\in\mathbb{N}} \alpha_k = \infty$, $\sum_{k\in\mathbb{N}} \alpha_k^2 < +\infty$
$(\theta_k)_{k\in\mathbb{N}}$ is bounded almost surely.

Then almost surely all accumulation points of $(\theta_k)_{k\in\mathbb{N}}$ satisfy $\nabla J(\theta_k) = 0$.

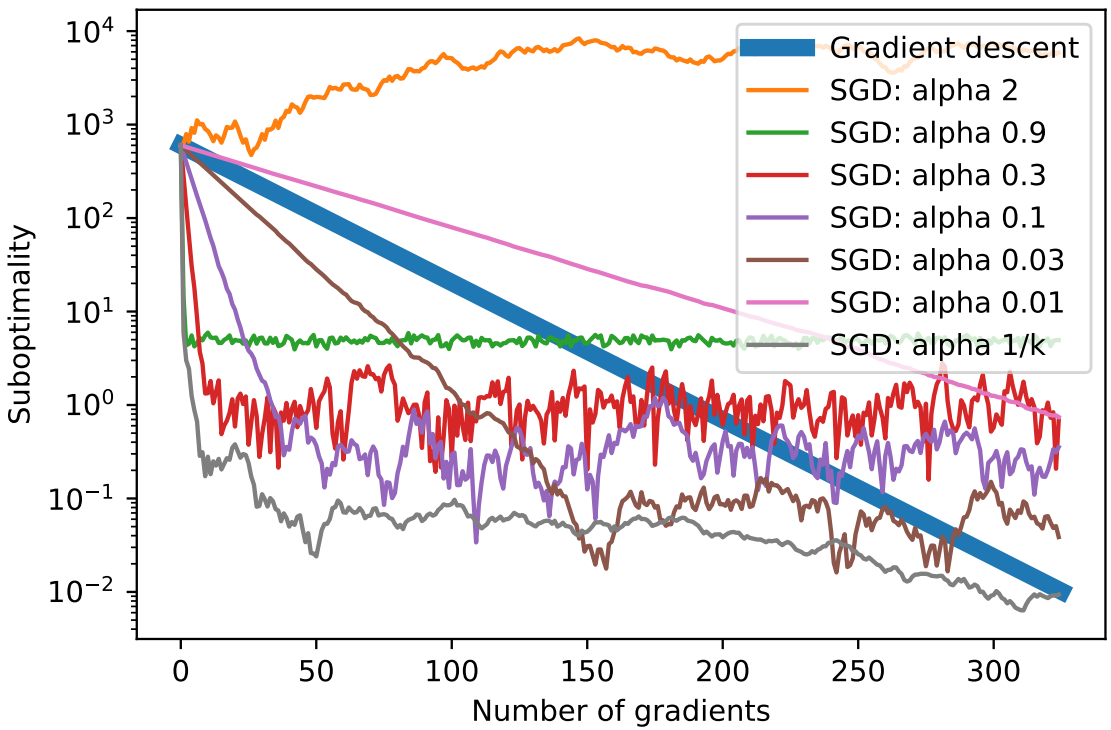
$$\mathrm{dist}(\theta_k, \mathrm{crit} J) \underset{\text{a.s.}}{\to} 0.$$

$$\min_{\theta} \quad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

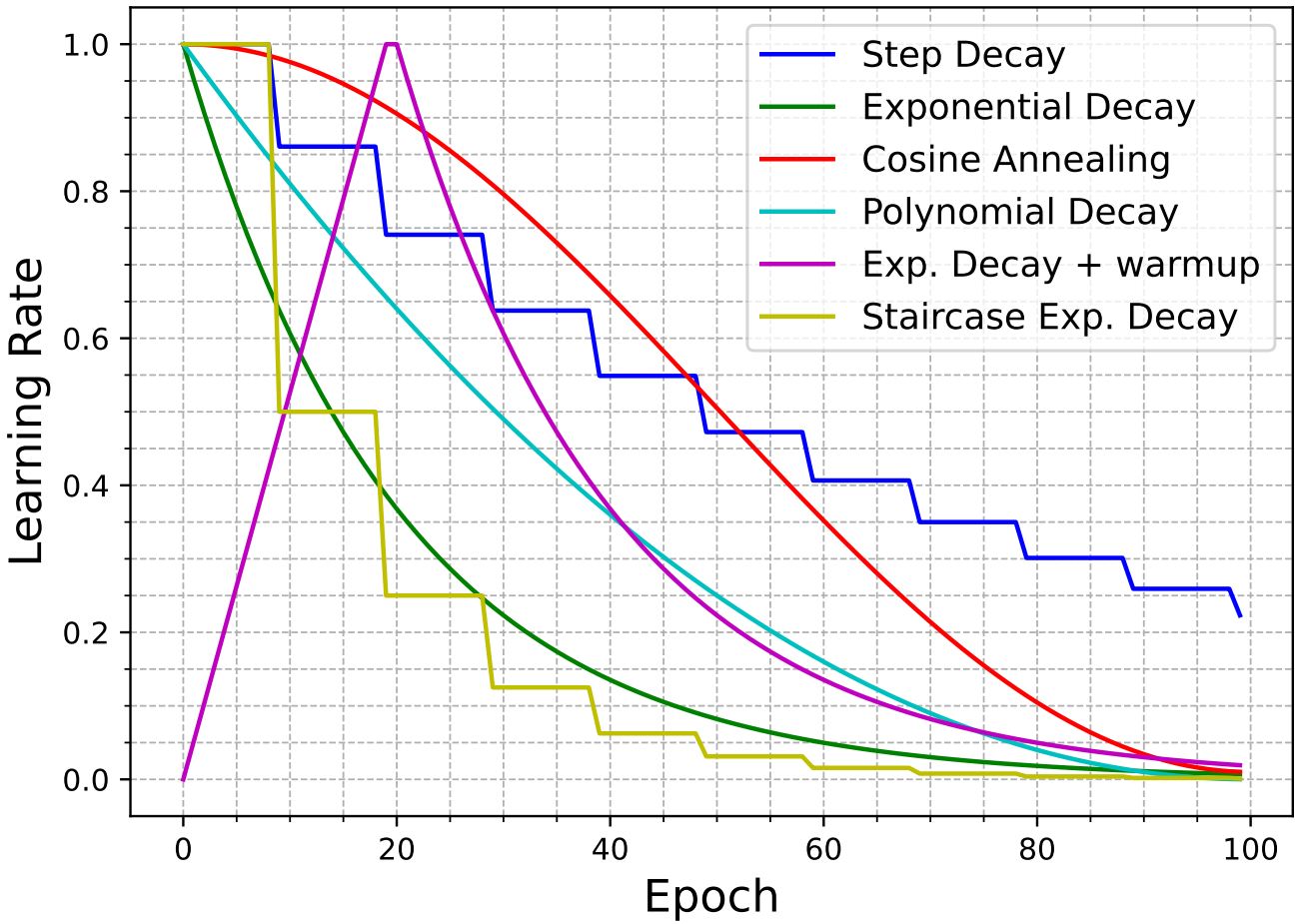
Typical algorithm: $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k).$

- $\alpha_k > 0$, $I_k \in \{1, \ldots, n\}$, independent, identically distributed, uniform.

Does it optimize? Yes, in various senses. Step sizes and variance are crucial.



Vocabulary: learning rate $\sim$ step-size.



$$\min_{\theta} \qquad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Typical algorithm: averaging preserves the mean and reduces the variance

$$\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k)$$

$$\theta_{k+1} = \theta_k - \alpha_k \frac{1}{b}\sum_{j=1}^{b} \nabla J_{I_{k,j}}(\theta_k)$$

Variance reduction: mechanisms to reduce the variance, possibly incuring bias.

Stochastic gradient algorithms, takeway

- Subsampling helps to treat large (infinite) datasets.
- Convergence arguments for small step sizes, find critical points.
- Local minima, saddle point etc ...
- Theory predicts behavior for large k, practical interest for small k.

Plan

- Adadelta (> 2010)
- Adagrad (> 2010)
- Adam (> 2010)
- Adamax (> 2010)
- Ftrl (> 2010)
- Nadam (> 2010)
- RMSprop (> 2010)
- SGD (1951)

- Adadelta (> 2010)
- Adagrad (> 2010)
- Adam (> 2010)
- AdamW (> 2010)
- SparseAdam (> 2010)
- Adamax (> 2010)
- Averaged SGD (90's)
- LBFGS (70's)
- RMSprop (> 2010)
- Rprop, signs (90's)
- SGD (1951)

Plug in subsampled estimates: first description without noise.

"Acceleration": momentum, Polyak's heavy ball, Nesterov's method (70's, 80's)

Gradient update:

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k)$$

$$\frac{\theta_{k+1} - \theta_k}{\alpha} + \nabla J(\theta_k) = 0$$

Gradient with momentum:

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k) + \beta(\theta_k - \theta_{k-1})$$

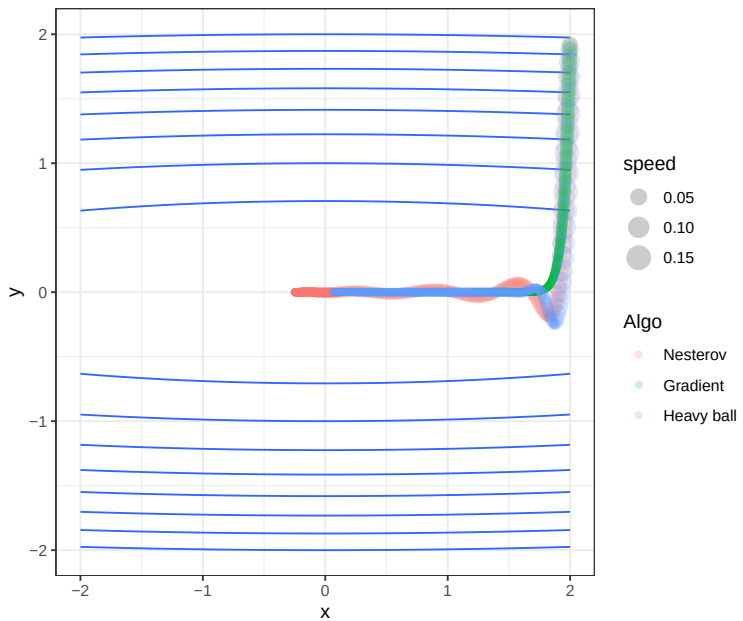
Alternatively:

$$v_{k+1} = \mu v_k + \nu \nabla J(\theta_k)$$

$$\theta_{k+1} = \theta_k - v_{k+1}$$

Continuous time (Antipin, Alvarez, Haraux, Jendoubi, Attouch, Su ...): $(\alpha \to 0)$

$$\dot{x}(t) + \nabla f(x(t)) = 0 \qquad \ddot{x}(t) + \delta(t)\dot{x}(t) + \nabla f(x(t)) = 0$$



Discounted average: $\theta_{k+1} = \theta_k - \mu \sum_{j=0}^{k} \nu^{k-j} \nabla J(\theta_j)$.

Diagonal scaling matrix: $D_k \in \mathbb{R}^{p \times p}$, diagonal with positive entries, (preconditioner)

$$\theta_{k+1} = \theta_k - D_k \nabla J(\theta_k)$$

- D_k estimated online based on gradients (and hyperparameters).
- Old idea in optimization (quasi-newton, preconditioning . . .).
- In machine learning
  - One preconditioner = one ada algorithm
  - Very popular (light hyperparameters tuning).
  - Mostly built by deep learning people for deep learning people.

Diagonal preconditioning: $\theta_{k+1} = \theta_k - D_k \nabla J(\theta_k)$
i-th entry of D_k:

- **Adagrad (Duchi et. al. 2011):** gradient coordinates

$$\frac{\gamma}{\sqrt{\epsilon + \sum_{l=0}^{k} [\nabla J(\theta_l)]_i^2}}$$

- **RMSprop (Hinton et. al. 2012):** discounted average $\beta \in (0, 1)$

$$\frac{\gamma}{\sqrt{\epsilon + (1-\beta) \sum_{l=0}^{k} \beta^{k-l} [\nabla J(\theta_l)]_i^2}}$$

- **Rprop (90's):** coordinatewise gradient sign

$$\frac{\gamma}{|[\nabla J(\theta_k)]_i|}$$

Remarks:

- If $J(\theta) = \sum_{i=1}^{p} w_i \theta_i^2$, then the algorithm depends lightly on $(w_i)_{i=1}^{p}$ (conditioning).
- Adagrad: step size morally scales like $1/\sqrt{k}$.
- Adaptive to different scenarios: smooth, nonsmooth, noise . . . (Levy *et. al.* 2018).
- RMSprop: step size does not vanish.

$$\min_{\theta} \qquad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

Plug in subsampled estimate:
replace $\nabla J(\theta)$ *mutatis mutandis* , by the stochastic estimate ($I_1, \ldots, I_b$ iid)

$$\hat{\nabla} J(\theta) = \frac{1}{b}\sum_{j=1}^{b} \nabla J_{I_j}(\theta) \qquad\qquad b < n$$

- Adaptive methods: stochastic denominator.
- Momentum: variance reduction effect.

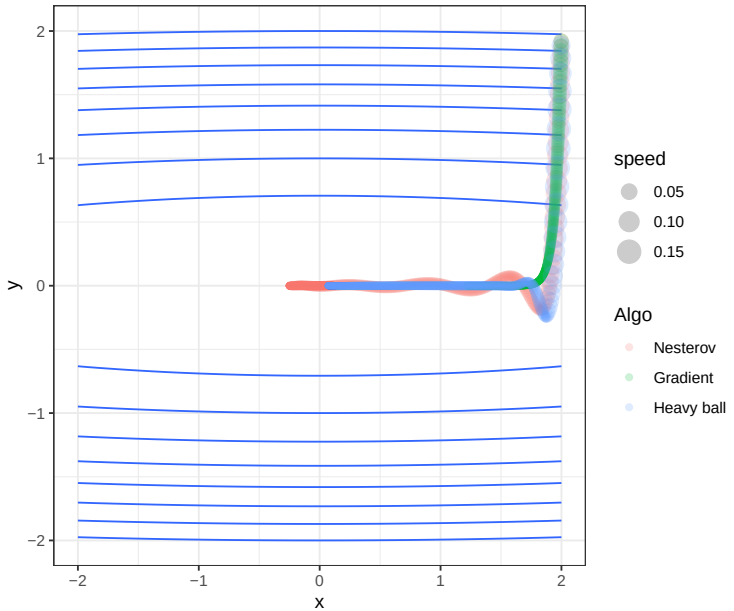
Adam (Kingma et. al. 2014) soon 2×10^5 citations $\sim$ most cited paper of all times

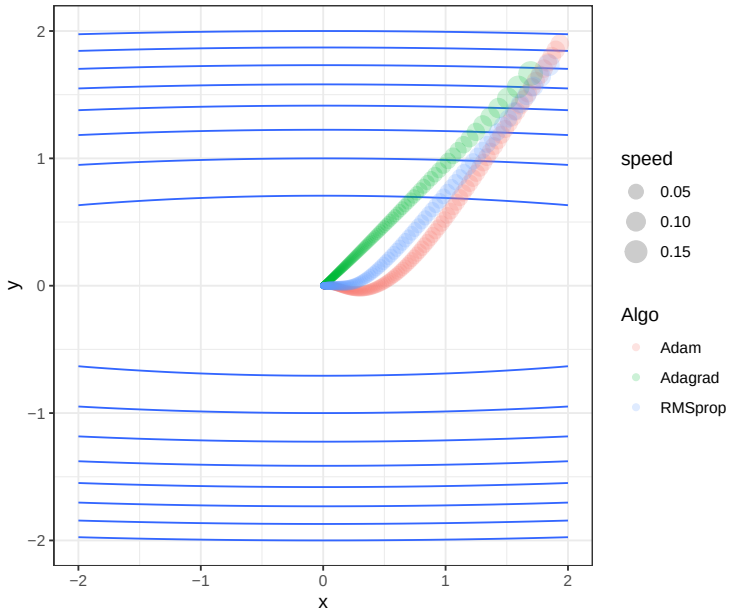
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

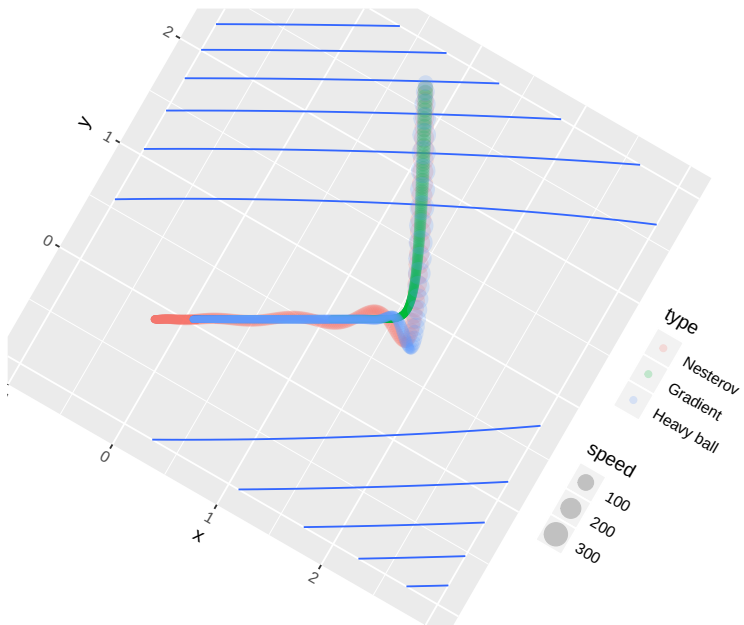
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

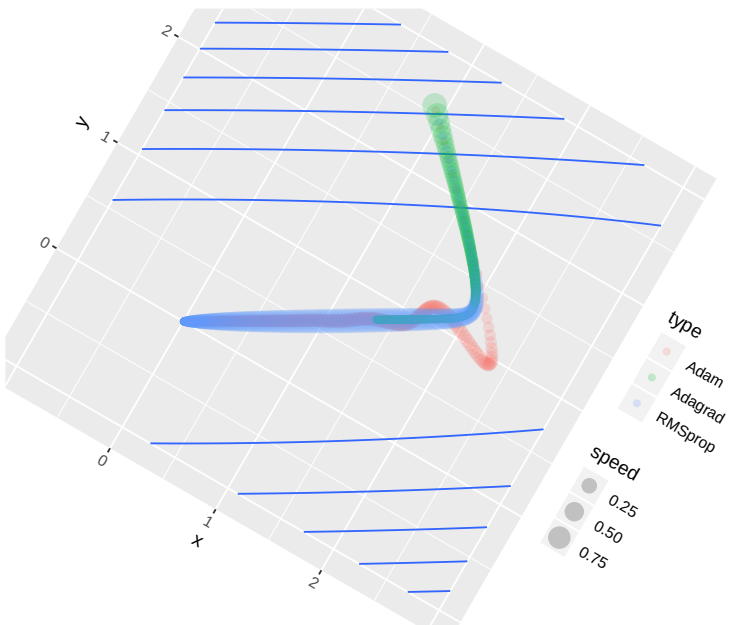
- Adaptive steps
- Momentum
- Discounted averages
- Large steps.

“Adaptive” step size illustration , rotation equivariance









Deep learning optimizers takeways

Combines stochastic gradient estimates with

- Momentum
- Preconditioning
- More . . .

Plan

Additional variations on training

Mixed with the network structure, affect the optimization process.

- Dropout (set randomly weights to 0 during backpropagation).
- Batch-normalization (center and scale intermediate layers (on each minibatch))
- Weight decay.
- Data augmentation: generate synthetic data (rotation, scaling . . .)

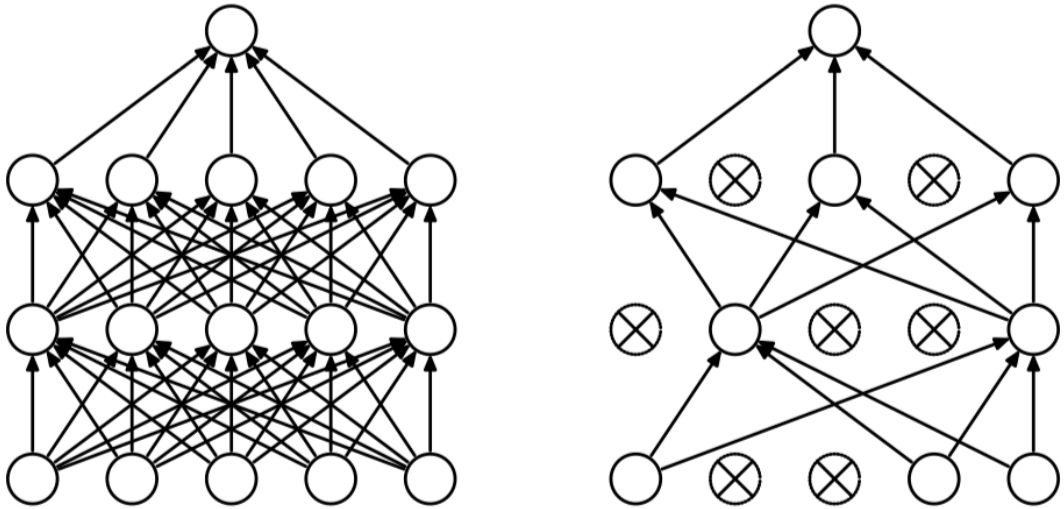
Network structure: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.
$F \colon \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$. $F(\theta, x) \sim y$.

$$\min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad := \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2$$

$$F(\theta, x) = \phi_L \left(W_L \quad \phi_{L-1} \left(W_{L-1} \left(\ldots \quad \phi_1 \left(W_1 x + b_1 \right) \quad \right) \quad + b_{L-1} \right) + b_L \right)$$

$\phi_i \colon \mathbb{R}^{p_i} \mapsto \mathbb{R}^{p_i}$ "activation functions", nonlinear.
$W_i \in \mathbb{R}^{p_i \times p_{i-1}}$, $b_i \in \mathbb{R}^{p_i}$, $\theta = (W_1, b_1, \ldots, W_L, b_L)$, model parameters.

Dropout



(a) Standard Neural Net (b) After applying dropout.

Srivastava *et.al.* (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR.

$$F(\theta, x) = \phi_L\left(W_L \quad \phi_{L-1}\left(W_{L-1}\left(\ldots \quad \phi_1\left(W_1 x + b_1\right) \quad\right) \quad + b_{L-1}\right) + b_L\right)$$

Implementation: $W_i \in \mathbb{R}^{p_i \times p_{i-1}}$, $D_i \in \mathbb{R}^{p_{i-1} \times p_{i-1}}$ diagonal with iid binary entries.

Run each stochastic gradient step with $W_i D_i$ instead of W_i.

Batch normalization (Ioffe, Szegedy, 2015)

$$F(\theta, x) = \phi_L\left(W_L \quad \phi_{L-1}\left(W_{L-1}\left(\ldots \quad \phi_1\left(W_1 x + b_1\right) \quad\right) \quad + b_{L-1}\right) + b_L\right)$$

Input: $z_0(x) = x \in \mathbb{R}^p$.

1: **for** $k = 1, 2, \ldots, L$ **do**

2: $\quad z_k(x) = \phi_i(W_i z_{i-1}(x) + b_i)$

3: **end for**

Return: $z_L(x) \in \mathbb{R}^m$.

Mini-batch: subsampled estimate $x_1, \ldots, x_b$. Center and scale each layer

$$z_i(x_1), \ldots, z_i(x_b) \qquad \rightarrow \qquad \tilde{z}_i(x_1), \ldots, \tilde{z}_i(x_b)$$

- Differentiable operation.
- Stabilizes training, crucial for some architectures (*e.g.* ResNet).
- After training, variance and mean are estimated on the whole training set.
- Mechanism not well understood.
- Breaks iid hypotheses.
- Breaks the empirical risk minimization interpretation.

Weight decay and L_2 regularization

$$\min_{\theta} \quad J(\theta) \quad := \quad \frac{1}{n}\sum_{i=1}^{n} J_i(\theta) + \frac{\lambda}{2}\|\theta\|^2$$

Typical algorithm: $\alpha_k > 0$, $I_k \in \{1, \ldots, n\}$, independent, identically distributed, uniform.

$$g_k = \nabla J_{I_k}(\theta_k)$$

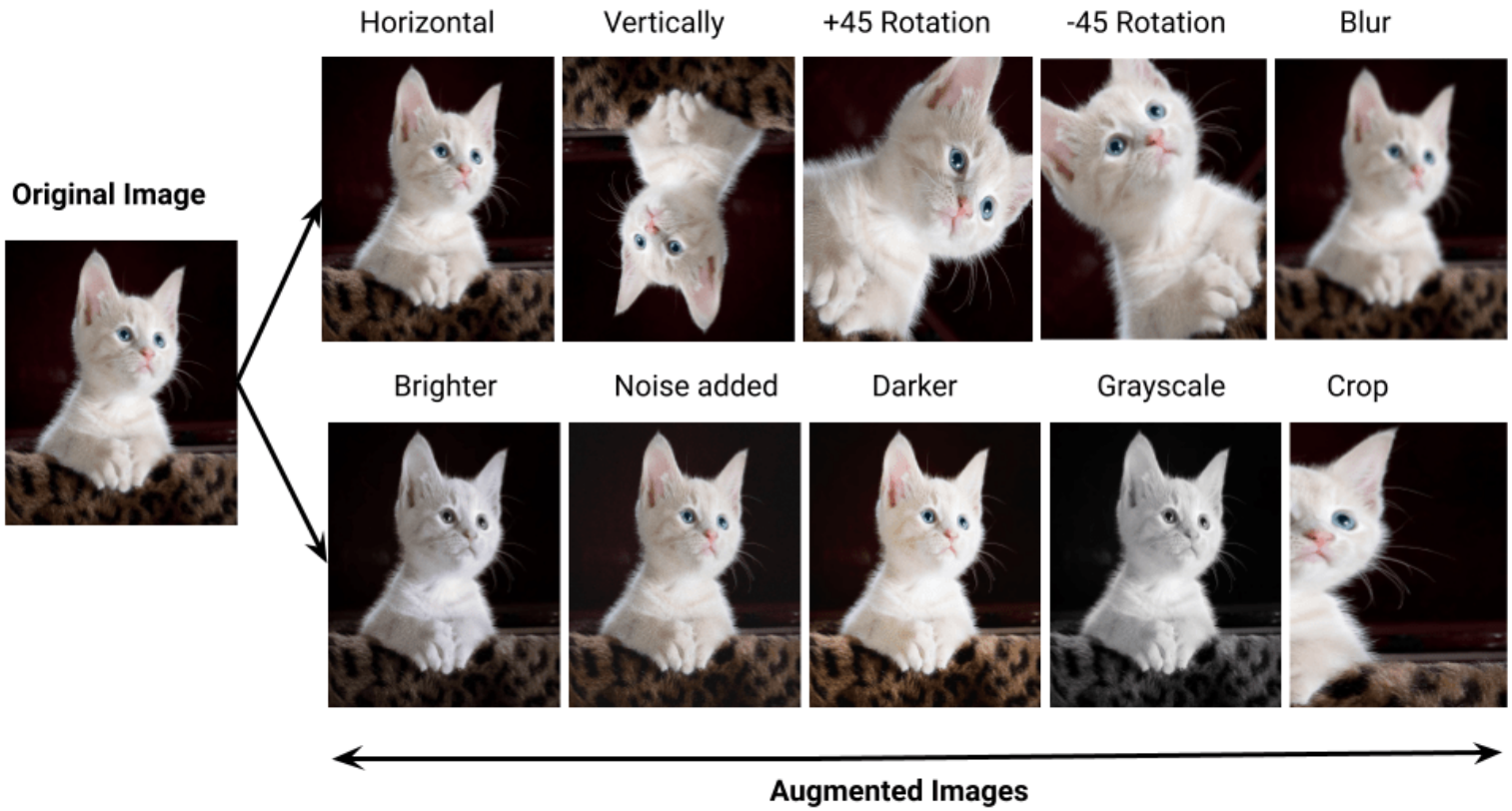
$$\theta_{k+1} = \theta_k - \alpha_k g_k$$

$$\begin{cases} \text{either} & g_k = g_k + \lambda\theta_k \\ \text{or} & \theta_k = (1 - \lambda\alpha)\theta_k \end{cases}$$

- Equivalent for SGD.
- Not equivalent for momentum or adaptive scaling.
- AdamW, baseline for some problems such as transformers language models.

Data augmentation

Training set: $(x_i, y_i)_{i=1}^n$, $\quad x_i \to \tilde{x}_{i1}, \ldots, \tilde{x}_{iN}$

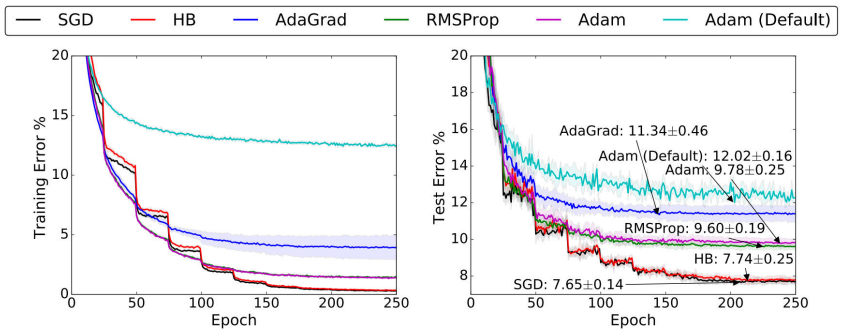


Credit UBIAI.

- Crucial for some vision models.
- Fits the iid interpretation in SGD.

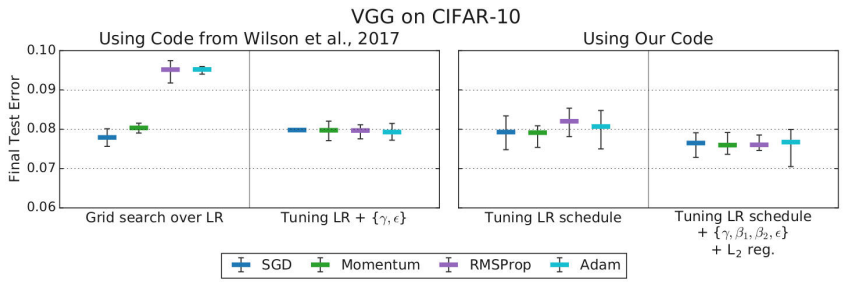
What is the best algorithm?

Wilson et. al. 2017: The Marginal Value of Adaptive Gradient Methods

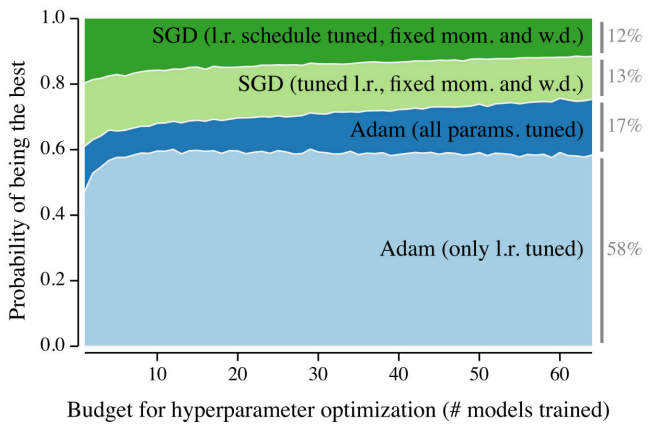


(a) CIFAR-10 (Train) **(b)** CIFAR-10 (Test)

Choi et. al. 2019: On Empirical Comparisons of Optimizers



Teja et. al. 2020: Optimizer Benchmarking Needs to Account for Hyperparameter Tuning



Difficult empirical comparison

- Parameter tuning.
- Huge variability: tasks (datasets), architectures.
- Many additional factors mixed between optimization and the network architecture.
- Computation cost: 1k euro for a single network training.
- No uniformly better algorithm.

Plan

Network structure: $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.
$F : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$. $F(\theta, x) \sim y$.

$$\begin{aligned} \min_{\theta \in \mathbb{R}^p} \quad J(\theta) \quad &:= \quad \frac{1}{n} \sum_{i=1}^{n} (F(\theta, x_i) - y_i)^2 \\ &= \quad \min_{\theta} \|F(\theta) - y\|^2 \end{aligned}$$

$F : \mathbb{R}^p \to \mathbb{R}^n$, $y \in \mathbb{R}^n$.

Gradient domination: quadratic growth property, there is $c > 0$.

$$\|\nabla J(\theta)\|^2 \geq c J(\theta)$$

- A long history in the study of gradient systems.
- One of the keys behind recent theoretical developments for neural network training.
- World leading expert in Toulouse.

In a deep network context: critical points are global minima interpolating the data.

A result from Lojasiewicz 1963 on gradient flow

One can relatively easily notice that the second theorem for analytic functions is equivalent to the following

INEQUALITY.

$$|\operatorname{grad} g(x)| \geq |g(x)|^{\Theta} \qquad \textit{in the neighbourhood of zero},$$

for any analytic function $g \not\equiv 0$ *in the neighbourhood of* 0 *in* $\mathbb{R}^n$, *with some exponent* Θ *such that*

$$0 < \Theta < 1.$$

This inequality is used in the proof of the

THEOREM ON LIMITS OF TRAJECTORIES of a dynamical system

$$\dot{x} = -\operatorname{grad} f(x),$$

where $f \geq 0$ is analytic in a neighbourhood of zero of the $\mathbb{R}^n$ space. Namely, *each of the trajectories* $x(t)$, *starting from the points sufficiently near to zero, has a limit when* $t \to \infty$.

A result from Polyak 1963 on the gradient algorithm

Theorem 4.2

In addition to the conditions of Theorem 4.1, let

$$\|f'(x)\|^2 \geqslant \lambda[f(x) - f^*], \quad \lambda > 0. \tag{4.6}$$

Then the sequence (4.1) is convergent to the minimum at the rate of a geometric progression.

The conditions of the theorem only need to be satisfied in a neighbourhood of x^*.

Lojasiewicz's trap mechanism in continuous time (same in discret time)

Let $f\colon \mathbb{R}^p \to \mathbb{R}$ be analytic, $f(0) = 0$, $f \geq 0$.

$\exists q \in (0,1)$, $R > 0$, $\|\nabla f(\theta)\| \geq f(\theta)^q$ and x, $\|\theta\| \leq R$.

Assume that $0 < \frac{f(\theta_0)^{1-q}}{1-q} + \|\theta_0\| < R$ and

$$\dot{\theta}(t) = -\nabla f(\theta(t)), \qquad \theta(0) = \theta_0$$

Lyapunov analysis: if $\|\theta(t)\| < R$,

$$\begin{aligned}\frac{d}{dt}\frac{f(\theta(t))^{1-q}}{1-q} &= f(\theta(t))^{-q}\left\langle \nabla f(\theta(t)), \dot{\theta}(t)\right\rangle \\ &= -f(\theta(t))^{-q}\|\nabla f(\theta(t))\|\,\|\dot{\theta}(t)\| \qquad \leq -\|\dot{\theta}(t)\|.\end{aligned}$$

Finite length and convergence, for any $T > 0$

$$\begin{aligned}\|\theta(T)\| &\leq \|\theta_0\| + \int_0^T \|\dot{\theta}(t)\|dt \leq \|\theta_0\| - \int_0^T \frac{d}{dt}\frac{f(\theta(t))^{1-q}}{1-q}dt \\ &= \|\theta_0\| + \frac{f(\theta_0)^{1-q}}{1-q} - \frac{f(\theta(T))^{1-q}}{1-q} < R.\end{aligned}$$

$f(\theta(t))$ converges: to 0 because $\liminf_{t\to\infty} \|\nabla f(\theta(t))\| = 0$.

Convergence rate in continuous time (same in discret time)

Let $f : \mathbb{R}^p \to \mathbb{R}$ be C^2 ($q = \frac{1}{2}$).

Assume $\|\nabla f(\theta)\|^2 \geq f(\theta) \geq 0$ for all θ.

Assume that

$$\dot{\theta}(t) = -\nabla f(\theta(t)), \qquad \theta(0) = 0$$

Lyapunov analysis:

$$\begin{aligned} \frac{d}{dt} f(\theta(t)) &= \left\langle \nabla f(\theta(t)), \dot{\theta}(t) \right\rangle \\ &= -\|\nabla f(\theta(t))\|^2 \qquad \leq -f(\theta(t)). \end{aligned}$$

Exponential convergence: $f(\theta(t)) \leq f(0) \exp(-t)$ (Gronwall).

Loss structure: $F\colon \mathbb{R}^p \to \mathbb{R}^n$ C^2, $y \in \mathbb{R}^n$.

$$\min_{\theta \in \mathbb{R}^p} J(\theta) \quad := \quad \|F(\theta) - y\|^2$$

$$\frac{\|\nabla J(\theta)\|^2}{J(\theta)} = 2\frac{\|J_F(\theta)^T(F(\theta)-y)\|^2}{\|F(\theta)-y\|^2} \geq 2\lambda_{\min}(J_F(\theta)J_F(\theta)^T)$$

$$J_F J_F^T = \sum_{i=1}^{p} \left(\frac{\partial F}{\partial \theta_i}\right)\left(\frac{\partial F}{\partial \theta_i}\right)^T \in \mathbb{R}^{n\times n}$$

Overparameterization intuition: For $p \gg n$, J_F is surjective, $J_F J_F^T$ is invertible.
Set $y = F(\theta_0)$, $\lambda > 0$

$$\exists R > 0,\ \|\nabla J(\theta)\|^2 \geq \lambda J(\theta),\ \forall \theta,\ \|\theta - \theta_0\| \leq R.$$

Main challenge: Estimate R and λ.

Historical remark: a classical argument, revisited in a deep learning context

$$\exists R > 0,\ \|\nabla J(\theta)\|^2 \geq \lambda J(\theta),\ \forall \theta,\ \|\theta - \theta_0\| \leq R.$$

GRADIENT DESCENT PROVABLY OPTIMIZES OVER-PARAMETERIZED NEURAL NETWORKS

Simon S. Du* Xiyu Zhai* Barnabás Poczós Aarti Singh

One hidden layer: Random initialization θ.
For large p, quantitative gradient domination, controlled w.h.p. λ, R as $p \to \infty$.
Trap mechanism, linear convergence. Many extensions.

Neural Tangent Kernel: Convergence and Generalization in Neural Networks

Arthur Jacot Franck Gabriel Clément Hongler

Infinite width limit: Gradient domination in function space. NTK: $J_F J_F^T$, stabilizes and remains constant during training.

Lazy training: both theories suggest that the length of the trajectory is very small.

$J = \frac{1}{n}\sum_{i=1}^{n} J_i$, each $J_i \geq 0$ has L Lipschitz gradient, fix an input θ, and $\|\nabla J\|^2 \geq \lambda J$, $\lambda > 0$.

Interpolation: If $\nabla J(\theta) = 0$, then $J_i(\theta) = 0$ and $\nabla J_i(\theta) = 0$, $i = 1, \ldots, n$, and $\sigma^2(\theta) = 0$.

For I uniform on $\{1, \ldots, n\}$, $\mathbb{E}_I[\nabla J_I(\theta)] = \nabla J(\theta)$,
Descent lemma: $0 \leq J_I(\theta - \alpha \nabla J_I(\theta)) \leq J_I(\theta) - \frac{1}{2L}\|\nabla J_I(\theta)\|^2$.

$$\begin{aligned}
\mathbb{E}_I[J(\theta - \alpha \nabla J_I(\theta))] &\leq J(\theta) - \alpha\|\nabla J(\theta)\|^2 + \frac{L\alpha^2}{2}\mathbb{E}_I[\|\nabla J_I(\theta)\|^2] \\
&\leq J(\theta)(1 - \lambda\alpha + L^2\alpha^2) \\
&= J(\theta)\left(1 - \frac{\lambda^2}{4L^2}\right)
\end{aligned}$$

where $\alpha \leq \lambda/(2L^2)$. Linear convergence to global min.

Non exhaustive historical landmarks

1963. Lojasiewicz: analyticity $\Rightarrow$ Lojasiewicz gradient inequality. Trap mechanism.

1963. Polyak: gradient domination (exponent $1/2$) implies linear convergence of GD.

Geometers mention Lojasiewicz's inequality.
Russian optimizers mention gradient dominated function.

1998. Kurdyka, generalizes Lojasiewicz arguments to o-minimal structures.

2005. Absil, Mahony, Andrews, Lojasiewicz's trap mechanism for GD on analytic losses.

2005's. Bolte, Daniilidis, Lewis, Shiota, extend Kurdyka's argument to nonsmooth losses. Introduce the name Kurdyka-Lojasiewicz inequality.

2010's. Bolte *et. al.* Extend the trap argument to many algorithm. Connection with error bounds. Convergence rates. Complexity for convex optimization ...

2015. Karimi, Nutini, Schmidt. Revisit Polyak's arguments in a machine learning context. Introduce the name Polyak-Lojasiewicz inequality for global gradient domination with power $1/2$.

Since then: trap argument ubiquitous in analysing training of overparameterized networks. under the name "PL" inequality.

Gradient domination and favorable landscape summary

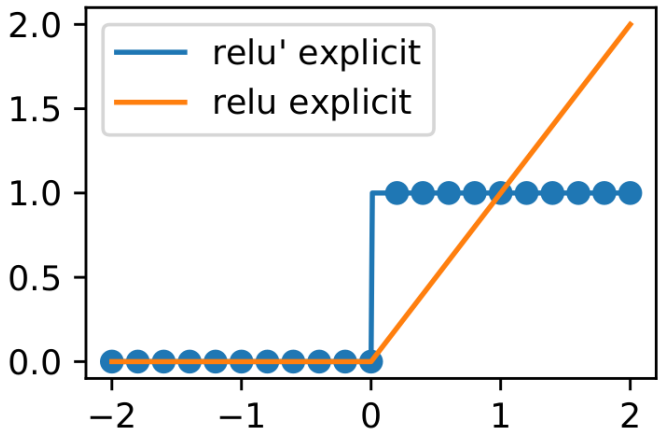
- Revisit classical arguments for convergence of gradient flows.
- High probability quantitative estimates for wide deep networks.
- Lazy training: small length trajectory, idealized explaination.

Plan

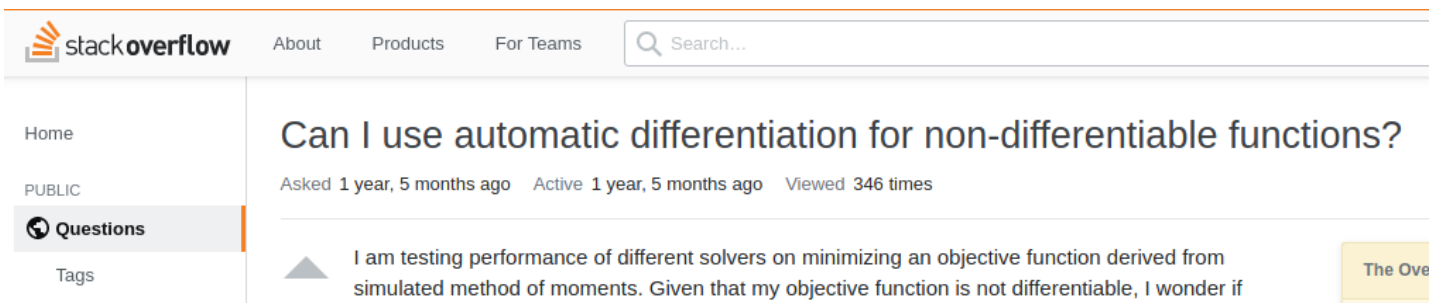
Nonsmoothness is needed

Differentiate programs: `if ... then ...` or `while`

```
def myRelu(x):
    if x<=0:
        return 0
    else:
        return x
```



Massive practice: elementary functions: relu, maxpool, sort, implicit layers
Ex: 75% of `torchvision` models.



Nonsmooth optimization (analysis):

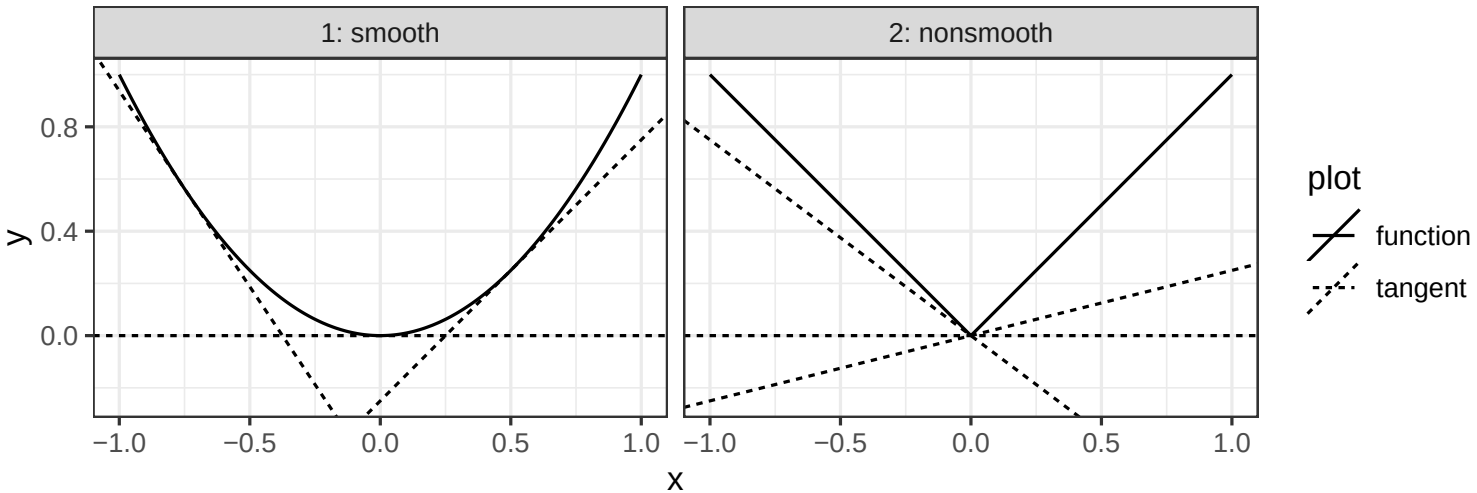
- Google book: > 150 results with "nonsmooth optimization" in the title.
- Important bibliography (starting $\sim$ 70's), well established concepts.
- Signal processing (inverse problems), machine learning (lasso, SVM ...).
- Stochastic approximation (subsampling / minibatching).

Nonsmooth algorithmic differentiation: requires special care.

F **convex (Moreau-Rockafellar):** global lower affine tangent

$$F(y) \geq F(x) + \nabla F(x)^T (y - x),\ \forall y \in \mathbb{R}^p \qquad \text{if } F \text{ is differentiable at } x$$

$$\partial_{\text{conv}} F(x) = \left\{ v \in \mathbb{R}^p,\ F(y) \geq F(x) + v^T (y - x),\ \forall y \in \mathbb{R}^p \right\}.$$



Example: $F : x \mapsto |x|$.

$$\partial_{\text{conv}} F(x) = \begin{cases} \{-1\} & \text{if } x < 0 \\ \{1\} & \text{if } x > 0 \\ [-1, 1] & \text{if } x = 0 \end{cases}.$$

F **general (Clarke):** Rademacher, the set $R \subset \mathbb{R}^p$ where F is differentiable has full measure.

Sequential closure: limits of neighboring gadients.

$$\partial_{\mathrm{cl}} F(x) = \left\{ v \in \mathbb{R}^p,\, \exists \left(y_k, v_k\right)_{k\in\mathbb{N}},\, y_k \underset{k\to\infty}{\to} x,\, v_k \underset{k\to\infty}{\to} v,\, y_k \in R,\, v_k = \nabla F(y_k), k \in \mathbb{N} \right\}.$$

Clarke subgradient: convex closure.

$$\partial_{\mathrm{Clarke}} F(x) = \mathrm{conv}(\partial_{\mathrm{cl}} F(x)).$$

Example: $F : x \mapsto |x|$.

$$\partial_{\mathrm{Clarke}} F(x) = \partial_{\mathrm{conv}} F(x) = \begin{cases} \{-1\} & \text{if } x < 0 \\ \{1\} & \text{if } x > 0 \\ [-1, 1] & \text{if } x = 0 \end{cases}.$$

Fermat rule: If x is a local minimum of F, then $0 \in \partial_{\mathrm{Clarke}} F(x)$.

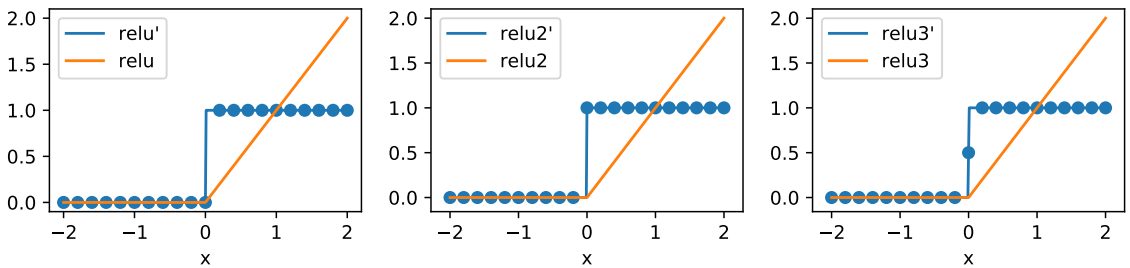
Calculus is partly unpredictible

$$\mathrm{relu}(t) = \max\{0, t\} \qquad \mathrm{relu}_2(t) = \mathrm{relu}(-t) + t \qquad \mathrm{relu}_3(t) = 1/2(\mathrm{relu}(t) + \mathrm{relu}_2(t))$$

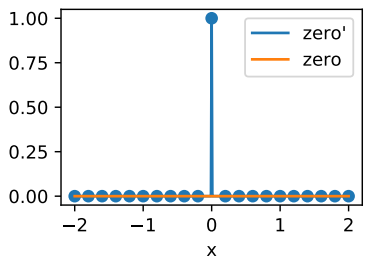
Then $\mathrm{relu} = \mathrm{relu}_2 = \mathrm{relu}_3$.

- TensorFlow (TF) set $\mathrm{backprop}\,\mathrm{relu}(0) = 0$. TF's gives

$$\mathrm{backprop}\,\mathrm{relu}_2(0) = 1 \text{ and } \mathrm{backprop}\,\mathrm{relu}_3(0) = 1/2.$$



- Artifacts: $\mathrm{zero}(x) = \mathrm{relu}_2(x) - \mathrm{relu}(x) = 0$.



- Actually $s \times \mathrm{zero} = 0$ and $\mathrm{backprop}\,[s \times \mathrm{zero}](0) = s \in \mathbb{R}$ arbitrary
- Spurious critical point: $\mathrm{identity}(x) := x - \mathrm{zero}(x) = x$ but $\mathrm{backprop}\,\mathrm{identity}(0) = 0$

No convexity, no calculus: $g_1 : \mathbb{R}^p \to \mathbb{R}$, $g_2 : \mathbb{R}^p \to \mathbb{R}$ locally Lipschitz.

$$\partial^c(g_1 + g_2) \subset \partial^c g_1 + \partial^c g_2.$$

- holds with equality if g_1 and g_2 are continuously differentiable.
- holds with equality if g_1 and g_2 are convex.
- no equality in general: $g : x \mapsto |x|$

$$\partial^c(g - g) = \partial^c(x \mapsto 0) = \{0\} \subset \quad \partial^c(g) + \partial^c(-g) = \quad \begin{cases} 0 & \text{if } x \neq 0 \\ [-2, 2] & \text{if } x = 0 \end{cases}.$$

Deep learning: no convexity, no smoothness. Calculus rules?

- Backpropagating subgradients does not produce subgradients.
- Sampling subgradients does not produce subgradients in expectation.

Important remark: it works extremely well in practice, despite artifacts.

J Lipschitz (locally), $\quad J(\theta_{k+1}) \leq J(\theta_k)$?

$$\theta_{k+1} = \theta_k - \alpha_k v_k \qquad \Leftrightarrow \qquad \frac{\theta_{k+1} - \theta_k}{\alpha_k} \in -\partial J(\theta_k)$$

$$v_k \in \partial J(\theta_k).$$

Chain rule along Lipschiz curves (Brézis 1973, Valadier 1989).
Hypothesis: For any Lipschitz $\gamma\colon [0,1] \mapsto \mathbb{R}^p$

$$\begin{aligned} \frac{d}{dt} J(\gamma(t)) &= \langle v, \dot\gamma(t) \rangle \qquad \forall v \in \partial J(\gamma(t)), \qquad \text{a.e.} \quad t \in [0,1] \\ &= -\|\dot\gamma(t)\|^2, \qquad \text{a.e.} \quad t \in [0,1] \end{aligned}$$

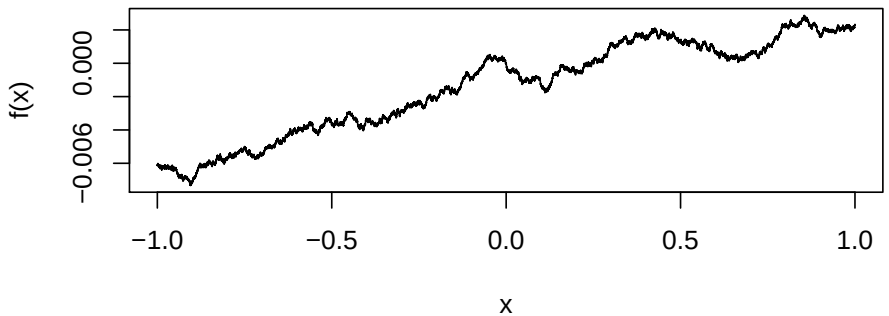
Suppose: $\dot\gamma(t) \in -\partial J(\gamma(t))$ for almost all $t \in [0,1]$,
then $t \mapsto J(\gamma(t))$ decreases, strictly if $0 \notin \partial J(\gamma(t))$.

Stochastic approximation (Benaim-Haufbauer-Sorin (2005), Faure-Roth (2013)),
subgradient plus zero mean noise, under proper assumptions:

Vanishing step sizes, almost surely all accumulation points are critical points.

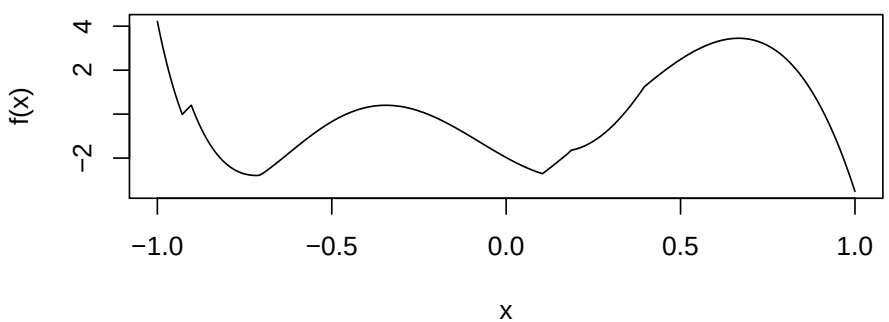
Borwein-Moors (2000),Loewen-Wang (2000): Let f be a typical 1-Lipschitz function (in sup norm), then

- ∂f is the unit ball everywhere (no chain rule, no subgradient algorithm).
- local minimizers are dense: there is a local minimizer arbitrarily close to any argument.



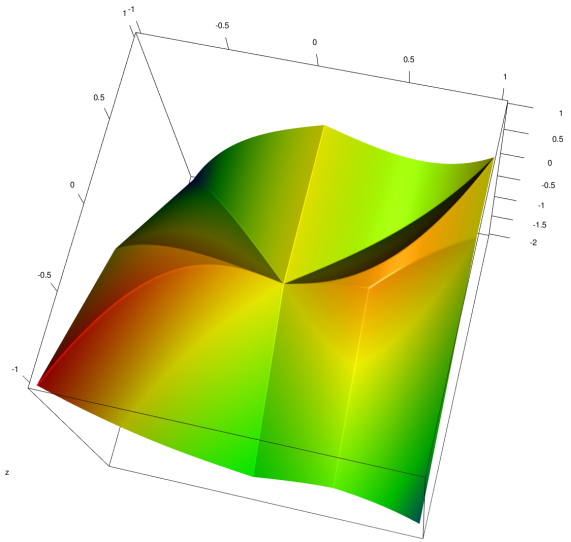
DL losses are tame: Let J be a typical locally Lipschitz deep learning loss, then

- Relu network + square loss: J piecewise polynomial.
- More generally: J semi-algebraic, definable.
- **Bolte-Daniilidis-Lewis 2007, Davis et.al. 2019:** Chain rule along Lipschitz curves.



Semi-algebraic?

Basic set: Solution set of finitely many polynomial inequalities.
Set: Finite union of Basic semi-algebraic sets.
Function, set valued map: Semi-algebraic graph.
Examples: polynomials, square root, quotients, norm, relu, rank ...



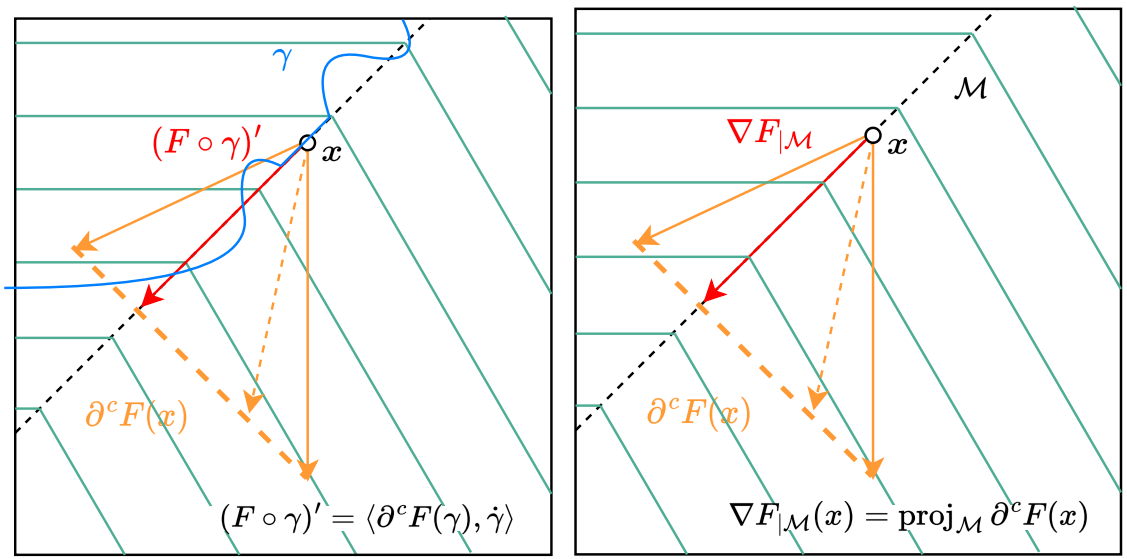
Tarski Seidenberg: first order formula involving semi-algebraic sets $\rightarrow$ semi-algebraic.

- gradient / subgradient of semi-algebraic function, partial minima, composition ....

Bolte-Daniilidis-Lewis 2007, Davis et.al. 2019: Chain rule along Lipschitz curves.

Chain rule (Davis *et.al.*): for any $\gamma\colon \mathbb{R} \to \mathbb{R}^p$ Lipschitz, for almost all $t \in \mathbb{R}$,

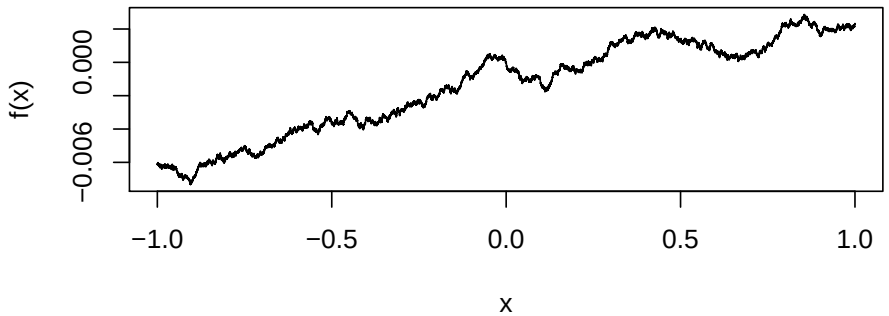
$$\frac{d}{dt}(J \circ \gamma)(t) = \langle v, \dot{\gamma}(t) \rangle\,, \qquad \forall v \in \partial^c J(\gamma(t)).$$



Chain rule from the projection formula of Bolte, Daniilidis, Lewis, Shiota.

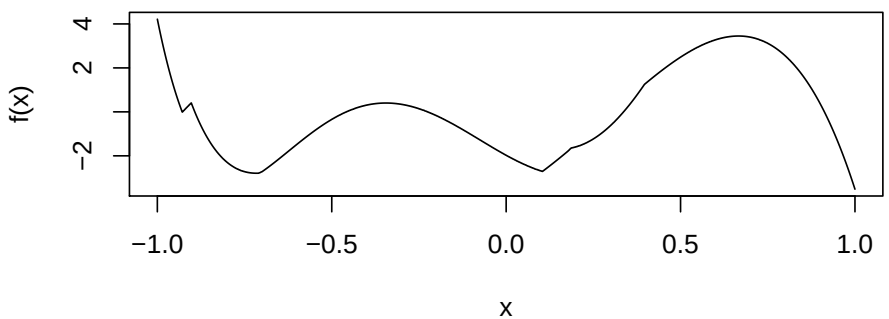
Typical Lipschitz function: pathological

- No chain rule (no convergence).



Typical DL loss: rigid (definable / semi-algebraic / piecewise polynomial)

- Chain rule (rich convergence theory).



$$\theta_{k+1} = \theta_k - \alpha_k \text{backprop } J_{I_k}(\theta_k).$$

Let D be the (set-valued) output of backpropagation applied to the DL loss J (with Clarke subgradient in place of gradients).

Conservativity:
For any Lipschitz $\gamma\colon [0,1] \mapsto \mathbb{R}^p$

$$\frac{d}{dt} J(\gamma(t)) = \langle v, \dot{\gamma}(t) \rangle \qquad \forall v \in D(\gamma(t)), \qquad \text{a.e.} \quad t \in [0,1].$$

Convergence of SGD for DL: under proper assumptions,

- almost surely accumulation points of SGD sequences are D-critical: $0 \in D(\theta)$
- for most sequences, accumulation points are Clarke critical $0 \in \partial J(\theta)$.

Main takeways regarding nonsmoothness

- Practice usually flows transparently as in the smooth case.
- Theoretical analysis is possible, details handled with care.

Plan

$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$, each J_i has L Lipschitz gradient.

"Convergence in quadratic mean": Suppose $\sigma(\theta) \leq \sigma$ and $J(\theta) \geq J^*$, for all $\theta \in \mathbb{R}^p$.
$(I_k)_{k\in\mathbb{N}}$ iid unif., $0 < \alpha \leq 1/L$, $\theta_0 \in \mathbb{R}^p$, $\theta_{k+1} = \theta_k - \alpha\nabla J_{I_k}(\theta_k)$

For $k \in \mathbb{N}$ large enough, and $\alpha > 0$ small enough, SGD finds approximate critical points, in quadratic mean.

$$\mathbb{E}_{I_k}[J(\theta_k - \alpha\nabla J_{I_k}(\theta_k))] \leq J(\theta_k) - \frac{\alpha}{2}\|\nabla J(\theta_k)\|^2 + \frac{L\alpha^2}{2}\sigma^2$$

So called: Martingale method.

ODE method: alternative arguments.

- Relate to continuous time dynamics.
- Main theoretical option to handle nonsmoothness.
- Qualitative.

Stochastic approximation: $\theta_{k+1} = \theta_k - \alpha_k(\nabla J(\theta_k) + \epsilon_{k+1})$.
Zero mean noise $\epsilon_{k+1} \in \mathbb{R}^p$, independent of the past.

Differentiable J (Ljung 1977): The sequence $(\theta_k)_{k\in\mathbb{N}}$ behaves in the small step limit as solutions to the differential equation

$$\dot{\theta} = -\nabla J(\theta)$$

Developments: Benaïm, Kushner, Yin ....

Nonsmooth setting:

$$\theta_{k+1} \in \theta_k - \alpha_k(H(\theta_k) + \epsilon_{k+1})$$

$H : \mathbb{R}^p \rightrightarrows \mathbb{R}^p$, set valued, locally bounded, convex non-empty values, closed graph.

- Clarke's subdifferential $\partial^c J$.
- Set-valued output of nonsmooth backpropagation $\mathrm{conv}(D)$.
- First order dynamics in phase space $\rightarrow$ second order dynamics.

Noise averaging: Zero mean + independent of the past + boundedness conditions $\rightarrow$ negligible effect, martingale argument.

Euler discretization: $J\colon \mathbb{R}^p \to \mathbb{R}$, Lipschitz, semi-algebraic,

$$\frac{\theta_{k+1} - \theta_k}{\alpha_k} \in -\partial^c J(\theta_k) \qquad \sim \qquad \dot{\theta}(t) \in -\partial^c J(\theta(t))$$

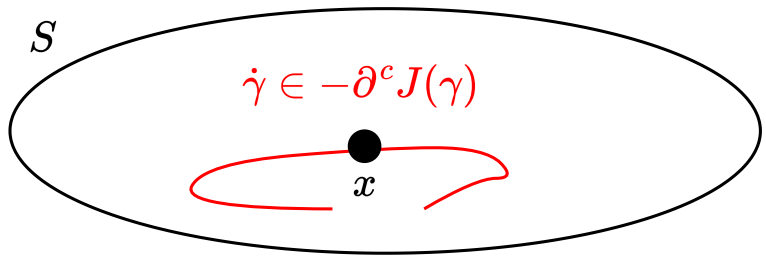
- Invariance.
- Regular values are repulsive.
- Leverage Semi-algebraicity.
- Noiseless setting.

Differential inclusion: $J\colon \mathbb{R}^p \to \mathbb{R}$, Lipschitz, semi-algebraic,

$$\dot\gamma(t) \in -(\partial^c J(\gamma(t))$$

Solutions: $\gamma\colon \mathbb{R} \to \mathbb{R}^p$ Lipschitz, almost all t (Aubin, Celina, Filippov . . .)

Invariant set: $S \subset \mathbb{R}^p$, fo all $x \in S$,
there is $\gamma\colon \mathbb{R} \to \mathbb{R}^p$, Lipschitz solution,
$\gamma(0) = x$ and $\gamma(\mathbb{R}) \subset S$.



Theorem: $\theta_{k+1} \in \theta_k - \alpha_k \partial J(\theta_k)$. Under boundedness assumptions.
acc denote the set of accumlation points, the following are invariant sets

- **Vanishing steps:** $\alpha_k \to 0$ as $k \to \infty$, $\sum \alpha_k = \infty$ (Benaim, Hoffbauer, Sorin),

$$S = \mathrm{acc}\,(\theta_k)_{k\to\infty}$$

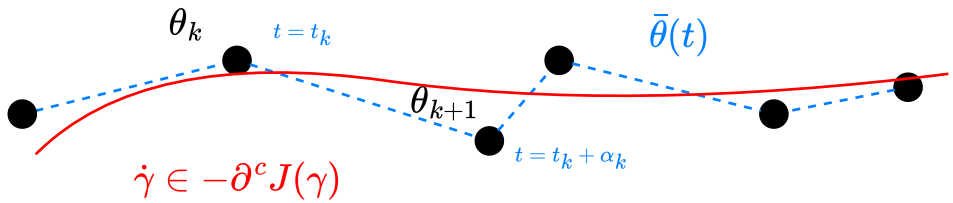
- **Constant steps:** $\theta_k(\alpha)$ with $\alpha_k = \alpha > 0$ for all k (Bolte, Le, Moulines, Pauwels)

$$S = \bigcap_{\alpha>0} \mathrm{cl} \bigcup_{0<s\le\alpha} \mathrm{acc}\,(\theta_k(s))_{k\to\infty} \quad = \quad \text{“}\mathrm{acc}\,(\,\mathrm{acc}\,(\theta_k(\alpha))_{k\to\infty}\,)_{\alpha\to 0+}\text{”}$$

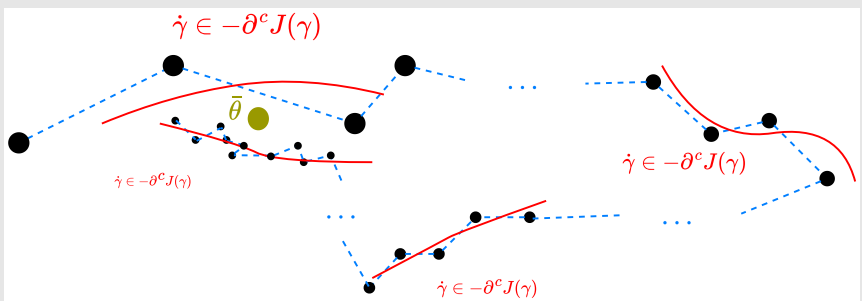
Euler discretization:

$$\frac{\theta_{k+1} - \theta_k}{\alpha_k} \in -\partial^c J(\theta_k) \qquad \sim \qquad \dot{\theta}(t) \in -\partial^c J(\theta(t))$$

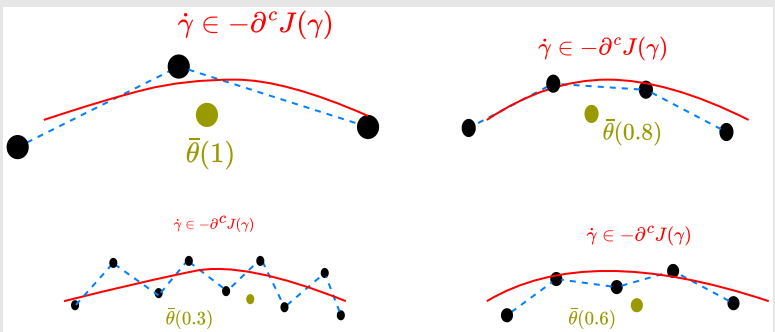
Sequences as Lipschitz functions: there is a nearby solution, closer as step size decreases



Vanishing step:



Constant step:



Recursion: $\alpha_k > 0$, $\quad \theta_{k+1} \in \theta_k - \alpha_k \partial^c J(\theta_k)$
$\operatorname{crit} J = \{\theta \ : \ 0 \in \partial^c J(\theta)\}$ $\quad \operatorname{vcrit} J = J\left(\operatorname{crit} J\right)$

Lemma: If $l \notin \operatorname{vcrit}$, there is $\bar{\alpha} > 0$ such that if $\alpha_k \leq \bar{\alpha}$ and $\sum \alpha_k = +\infty$,
either $\limsup_{k\to\infty} \|\theta_k\| = +\infty$, or $\liminf_{k\to\infty} J(\theta_k) > l$ or $\limsup_{k\to\infty} J(\theta_k) < l$.

Proof crucially relies on the chain rule and Lyapunov decrease.

Consequence: $\theta_{k+1} \in \theta_k - \alpha_k \partial J(\theta_k)$. Under boundedness assumptions.
acc denote the set of accumlation points, the following are invariant sets

- **Vanishing steps:** $\alpha_k \to 0$ as $k \to \infty$, $\sum \alpha_k = \infty$

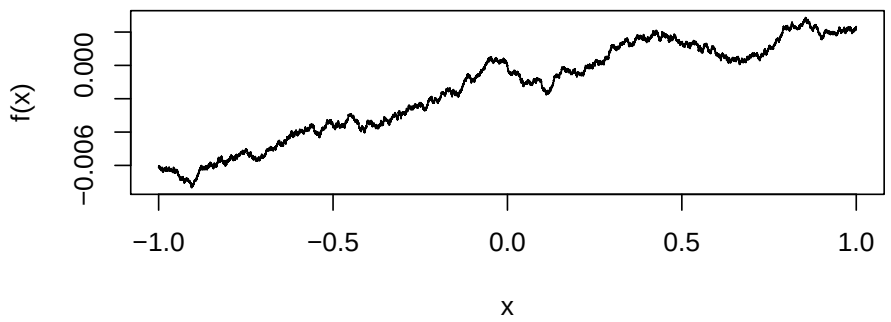
$$S = \operatorname{acc}\left(\theta_k\right)_{k\to\infty} \qquad J(S) \subset \operatorname{vcrit}.$$

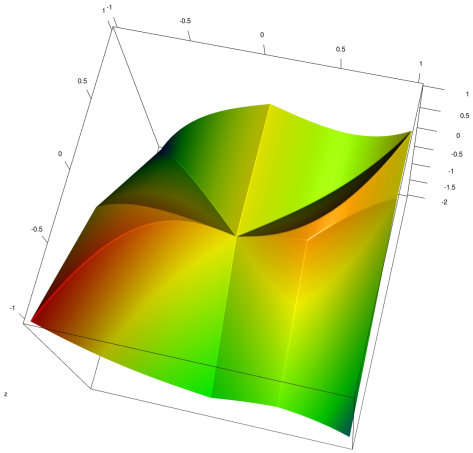
- **Constant steps:** $\theta_k(\alpha)$ with $\alpha_k = \alpha > 0$ for all k

$$S = \bigcap_{\alpha>0} \operatorname{cl} \bigcup_{0<s\leq\alpha} \operatorname{acc}\left(\theta_k(s)\right)_{k\to\infty} \qquad J(S) \subset \operatorname{vcrit}$$

Let $S \subset \mathbb{R}^p$ be invariant and $J(S) \subset \operatorname{vcrit} J$, what can I say about S?

- In general: not much. For well structured f: a lot more.





Semi-algebraic J: $S \subset \operatorname{crit}$.

- $\operatorname{vcrit} J$ is finite (Morse-Sard), $J(S)$ is constant.
- For any Lipschitz γ, $\frac{d}{dt} J(\gamma(t)) = -\min_{v \in \partial^c J(\gamma(t))} \|v\|^2$ a.a. t (chain rule).
- Invariance, for any $x \in S$, $\gamma(0) = x$, $\gamma(\mathbb{R}) \subset S$, $J(\gamma(\mathbb{R}))$ singleton, $\frac{d}{dt} J(\gamma(t)) = 0$.

Conclusion regarding the ODE method

Theorem: $\theta_{k+1} \in \theta_k - \alpha_k \partial J(\theta_k)$. Under boundedness assumptions.
acc denote the set of accumlation points

- **Vanishing steps:** $\alpha_k \to 0$ as $k \to \infty$, $\sum \alpha_k = \infty$ (Benaim, Hoffbauer, Sorin),

$$\mathrm{acc}\,(\theta_k)_{k\to\infty} \subset \mathrm{crit} \qquad \lim_{k\to\infty} \mathrm{dist}(\theta_k, \mathrm{crit}\,) = 0.$$

- **Constant steps:** $\theta_k(\alpha)$ with $\alpha_k = \alpha > 0$ for all k (Josz *et. al.*)

$$\bigcap_{\alpha>0} \mathrm{cl} \bigcup_{0<s\leq\alpha} \mathrm{acc}\,(\theta_k(s))_{k\to\infty} \subset \mathrm{crit} \qquad \lim_{\alpha\to 0} \limsup_{k\to\infty} \mathrm{dist}(\theta_k, \mathrm{crit}\,) = 0.$$

Many extensions:
Averaging out noise, more complicated dynamics, avoidance of traps . . . .

Plan

Understanding deep networks

- Why can we train deep networks (non convex, NP-hard)?
- Why do deep network generalize?

Statistical learning theory: Understanding Deep Learning Requires Rethinking Generalization (Zhang *et. al.* 2017). Challenge the notion of overfitting.

Important factor:

- Optimization algorithms
- Beyond computational efficiency (contrary to traditional statistical learning).
- Loss structure (compositional, rigidity).

Absence of convexity: cannot garanty better than critical points.

Heuristic explaination: gradient methods succeed in deep network training

- All (most) local minima are close to global
- (most) saddle points have negligible effect

Many results for specific model classes (linear, approximation by physics models . . .).

Large dimension, many escapes

Extreme overparametrization: many more parameters than data.

Classical view: red flag.

Theoretical studies (starting 2018): this could have some benefit

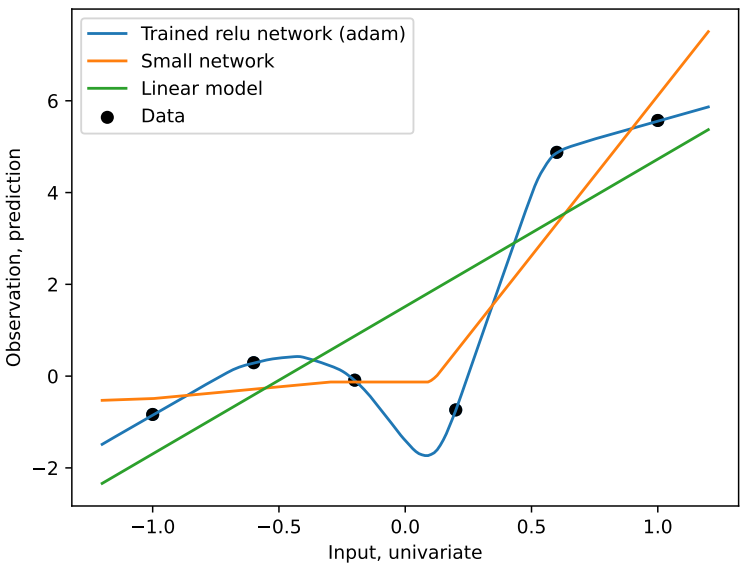
- Approximate well optimization in function space (possibly convex).
- Global minima almost dense, SGD converges to them (lazy training).
- Neural tangent kernel.
- Mean field approximation.

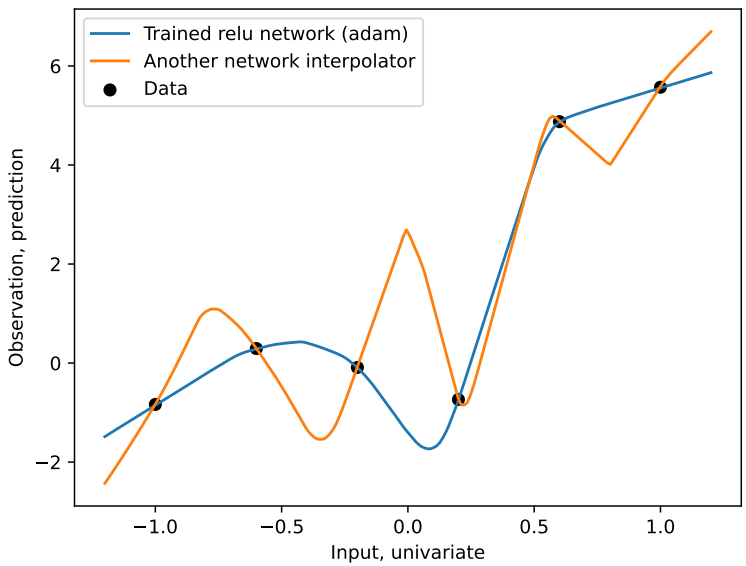
Many minima: Gradient method "chooses" good global minima, $A \in \mathbb{R}^{n \times p}$, $p > n$, $\bar{x} \in \mathbb{R}^p$ unknown, $A\bar{x}$ known:

$$\min_x \|Ax - A\bar{x}\|_2^2$$

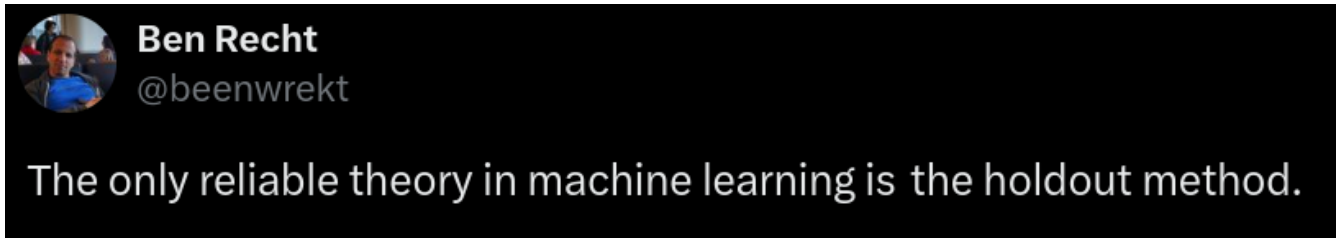
Gradient descent initialized at 0 will converge (linearly) to

$$\begin{aligned} \arg\min_x \quad & \|x\|_2^2 \\ \text{s.t.} \quad & Ax = A\bar{x} \end{aligned}$$





Implicit bias: solution algorithm is key, directional convergence, benign overfitting . . .



Donoho: Data science at the singularity 2024.

The emergence of frictionless reproducibility follows from the maturation of 3 data science principles that came together after decades of work by many technologists and numerous research communities. The mature principles involve data sharing, code sharing, and competitive challenges, *however* implemented in the particularly strong form of frictionless open services. Empirical Machine Learning (EML) is today's leading adherent field, and its consequent rapid changes are responsible for the AI progress we see.

Plan

Optimization for deep learning:

- Renewed interest in nonconvex optimization, algorithm design and analysis.
- Specific community / practice, sometimes different from classical math programming
- Practice is evolving extremely fast.
- Algorithmic ideas are difficult to evaluate (benchmarking is a full time job).
- Theoretical arguments to explain empirical successes for idealized situations.

Thanks