

An introduction to optimization for deep learning

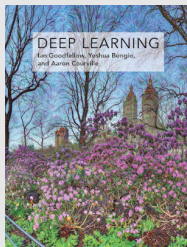
EDOUARD PAUWELS

CIMI Virtual School on Optimisation (September 2021)



Institut de Recherche
en Informatique de Toulouse
CNRS - INP - UTS - UT1 - UT2J





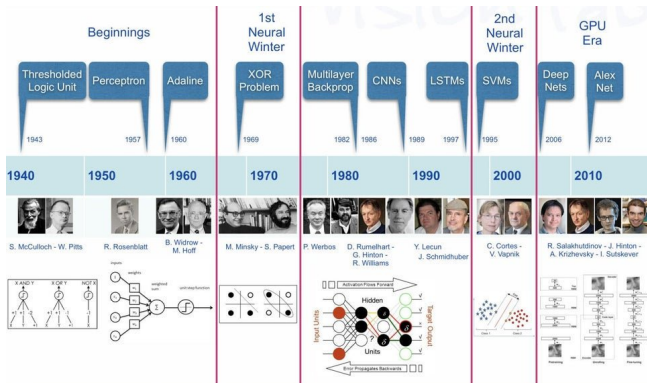
- Deep learning = machine learning with a specific class of models
- Deep network training reduces to an optimization problem
- Two important structural specificities: large sum, compositional model.
- Specific context, with its own goals and difficulties, dedicated algorithms.

Ongoing research with many collaborators (Bolte, Févotte, Castera, Rios-Zertuche, Glaudin, Le, Traoré, Silveti, . . .)

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness
- 4 Deep learning optimizers
- 5 Further questions
- 6 Conclusion

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness
- 4 Deep learning optimizers
- 5 Further questions
- 6 Conclusion

Brief historical perspective

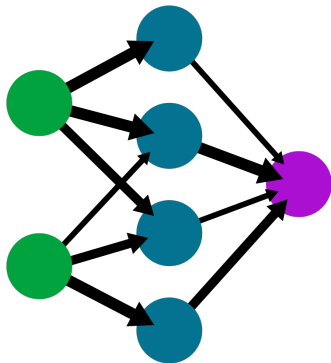


- Research
- Computer hardware
- Large amount of data
- Private sector efforts to develop softwares

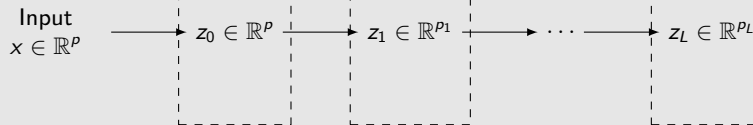


IMAGENET

TensorFlow



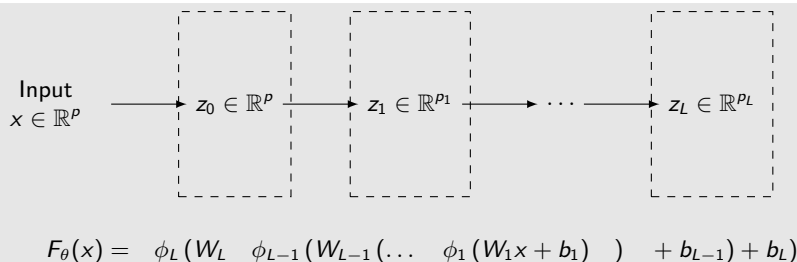
- Directed acyclic graph (DAG): partial order, parents, children.
- Each node is an artificial neuron which inputs are the parent nodes.
- Nodes organised in layers: input, hidden, output (biological inspiration).
- Notion of depth: number of layers.



For $i = 1, \dots, L$:

- $z_i \in \mathbb{R}^{p_i}$ “layer”.
- $z_i = \phi_i(W_i z_{i-1} + b_i)$
- $\phi_i: \mathbb{R}^{p_i} \mapsto \mathbb{R}^{p_i}$ “activation functions”, nonlinear.
- $W_i \in \mathbb{R}^{p_i \times p_{i-1}}$, $b_i \in \mathbb{R}^{p_i}$, $\theta = (W_1, b_1, \dots, W_L, b_L)$, model parameters.

$$\begin{aligned}
 F_\theta(x) &= z_L \\
 &= \phi_L(W_L \phi_{L-1}(W_{L-1}(\dots \phi_1(W_1 x + b_1) \dots) + b_{L-1}) + b_L)
 \end{aligned}$$



Training set: input output pairs $\{(x_i, y_i)\}_{i=1}^n$ in $\mathbb{R}^p \times \mathbb{R}^{p_L}$.

Ultimate goal: Find θ such that

$$F_{\theta}(x_{n+1}) \simeq y_{n+1} \quad (\text{new "unseen" data})$$

Training: Find θ such that

$$F_{\theta}(x_i) \simeq y_i, \quad i = 1, \dots, n.$$

An optimization problem: loss $\ell: \mathbb{R}^{p_L} \times \mathbb{R}^{p_L} \rightarrow \mathbb{R}_+$.

$$\min_{\theta} J(\theta) := \frac{1}{n} \sum_{i=1}^n \ell(F_{\theta}(x_i), y_i) = \frac{1}{n} \sum_{i=1}^n J_i(\theta).$$

Minimize an expectation: n potentially very large (10^5)

$$\begin{aligned} \min_{\theta} \quad J(\theta) &:= \frac{1}{n} \sum_{i=1}^n \ell(F_{\theta}(x_i), y_i) = \frac{1}{n} \sum_{i=1}^n J_i(\theta) \\ &= \mathbb{E}_{I \sim U(\{1, \dots, n\})} [J_I(\theta)]. \end{aligned}$$

Typical size of θ , 10^6 , no convexity

Main algorithmic tool: gradient descent .

Minimize an expectation: n potentially very large (10^5)

$$\begin{aligned} \min_{\theta} \quad J(\theta) &:= \frac{1}{n} \sum_{i=1}^n \ell(F_{\theta}(x_i), y_i) = \frac{1}{n} \sum_{i=1}^n J_i(\theta) \\ &= \mathbb{E}_{I \sim U(\{1, \dots, n\})} [J_I(\theta)]. \end{aligned}$$

Typical size of θ , 10^6 , no convexity

Main algorithmic tool: gradient descent.

Stochastic (minibatch) gradient algorithm: Given $(I_k)_{k \in \mathbb{N}}$ iid, uniform on $\{1, \dots, n\}$, $(\alpha_k)_{k \in \mathbb{N}}$ positive, iterate,

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

Backpropagation: Backward mode of algorithmic differentiation used to compute ∇J_i

Algorithmic differentiation (AD, 70s):

Automatized numerical implementation of the chain rule:

$$H: \mathbb{R}^p \mapsto \mathbb{R}^p, \quad G: \mathbb{R}^p \mapsto \mathbb{R}^p, \quad f: \mathbb{R}^p \rightarrow \mathbb{R}, \quad (\text{differentiable}).$$

$$f \circ G \circ H: \mathbb{R}^p \mapsto \mathbb{R}.$$

$$\nabla(f \circ G \circ H)^T(x) = \nabla f^T(G \circ H(x)) \times \text{Jac}_G(H(x)) \times \text{Jac}_H(x)$$

Function implemented by program: composition of smooth functions.

$$x \mapsto (H(x), G(H(x)), f(G(H(x))))$$

Forward mode of AD: $\nabla f^T \times (\text{Jac}_G \times \text{Jac}_H)$.

Backward mode of AD: $(\nabla f^T \times \text{Jac}_G) \times \text{Jac}_H$.

Backpropagation: Backward AD for neural network training.

It computes gradient (and it works beyond smoothness).

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

Given $(I_k)_{k \in \mathbb{N}}$ iid, uniform on $\{1, \dots, n\}$, $(\alpha_k)_{k \in \mathbb{N}}$ positive, iterate,

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

- **Neural network training:** optimization.
- **Large scale:** gradient algorithms.
- **Large sum:** Stochastic subsampling approximation.
- **Compositional structure:** algorithmic differentiation.

Profusion of numerical tools: e.g. Tensorflow, Pytorch. Democratized the usage of these models. Goes beyond neural nets (differentiable programming).

Define a neural network model:

- Define a structure (a directed acyclic graph).
- Define the activation functions (nonlinearities).
- A lot more ...

Described by a numerical program.

Training mechanisms:

- What one can program, one can differentiate.
- Minibatch stochastic approximation for large sums or more general expectations
- Optimize using available numerical solvers.

Tensorflow, Pytorch:

~ transparent, efficient and flexible implementation of training.

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms**
- 3 Nonsmoothness
- 4 Deep learning optimizers
- 5 Further questions
- 6 Conclusion

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

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.

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

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

$$J(\theta - \alpha \nabla J_i(\theta)) \leq J(\theta) - \alpha \langle \nabla J(\theta), \nabla J_i(\theta) \rangle + \frac{L\alpha^2}{2} \|\nabla J_i(\theta)\|^2$$

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

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

From “descent” to convergence

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

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

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

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

Robbins-Monro (1951), Robbins-Sigmund (1971): $(I_k)_{k \in \mathbb{N}}$ iid, $(\alpha_k)_{k \in \mathbb{N}}$ positive:
 $\theta_{k+1} = \theta_k - \alpha_k \nabla J_{I_k}(\theta_k)$

Suppose

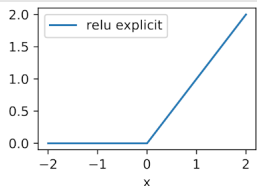
- $\sup_{\theta} \sigma(\theta) < +\infty$
- J is lower bounded.
- $\sum_{k \in \mathbb{N}} \alpha_k = \infty$, $\sum_{k \in \mathbb{N}} \alpha_k^2 < +\infty$
- $(\theta_k)_{k \in \mathbb{N}}$ is bounded almost surely.

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

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness**
- 4 Deep learning optimizers
- 5 Further questions
- 6 Conclusion

Positive part: $\text{relu}(t) = \max\{0, t\}$,

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



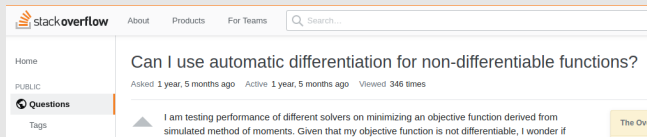
Less straightforward examples:

- Max pooling in convolutional networks.
- knn grouping layers, farthest point subsampling layers.
Qi *et. al.* 2017. PointNet++: Deep Hierarchical Feature Learning on point Sets in a Metric Space.
- Sorting layers.
Anil *et. al.* 2019. Sorting Out Lipschitz Function Approximation. ICML.

Nonsmooth optimization (analysis):

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

Nonsmooth algorithmic differentiation:



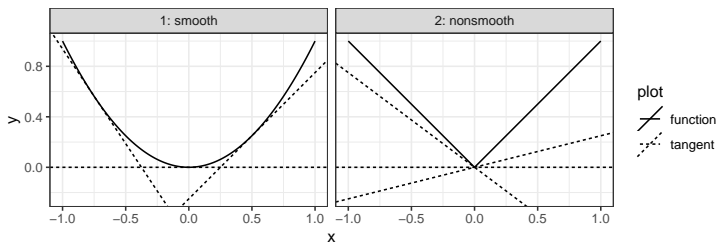
- A few contribution from algorithmic differentiation community (2000's).
- Empirical, not completely “mature”.

In deep learning: largely overlooked.

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

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

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



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 gradients.

$$\partial_{\text{cl}} F(x) = \left\{ v \in \mathbb{R}^p, \exists (y_k, v_k)_{k \in \mathbb{N}}, y_k \xrightarrow[k \rightarrow \infty]{} x, v_k \xrightarrow[k \rightarrow \infty]{} v, y_k \in R, v_k = \nabla F(y_k), k \in \mathbb{N} \right\}.$$

Clarke subgradient: convex closure.

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

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

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

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

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

$$\begin{aligned} \theta_{k+1} = \theta_k - \alpha_k v_k & \Leftrightarrow \frac{\theta_{k+1} - \theta_k}{\alpha_k} \in -\partial J(\theta_k) \\ v_k \in \partial J(\theta_k). & \end{aligned}$$

Chain rule along Lipschitz curves (Brézis 1973, Valadier 1989).

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

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

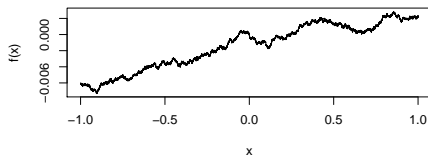
Suppose: $\dot{\gamma}(t) \in -\partial J(\gamma(t))$ for almost all $t \in [0, 1]$,
then $t \mapsto J(\gamma(t))$ decreases, strictly if $0 \notin \partial J(\gamma(t))$.

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.

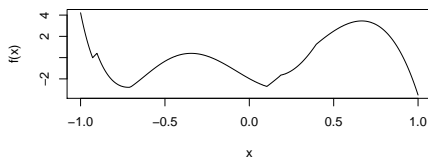
Borwein-Moors (2000), Loewen-Wang (2000): Let f be a typical 1-Lipschitz function (in sup norm), then

- ∂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.

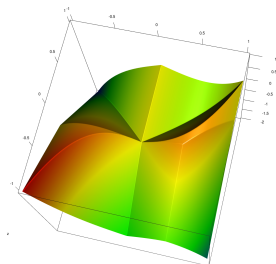


Basic set: Solution set of finitely many polynomial inequalities.

Set: Finite union of Basic semi-algebraic sets.

Function, set valued map: Semi-algebraic graph.

Examples: polynomials, square root, quotients, norm, relu, rank . . .



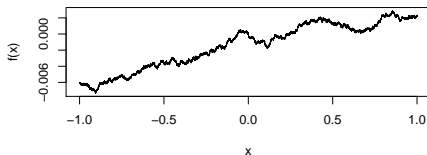
Tarski Seidenberg: first order formula involving semi-algebraic sets \rightarrow semi-algebraic.

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

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

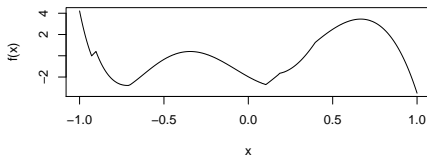
Typical Lipschitz function: pathological

- No chain rule (no convergence).



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

- Chain rule (rich convergence theory).



No convexity, no calculus:

$$\partial(f + g) \subset \partial f + \partial g.$$

- holds with equality if f and g are continuously differentiable.
- holds with equality if f and g are convex (full domain).
- does not hold in general: $f: x \mapsto |x|$

$$\begin{aligned} \partial(f - f) &= \partial(x \mapsto 0) = \{0\} \\ \subset \partial(f) + \partial(-f) \\ &= \begin{cases} \{0\} & \text{if } x < 0 \\ \{0\} & \text{if } x > 0 \\ [-2, 2] & \text{if } x = 0 \end{cases} \end{aligned}$$

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

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

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 $\gamma: [0, 1] \mapsto \mathbb{R}^p$

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

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 $0 \in \partial J(\theta)$.

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness
- 4 Deep learning optimizers**
- 5 Further questions
- 6 Conclusion

 TensorFlow

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

 PyTorch

- 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)

Gradient update:

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

$$\frac{x_{k+1} - x_k}{\alpha} + \nabla f(x_k) = 0$$

Gradient with momentum:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \beta(x_k - x_{k-1})$$

Alternatively:

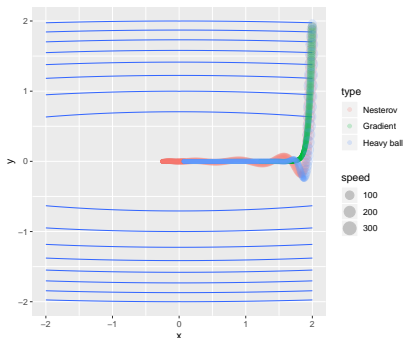
$$v_{k+1} = \mu v_k + \nu \nabla f(x_k)$$

$$x_{k+1} = x_k - v_{k+1}$$

Continuous time counterparts: ($\alpha \rightarrow 0$)

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

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



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

$$\mathbf{x}_{k+1} = \mathbf{x}_k - D_k \nabla f(\mathbf{x}_k)$$

- 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: $x_{k+1} = x_k - D_k \nabla f(x_k)$

i -th entry of D_k :

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

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

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

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

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

$$\frac{\gamma}{|[\nabla f(x_k)]_i|}$$

Remarks:

- If $f(x) = \sum_{i=1}^p w_i x_i^2$, 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.
- Stochastic variants: plug and play. Convergence analysis (Li & Orabona 2018 ...)

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)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

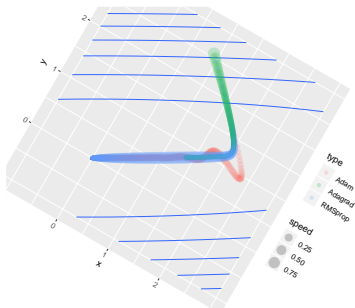
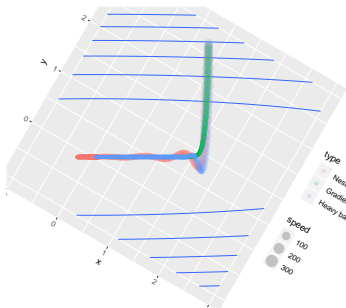
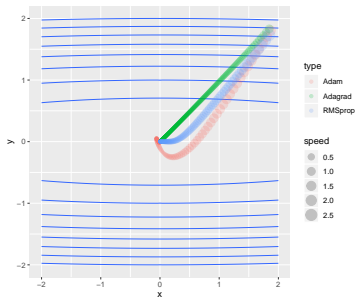
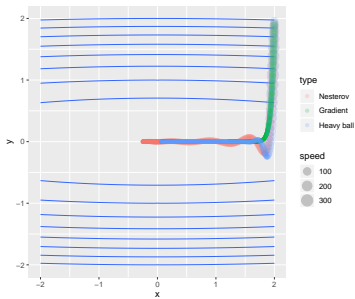
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t (Resulting parameters)

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

“Adaptive” step size illustration , rotation equivariance

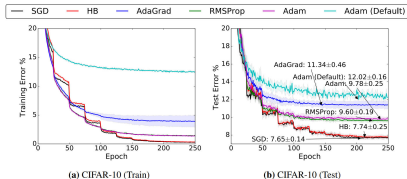


Mixed with the network structure, affect the optimization process.

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

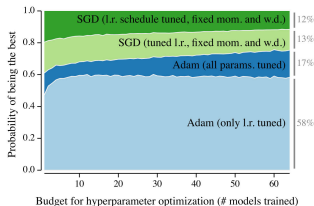
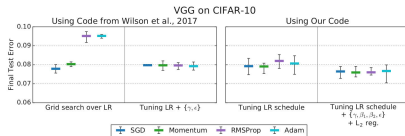
What is the best algorithm?

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



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

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

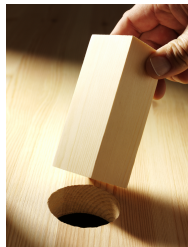


- Parameter tuning.
- Huge variability: tasks (datasets), architecture, additional factors.
- Computation cost: 1k euro for a single network training.
- No uniformly better algorithm.

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness
- 4 Deep learning optimizers
- 5 Further questions**
- 6 Conclusion

- 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).



Important factor:

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

Absence of convexity: cannot guarantee better than critical points.

Heuristic explanation: 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 . . .), no clear general result.

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

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

Implicit bias: gradient dynamics is a key directional convergence, benign overfitting, gradient flows ...

- 1 A primer on deep neural networks
- 2 Stochastic gradient algorithms
- 3 Nonsmoothness
- 4 Deep learning optimizers
- 5 Further questions
- 6 Conclusion**

Optimization for deep learning:

- Triggered renewed interest in nonconvex optimization and nonsmooth analysis.
- Specific community / practice, sometimes different from classical math programming
- Practice has evolved extremely fast.
- Algorithmic ideas are difficult to evaluate (benchmarking is a full time job).
- A lot of open questions.

Thanks