Advanced Course on Deep Learning and Geophysical Dynamics

Course 1: Deep Learning and Optimization

Lucas Drumetz

IMT Atlantique

November, 15th, 2022

Outline

1. Optimization and machine learning
   - The very basics of Deep Learning
   - (Nondifferentiable) convex optimization
2. Bilevel optimization and meta learning
   - Motivation and examples
   - Dealing with bilevel problems
     - Algorithm Unrolling
     - Implicit Differentiation
     - Learning Solvers

Table of Contents

1 Optimization and machine learning
- The very basics of Deep Learning
- (Nondifferentiable) convex optimization

2 Bilevel optimization and meta learning
- Motivation and examples
- Dealing with bilevel problems
  - Algorithm Unrolling
  - Implicit Differentiation
  - Learning Solvers

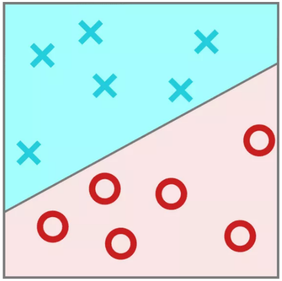
Supervised learning problem

Supervised learning: learn a relationship of the form

$$\mathbf{y} = f_\theta(\mathbf{x})$$

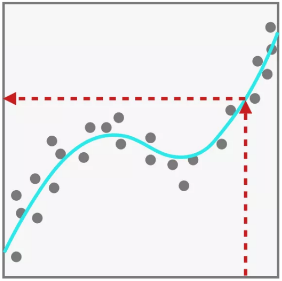
We are given a dataset of input/output pairs $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1,..,N}$.
$\mathbf{x} \in \mathbb{R}^d$ is the input variable. Depending on the problem, $\mathbf{y}$ can be categorical or continuous

Classification Groups observations into "classes"



Here, the line classifies the observations into X's and O's

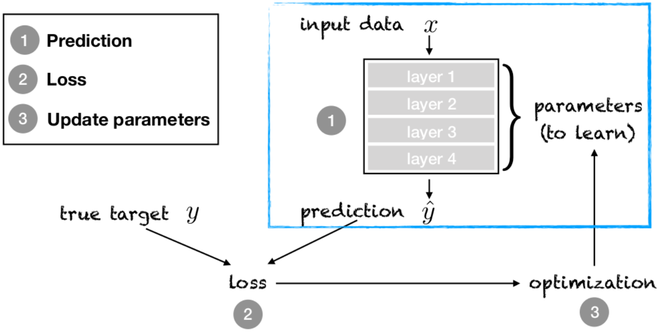
Regression predicts a numeric value



Here, the fitted line provides a predicted output, if we give it an input

Supervised learning ingredients

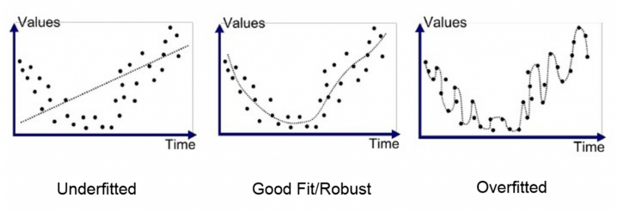
- Training Data and problem $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1,..,N}$
- Hypothesis set f_θ, a set of functions parameterized by $\theta \in \mathbb{R}^d$
- Loss function $\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$. Typically:
  - For regression: squared L_2 distance
  - For classification: Cross Entropy
- Optimization algorithm, usually gradient based



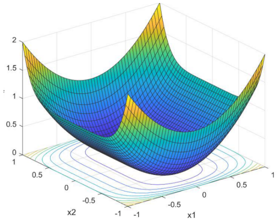
Hypothesis set

Or simply "model". The model is ideally:

- Expressive (but not too complex in order to generalize well)



- Leads to "simple" optimization problems



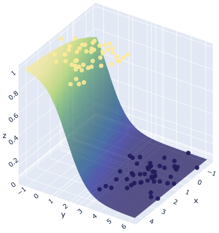
- Interpretable

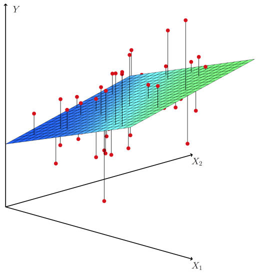
Linear Models

Linear models: Easy to optimize, interpretable, but not very expressive: Logistic regression or linear regression

$$f_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{w}_0) = \phi(\mathbf{x})$$

with $\theta = \{\mathbf{W}, \mathbf{w}_0\}$. $\sigma = \mathrm{softmax}$ for classification, the identity for regression.





Multilayer Perceptron

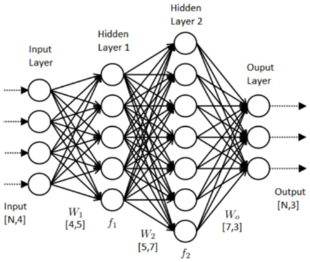
("Pure") Neural Networks: Expressive (universal approximators), less interpretable:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \phi_K \circ \phi_{K-1} \circ \cdots \circ \phi_1(\mathbf{x})$$

parameters $\boldsymbol{\theta}$ are made of all $\mathbf{W}_k$ ("weights") and $\mathbf{b}_k$ ("biases").

$$\mathbf{a}_{k+1} = \sigma(\mathbf{W}_k \mathbf{a}_k + \mathbf{b}_k) = \phi_k(\mathbf{a}_k)$$

with σ a nonlinear function applied componentwise (activation function).



Classical CNNs and RNNs are just particular instances of these with structured weights and weight sharing schemes.

Gradient-based optimization

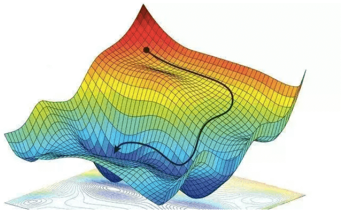
Are those models "easy to optimize" in supervised learning?

$$\arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_n))$$

- Optimization is no longer convex and does not have a closed form

$\rightarrow$ iterative optimization algorithms (usually first-order). The simplest is (stochastic) gradient descent.

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \rho \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_i)$$



More generally: Differentiable programs

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}$$

What about differentiability? $\rightarrow$ Backpropagation algorithm to obtain $\frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}$

More generally: reverse mode automatic differentiation. Applies to any "differentiable program", i.e. composition of differentiable building blocks.

In practice, code your model in Pytorch, Tensorflow, JAX, JuliaDiff, etc.

A computation graph keeps track of all the elementary operations involved. The chain rule is used to backtrack in the graph and obtain the exact derivatives of the output value wrt to the parameters to train.

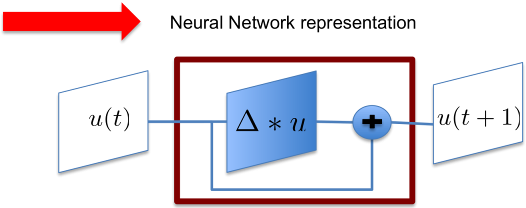
What automatic differentiation allows to do

- Apply gradient descent to any real valued differentiable function without computing derivatives by hand
- Hard cable structural or physical constraints into algorithms

Partial Differential Equation/
Ordinary Differential Equation

Neural Network representation

$$\frac{\partial u}{\partial t} = \kappa \Delta u$$



- Differentiate through iterative algorithms, e.g. ODE/PDE solvers or optimization algorithms (more on that next week)

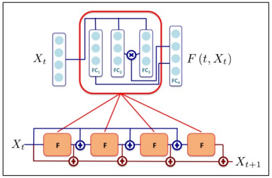


Table of Contents

1 Optimization and machine learning
- The very basics of Deep Learning
- (Nondifferentiable) convex optimization

2 Bilevel optimization and meta learning
- Motivation and examples
- Dealing with bilevel problems
  - Algorithm Unrolling
  - Implicit Differentiation
  - Learning Solvers

Brief recap on convex optimization

In many domains we oppose linear ("easy") and nonlinear ("hard") problems.

In optimization: convex ("easy") vs nonconvex ("hard") problems.

$$\arg\min_{\mathbf{x}} f(\mathbf{x})$$
$$\text{s.t. } \mathbf{g}(\mathbf{x}) \leq 0, \mathbf{h}(\mathbf{x}) = 0$$

with **g** and **h** two (possibly vector valued) functions defining inequality and equality constraints.

An optimization problem is convex iff those two conditions are met:

- the objective function is convex
- the constraints define a convex set.

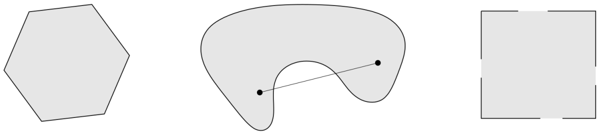
Convex sets

Convex set

A set $A \subset \mathbb{R}^n$ is convex if

$$\forall \mathbf{x}, \mathbf{y} \in A,\ [\mathbf{x}, \mathbf{y}] \subset A,\ \text{i.e. } \forall \lambda \in\ [0, 1],\ \lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in A.$$

- A set is convex if any segment whose endpoints are in A is fully included in A.
- Examples:



Some convex and nonconvex sets [Boyd & Vandenberghe]

- Other Examples:
  - A disk in $\mathbb{R}^2$ is convex
  - A circle in $\mathbb{R}^2$ is **not** convex

Convex functions

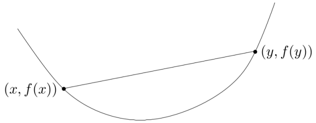
Convex functions

The function $f : A \subset \mathbb{R}^n \to \mathbb{R}$, where A is a convex set, is convex iff

$$\forall \mathbf{x}, \mathbf{y} \in A,\ \forall \lambda \in [0,1],\ f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}).$$

A function is strictly convex when the inequality is strict.

- A function is convex iff any segment between two images is always above the graph of the function for the segment joining the preimages.



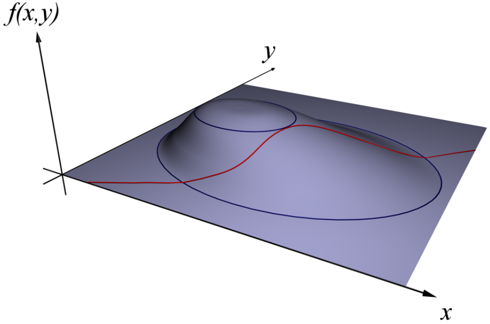
Graph of a convex function. [Boyd & Vandenberghe]

- The real exponential function is convex
- A linear function $\mathbf{c}^T\mathbf{x}$ is convex
- Every norm is convex

Example of a constrained optimization problem

Maximizing a two variable function subject to a single (linear) constraint ($ax + by + c = 0$):

$$\arg\min_{\mathbf{x}} f(x, y)$$
$$\text{s.t. } ax + by + c = 0$$



Optimality conditions for convex problems

Optimality conditions for convex problems

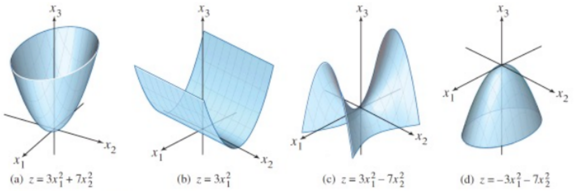
In the unconstrained case, the equation (whose solutions are called "critical points")

$$\nabla f(\mathbf{x}) = 0$$

is a necessary and *sufficient* optimality condition (generalizes to Karush-Kuhn-Tucker (KKT) conditions when there are constraints)

Otherwise this condition is only necessary and critical points can be:

- (local) minima
- (local) maxima
- saddle points


x_3
x_1
x_2
(a) $z = 3x_1^2 + 7x_2^2$
(b) $z = 3x_1^2$
(c) $z = 3x_1^2 - 7x_2^2$
(d) $z = -3x_1^2 - 7x_2^2$

Types of critical points for quadratic functions in 2D

Gradient descent

Consider $f : \mathbb{R}^n \to \mathbb{R}$ differentiable that we want to minimize:

$$\arg\min_{\mathbf{x}} f(\mathbf{x})$$

Gradient descent

To find a local minimum (global if the function is convex), we can use conventional gradient descent:

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

starting from an initial point $\mathbf{x}_0$, and with ρ small enough to guarantee convergence

In DL, typically we deal with differentiable (except possibly on a finite number of points $\to$ ReLU...) unconstrained optimization.

But nonconvex problems in almost all cases!!

Convex optimization under convex constraints

What if we have (convex) constraints?

$$\underset{\mathbf{x}\in\mathcal{C}}{\arg\min}\ f(\mathbf{x})$$

where $\mathcal{C}$ is a convex set, on which projecting any point is easy, i.e. solving

$$\mathbf{proj}_{\mathcal{C}}(\mathbf{y}) = \underset{\mathbf{x}\in\mathcal{C}}{\arg\min}\ \frac{1}{2}||\mathbf{x}-\mathbf{y}||_2^2$$

Examples:

- $\mathbf{x}$ is a vector of positive coefficients: $\mathbf{proj}_{\mathbb{R}_+^n}(\mathbf{y}) = \max(\mathbf{y}, \mathbf{0})$
- $\mathbf{x}$ is a vector of proportions: $\mathbf{proj}_{\{u\in\ \mathbb{R}_+^n, \sum_{i=1}^n u_i = 1\}}$ can be computed efficiently
- $\mathbf{x}$ is a vector whose norm cannot exceed a given value : $\mathbf{proj}_{\{\mathbf{u}, ||\mathbf{u}|| \leq K\}} = \max(1, K/||\mathbf{y}||)\ \mathbf{y}$

Projected gradient

Rewrite the problem as an unconstrained problem

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) + \mathcal{I}_{\mathcal{C}}(\mathbf{x})$$

where

$$\mathcal{I}_{\mathcal{C}}(\mathbf{x}) = \begin{cases} 0 \text{ if } \mathbf{x} \in \mathcal{C} \\ +\infty \text{ otherwise} \end{cases}$$

is the indicator function of the convex set $\mathcal{C}$. New problem is unconstrained and convex if f is, but nondifferentiable. we have

$$\mathbf{proj}_{\mathcal{C}}(\mathbf{y}) = \arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + \mathcal{I}_{\mathcal{C}}(\mathbf{x})$$

Projected gradient descent

$$\mathbf{x}_{k+1} = \mathbf{proj}_{\mathcal{C}}(\mathbf{x}_k - \rho_k \nabla f(\mathbf{x}_k))$$

starting from an initial point $\mathbf{x}_0$, and with ρ small enough to guarantee convergence

Proximal gradient

Generalizing, what happens if we want to solve instead

$$\arg\min_{\mathbf{x}} \ f(\mathbf{x}) + g(\mathbf{x})$$

where g is not necessarily differentiable (but assumed convex)?

By analogy with projection, let us define

$$\mathbf{prox}_g(\mathbf{y}) = \arg\min_{\mathbf{x}} \ \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + g(\mathbf{x})$$

$\rightarrow$ result is close to $\mathbf{y}$ but also locally leads to a small value for g

Proximal gradient descent

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

starting from an initial point $\mathbf{x}_0$, and with ρ small enough to guarantee convergence

Proximal operator

It turns out that for a certain number of useful nondifferentiable functions g, $\mathbf{prox}_g$ admits a closed form! Examples:

- indicator function of a convex set: $\mathbf{prox}_{\mathcal{I}_C}(\mathbf{x}) = \mathbf{proj}_C(\mathbf{x})$
- $\mathcal{L}_1$ norm: $\mathbf{prox}_{\tau||\cdot||_1}(\mathbf{x}) = \mathrm{soft}_\tau(\mathbf{x})$ and $\mathrm{soft}_\tau(\mathbf{x})_i = \mathrm{sign}(x_i)(|x_i| - \tau)_+$
- $\mathcal{L}_2$ norm: $\mathbf{prox}_{\tau||\cdot||_2}(\mathbf{x}) = \mathbf{soft}_\tau(\mathbf{x}) = \left(1 - \frac{\tau}{||\mathbf{x}||_2}\right)_+ \mathbf{x}$
- $\mathcal{L}_\infty$ norm: $\mathbf{prox}_{\tau||\cdot||_\infty}(\mathbf{x}) = \mathbf{x} - \tau\ \mathbf{proj}_{(||\cdot||_1 \leq \tau)}(\mathbf{x})$

Many inverse problems can be formulated as optimization of convex but nondifferentiable functions.

[Parikh & Boyd,. 2014]. Proximal algorithms. Foundations and Trends in optim.

Sparse regression

Let $\mathbf{b} \in \mathbb{R}^n$ a signal that can be decomposed a linear combination of large number $m \gg n$ of potential "modes" or "primitive signals" $\mathbf{a}_i \in \mathbb{R}^n$:

$$\mathbf{b} = \sum_{i=1}^{m} x_i \mathbf{a}_i = \mathbf{A}\mathbf{x}$$

where $\mathbf{A} \in \mathbb{R}^{n \times m}$ gathers all the $\mathbf{a}_i$ in its columns, and $\mathbf{x} \in \mathbb{R}^m$ gathers all the coefficients x_i. Recovering $\mathbf{x}$ means solving an underdetermined linear system:

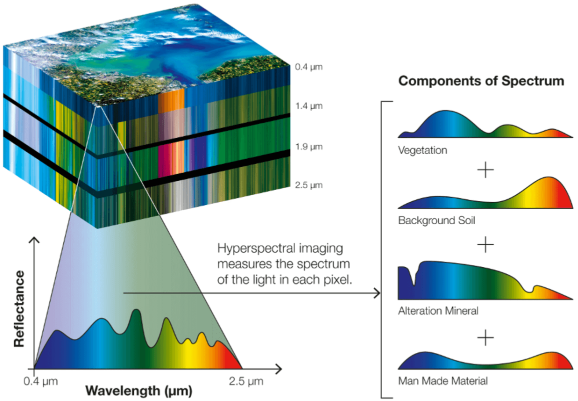
$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A} \in \mathbb{R}^{n \times m}$, with $n < m$ (less equations than unknowns).

$\rightarrow$ There are an infinite number of solutions.

Spectral Unmixing

Hyperspectral images are multivariate images with a large number n of narrow and contiguous observed wavelengths



Elementary signatures corresponding to materials can come from a very large dictionary of m in-situ acquired signatures: but for a given image/pixel, only a few are used.

Sparse regression

If the solution is known to be sparse, i.e. to involve only a "small enough" number of "atoms" out of the *m* possible ones.

Then we know that (under some conditions on the dictionary) the exact solution may be recovered by solving:

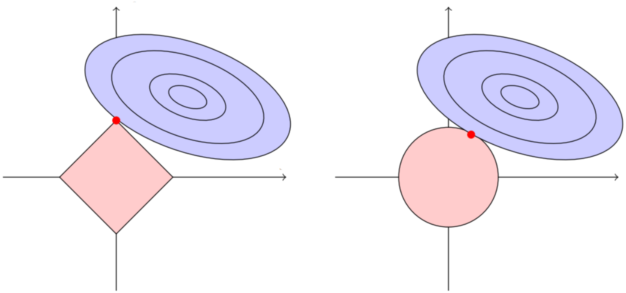
$$\arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{Ax} - \mathbf{b}||_2^2 + \lambda||\mathbf{x}||_1$$

where $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$ is the $\mathcal{L}_1$ norm, and λ is a regularization parameter that controls the sparsity of the solution (the number of zeros in the coefficients of **x**)
We can constrain coefficients to be positive and even to sum to one if they represent proportions.

Sparsity

Equivalent formulation

$$\arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{Ax}-\mathbf{b}||_2^2$$
$$\text{s.t} \quad ||\mathbf{x}||_1 \leq \lambda$$



Problem: the $\mathcal{L}_1$ norm is not differentiable... Also, parameter λ needs to be manually tuned

Image denoising

Let $\mathbf{x} \in \mathbb{R}^n$ be an image with n pixels, that has been corrupted by i.i.d. Gaussian noise $b \in \mathbb{R}^n$, such that the observed image $\mathbf{y}$ is:

$$\mathbf{y} = \mathbf{x} + \mathbf{b}$$

Typical denoising formulation:

$$\arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x} - \mathbf{y}||_2^2 + \lambda \mathcal{R}(\mathbf{x})$$

Image denoising

A good idea is to make sure that close by pixels have similar values:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\left(||\mathbf{x}-\mathbf{y}||_2^2 + \lambda||\nabla\mathbf{x}||_2^2\right)$$

Where $\nabla\mathbf{x} \in \mathbb{R}^{2n}$ denotes the spatial gradient, for each pixel. Discretizing via finite difference operators:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\left(||\mathbf{x}-\mathbf{y}||_2^2 + \lambda(||\mathbf{H}_h\mathbf{x}||_2^2 + ||\mathbf{H}_v\mathbf{x}||_2^2)\right)$$

where $\mathbf{H}_h$ and $\mathbf{H}_h \in \mathbb{R}^{n\times n}$ compute horizontal and vertical gradients by finite differences. This can be rewritten:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\left(||\mathbf{x}-\mathbf{y}||_2^2 + \lambda||\mathbf{H}\mathbf{x}||_2^2\right)$$

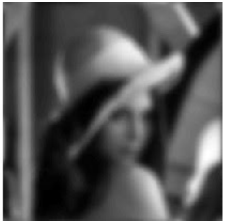
with $\mathbf{H} = \begin{bmatrix}\mathbf{H}_h \\ \mathbf{H}_v\end{bmatrix} \in \mathbb{R}^{2n\times n}$

Tikhonov regularization

The objective is differentiable, convex and easy to optimize (closed form solution!):

$$\mathbf{x} = (\mathbf{I} + \lambda \mathbf{H}^T \mathbf{H})^{-1} \mathbf{y}$$

But the solution will typically result in a blurry image because edges are not preserved:



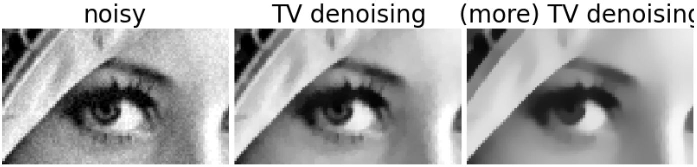
Total variation denoising

- We consider a slightly different formulation:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\left(||\mathbf{x}-\mathbf{y}||_2^2 + \lambda||\mathbf{H}\mathbf{x}||_{2,1}\right)$$

where $\mathbf{H} = \left[\mathbf{H}_h^T, \mathbf{H}_v^T\right]$ computes spatial gradient operators, and

$$||\mathbf{H}\mathbf{x}||_{2,1} \triangleq \sum_{i=1}^{n} \sqrt{(\mathbf{H}_h\mathbf{x})_i^2 + (\mathbf{H}_v\mathbf{x})_i^2}$$



- This regularizer penalizes edges in images, but not as much as Tikhonov regularization: promotes piecewise smooth solutions

  But the cost function is no longer differentiable...

Handling more complex objectives

Proximal gradient only applies to minimize

$$f(\mathbf{x}) + g(\mathbf{x})$$

where f is differentiable and g is "proximable".

What happens if we want to minimize functions with several nondifferentiable terms (g and h)?

$$f(\mathbf{x}) + g(\mathbf{x}) + h(\mathbf{x})$$

or with linear operators involved?

$$f(\mathbf{x}) + g(\mathbf{Hx})$$

where $\mathbf{H} \in \mathbb{R}^{m \times n}$ is a matrix?

$\rightarrow$ use of more general "proximal splitting" schemes...

Example: Condat-Vu hybrid gradient scheme

The objective here is to minimize the sum of three convex functions, one only (f) being differentiable and one involving a linear operator $\mathbf{H}$.

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x}) + h(\mathbf{Hx})$$

Condat-Vu algorithm

$$\mathbf{x}_{k+1} = \mathbf{prox}_{\rho g}(\mathbf{x}_k - \rho(\nabla f(\mathbf{x}_k) + \mathbf{H}^\top \mathbf{s}_k))$$
$$\mathbf{s}_{k+1} = \mathbf{prox}_{\gamma h^*}(\mathbf{s}_k + \gamma \mathbf{H}(2\mathbf{x}_{k+1} - \mathbf{x}_k)).$$

starting from $\mathbf{x}_0 \in \mathbb{R}^n$ and $\mathbf{s}_0 \in \mathbb{R}^m$ and ρ and γ are well chosen to guarantee convergence. h^* is a function closely related to h (convex conjugate).

[Condat, 2013]. A primal–dual splitting method for convex optim. involving Lipschitzian, proximable and linear composite terms. Jour. of optim. theory. and app.

Outline

1. Optimization and machine learning
   - The very basics of Deep Learning
   - (Nondifferentiable) convex optimization

2. Bilevel optimization and meta learning
   - Motivation and examples
   - Dealing with bilevel problems
     - Algorithm Unrolling
     - Implicit Differentiation
     - Learning Solvers

Table of Contents

1 Optimization and machine learning
- The very basics of Deep Learning
- (Nondifferentiable) convex optimization

2 Bilevel optimization and meta learning
- Motivation and examples
- Dealing with bilevel problems
  - Algorithm Unrolling
  - Implicit Differentiation
  - Learning Solvers

Inverse problems: the example of image denoising

Let $\mathbf{x} \in \mathbb{R}^n$ be an image with n pixels, that has been corrupted by i.i.d. Gaussian noise $\mathbf{b} \in \mathbb{R}^n$, such that the observed image $\mathbf{y}$ is:

$$\mathbf{y} = \mathbf{x} + \mathbf{b}$$

Typical denoising formulation:

$$\arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x} - \mathbf{y}||_2^2 + \lambda \mathcal{R}(\mathbf{x})$$

with $\mathcal{R}$ a function that penalizes images that are not (piecewise) smooth.

Solving variational inverse problems

Typically, optimization algorithms are used to solve these problems:

- In many cases they are convex
- They rarely have closed form solutions: *iterative algorithms*
- The objective function may or may not be differentiable

Problems:

- How do we choose λ? More generally, how do we choose the best $\mathcal{R}$?
- Once an algorithm has been selected, how do we tune its parameters (e.g. step size)

Hyperparameter Learning

In a "supervised setting", we can try to learn the best values of this parameter for a given input $\mathbf{y}$.

$$\begin{aligned} &\arg\min_{\lambda} ||\mathbf{x}(\lambda) - \mathbf{x}^{GT}||_2^2 \\ &\text{s.t. } \mathbf{x}(\lambda) = \arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + \lambda\mathcal{R}(\mathbf{x}) \end{aligned}$$

where $\mathbf{x}^*$ is a ground truth image (noiseless).

By doing this for a number of supervised examples, one can finally build a dataset made of input/output pairs : $\{\mathbf{y}_n, \hat{\lambda}_n\}_{n=1,\dots,N}$.

Finally, we can learn a regressor between an input image $\mathbf{y}$ and the associated regularization parameter λ, e.g via a neural network.

Regularization Learning

More ambitiously, one may want to learn a part of (or even the whole) regularizer $\mathcal{R}_\theta$:

$$\arg\min_{\theta} \sum_{i=1}^{N} ||\mathbf{x}_n(\theta) - \mathbf{x}_n^{GT}||_2^2$$

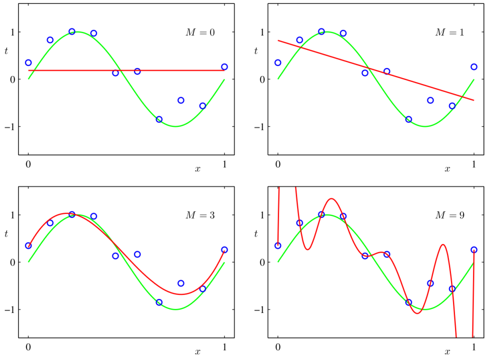
$$\text{s.t. } \forall n, \; \mathbf{x}_n(\theta) = \arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{x}_n - \mathbf{y}_n||^2 + \mathcal{R}_\theta(\mathbf{x}_n)$$

Weight decay in supervised learning

Example: polynomial regression:

$$\hat{y} = \mathbf{w}^T \phi(x)$$

where $\phi(x) \in \mathbb{R}^{M+1}$ contains all the powers of x from x^0 to x^M.



$M = 9$ overfits because it has to introduce very large derivatives to go through all the points (and hence large weights).

Weight decay

Here too there is a need to regularize. Weights with smaller magnitude overfit less:

$$\arg\min_{\mathbf{w}} \sum_{n=1}^{N} \mathcal{L}(y_n, \mathbf{w}^T \mathbf{x}) + \lambda ||\mathbf{w}||^2$$

Problem: How can we tune the regularization parameter λ?

Hyperparameter Learning

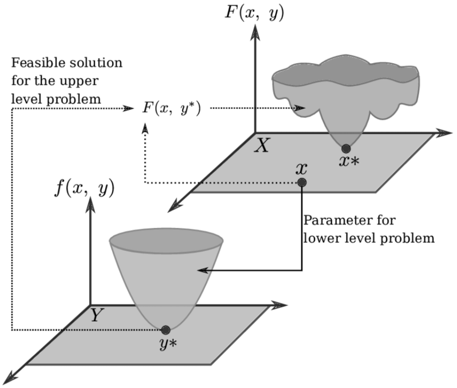
To avoid overfitting, tune the hyperparameter λ so that it performs well on a validation set:

$$\begin{aligned}&\arg\min_{\lambda} \sum_{n \in \mathcal{D}_{val}} \mathcal{L}(y_n, \mathbf{w}^T \mathbf{x}_n)\\ &\text{s.t. } \mathbf{w}(\lambda) = \arg\min_{\mathbf{w}} \sum_{n \in \mathcal{D}_{train}} \mathcal{L}(y_n, \mathbf{w}^T \mathbf{x}_n) + \lambda ||\mathbf{w}||^2\end{aligned}$$

Bilevel optimization problems

Both problems are hyperparameterization problems: simply formulating them involves the solution to another problem

- The upper level problem involves the hyperparameter to learn.
- The lower level solution becomes the constraint set for the upper level



$F(x,\ y)$
Feasible solution for the upper level problem
$F(x,\ y^*)$
X
x
$x*$
$f(x,\ y)$
Parameter for lower level problem
Y
$y*$

$$\arg\min_{\mathbf{x}} F(\mathbf{x},\mathbf{y})$$
$$\text{s.t. } \mathbf{y}(\mathbf{x}) = \arg\min_{\mathbf{y}} f(\mathbf{x},\mathbf{y})$$

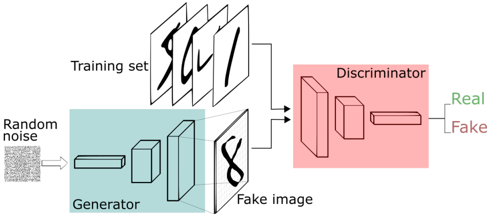
More examples

Min-Max problems (zero-sum games, e.g. training GANs)

$$\arg\min_{\mathbf{x}} F(\mathbf{x})$$
$$\text{s.t. } F(\mathbf{x}) = \max_{\mathbf{y}} F(\mathbf{x}, \mathbf{y})$$

For GANs, we need to train a generator and a discriminator with parameters $\mathbf{x}$ and $\mathbf{y}$, respectively:

$$F(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\mathbf{u} \sim p_{data}}[\log(D_{\mathbf{y}}(\mathbf{u}))] + \mathbb{E}_{\mathbf{z} \sim p_{latent}}[\log(1 - D_{\mathbf{y}}(G_{\mathbf{x}}(\mathbf{z})))]$$



D must output values close to $1/0$ for real/fake data $\rightarrow$ max
G must produce data so the discriminator outputs values close to 1 $\rightarrow$ min

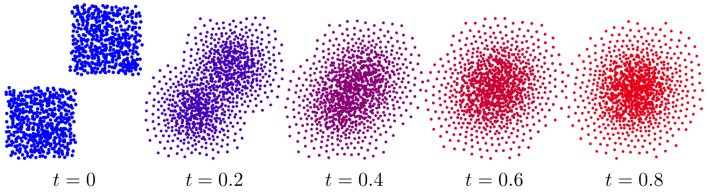
More examples

Optimal transport (see lecture in two weeks)

$$\underset{\mu}{\operatorname{argmin}}\ F(\mu) + W_2^2(\mu, \mu_0)$$

Where μ and μ_0 are probability distributions, and W_2^2 is the (squared L_2) Wasserstein distance between probability distributions (computing it requires solving an optimization problem)

Depending on the choice of F, useful to obtain solutions of some PDE using variational methods...



What about Convexity?

Bilevel optimization algorithms usually have convergence results for:

- Strongly convex lower level f wrt y
- Lipschitz continuous F,f, $\nabla_{\mathbf{x}} f$, $\nabla_{\mathbf{y}} g$ and $\nabla^2_{\mathbf{y}} F$ (differentiable case)

i.e. for *very* well behaved functions on both levels...

Many cases of practical interest do not fall into this framework.

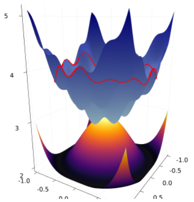
No guarantees at all anytime a NN is used in the lower level cost function...

Which is what we want to do in practice...

Chen, T., et al. (2021). Closing the gap: Tighter analysis of alternating stochastic gradient methods for bilevel problems. Neurips 2021

Several critical points on inner level

- several global minima mean several branches for the upper level
- worse: local minima are actually not feasible points for the upper level



Case where the upper level $F(\mathbf{x})$ does not directly depend on $\mathbf{y}$ and lower level (not convex in $\mathbf{y}$) does not depend on $\mathbf{x}$.

In this case the nonconvexity of f wrt $\mathbf{y}$ is not too bad since the lower level can be turned into a constraint set

Q: How bad is it not to find the exact solution of the bottom level?

Table of Contents

1 Optimization and machine learning
- The very basics of Deep Learning
- (Nondifferentiable) convex optimization

2 Bilevel optimization and meta learning
- Motivation and examples
- Dealing with bilevel problems
  - Algorithm Unrolling
  - Implicit Differentiation
  - Learning Solvers

Unrolled algorithms

Idea: replace the solution of the inner optimization problem by either:

- A differentiable truncated iterative solution
- A close approximation of an iterative solution, with trainable parameters

$$\arg\min_{\mathbf{x}} F(\mathbf{x}, \mathbf{y})$$
$$\text{s.t. } \mathbf{y}(\mathbf{x}) = \arg\min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$$

Let us assume that $\mathbf{y}(\mathbf{x})$ can be obtained via an iterative algorithm

$$\mathbf{y}_{k+1}(\mathbf{x}) = \mathbf{y}_k(\mathbf{x}) + g_k(\mathbf{y}_k(\mathbf{x}))$$

Then just code that algorithm in an autodiff framework for a finite number of iterations K, and differentiate the result (or a function of it) wrt $\mathbf{x}$!

Unrolling algorithms

In practice, we solve

$$\arg\min_{\mathbf{x}} F(\mathbf{x}, \mathbf{y})$$
$$\text{s.t. } \mathbf{y}(\mathbf{x}) = \mathbf{y}_K(\mathbf{x})$$

```
Data: $\mathbf{x}_0$,$\mathbf{y}_0$,$N \geq 1$,$K \geq 1$,lr
Result: $\mathbf{x}_N$
$i \leftarrow 0$, $k \leftarrow 0$
while $i < N$ do
    while $k < K$ do
        $\mathbf{y}_{k+1}(\mathbf{x}_i) = \mathbf{y}_k(\mathbf{x}_i) + g_k(\mathbf{x}_i, \mathbf{y}_k(\mathbf{x}_i))$ # Within the computation graph
        $k \leftarrow k + 1$
    end
    $\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}_i, \mathbf{y}_K(\mathbf{x}_i)) = \texttt{autograd}_{\mathbf{x}}(F, (\mathbf{x}_i, \mathbf{y}_K(\mathbf{x}_i))$ # This is simple to obtain via AD
     $\mathbf{x}_{i+1} = \mathbf{x}_i - \texttt{lr}\ \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}_i, \mathbf{y}_K(\mathbf{x}_i))$ # For a simple gradient descent on the upper
     level
    $i \leftarrow i + 1$
end
```

Image denoising example

Optimizing regularization parameters, knowing the solution (supervised learning):

$$\begin{aligned}&\arg\min_{\lambda} ||\mathbf{x}(\lambda) - \mathbf{x}^{GT}||_2^2\\ &\text{s.t. } \mathbf{x}(\lambda) = \arg\min_{\mathbf{x}} \left(f(\mathbf{x}) + \lambda\mathcal{R}(\mathbf{x})\right)\end{aligned}$$

There are a large class of algorithms available to solve the lower level problem for a given λ

Idea: replace the nested optimization problem by a differentiable "unrolling" of an iterative algorithm

$\rightarrow \lambda$ can be optimized via automatic differentiation

[Afkham et al. 2021]. Learning Regularization Parameters of Inverse Problems via Deep Neural Networks. arXiv preprint arXiv:2104.06594.

Guarantees in the differentiable convex case

We would like to recover the true derivative of the solution to the lower level when $k \to +\infty$.

$$\begin{tikzcd}
\mathbf{x}_k(\lambda) \arrow[r, "\frac{\partial}{\partial \lambda}"] \arrow[d, "k \to +\infty"] & \frac{\partial \mathbf{x}_k}{\partial \lambda} \arrow[d, "k \to +\infty"] \\
\mathbf{x}^*(\lambda) \arrow[r, "\frac{\partial}{\partial \lambda}"] & ?
\end{tikzcd}$$

Things will be okay when the lower level cost function is

- convex $\to$ unicity of the lower level solution $\mathbf{x}^*(\lambda)$
- differentiable $\to$ the derivative $\frac{\partial \mathbf{x}_k}{\partial \lambda}$ is well defined

K small: bias because the solution of the lower level is not exact
K large: requires more memory and vanishing gradient issues can arise.

Nondifferentiable algorithms?

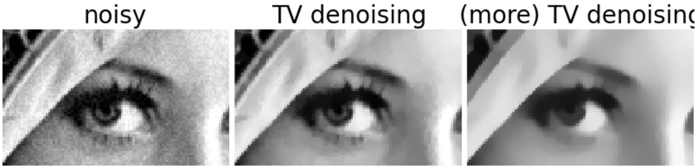
Frequently, we differentiate through nonsmooth algorithms.

- Inverse problems often involve sparsity inducing norms

$$\arg\min_{\mathbf{x}} \frac{1}{2}\left(||\mathbf{x}-\mathbf{y}||_2^2 + \lambda||\mathbf{H}\mathbf{x}||_{2,1}\right)$$

  where $\mathbf{H} = \left[\mathbf{H}_h^T, \mathbf{H}_v^T\right]$ computes spatial gradient operators, and

$$||\mathbf{H}\mathbf{x}||_{2,1} \triangleq \sum_{i=1}^{n} \sqrt{(\mathbf{H}_h\mathbf{x})_i^2 + (\mathbf{H}_v\mathbf{x})_i^2}$$



- Even convex constraints can be a problem, e.g. for a projected gradient descent

Guarantees in the nondifferentiable convex case

Recent results for mild conditions on the nondifferentiable iterations (for convex functions):

- A notion of generalized derivative of $\mathbf{x}_k$ is well defined for almost all λ
- When $k \to \infty$, it converges to the *true* derivative of the unique fixed point of the iterations wrt λ.

$\rightarrow$ In practice we can safely differentiate through most proximal algorithms for convex objectives...

LAB SESSION: optimizing hyperparameters for image denoising, and tuning them automatically.

[Bolte et al. 2022]. Automatic differentiation of nonsmooth iterative algorithms. arXiv preprint arXiv:2206.00457.

Implicit differentiation

Is there a way to obtain the derivative of the solution $\mathbf{x}^*(\lambda)$ *directly*?

Idea: replace the lower level by its optimality conditions (fixed point equation).

Example of weight decay in logistic regression:

$$\begin{aligned}&\arg\min_{\lambda\in\mathbb{R}^m} F(\mathbf{w}) := \sum_{n\in\mathcal{D}_{val}} CE_n(\mathbf{w})\\ &\text{s.t. } \mathbf{w}(\boldsymbol{\lambda}) = \arg\min_{\mathbf{w}\in\mathbb{R}^m} f(\mathbf{w},\boldsymbol{\lambda}) := \sum_{n\in\mathcal{D}_{train}} CE_n(\mathbf{w}) + \boldsymbol{\lambda}^T(\mathbf{w}\odot\mathbf{w})\end{aligned}$$

can be rewritten as

$$\begin{aligned}&\arg\min_{\boldsymbol{\lambda}} F(\mathbf{w})\\ &\text{s.t. } g(\boldsymbol{\lambda},\mathbf{w}(\boldsymbol{\lambda})) = 0\end{aligned}$$

with $g(\boldsymbol{\lambda},\mathbf{w}(\boldsymbol{\lambda})) = \nabla_{\mathbf{w}} f(\mathbf{w},\boldsymbol{\lambda})$ (optimality condition on the lower problem)

Implicit Differentiation

$g : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ defines an *implicit* function (m is the number of parameters of the logistic regression). The solution of the lower level problem then verifies:

$$g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) = 0$$

Using the chain rule:

$$\frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}} + \frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}} = 0$$

This means we can obtain the derivative of the *optimum* of the lower level problem wrt the *upper level* variable $\boldsymbol{\lambda}$ as:

$$\underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\mathbb{R}^{m\times m}} = - \underbrace{\left(\frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}} \right)^{-1}}_{\mathbb{R}^{m\times m}} \underbrace{\frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\mathbb{R}^{m\times m}}$$

We just need to solve a linear system!

Implicit Differentiation

In this case,

$$\frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}} = \nabla^2_{\mathbf{w}} f(\mathbf{w}^*(\boldsymbol{\lambda}), \boldsymbol{\lambda})$$

is the Hessian of the lower level cost function, and here

$$\frac{\partial g(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}} = 2\mathrm{diag}(\mathbf{w})$$

$\rightarrow$ Both can be computed via autodiff.

We can obtain the derivative of the fixed point without differentiating through the computation of $\mathbf{w}^*(\boldsymbol{\lambda})$!

Any algorithm can be used to compute the fixed point *outside* the computation graph. In particular, no need to unroll an iterative algorithm

$\rightarrow$ saves memory and no vanishing gradient issues

[Blondel et al. (2021)]. Efficient and modular implicit differentiation. arXiv preprint arXiv:2105.15183.

Implicit differentiation

We want to perform a gradient descent on the upper level problem:

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k - \alpha \nabla_{\boldsymbol{\lambda}}(F(\mathbf{w}(\boldsymbol{\lambda})))$$

The gradient can be computed via the chain rule:

$$\frac{\partial F}{\partial \boldsymbol{\lambda}} = \frac{\partial F}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$$

Again, $\frac{\partial F}{\partial \boldsymbol{\lambda}}$ can be computed via autodiff.

Drawback: This is only realistic for a relatively small number of parameters m in the lower level.

LAB SESSION: optimizing hyperparameters for logistic regression via implicit differentiation

Algorithm

Implicit formulation (here to simplify no direct dependence on $\mathbf{x}$ in F):

$$\arg\min_{\mathbf{x}} F(\mathbf{y}^*(\mathbf{x}))$$
$$\text{s.t. } g(\mathbf{x}, \mathbf{y}^*(\mathbf{x})) = 0$$

```
Data: $\mathbf{x}_0$,$\mathbf{y}_0$,$N \geq 1$,lr
Result: $\mathbf{x}_N$
$i \leftarrow 0$, $k \leftarrow 0$
while $i < N$ do
    while $\mathbf{y}_k$ has not converged to $\mathbf{y}^*(\mathbf{x}_i)$ do
        $\mathbf{y}_{k+1}(\mathbf{x}_i) = \mathbf{y}_k(\mathbf{x}_i) + g_k(\mathbf{y}_k(\mathbf{x}_i), \mathbf{x}_i)$ # Outside the computation graph
        $k \leftarrow k + 1$
    end
    $\frac{\partial g(\mathbf{x},\mathbf{y}^*(\mathbf{x}_i))}{\partial \mathbf{y}} = \texttt{autograd}_{\mathbf{y}}(g, \mathbf{y}^*(\mathbf{x}_i))$ # This is simple to obtain via AD
    $\frac{\partial g(\mathbf{x},\mathbf{y}^*(\mathbf{x}_i))}{\partial \mathbf{x}} = \texttt{autograd}_{\mathbf{x}}(g, \mathbf{x}_i)$
    $\frac{\partial F(\mathbf{y}^*(\mathbf{x}_i))}{\partial \mathbf{y}} = \texttt{autograd}_{\mathbf{y}}(F, \mathbf{y}^*(\mathbf{x}_i))$
    $\frac{\partial F}{\partial \mathbf{x}} = \frac{\partial F}{\partial \mathbf{y}} \frac{\partial \mathbf{y}^*(\mathbf{x}_i)}{\partial \mathbf{x}} = -\frac{\partial F}{\partial \mathbf{y}} \left(\frac{\partial g(\mathbf{x},\mathbf{y}^*(\mathbf{x}_i))}{\partial \mathbf{y}}\right)^{-1} \frac{\partial g(\mathbf{x},\mathbf{y}^*(\mathbf{x}))}{\partial \mathbf{x}}$
    $\mathbf{x}_{i+1} = \mathbf{x}_i - \texttt{lr}\ \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}_i)$ # For a simple gradient descent on the upper level
    $i \leftarrow i + 1$
end
```

Implicit layers

In particular, one may want to include layers defined via (uniquely defined) fixed points to classical DL models:

$$\mathbf{x} \leftarrow \{\mathbf{x}, g(\mathbf{x}) = \mathbf{x}\}$$

For example, we can compute networks with an "infinite" number of layers:

$$\begin{aligned}\mathbf{x} &\leftarrow f_\theta(\mathbf{x})\\ \mathbf{x} &\leftarrow \{\mathbf{x}, \mathbf{x} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})\}\\ \mathbf{x} &\leftarrow g_\phi(\mathbf{x})\end{aligned}$$

This is effectively applying an infinite number of times the operation $\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$. We can integrate such layers in a differentiable algorithm by defining a custom backward method for them (as before).

$\rightarrow$ layers can now be "infinite neural nets", or even solutions to (convex) optimization problems... with trainable parameters!

`https://implicit-layers-tutorial.org/`

Learning faster optimization algorithms

$\mathcal{L}_1$ regularized least squares:

$$\arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2 + \lambda||\mathbf{x}||_1$$

An iteration of a proximal gradient descent (called ISTA in this case) writes:

$$\mathbf{x}_{k+1} = \mathbf{prox}_{\lambda/\rho||.||_1}(\mathbf{x}_k - \rho\mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}))$$

Idea of "LISTA" (Learned ISTA), replace this with a close version with trainable parameters:

$$\mathbf{x}_{k+1} = \mathbf{prox}_{\theta_k||.||_1}(\mathbf{x}_k - \mathbf{W}_1^k\mathbf{x} - \mathbf{W}_2^k\mathbf{b})$$

and optimize the MSE between real solutions and LISTA iterates. The unrolled algorithm can be implemented as a simple RNN.

[Gregor & LeCun, 2010]. Learning fast approximations of sparse coding. ICML 2010

LISTA bilevel formulation

Here we want to learn parameters of the lower level algorithm in order to make it faster on new problems $\rightarrow$ learning to optimize!
This problem can also be framed as a bilevel optimization problem:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} ||\mathbf{x}_i(\boldsymbol{\theta}) - \mathbf{x}_i^{GT}||_2^2$$
$$\text{s.t. } \mathbf{x}_i(\boldsymbol{\theta}) = g_i(\mathbf{x}, \boldsymbol{\theta})$$

- $\mathbf{x}_i^{GT}$ are examples of solutions to the problem (can be synthetic)
- $\boldsymbol{\theta}$ contains all the trainable parameters
- the lower level is a fixed point equation encoding the optimality conditions of the sparse regression problem.

$\rightarrow$ Unrolling is a way to compute an *approximate* solution of an optimization problem in a differentiable way.

Learning to optimize (model free)

Objective: Leverage training data to learn optimization algorithms, or to improve the performance of existing ones.

Some classes of problems are such that generating training pairs of possible inputs/solution is doable.

[Chen et al. 2021]. Learning to optimize: A primer and a benchmark. arXiv preprint arXiv:2103.12828.

Model free approach

Exact line search in a descent algorithm (unconstrained minimization of f):

$$\underset{\rho_k}{\arg\min}\ f(\mathbf{x}_k + \rho_k \mathbf{d}_k)$$

where $\mathbf{d}_k$ is a descent direction, i.e. a vector such that $\langle \mathbf{d}_k, \nabla_f(\mathbf{x}_k)\rangle \leq 0$ (typically $\mathbf{d}_k = -\nabla_f(\mathbf{x}_k)$).

Typically no closed form solution except in simple cases. Also, the best $\mathbf{d}_k$ is a priori unknown.

Model free approaches

Given a distribution of functions to optimize $\mathcal{T}$, find an algorithm of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + g(\mathbf{z}_k, \phi)$$

where g is an update rule, with $\mathbf{z}_k$ the available information, and ϕ trainable parameters. g is optimized such that

$$\arg\min_{\phi} \mathbb{E}_{f \in \mathcal{T}} \left[\sum_{k=1}^{K} w_k f(\mathbf{x}_k) \right]$$

is minimal, for a fixed K and fixed $w_k, k = 1, ..., K$.

A possible choice is $w_K = 1$, and $w_1, ..., w_{K-1} = 0$.

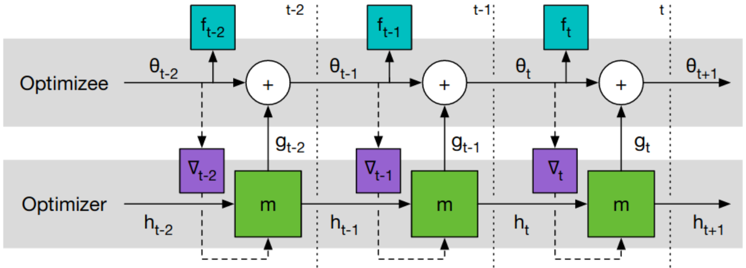
[Andrychowicz et al., 2016], Learning to learn by gradient descent by gradient descent, Neurips 2016

Model-free approaches

$\{\mathbf{x}_k\}_{k=1,\dots K}$ being sequential, a natural choice is to use LSTMs.

$\{\mathbf{z}_k\}_{k=1,\dots K}$ incorporates $\{\mathbf{x}_k\}_{k=1,\dots K}$, and possibly $\{\nabla f(\mathbf{x}_k)\}_{k=1,\dots K}$, computed by automatic differentiation.

- With current values of ϕ, sequences $\{\mathbf{x}_k\}_{k=1,\dots K}$ and $\{\mathbf{z}_k\}_{k=1,\dots K}$ are computed
- They are passed through an LSTM that produces a sequence $g(\mathbf{z}_k, \phi)$
- parameters ϕ of the LSTMs are updated (using any off-the-shelf optimizer) to decrease the the cost function



Model free approaches

Advantages:

- Can be specialized to certain types of problems
- Gets better results than conventional optimizers in several cases

Yet another framework: optimal control

Two questions in image denoising:

- How to learn a regularizer?
- If the lower level is unrolled, how many iterations to use?

$$\arg\min_{\theta} \sum_{i=1}^{N} ||\mathbf{x}(\theta)_i - \mathbf{x}_i^*||_2^2$$

$$\text{s.t. } \mathbf{x}(\theta)_i = \arg\min_{\mathbf{x}_i} \frac{1}{2}||\mathbf{x} - \mathbf{y}_i||^2 + \mathcal{R}_\theta(\mathbf{x}) = f_\theta(\mathbf{x})$$

can be rewritten in continuous time using the *gradient flow* of f_θ

$$\arg\min_{\theta, T} \sum_{i=1}^{N} ||\mathbf{x}(\theta)_{T\,i} - \mathbf{x}_i^*||_2^2$$

$$\text{s.t. } \frac{d\mathbf{x}(\theta)}{dt} = -\nabla f_\theta(\mathbf{x}(t)),\ \{\mathbf{x}_i(0) = \mathbf{y}_i\}_{i=1,\dots,N}, t \in [0, T]$$

[Efflandet et al. (2020)] Variational networks: An optimal control approach to early stopping variational methods for image restoration. JMIV

Optimal control

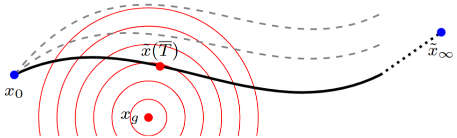


Fig. 3 Schematic drawing of optimal trajectory (black curve) as well as suboptimal trajectories (gray dashed curves) emanating from x_0 with ground truth x_g, optimal restored image $\tilde{x}(\overline{T})$, sink/stable node $\tilde{x}_\infty$ and energy isolines (red concentric circles) (Color figure online)

Optimal control theory (Pontryagin's maximum principle) gives necessary optimality conditions.
Those can be discretized to obtain a practical algorithm:

1. Integrate the dynamics of $\mathbf{x}$ and other variables with current values of T and θ
2. Optimize the cost function wrt T and θ for current trajectory of $\mathbf{x}$
3. Repeat steps 1 and 2 until convergence

Actually not that different from continuous time backpropagation...

Summary

For hyperparameterization, multiple options are possible

- Differentiable unrolling
- Implicit differentiation
- Optimal control

No guarantees when the lower level is not convex in $\mathbf{y}$:

- In practice, we want to control the parameters of neural networks...
- But multiple global and local minima make us lose all guarantees
- In practice, deep unrolling works for a small number of iterations
- Implicit differentiation is possible for small problems and when we can compute the solution of the lower level with good precision
- It is hard to assess if a suboptimal inner solution is "good enough"