T4: Gradient descent optimization algorithms for training neural networks

Pablo Arias Martínez - ENS Paris-Saclay, UPF
pablo.arias@upf.edu

October 19, 2021

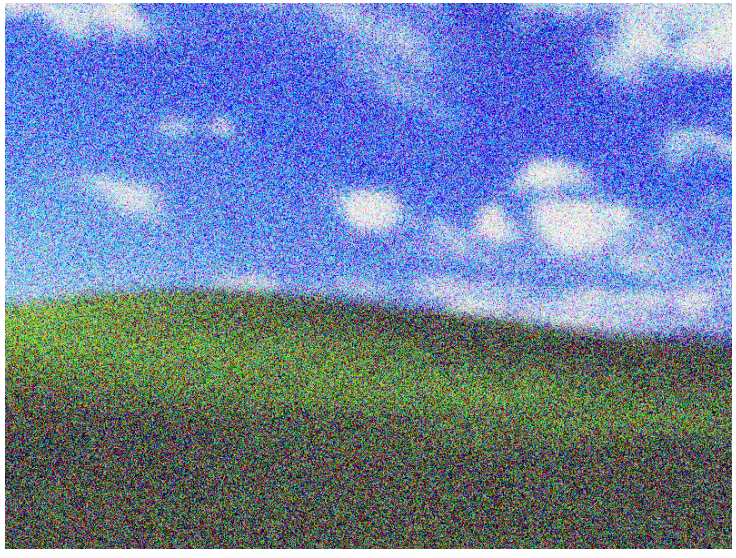
Optimization and inference techniques for Computer Vision

Previously on . . .

Model image restoration problems as optimization problems

degradation operator

minimize energy



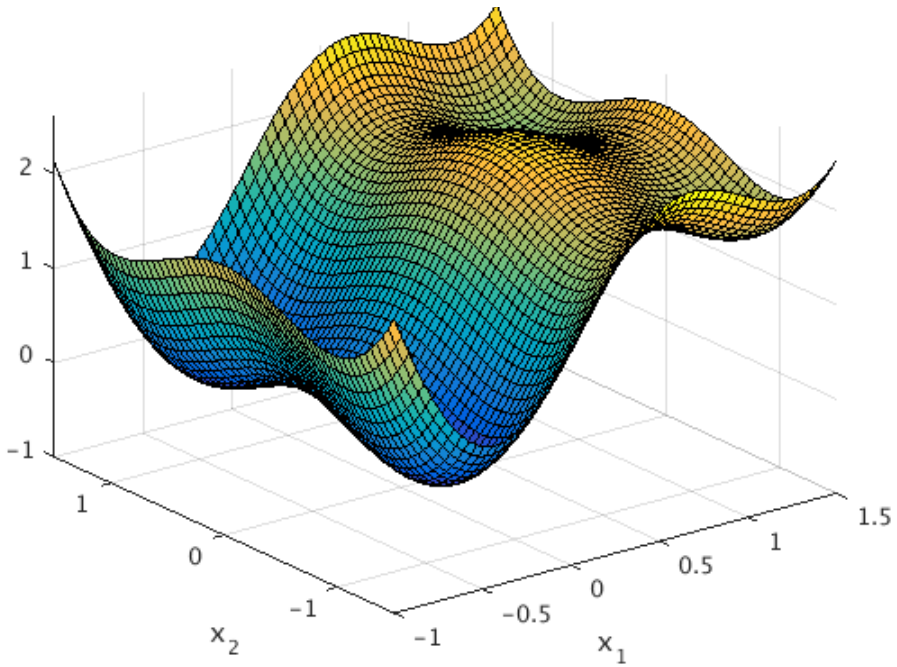
u: original image.

f: degraded image.

Goal: estimate the (unknown) clean image given the degraded observation

Model image restoration problems as optimization problems

Approach: design a suitable "energy" such that its minimizer(s) are good estimations.

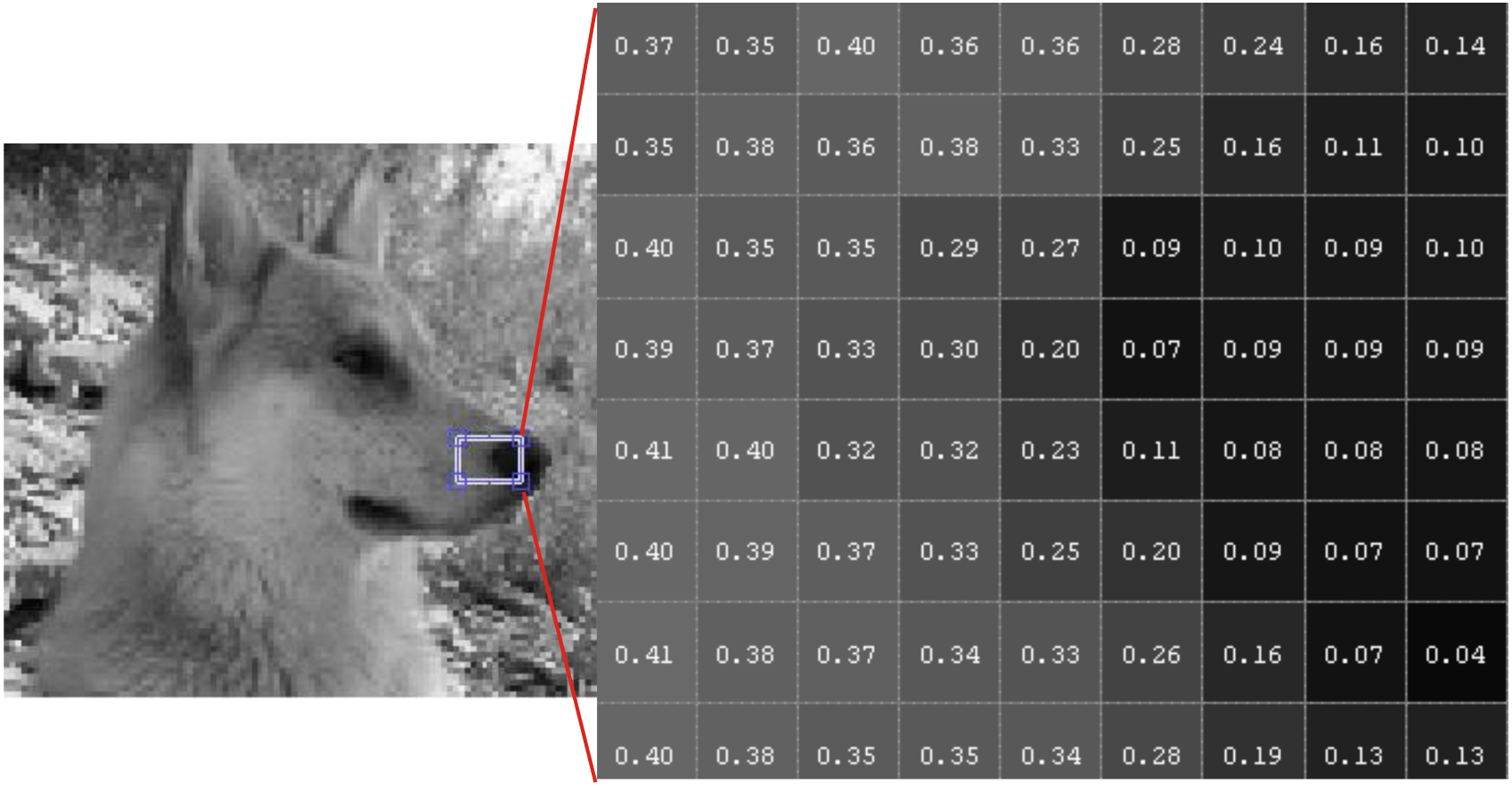


Each point in the domain of our function corresponds to an image!
$\rightarrow$ large scale optimization problem, with 10^4 to 10^6 variables!

Modelling images as matrices (or 3D tensors) . . .

We can model scalar images as

- $H \times W$ matrices (over a 2D regular grid) or
- vectors with HW components (vectorized, "flattened")



Modelling images as matrices (or 3D tensors) . . .

Color is represented by a **vector (of 3 components)** at each pixel. We can model color images as

- $H \times W \times 3$ 3D tensors (over a 3D regular grid) or
- vectors with $3HW$ components (vectorized, "flattened")
- 3 $H \times W$ 2D tensors (over a 2D regular grid)

R:193 G:163 B:113	R:169 G:145 B:109	R:152 G:129 B: 97	R:129 G:120 B: 79	R:111 G: 91 B: 84	R:144 G: 20 B: 70	R: 91 G: 69 B: 71	R: 80 G: 74 B: 60	R: 84 G: 72 B: 58	R:106 G: 77 B: 71
R:192 G:160 B:122	R:171 G:142 B:112	R:151 G:130 B:109	R:137 G:132 B: 74	R:120 G: 90 B: 82	R:146 G: 25 B: 76	R: 96 G: 73 B: 83	R: 77 G: 78 B: 64	R: 89 G: 73 B: 60	R:113 G: 74 B: 79
R:190 G:164 B:115	R:172 G:147 B:116	R:150 G:132 B:108	R:143 G:139 B: 78	R:126 G:100 B: 87	R:137 G: 53 B: 79	R: 99 G: 87 B: 89	R: 83 G: 79 B: 68	R: 88 G: 76 B: 76	R:115 G: 82 B: 89
R:189 G:166 B:116	R:173 G:146 B:119	R:160 G:134 B:111	R:147 G:135 B: 87	R:119 G:103 B: 78	R:139 G: 56 B: 84	R:108 G: 89 B: 91	R: 88 G: 88 B: 88	R: 87 G: 92 B: 95	R:111 G: 83 B: 97
R:185 G:158 B:115	R:171 G:147 B:123	R:155 G:134 B:117	R:147 G:134 B: 90	R:124 G:111 B: 79	R:140 G: 57 B: 85	R:125 G: 95 B: 97	R:119 G:107 B: 91	R:124 G:104 B:116	R:141 G: 95 B:108
R:186 G:157 B:125	R:172 G:148 B:120	R:162 G:133 B:125	R:154 G:137 B: 91	R:125 G:111 B: 74	R:134 G: 59 B: 82	R:112 G: 97 B:100	R: 93 G: 94 B: 96	R:101 G: 91 B: 90	R:125 G: 96 B: 92
R:179 G:154 B:124	R:172 G:149 B:118	R:166 G:139 B:120	R:157 G:141 B: 89	R:130 G:113 B: 70	R:134 G: 59 B: 82	R: 99 G:106 B: 99	R: 74 G: 97 B: 79	R: 78 G: 76 B: 81	R:114 G: 84 B: 84
R:180 G:150 B:122	R:174 G:150 B:124	R:160 G:144 B:108	R:153 G:140 B: 88	R:137 G:115 B: 74	R:132 G: 57 B: 90	R: 91 G:108 B:100	R: 73 G: 96 B: 86	R: 85 G: 75 B: 83	R:128 G: 90 B: 81

. . . and image restoration problems as energy minimization problems in $\mathbb{R}^n$

Given the degraded image f, we describe the characteristics of the (unknown) restored image u:

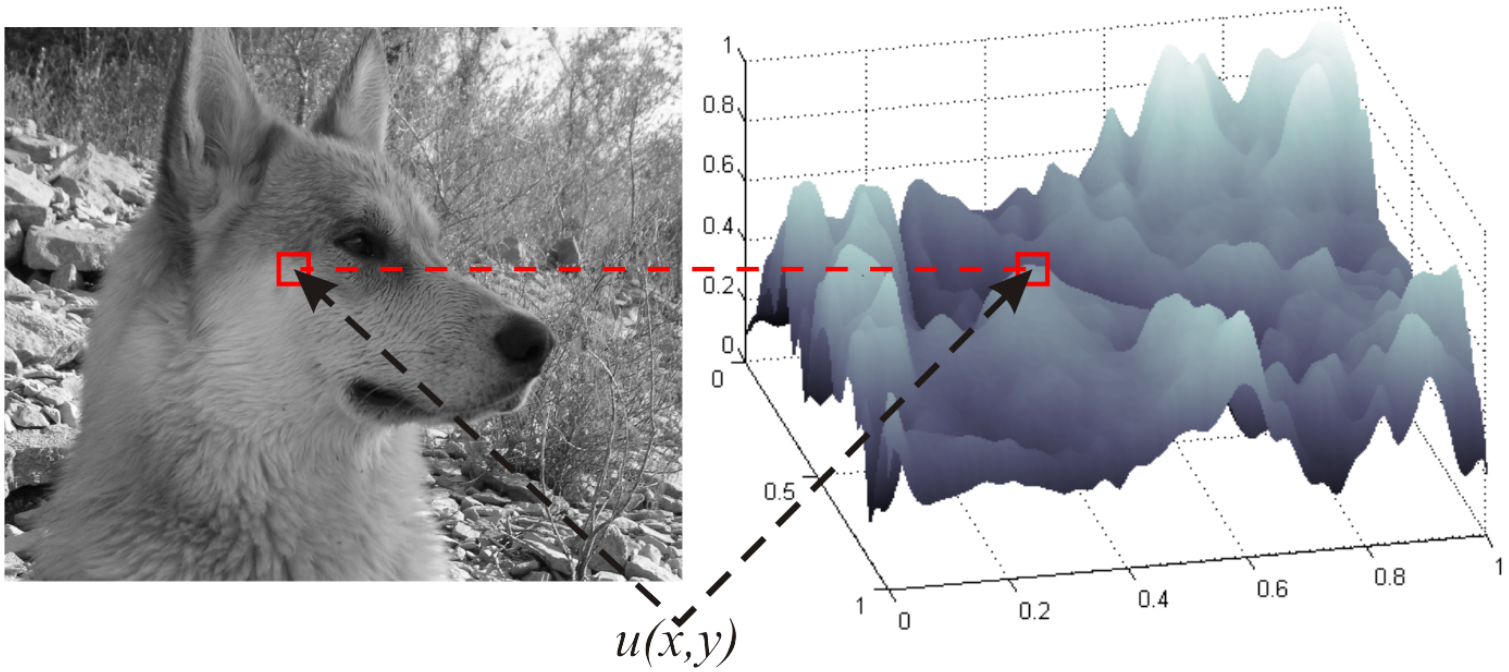
1. No noise: we assume that noise-less images are "smooth".
2. Compatible with degraded image f: difference between u and f is small.

$$\textbf{Energy function:} \quad J(u; f) = \underbrace{\frac{1}{2}\|\nabla^h u\|^2}_{\text{1. regularity of } u} + \underbrace{\frac{\lambda}{2}\|u - f\|^2}_{\text{2. data fidelity}}$$

- $f \in \mathbb{R}^{H \times W}$, **known**, the **given** noisy image.
- $u \in \mathbb{R}^{H \times W}$, **unknown**, the restored image.
- λ as trade-off parameter
- Find $u^* = \operatorname{argmin}_{u \in W} J(u; f)$, where $W \subset \mathbb{R}^{H \times W}$.

Modelling images as functions over a continuous domain . . .

For each position (x, y) there is a value $u(x, y)$.



- Image u(x,y) with $u : \Omega \to \mathbb{R}$
- $\Omega \subset \mathbb{R}^2$, $\Omega = [0, 1]$x$[0, 1]$
- $(x, y) \in \Omega$ denotes pixel location

. . . and image restoration problems as variational problems

Given the degraded image f, we describe the characteristics of the (unknown) restored image u:

1. No noise: we assume that noise-less images are "smooth".
2. Compatible with degraded image f: difference between u and f is small.

$$\textbf{Energy functional:} \quad J(u;f) = \underbrace{\frac{1}{2}\int_\Omega |\nabla u|^2 dx}_{\text{1. regularity of } u} + \underbrace{\frac{\lambda}{2}\int_\Omega |u-f|^2 dx}_{\text{2. data fidelity}}$$

- $f : \Omega \to \mathbb{R}$, **known**, the **given** noisy image.
- $u : \Omega \to \mathbb{R}$, **unknown**, the restored image.
- λ as trade-off parameter
- Find $u^* = \operatorname{argmin}_{u \in W} J(u;f)$, where W is a suitable function space.

Continuous vs discrete images

It depends on the problem, on you background, and on your goals.

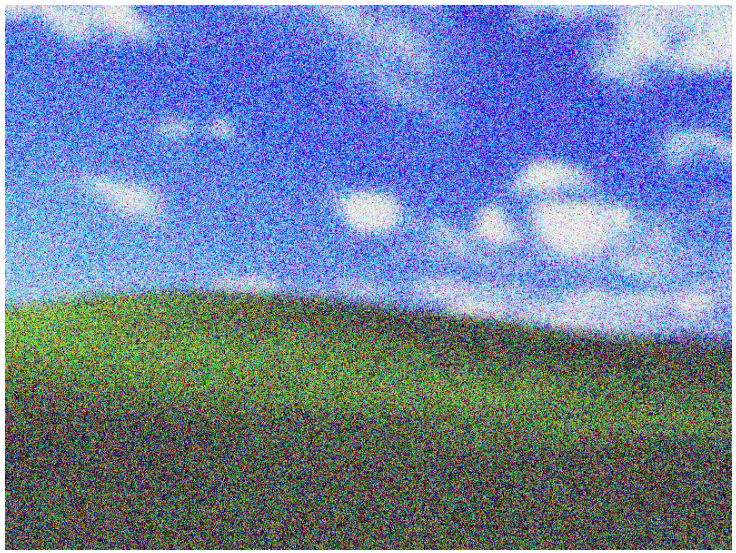
- Continous models allow us to use differential operatos (∇, div, etc.) which have a well understood theory and physical interpretations.
- Some properties or phenomenoms of natural images are often easier to model with continuous images: e.g. discontinuities created by occlusions.
- Discrete version of differential operators exist, often as approximations of continuous ones, but not necessarily. For example, there is a well develop theory of calculus on graphs.
- Others are easier by modelling images as elements in $\mathbb{R}^n$. For example, probabilistic models.

The neural networks approach

Train a NN to restore a degraded image

degradation operator

network neural



y: original image.

x: degraded image.

Goal: train a neural network so that it "learns" estimate the (unknown) clean images from the degraded observations

Supervised learning

We have a set of dataset points with corresponding **labels**

$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, with $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$, (e.g. $\mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \mathbb{R}^t$)

We assume that the data pairs are IID samples of a joint PDF:
$(x_i, y_i) \sim p(x, y)$.

In our case:

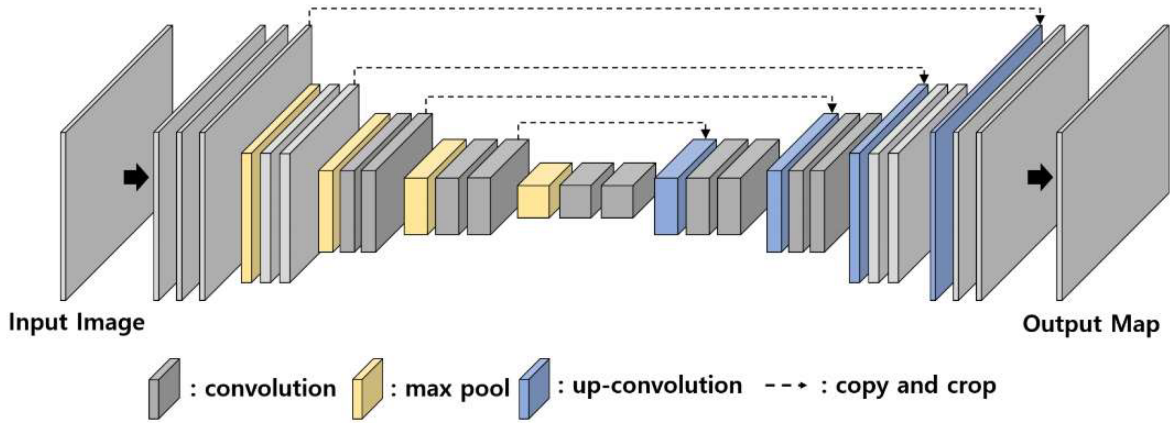
- $y_i \sim p(y)$ (distribution of "normal" images)
- $x_i \sim \mathcal{D}(y_i) = p(x|y_i)$ (distribution of degraded images from y)

We want a function $\mathcal{F}_\theta : \mathcal{X} \to \mathcal{Y}$ such that $\mathcal{F}_\theta(x)$ "$=$" y, for $(x, y) \sim p(x, y)$.

Supervised learning

For $\mathcal{F}_\theta$ we can think of a (convolutional) neural network, where θ are the network parameters (e.g. the kernels in the convolutions).

$\mathcal{F}_\theta =$



We define a **loss function** ℓ which measures the error between $\mathcal{F}_\theta(x)$ and y, and set the **parameters** θ to minimize the expected loss:

$$\underbrace{\mathbb{E}\{\ell(\mathcal{F}_\theta(x), y)\} = \mathcal{R}(\theta)}_{\text{risk}}.$$

Supervised learning

In practice, we can't compute the expected loss, since we don't know $p(x, y)$. Instead we minimize the empirical loss over a large training dataset:

$$\underbrace{\mathcal{R}^{\text{emp}}(\theta) = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathcal{F}_\theta(x_i), y_i)}_{\text{empirical risk}} \to_{m\to\infty} \underbrace{\mathbb{E}\{\ell(\mathcal{F}_\theta(x), y)\} = \mathcal{R}(\theta)}_{\text{risk}}.$$

There are two different stages:

Learning (or training):

$$\theta^* = \text{argmin}_\theta \mathcal{R}^{\text{emp}}(\theta) = \text{argmin}_\theta \frac{1}{m}\sum_{i=1}^{m} \ell(\mathcal{F}_\theta(x_i), y_i)$$

Inference Given a degraded image x, we restore it by applying the network:

$$\hat{y} = \mathcal{F}_{\theta^*}(x)$$

Training is not exactly optimization

We use optimization techinques to train networks, but is good to keep in mind that the **real goal is to minimize the unknown expected loss**, not the empirical loss.

This leads to a few differences between training and traditional optimization, with the objective to have a small **generalization gap** between the empirical loss and the expected loss.

From classical variational methods to training NNs

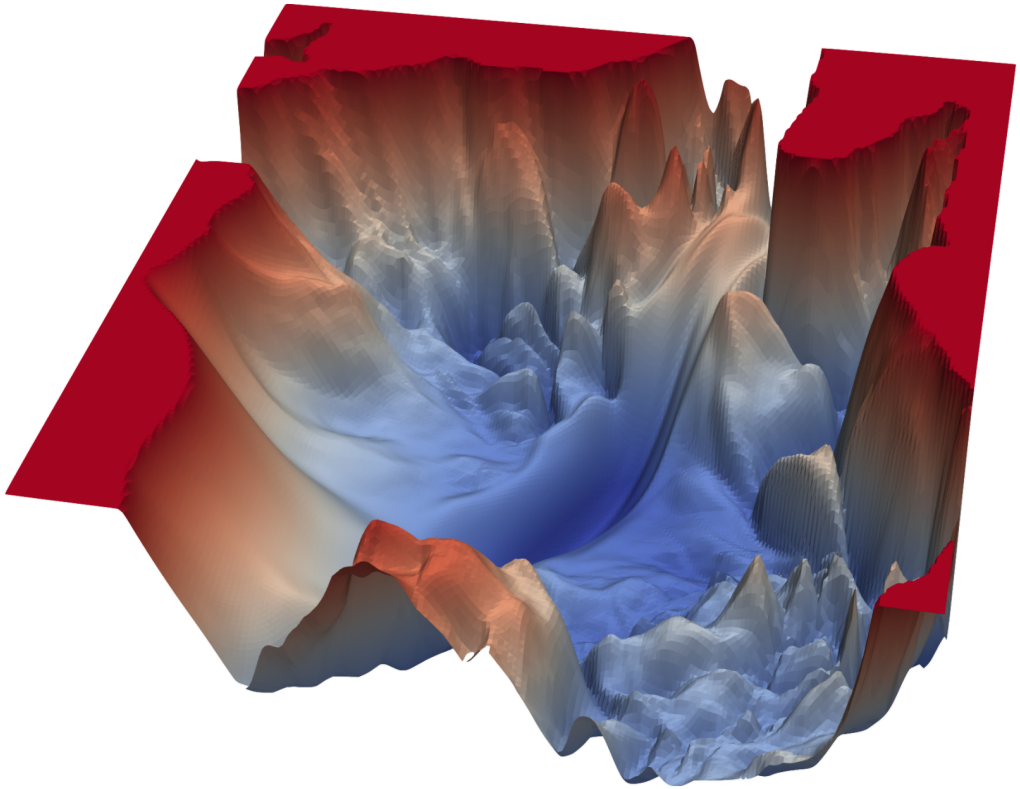
Variational methods:

- For each degraded image, solve a different optimization problem.
- The variable is the image. The number of variables up to 10^6.
- The objective function is easy to evaluate.
- Depending on the restoration problem the energy can be complicated, but sometimes it's possible to find a convex energy (a.k.a an easy optim problem).

Learning-based approach:

- Solve an optimization problem once over a training set ("learn" to invert the degradation).
- The variable is the parameters vector (between 10^4 and –if you work at Google or Facebook– 10^9).
- The energy is **highly non-convex** (and non-differentiable).
- Evaluating the objective function is **very costly (or intractable)**, since it requires computing the loss on the entire training set.

Non-convexity of loss functions



The exact shape is unknown. Modern DL architectures have millions of parameters and we can only visualize representations.

Created with the loss visualization web from the project

`https://www.cs.umd.edu/~tomg/projects/landscapes/`.

Questions so far?

How do we minimize a function defined over millions of variables which we can't even evaluate?

With our old friend, the gradient descent.

Gradient descent techniques for training neural networks

Gradient descent

Gradient descent is an algorithm to minimize a function $\mathcal{R}(\theta)$ by updating the parameters ($\theta \in \mathbb{R}^d$) in the direction $-\nabla_\theta \mathcal{R}(\theta)$:

$$\begin{cases} \text{initialize} & \theta^0 \\ \text{iterate} & \theta^{k+1} = \theta^k - \eta \, \nabla_\theta \mathcal{R}(\theta^k) \quad \text{until convergence.} \end{cases}$$

The step size η determines the size of the steps we take. The machine learning community calls it **learning rate**.

Remember:

- The negative gradient is the direction along which the function decreases faster: the direction of the steepest descent.
- The gradient is orthogonal to the level surface.

Gradient descent

Gradient descent demo.

Gradient descent variants

Batch gradient descent uses the entire training set in each iteration (also called deterministic gradient descent).

$$\theta^{k+1} = \theta^k - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \ell \left(\mathcal{F}(x_i; \theta^k), y_i \right)$$

Stochastic gradient descent (SGD) uses a single datapoint sampled randomly at each iteration.

$$\theta^{k+1} = \theta^k - \eta \nabla_\theta \ell \left(\mathcal{F}(x_{i_k}; \theta^k), y_{i_k} \right)$$

Mini-batch gradient descent uses $b << m$ datapoints randomly sampled at each iteration. (Nowadays most people refer to the mini-batch gradient descent as SGD).

$$\theta^{k+1} = \theta^k - \eta \frac{1}{b} \sum_{i_k=1}^{b} \nabla_\theta \ell \left(\mathcal{F}(x_{i_k}; \theta^k), y_{i_k} \right)$$

Three main variants differing in how much data we use to compute the gradient of the objective function, trading-off computational cost for accuracy.

We will mainly follow the review work from Sebastian Ruder (click here).

Batch gradient descent

- Deterministic trajectory

✓ Guaranteed to converge to global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

✗ Computing all the gradient for one update is very slow.

✗ Intractable for datasets that do not fit in memory.

✗ No online learning.

Stochastic gradient descent (SGD): how can it work?

$$
\begin{aligned}
\theta^{k+1} &= \theta^k - \eta \nabla_\theta \ell \left(\mathcal{F}(x_{i_k}; \theta^k), y_{i_k} \right) \\
&= \theta^{k-1} - \eta \left(\nabla_\theta \ell \left(\mathcal{F}(x_{i_{k-1}}; \theta^{k-1}), y_{i_{k-1}} \right) + \nabla_\theta \ell \left(\mathcal{F}(x_{i_k}; \theta^k), y_{i_k} \right) \right) \\
&= \theta^{k-s} - \eta \sum_{j=0}^{s} \nabla_\theta \ell \left(\mathcal{F}(x_{i_{k-j}}; \theta^{k-j}), y_{i_{k-j}} \right) \\
&\approx \theta^{k-s} - \eta \sum_{j=0}^{s} \nabla_\theta \ell \left(\mathcal{F}(x_{i_{k-j}}; \theta^{k-s}), y_{i_{k-j}} \right)
\end{aligned}
$$

Intuition: assuming that the parameters are updated slowly (small η) then we can roughly approximate that for $j = 0, \ldots, s$

$$\theta^{k-j} \approx \theta^{k-s}.$$

$\Rightarrow$ we can think of SGD as slowly computing an average gradient.

Stochastic gradient descent (SGD)

- Batch gradient descent performs a lot of redundant computation. The gradient parameters very little between θ^k and θ^{k+1}. The SGD exploits this.

- Performs an update of the parameters for each training sample (x_{i_k}, y_{i_k}).

- The dataset is shuffled and then in each iteration a sample is extracted. Once the dataset is consumed, it is re-shuffled. Each run over the dataset is called an **epoch**.

Stochastic gradient descent

- Stochastic trajectory

✗ Slower convergence rate than batch gradient descent (requires more iterations)

✓ . . . but each iteration is much faster.

✓ Allows online learning.

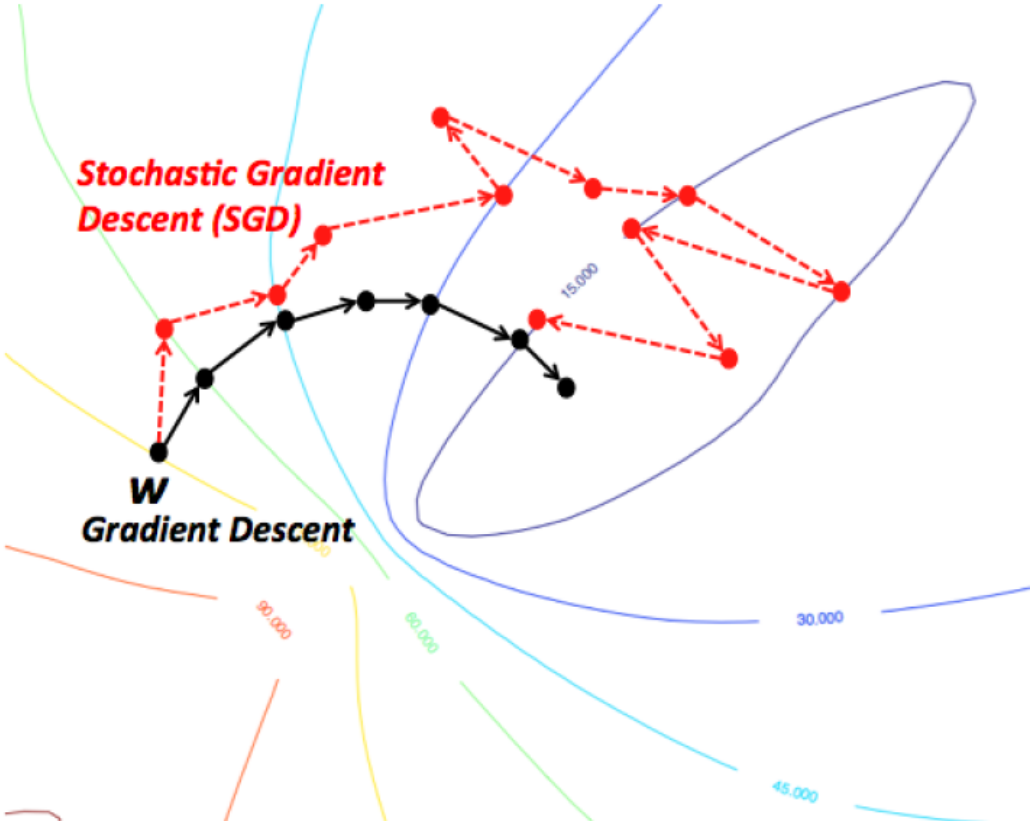
✓ Fluctuation can allow SGD to escape local minima.

✗ At convergence will fluctuate around the minimizer.

✗ Learning rate controls convergence speed, and variance of fluctuations.

SGD vs Batch gradient descent

SGD shows same convergence behaviour as batch gradient descent if learning rate is slowly decreased (annealed) over iterations.



Source: wikidocs.net

Mini-batch gradient descent

- Stochastic trajectory

✓ Reduce variance of the updates.

✓ Can exploit efficient parallelization on GPUs.

✗ Mini-batch size is an hyper-parameter (commonly limited by GPU memory).

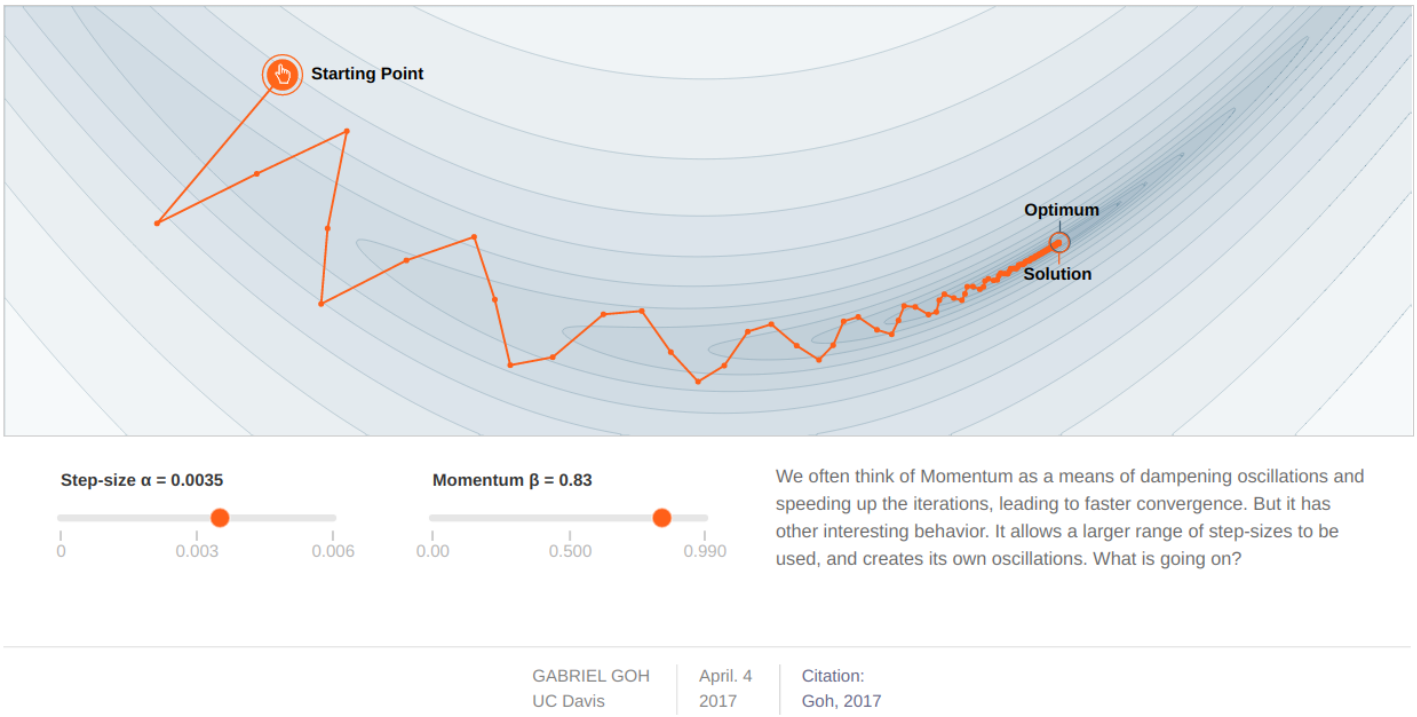
- Typically is the algorithm of choice.

Improvements over SGD

Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another oscillating across the slopes.

Demo!



Source: Goh, 2017 (click here)

Momentum

Momentum helps SGD accelerate by adding a fraction γ from the past update vector (v^k) to current update (v^{k+1})

$$v^{k+1} = \gamma\, v^k + \eta\, \nabla_\theta \mathcal{R}(\theta^k)$$
$$\theta^{k+1} = \theta^k - v^{k+1}$$

- Reduces updates in directions in which the gradient oscillating.
- Increases updates for directions in which the gradient is consistent.

Nesterov Accelerated Gradient

Momentum:

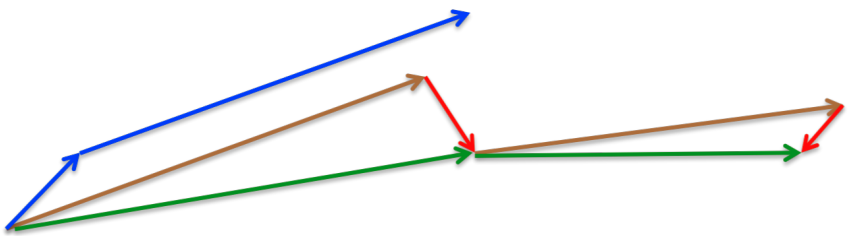
1. Computes the gradient at the current location.
2. Takes a big jump in the accumulated gradient direction.

$$v^{k+1} = \gamma\, v^k + \eta\, \nabla_\theta \mathcal{R}(\theta^k)$$
$$\theta^{k+1} = \theta^k - v^{k+1}$$

Nesterov Accelerated Gradient:

1. Looks ahead in the previous accumulated gradient direction.
2. Measures where it ends up and makes a correction.

$$v^{k+1} = \gamma\, v^k + \eta\, \nabla_\theta \mathcal{R}(\theta^k - \gamma v^k)$$
$$\theta^{k+1} = \theta^k - v^{k+1}$$



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Source: Lecture 6, G. Hinton Coursera Class.(click here)

Annealing the learning rate

Typically a large learning rate is preferred at the beginning:

- Large initial updates when far from the solution
- Higher variance helps avoiding getting stuck in local minima or plateaus

During the optimization the learning rate is gradually reduced:

- By reducing the variance the solution "sinks" into the basin of the minimum

Most common solutions:

- Step decay: $\eta = \eta / K$
- Exponential decay: $\eta = \eta_0 e^{-kt}$.
- 1/t decay: $\eta = \eta_0 / (1 + kt)$

Adagrad

In some ML problems, for some parameters θ_j the stochastic gradient is often 0. Parameters which are updated less frequently, need larger updates.

Adagrad[1] adapts the learning rate of each parameter by dividing the learning rate by the square root of the accumulated sum of squared gradients of previous iterations.

Applied to the SGD update it will result in:

$$
\begin{aligned}
g^k &= \nabla_\theta \mathcal{R}(\theta^k) \\
G^k &= G^{k-1} + g^k \odot g^k \\
\theta^{k+1} &= \theta^k - \frac{\eta}{\sqrt{G^k + \epsilon}} \odot g^k \qquad (\epsilon \text{ small to avoid division by } 0)
\end{aligned}
$$

[1] Duchi et al (2011). J Mach Learn Res, 12(7).

Adagrad

✓ Significantly improves SGD robustness.

✓ Well suited for unbalanced or sparse data.

✓ Reduces importance of learning rate scheduling.

✗ Learning rate shrinks and becomes infinitesimally small because of the accumulation of squared gradients.

RMSprop

RMSprop[2] aims to reduce the aggressive decreasing of Adagrad.

Divides the learning rate by a moving average of squared gradients instead of accumulating them.

$$
\begin{aligned}
g^k &= \nabla_\theta \mathcal{R}(\theta^k) \\
v^k &= \beta\, v^{k-1} + (1-\beta)\,(g^k \odot g^k) \qquad \rightarrow \beta \in [0,1] \text{ the decay rate} \\
\theta^{k+1} &= \theta^k - \frac{\eta}{\sqrt{v^k + \epsilon}} \odot g^k
\end{aligned}
$$

[2] Slide 29, Lecture 6, G. Hinton Coursera Class.

Adam

Adaptive Moment Estimation[3] improves RMSprop by adding momentum to the gradient as a running average.

The simplified update is:

$$
\begin{aligned}
g^k &= \nabla_\theta \mathcal{R}(\theta^k) \\
m^k &= \beta_1\, m^{k-1} + (1-\beta_1)\, g^k \qquad \rightarrow \beta_1 \in [0,1] \\
v^k &= \beta_2\, v^{k-1} + (1-\beta_2)\, (g^k \odot g^k) \qquad \rightarrow \beta_2 \in [0,1] \\
\theta^{k+1} &= \theta^k - \frac{\eta}{\sqrt{v^k + \epsilon}} \odot m^k
\end{aligned}
$$

[3] Kingma & Ba (2015). ICLR.

Adam

Typically $m^0 = v^0 = 0$. This causes a bias of m^k, v^k towards 0, specially in the first iterations. This can be corrected:

$$
\begin{aligned}
g^k &= \nabla_\theta J(\theta^k) \\
m^k &= \beta_1 \, m^{k-1} + (1 - \beta_1) \, g^k \\
\hat{m}^k &= m^k / (1 - \beta_1^k) \\
v^k &= \beta_2 \, v^{k-1} + (1 - \beta_2) \, (g^k \odot g^k) \\
\hat{v}^k &= v^k / (1 - \beta_2^k) \\
\theta^{k+1} &= \theta^k - \frac{\eta}{\sqrt{\hat{v}^k + \epsilon}} \odot \hat{m}^k
\end{aligned}
$$

Which one should you use?

- Adaptive methods work very well with sparse/unbalanced data.
- Adagrad, RMSprop, Adam work well in similar circumstances, among them bias-corrected Adam outperform slightly the others.
- Nesterov Accelerated Gradient SGD with a learning rate annealing is competitive with Adam depending on the problem.

Additional strategies for optimizing SGD

- Shuffling and Curriculum learning.
- Batch normalization.
- Early stopping.
- Gradient noise.

Coming Soon

We still have many open questions

Does our problem have a solution?
(Existence)

Does our problem have an unique solution?
(Uniqueness)

Does our problem still have solutions if we have restrictions on them?
(Constrained Optimization)

Can we still find solutions for non-differentiable problems?
(Non-smooth Optimization)

Convex Optimization is coming

In the following lectures we will be working with problems in where we can answer some of the previous questions.

In convex problems we can assure the existence of solutions.

In strictly convex problems we can assure the uniqueness of their solutions.

If the general problem is (strictly) convex and the restrictions on our solutions enclose them in convex sets then we can assure the (uniqueness and) existence of solutions.

If the problem is convex but not differentiable we will see methods to solve it without using derivatives using duality principles.