Lecture 2: Optimization of Neural Networks

26 September 2022

Alessandro Betti `alessandro.betti@inria.fr`

Office F319, Fermat Building Inria

A plan for this lecture

- Definition of a learning problem
- Functional and Empirical Risks
- Uniform Convergence and Bias-Variance dilemma
- Gradient Flows and Gradient Descent
- Short review of Backpropagation
- Gradient vanishing
- Exercise on the convergence of GD
- Exercise on the curse of dimensionality
- Setting up the learning of a MLP in Python

A plan for the next lecture

- Initialization, normalization, cross-validation
- SGD in details: convergence analysis
- SGD vs Online GD
- Important Variants of GD
- Activity: experimental comparison

Our general setting

- Data
$$\Omega = \begin{cases} X \times Y & \text{for supervised learning} \\ X & \text{for unsupervised learning} \end{cases}$$
with $X \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}^p$.
- A probability measure
$$\pi \colon \mathcal{M} \to [0,1]$$
where $\mathcal{M}$ is the set of events (measurable sets).
- In these lectures we will assume, for definiteness that $\Omega = X \times Y$.

Our general setting (cont.)

- A "local measure of goodness" encoded by a loss function

$$\ell: Y \times Y \to \overline{\mathbb{R}}_+$$

- From which we can define the risk functional

$$f \mapsto \mathcal{R}(f) := \int_\Omega \ell(f(x), y)\, d\pi((x, y))$$

- Then the goal of ML is

find a function f that makes the risk functional as small as possible

Handling the Risk Functional

There are at least two main reason why we cannot directly attack the problem directly using $\mathcal{R}$:

1. Minimizing a functional is in general a hard problem of calculus of variations
2. We do not know π.

Handling the Risk Functional (cont.)

- 1. it is usually addressed by choosing a specific class of parametric functions $\mathcal{F}$ with suitable contraints on the parameters (for instance a deep NN with a particular architecture).
  - Given a family of functions (for instance CNN with 4 layers) labeled by the parameter $w \in \mathbb{R}^N$ we can assume for instance $\mathcal{F} = \{f(\cdot, w)\}_{w \in \mathbb{R}^N}$.
- When we do this we can define a corresponding risk function $R \colon \mathbb{R}^N \to \overline{\mathbb{R}}_+$ defined as the map

$$w \in \mathbb{R}^N \mapsto R(w) := \mathcal{R}(f(\cdot, w))$$

Handling the Risk Functional (cont.)

- 2. can be tackled by approximating the measure π with an empirical probability measure
  - Assume to have access to a training sample $z_1, \ldots, z_n \in \Omega$ drawn independently from the distribution π and define the measure

$$\pi_n := \frac{1}{n} \sum_{i=1}^{n} \delta_{z_i}$$

- This new distribution induces a corresponding empirical risk functional and an empirical risk function:

$$f \mapsto \mathcal{R}_n(f) := \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i), \qquad w \in \mathbb{R}^N \mapsto R_n(w) := \mathcal{R}_n(f(\cdot, w)).$$

Prospectives on Optimization

- In the remaining of this and the next lecture we will discuss how to solve the problem

$$\min_{w \in \mathbb{R}^N} R_n(w)$$

- But still we need to remember that the objective of machine learning is to find a predictor that performs well on new data (unseen in training). That is to say a function that makes the risk (not only the empirical risk) small.
- Then before going on we will briefly discuss this issue
  - Consistency and uniform convergence of the empirical risk ($n \to +\infty$)
  - Bias-Variance dilemma (fixed n)

Uniform Convergence

Let us choose $f_n^* \in \arg\min_{f \in \mathcal{F}} \mathcal{R}_n(f)$, then

$$\mathcal{R}(f_n^*) - \inf_f \mathcal{R}(f) = \left(\mathcal{R}(f_n^*) - \inf_{f \in \mathcal{F}} \mathcal{R}(f)\right) + \left(\inf_{f \in \mathcal{F}} \mathcal{R}(f) - \inf_f \mathcal{R}(f)\right)$$

- The first term measure the statistical cost of estimating the optimal prediction with a finite sample.
- The second term measure how well the functions in the class $\mathcal{F}$ approximate the optimal solution.

Uniform Convergence: first term

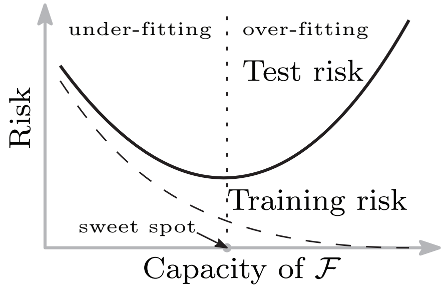
Suppose that $f^* \in \arg\min_{f\in\mathcal{F}} \mathcal{R}(f)$ then

$$\begin{aligned}
\mathcal{R}(f_n^*) - \inf_{f\in\mathcal{F}} \mathcal{R}(f) &= (\mathcal{R}(f_n^*) - \mathcal{R}_n(f_n^*)) + (\mathcal{R}_n(f_n^*) - \mathcal{R}_n(f^*)) + (\mathcal{R}_n(f^*) - \mathcal{R}(f^*)) \\
&\le (\mathcal{R}(f_n^*) - \mathcal{R}_n(f_n^*)) + 0 + (\mathcal{R}_n(f^*) - \mathcal{R}(f^*)) \\
&\le \sup_{f\in\mathcal{F}} |\mathcal{R}(f) - \mathcal{R}_n(f)| + (\mathcal{R}_n(f^*) - \mathcal{R}(f^*))
\end{aligned}$$

- The last term as $n \to \infty$ converge in probability to 0 by LLN
- The first term can again be controlled for a large class of loss functions (but it requires a little bit more work...)

Bias-Variance dilemma

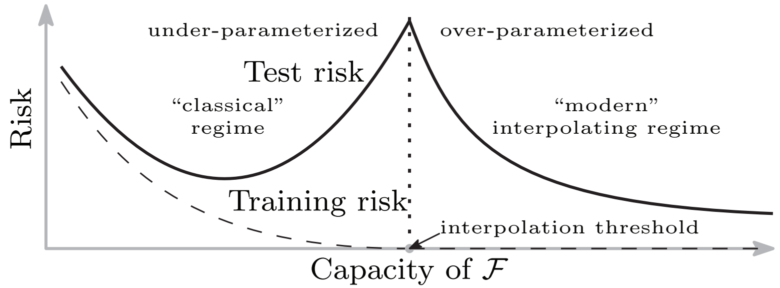
- Traditionally in ML we deal with the discrepancy of empirical risk and the "true" risk by controlling (implicitly or explicitly) the capacity of the family $\mathcal{F}$ with the aim trying to find a "sweet spot" between under-fitting (large bias, small variance) and overfitting (small bias and large variance):
  - If $\mathcal{F}$ is too small we may end up with a large empirical risk
  - If $\mathcal{F}$ is too large we may end up with a solution that fits perfectly the training set but does poorly on new data.



Taken from `https://arxiv.org/pdf/1812.11118.pdf`

Bias-Variance dilemma (cont.)

- In modern machine learning methods (deep networks) it seems that the high capacity of the model and the perfect fitting of the training data in an “interpolating regime” gives low values of the risk and hence good generalization.
- The theoretical reasons under this phenomena are not clearly understood.



Taken from `https://arxiv.org/pdf/1812.11118.pdf`

Gradient flows

- Sometimes it is useful to think of pre-algorithmic processes in continuous time to build some intuition.
- The reason is that in continuous time we can use calculus which is usually easy and rather powerful.
- Consider the following Cauchy problem for an ODE

$$\begin{cases} w'(t) = -\nabla\phi(w(t)) & \text{if } t > 0 \\ w(0) = w^0 & \text{for } t = 0 \end{cases}$$

$t \mapsto w(t) \in \mathbb{R}^N$, $\phi\colon \mathbb{R}^N \to \mathbb{R}$, $\phi \in C^{1,1}_{\mathrm{loc}}(\mathbb{R}^N; \mathbb{R})$.

- A solution of this problem is called a gradient flow.

Gradient flows, $N = 1$

Let us build a little bit of intuition.

- Choose $\phi(x) = x^2/2$, then
$$w(t) = w^0 e^{-t}$$
- Choose $\phi(x) = (x^2 - 1)^2$, then
$$w(t) = \begin{cases} 0 & \text{if } w^0 = 0 \\ 1 & \text{if } w^0 = 1 \\ -1 & \text{if } w^0 = -1 \\ \frac{w^0}{\sqrt{(w^0)^2 - ((w^0)^2 - 1)e^{-8t}}} & \text{otherwise} \end{cases}$$

Gradient flows, energetic analysis

The continuous formulation is particularly nice because it offers an energetic interpretation of the process.

- We have that

  $$\phi \text{ decreases along the gradient flow } w(t),$$

  which means precisely that

  $$\frac{d}{dt}\phi(w(t)) = \nabla\phi(w(t))\cdot w'(t) = -|w'(t)|^2 \leq 0$$

- If we also assume that $\inf \phi > -\infty$ than we may conclude that $\lim_{t\to\infty} \phi(w(t))$ exists, i.e. the value of ϕ along the flow is asymptotically convergent.

Gradient Descent

- A gradient descent on the function ϕ can be regarded as a discretization of the gradienr flow. In particular we have ($\tau > 0$)
- Explicit Euler method:
$$w^{k+1} = w^k - \tau \nabla \phi(w^k)$$
- Implicit Euler method:
$$w^{k+1} = w^k - \tau \nabla \phi(w^{k+1})$$

A “better” way to define Gradient Descent

- The (implicit) gradient step can be alternatively characterized by

$$w^{k+1} \in \operatorname*{arg\,min}_{s \in \mathbb{R}^N} \phi(s) + \frac{1}{\tau} |s - w^k|^2$$

- The (explicit) gradient step can be alternatively characterized by

$$w^{k+1} \in \operatorname*{arg\,min}_{s \in \mathbb{R}^N} \phi(w^k) + \nabla \phi(w^k)(s - w^k) + \frac{1}{\tau} |s - w^k|^2$$

Backpropagation: Introduction

- In Machine Learning we usually choose $\phi = R_n$
- But how do we compute ∇R_n?
- Remember that modern architectures we have N in the range 10^7 (ResNet) 10^{11} (GPT3).

FNN

- A FNN is a NN whose computational structure is that based on a dag $G = (A, V)$.

$$V = I \cup H \cup O = \{1, 2, \cdots, m\}$$

$$I = \{i \in V : \mathrm{pa}(i) = \emptyset\}, \quad O = \{i \in V : \mathrm{ch}(i) = \emptyset\}, \quad H = V \setminus (I \cup O)$$

  Without loss of generality we will assume

$$I = \{1, 2, \ldots, d\}, \qquad O = \{m, m-1, \ldots, m-p+1\}$$

- Clearly $|I| = d$, $|O| = p$, $|H| = m - d - p$.

FNN (cont)

- More precisely a FNN is G together with a set of variables (neuron outputs) for each vertex and a set of weights for each arc.

$$i \mapsto x^i \in \mathbb{R}, \qquad j \longrightarrow i \mapsto w_{ij} \in \mathbb{R}$$

- Given an input pattern $y \in \mathbb{R}^d$ the value of the neuron outputs are recursively computed as

$$x^i = \begin{cases} y_i & \text{if } i \in I \\ \sigma\left(\sum_{j \in \mathrm{pa}(i)} w_{ij} x^j\right) & \text{otherwise} \end{cases}$$

- The maps that takes

$$y \mapsto (x^m, \ldots x^{m-p+1})$$

according to this recursive computations is the FNN.

Informal derivation of Backpropagation

- Suppose we want to compute

$$\frac{\partial}{\partial w_{ij}} R_n = \frac{1}{n} \sum_{k=1}^{n} \frac{\partial}{\partial w_{ij}} \ell(f(x_k), y_k)$$

- Let us denote with a_i the activation of neuron i, that is

$$a_i := \sum_{j \in \mathrm{pa}(i)} w_{ij} x^j, \qquad i \in H \cup O,$$

  then for each sample (x_k, y_k) we have

$$\frac{\partial}{\partial w_{ij}} \ell = \frac{\partial \ell}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}}$$

Informal derivation of Backpropagation (cont./)

- Hence
$$\frac{\partial}{\partial w_{ij}}\ell = \delta_i x^j$$
where $\delta_i := \partial \ell / \partial a_i$ for $i \in H \cup O$.
- Now the crucial point is that we can starting from the values of $\delta_i \in O$ recursively compute backward all the other δ_i:

$$\delta_i = \frac{\partial \ell}{\partial a_i} = \sum_{k \in \mathrm{ch}(i)} \frac{\partial \ell}{\partial a_k} \frac{\partial a_k}{\partial a_i}$$

Informal derivation of Backpropagation (cont./)

- Hence since

$$a_k = \sum_{j \in \mathrm{pa}(k)} w_{kj} \sigma(a_j),$$

$$\delta_i = \sigma'(a_i) \sum_{k \in \mathrm{ch}(i)} \delta_k w_{ki}.$$

- The recurrence relation is closed on the output neurons on which we can directly compute the δ term. For instance if ℓ is a quadratic loss we will have

$$\delta_i = (\sigma(a_i) - y_i)\sigma'(a_i), \qquad i \in H$$

Common problems of training NN: Gradient vanishing

- The classical sigmoidal activation function has $0 < \sigma' < 1/4$
- The delta error has, as we have just seen is

$$\delta_i = \sigma'(a_i) \sum_{k \in \mathrm{ch}(i)} \delta_k w_{ki}$$

- Usually w_{ij} are initialized with $|w_{ij}| < 1$.