

Theoretical Principles of Deep Learning

Class III: Training in the linear regime

Hédi Hadiji

Université Paris-Saclay - CentraleSupélec
hedi.hadiji@l2s.centralesupelec.fr

December 2024

Table of Contents

- 1 **Reminder of Last Time and Plan for the Day**
- 2 Empirical Risk Minimization II
- 3 Convex Optimisation
- 4 Neural networks and Lazy Training
- 5 Lazy training
- 6 Linear models

Plan

Last time: Approximation

- 2-layer nets with enough nodes are universal approximators
- Smooth functions (in the sense of Barron) can be well approximated by small(-ish) nets, even for high-dim inputs

Today: Optimization. Understanding the behavior of neural nets under gradient descent.

Reading Material:

- *On Lazy Training in Differentiable Programming*, Chizat, Oyallon and Bach '19
- *Deep learning theory: A statistical viewpoint*, Bartlett, Montanari and Rakhlin '21

Table of Contents

- 1 Reminder of Last Time and Plan for the Day
- 2 Empirical Risk Minimization II**
- 3 Convex Optimisation
- 4 Neural networks and Lazy Training
- 5 Lazy training
- 6 Linear models

ERM and SGD

Fix a set of hypotheses, parameterized by some $w \in \mathbb{R}^p$.

Given a dataset, find a model that minimizes the empirical loss

$$\arg \min_{w \in \mathbb{R}^p} F(w) := \frac{1}{n} \sum_{i=1}^n \ell(h(x_i; w), y_i),$$

using (Stochastic) Gradient Descent.

In practice, SGD easily finds a model with small train loss. **Why?**

Plan for the day

Bringing **partial** answer to why (S)GD works with neural nets

- Describe a typical cases we understand: linear models
- Identify the Linear Regime (i.e. sets of dimensions, or parameters) in which neural nets can be seen as almost linear models

Table of Contents

- 1 Reminder of Last Time and Plan for the Day
- 2 Empirical Risk Minimization II
- 3 Convex Optimisation**
- 4 Neural networks and Lazy Training
- 5 Lazy training
- 6 Linear models

Linear models

Linear models

Fix a feature map $\phi : \mathcal{X} \rightarrow \mathbb{R}^p$ for some $p \in \mathbb{N}$, and consider the corresponding set of linear models

$$\mathcal{H} = \{h(\cdot; w) : x \mapsto w^\top \phi(x)\}.$$

Important Models are linear in w , not in x . In fact, this makes sense even if x is not an element of a vector space.

Convex losses

If the model is linear and the loss is convex in its first argument then the ERM objective is a convex function of w ,

$$F(w) = \frac{1}{n} \sum_{i=1}^n \ell(w^\top \phi(x_i), y_i)$$



Convex functions are easy to optimize

Gradient descent for convex optimization

Let $F : \mathbb{R}^p \rightarrow \mathbb{R}$ be a convex differentiable function. Gradients point in the direction where the function is locally growing the fastest. To minimize, move in opposite direction.

$$\nabla F(w) = \left(\frac{\partial F}{\partial w_i}(w) \right)$$

Gradient descent

Fix a learning rate (or step-size) $\eta > 0$, an initialization point w_0 . Gradient descent generates the sequence of points defined by

$$w_{t+1} = w_t - \eta \nabla F(w_t).$$

Stochastic Gradient Descent

Sometimes, only have access to noisy estimates of the gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t \quad \text{where} \quad \mathbf{g}_t \quad \text{is such that} \quad \mathbb{E}[\mathbf{g}_t] = \nabla F(\mathbf{w}_t).$$

Example (SGD, sampling with replacement for ERM)

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \ell_i(\mathbf{w})$$

Pick I_t at random in $\{1, \dots, n\}$ and $\mathbf{g}_t = \nabla \ell_{I_t}(\mathbf{w}_t)$, then

$$\mathbb{E}[\mathbf{g}_t] = \sum_{i=1}^n \nabla \ell_i(\mathbf{w}_t) \mathbb{P}[I_t = i] = \frac{1}{n} \sum_{i=1}^n \nabla \ell_i(\mathbf{w}_t) = \nabla \left(\frac{1}{n} \sum_{i=1}^n \ell_i \right) (\mathbf{w}_t) = \nabla F(\mathbf{w}_t)$$

Remark: SGD can also be interpreted as directly minimizing the true average loss $\mathbb{E}[\ell(h(X, \mathbf{w}), Y)]$

GD: Typical behavior for convex and smooth objectives

If F is smooth and the learning rate small enough, GD will converge to a critical point $\nabla F(w^*) = 0$. If function is convex, critical points are global optima.

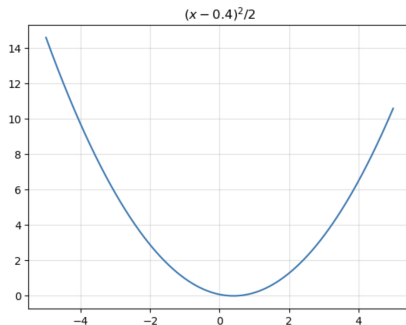
Regardless of initialization:

- reasonable η function value will decrease to the minimum
- too small η small steps, not moving much, slow convergence
- too large η big steps, instability, divergence or oscillations

Could do a (multiple) whole course on convex optimization but let us just look at important examples.

GD examples. Curvature and smoothness.

$$f(w) = \frac{1}{2}(w - w^*)^2$$



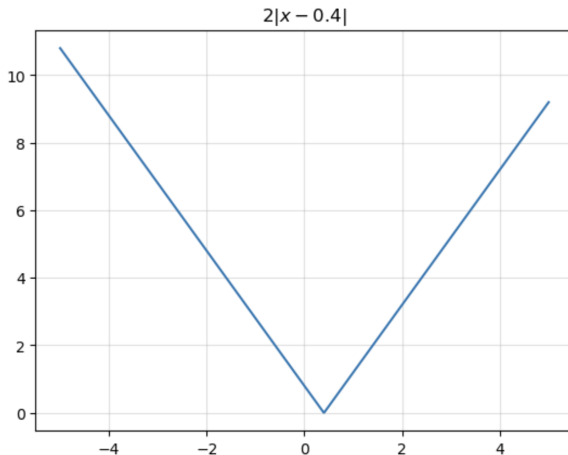
$$w_{t+1} = w_t - \eta(w_t - w_*) \quad \text{so} \quad w_{t+1} - w_* = (1 - \eta)(w_t - w_*)$$

- Exponential convergence iff $0 < \eta < 2$
- Exponential divergence if $\eta > 2$

GD examples: Flat and not smooth

Absolute value: flat and not smooth at the optimum

$$f(w) = G|w - w^*|$$



GD examples: Flat and not smooth II

$$w_t = \begin{cases} w_t - \eta G & \text{if } w_t > w^* \\ w_t + \eta G & \text{if } w_t < w^* \end{cases}$$

Move to a small neighborhood of w^* , then oscillate around w^* .

To find w such that $f(w) - f(w^*) \leq \varepsilon$, pick $\eta = G/\varepsilon$ and wait for $T = |w_0 - w^*|G/\varepsilon$ steps.

Exercise: think about Huber loss and local vs. global effects of curvature and smoothness

$$F(w) = \begin{cases} \frac{|w|^2}{2} & \text{if } |w| \leq 1 \\ |w| - \frac{1}{2} & \text{if } |w| > 1 \end{cases}$$

GD: Fundamental examples

Quadratic:

$$f(w) = \frac{1}{2}(w - w^*)^\top M(w - w^*), \quad M \text{ positive definite}$$

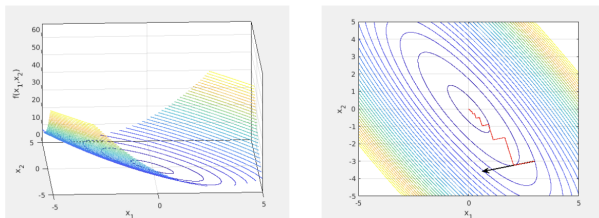


Figure 2. Gradient, level set, and behavior of GD.

Source: Francesco Orabona's [blog](#)

Best learning rate and convergence speed is determined by the condition number of M , i.e. how 'round' the level sets are.

Gradient descent

Exercise: Considering both $\|w - w^*\|$ and $F(w) - F(w^*)$. Describe the behavior of the iterates of gradient descent on the functions

a)

$$f(x, y) = \frac{x^2}{2}$$

b)

$$g(x, y) = \frac{x^2}{2} + 0.00001 \frac{y^2}{2}$$

c)

$$h(x, y) = \frac{x^2}{2} + |y|.$$

Gradient flow

The *Gradient flow* is the solution to the differential equation:

$$\begin{cases} w(0) = w_0 \\ w'(t) = -\nabla F(w(t)) \end{cases}$$

Gradient descent is a discretization scheme to the gradient flow:

$$\frac{w(t + \varepsilon) - w(t)}{\varepsilon} = -\nabla F(w(t))$$

$$w(t + \varepsilon) = w(t) - \varepsilon \nabla F(w(t))$$

Gradient flow has the same trajectory as gradient descent with small step-size. Interesting to study the limit of gradient descent.

Caveat: Step-sizes in practice are not vanishingly small, and some recent investigations suggest that some generalization properties of NNs come from large step-size. (Sharp minima repel large-step size trajectories.)

Table of Contents

- 1 Reminder of Last Time and Plan for the Day
- 2 Empirical Risk Minimization II
- 3 Convex Optimisation
- 4 Neural networks and Lazy Training**
- 5 Lazy training
- 6 Linear models

Recall: Back to neural networks

Feedforward neural networks

For dimensions p, q, r , a **layer** is a function $\mathbb{R}^p \rightarrow \mathbb{R}^r$

$$\Phi_{\sigma, A, b} : X \mapsto \sigma(Ax + b)$$

where $\sigma : \mathbb{R}^q \rightarrow \mathbb{R}^r$ is a simple non-linear function, A is a $q \times p$ matrix and $b \in \mathbb{R}^q$ is a vector.

A **neural network** is a function of the form

$$h : X \mapsto \Phi_{\sigma_L, A_L, b_L} \circ \dots \circ \Phi_{\sigma_0, A_0, b_0}(x).$$

The trainable parameters are

$$w = (A_0, b_0, \dots, A_L, b_L).$$

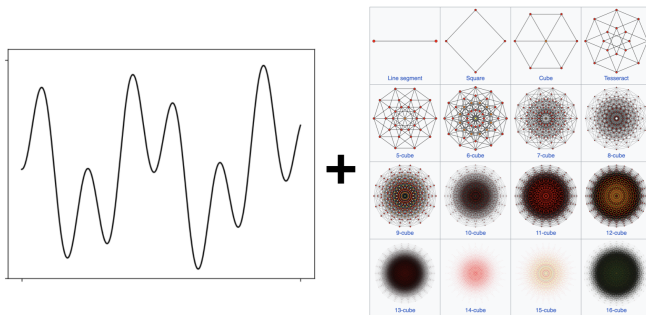
The (A_i) are the weights and the (b_i) the biases, but we often forget the biases and call w the weights.

Non-convexity

Even one hidden layer with square loss

$$\frac{1}{2} (h(x, w) - y)^2 = \frac{1}{2} \left(\sum_{k=1}^m c_k \sigma(a_k^\top x + b_k) - y \right)^2$$

Dependence on (c_k, a_k, b_k) looks complicated (and this is only one layer).



non-convex + high-dimension = trouble

Some empirical observations about neural nets

Neural net that are

- very large
- randomly initialized with an appropriate scaling (Le Cun initialization)

are such that

- training loss decreases very fast
- weights move very slowly

Called the **lazy regime** because the network learns by making small moves.

Table of Contents

- 1 Reminder of Last Time and Plan for the Day
- 2 Empirical Risk Minimization II
- 3 Convex Optimisation
- 4 Neural networks and Lazy Training
- 5 Lazy training**
- 6 Linear models

Notation

Identify $h(w)$ with the vector $(h(w, x_i))$.

Linearization

What about non-linear models? Consider linearization:

$$h(w, x) = \underbrace{h(w_0, x) + Dh(w_0, x) \cdot (w - w_0)}_{\text{linear in } w} + o(\|w - w_0\|)$$

For convenience, omit the dependence on x . (Data is fixed throughout training anyway.)

Linearized model around w_0

$$\bar{h}(w) := h(w_0) + Dh(w_0) \cdot (w - w_0)$$

$$h(w) = \bar{h}(w) + o(\|w - w_0\|)$$

Check type:

$$w \in \mathbb{R}^d, \quad h(w) : \mathcal{X} \rightarrow \mathbb{R}^m, \quad Dh(w) : \mathcal{X} \rightarrow \mathbb{R}^{m \times d}, \quad Dh(w) \cdot (w - w_0) : \mathcal{X} \rightarrow \mathbb{R}^m$$

Lazy training: some heuristic computations

Lazy training

Lazy training occurs when the Taylor expansion of the model stays valid through the whole training procedure.

Hand-wavy, but useful for intuition.

$$\underbrace{\frac{|F(w_1) - F(w_0)|}{F(w_0)}}_{\text{scale of loss variations}} \gg \underbrace{\frac{\|D^2 h(w_0)[w_1 - w_0, w_1 - w_0]\|}{\|Dh(w_0)[w_1 - w_0]\|}}_{\text{scale of variations of the linear approximations}}$$

i.e., using $w_1 - w_0 = \eta \nabla F(w_0)$ when

$$\frac{\|\nabla F(w_0)\|}{|F(w_0)|} \gg \frac{\|D^2 h(w_0)\|}{\|Dh(w_0)\|}.$$

The weights change in a way that keeps the linear approximation valid, but the loss moves.

Heuristic computations II: Square loss

Consider a square loss of the form

$$F(w) = \frac{1}{2n} \sum_{i=1}^n (h(w, x_i) - y_i)^2 = \frac{\|h(w) - y\|_2^2}{2}$$

then $\|\nabla F(w_0)\| = \|Dh(w_0)^\top (h(w_0) - y)\| \approx \|Dh(w_0)\| \|h(w_0) - y\|$, so

$$\frac{\|\nabla F(w_0)\|}{|F(w_0)|} \gg \frac{\|D^2 h(w_0)\|}{\|Dh(w_0)\|} \quad \text{iff} \quad \|h(w_0) - y\| \frac{\|D^2 h(w_0)\|}{\|Dh(w_0)\|^2} \ll 1.$$

Definition (Lazy regime criterion for the square loss)

$$\kappa(w_0) := \|h(w_0) - y\|_2 \frac{\|D^2 h(w_0)\|}{\|Dh(w_0)\|^2} \ll 1$$

Morally, if $\kappa(w_0)$ is small, then the model stays close to its linear approximation during the whole training.

Gradient flow and linearized gradient flow

Consider the train loss of the model and its linearization

$$F(w) = \frac{\|h(w) - y\|_2^2}{2} \quad \text{and} \quad \bar{F}(w) = \frac{\|\bar{h}(w) - y\|_2^2}{2}$$

and their respective gradient flows

$$w'(t) = -\nabla F(w(t)) \quad \text{and} \quad \bar{w}'(t) = -\nabla \bar{F}(\bar{w}(t)).$$

Let $\sigma_{\min}, \sigma_{\max}$ be the min and max singular values of $Dh(w_0)$, and

$$\text{Lip}(Dh) = \sup_{w \neq w'} \frac{\|Dh(w) - Dh(w')\|}{\|w - w'\|}.$$

Gradient flow linearization: Some guarantees

Theorem (Oymak, Soltanolkotabi '19)

If

$$\|h(w_0) - y\|_2 \frac{\text{Lip}(Dh)}{\sigma_{\min}^2(Dh(w_0))} < \frac{1}{4}$$

then for all t ,

$$\|w(t) - \bar{w}(t)\| \leq c \frac{\sigma_{\max}}{\sigma_{\min}^2} \|y - h(w_0)\|_2^2 + \frac{\text{Lip}(Dh)}{\sigma_{\min}^3} \|y - h(w_0)\|_2.$$

ToDo: and the loss decreases.

This is a formally correct version of the criterion on κ .

Linear regime

Let us describe two cases of the linear regime training:

- Scaling + 0-prediction initialization
- Large width + random initialization

Linear regime by scaling

Simple way to ensure lazy training: multiply the model outputs by a scale factor α , while maintaining $h(w_0) = 0$. Then

$$\|y\| \frac{\| \text{Lip}(D(\alpha h)) \|}{\sigma_{\min}^2(D(\alpha h)(w_0))} \xrightarrow{\alpha \rightarrow \infty} 0$$

Theorem (Chizat, Oyallon, Bach '19)

For the square loss, if $h(w_0) = 0$, the trajectories of gradient descent following αh and $\bar{\alpha} \bar{h}$ stay within $O(1/\alpha)$ of each other.

Linear regime in ∞ -width two layer nets

Proposition (Linear regime in infinite-width)

Consider a one-layer hidden layer neural network with m hidden nodes with Le Cun initialization, then almost surely

$$\kappa(W_0) \rightarrow 0 \quad \text{as} \quad m \rightarrow \infty.$$

Le Cun initialization: all weights are randomly initialized with variance of inputs of every node that sum to 1, e.g., if $x \in \mathbb{R}$, is a 1-dim input,

$$h(W_0, x) = \sum_{j=1}^m C_j \sigma(x A_j)$$

$$A_j \sim \mathcal{N}(0, 1) \quad C_j \sim \mathcal{N}\left(0, \frac{1}{m}\right)$$

More details in two lectures: the Neural Tangent Kernel.

Table of Contents

- 1 Reminder of Last Time and Plan for the Day
- 2 Empirical Risk Minimization II
- 3 Convex Optimisation
- 4 Neural networks and Lazy Training
- 5 Lazy training
- 6 Linear models**

A closer look at least-squares regression

Least-squares regression

$\mathcal{Y} = \mathbb{R}$ and $\phi(\mathbf{x}) \in \mathbb{R}^p$. Minimize the empirical square loss

$$\arg \min_{\mathbf{w} \in \mathbb{R}^p} \frac{1}{2n} \sum_{i=1}^n (\mathbf{w}^\top \phi(\mathbf{x}_i) - y_i)^2$$

Useful notation

$$\mathbf{X} = \begin{pmatrix} \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_n) \end{pmatrix} \in \mathbb{R}^{p \times n} \quad \text{and} \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^n$$

Least-squares can be written as (with euclidean norm in \mathbb{R}^n)

$$\arg \min_{\mathbf{w} \in \mathbb{R}^p} \frac{1}{2n} \|\mathbf{X}^\top \mathbf{w} - \mathbf{Y}\|_2^2$$

Overparameterization in linear regression

$$\arg \min_{w \in \mathbb{R}^p} \frac{1}{2n} \|\mathbf{X}^\top w - \mathbf{Y}\|_2^2$$

Consider the system of equations in $w \in \mathbb{R}^p$

$$\mathbf{X}^\top w = \mathbf{Y}, \quad \mathbf{X}^\top \in \mathbb{R}^{n \times p}$$

Underparameterization. If $\text{Rank}(\mathbf{X}^\top) < n$, then system can have no solutions (generically for \mathbf{Y}). Loss has unique minimizer, reached at the projection of \mathbf{Y} on the linear span of the feature vectors.

Overparameterization. If $\text{Rank}(\mathbf{X}^\top) = n$, then system can have infinitely many solutions, and the minimal loss value is 0. The feature vectors have the capacity to fit perfectly the data in many ways.

GD in overparameterized least-squares

GD (with learning rate $n\eta$)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{X}(\mathbf{X}^\top \mathbf{w}_t - \mathbf{Y}) = (I_p - \eta \mathbf{X}\mathbf{X}^\top) \mathbf{w}_t + \eta \mathbf{X}\mathbf{Y}.$$

In overparameterized regime, denoting by λ_i the eigenvalues of $\mathbf{X}\mathbf{X}^\top$

$$I_p - \eta \mathbf{X}\mathbf{X}^\top \sim \begin{pmatrix} 1 - \eta\lambda_1 & & & & \\ & \ddots & & & \\ & & 1 - \eta\lambda_n & & \\ & & & 1 & \\ & \mathbf{0} & & & \ddots & \\ & & & & & 1 \end{pmatrix}$$

- GD converges exponentially fast to a minimizer of F , and the speed of convergence depends on eigenvalues of the feature matrix $\mathbf{X}\mathbf{X}^\top$.
- (\mathbf{w}_t) only moves in an n -dimensional: the image of \mathbf{X}

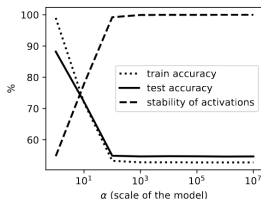
→ Overparameterized linear models are lazy

Lazy training: so what?

Benefits of the lazy training regime

- Neural nets are easy to train in the lazy training
- Great for theory: we can also identify the point of convergence and study the statistical properties of that point.

BUT limited for practice. Linearized nets behave worse than real life nets.



(a)

Model	Train acc.	Test acc.
ResNet wide, linearized	55.0	56.7
VGG-11 wide, linearized	61.0	61.7
Prior features [25]	-	82.3
Random features [28]	-	84.2
VGG-11 wide, standard	99.9	89.7
ResNet wide, standard	99.4	91.0

(b)

Figure 3: (a) Accuracies on CIFAR10 as a function of the scaling α . The stability of activations suggest a linearized regime when high. (b) Accuracies on CIFAR10 obtained for $\alpha = 1$ (standard, non-linear) and $\alpha = 10^7$ (linearized) compared to those reported for some linear methods without data augmentation: random features and prior features based on the scattering transform.

source [the paper](#)

ToDo: relevance in practice?

Conclusion and Next time

Today

- Optimization in linear models
- The Lazy regime

Next time: A closer look at the linearized model for infinite-width networks **Neural Tangent Kernel**

Conclusion and Next time

Thanks !!
Break