Neural Networks

Clément Lalanne

Notation for this lecture

$(x, y) \sim \mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$, sample $S = ((x_1, y_1), \dots, (x_n, y_n)) \sim \mathbb{P}^{\otimes n}$, empirical and population risks

$$\widehat{R}_S(f) = \frac{1}{n} \sum_{i \le n} \ell(y_i, f(x_i)), \qquad R(f) = \mathbb{E}_{(x,y)\sim\mathbb{P}}[\ell(y, f(x))].$$

Depth. We write L for the depth of a network.

Parameters. $\theta = (\theta_1, \dots, \theta_L) \in \mathbb{R}^{d_1} \times \cdots \times \mathbb{R}^{d_L}$, and $P = d_1 + \cdots + d_L$ is the total number of scalar parameters.

Simplex. $\Delta_{d-1} = \{u \in \mathbb{R}^d : u \ge 0, \mathbf{1}^\top u = 1\}$, the set of probability vectors on d classes.

Part I. Architectures

The class

Definition. A **neural network** is a function obtained by composing L parametrized maps,

$$f_{(\theta_1,\dots,\theta_L)}(x) = \rho_{\theta_L}^{(L)}(\cdots(\rho_{\theta_1}^{(1)}(x))),$$

and the associated class is

$$\mathcal{F} = \{f_{(\theta_1,\dots,\theta_L)} : (\theta_1,\dots,\theta_L) \in \mathbb{R}^{d_1} \times \cdots \times \mathbb{R}^{d_L}\}.$$

General principle. Each $\rho^{(i)}$ is in general very simple. The complexity and the expressivity come from the depth, not from any single layer.

Vocabulary

The map f, meaning the sequence of shapes and of layer types with the parameters left free, is the **architecture**.

$(\theta_1, \ldots, \theta_L)$ are the **weights**.

L is the **depth**.

$\rho^{(i)} \circ \cdots \circ \rho^{(1)}$ is the **layer**, or level, i.

$(\rho^{(i)} \circ \cdots \circ \rho^{(1)})(x)$ are the **activations** at layer i.

1) Linear layers

Definition. A **linear layer** is

$$\rho^{(i)}_{\theta_i}(x) = W_i x + b_i, \quad \theta_i = (W_i, b_i).$$

W_i is the matrix of **weights** and b_i is the **bias**.

Two warnings. The word *weights* has a double usage: it is both the full parameter θ_i and the matrix W_i inside it. And a *linear* layer is affine, not linear, as soon as $b_i \neq 0$.

Convention. Appending a constant coordinate 1 to x turns $Wx + b$ into a genuinely linear map of the augmented vector. We keep the bias explicit because it is what the code does.

2) Why a non-linearity is needed

> **Proposition.** A composition of affine maps is affine.

Proof. $W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$, and induction on the depth. ■

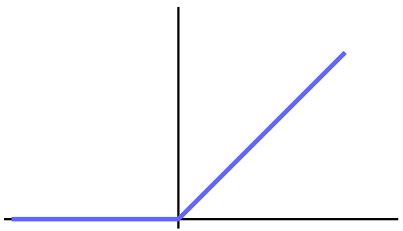
So depth alone is not enough: a network of affine layers is a single affine layer. Worse, $\mathrm{rank}(W_2W_1) \leq \min(\mathrm{rank}W_1, \mathrm{rank}W_2)$, so stacking narrow affine layers can only lose expressivity.

The fix. Insert a fixed non-linear map between the affine ones. Expressivity comes from alternating affine maps and non-linear rectifications.

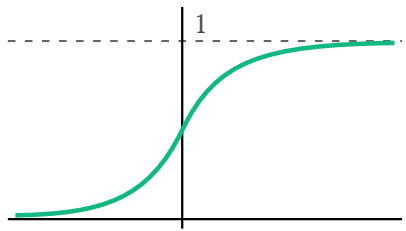
Activation functions

Definition. A **non-linear layer** is $\rho_{\theta_i}^{(i)}(x) = \sigma(W_i x + b_i)$ with $\theta_i = (W_i, b_i)$, where σ is a fixed function applied coordinatewise, the **activation function**.

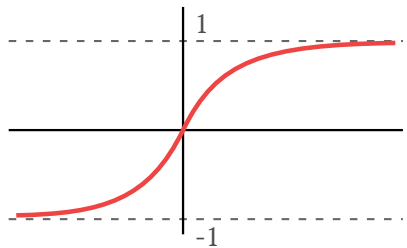
ReLU



sigmoid



tanh



$$\sigma(u) = (u)_+ = \max(0, u), \quad \sigma(u) = \frac{1}{1 + e^{-u}}, \quad \sigma(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}.$$

The softmax

In classification with d classes we want $f(x) \in \Delta_{d-1}$, so that $f(x)$ can be read as the vector of probabilities of belonging to each class. A network produces an arbitrary vector of $\mathbb{R}^d$, so it has to be converted.

Definition. The **softmax** is $\mathrm{sm} : \mathbb{R}^d \to \Delta_{d-1}$,

$$\mathrm{sm}(x)_j = \frac{e^{x_j}}{\sum_{k \le d} e^{x_k}}.$$

It lands in Δ_{d-1}: each coordinate is positive and they sum to 1 by construction. It is applied once, as the last layer, and its input coordinates are called the **logits**.

Not coordinatewise. Unlike ReLU, sigmoid and tanh, the softmax couples the coordinates. That is what makes its Jacobian, computed in Part II, not diagonal.

3) Loss functions, regression

A network is trained with a loss $\ell(\cdot, \cdot)$. In regression, with $\mathcal{Y} = \mathbb{R}^d$, the loss is usually quadratic,

$$\ell(y, f(x)) = \|y - f(x)\|_2^2.$$

Nothing here is specific to networks: this is the loss of the introductory lecture, whose Bayes predictor is the conditional mean $\mathbb{E}_{y|x=x'}[y]$. Choosing an architecture changes the class $\mathcal{F}$, not the target.

3) Loss functions, classification

In classification both y and $f(x)$ live in Δ_{d-1}, and two probability vectors have to be compared. This is done with the **cross-entropy loss**,

$$\ell(y, p) = -\sum_{i \le d} y_i \log(p_i), \quad y, p \in \Delta_{d-1},$$

with the convention $0 \log 0 = 0$.

Proposition. For all $p, q \in \Delta_{d-1}$, $\ell(p, q) \geq \ell(p, p)$, with equality if and only if $p = q$. In particular $q \mapsto \ell(p, q)$ is minimized exactly at $q = p$.

Warning. It is *not* true that $\ell(p, q) = 0$ if and only if $p = q$: for $p = q = (\frac{1}{2}, \frac{1}{2})$ the loss equals $\log 2$. The quantity that vanishes exactly on the diagonal is the difference $\ell(p, q) - \ell(p, p)$.

Proof of the proposition

$$\begin{aligned}
\ell(p,q) - \ell(p,p) &= -\sum_i p_i \log q_i + \sum_i p_i \log p_i = -\sum_i p_i \log \frac{q_i}{p_i} \\
&\geq -\log\left(\sum_i p_i \frac{q_i}{p_i}\right) && \text{Jensen, log concave} \\
&= -\log\left(\sum_i q_i\right) = -\log 1 = 0
\end{aligned}$$

All sums are over $\{i : p_i > 0\}$. Jensen is an equality exactly when q_i/p_i is constant on that set, and since p and q both sum to 1 the constant is 1, that is $p = q$. ■

Reading it. The difference $\ell(p,q) - \ell(p,p)$ is the Kullback and Leibler divergence $\mathrm{KL}(p\|q)$, and $\ell(p,p)$ is the Shannon entropy of p. Cross-entropy is entropy plus divergence, and only the divergence depends on the prediction.

Hard labels, and the link with maximum likelihood

In practice y is a **one-hot** vector, $y = e_c$ for the true class c. Then $\ell(p, p) = 0$ and

$$\ell(e_c, p) = -\log p_c.$$

On one-hot labels the claim of the notes is exactly right: the loss is nonnegative and vanishes only at $p = e_c$. The counterexample above needs a genuinely spread out p, which only occurs with soft labels.

Summing over the sample, $\frac{1}{n}\sum_i -\log p_{c_i}$ is the negative log-likelihood of the model $\mathbb{P}(y = c \mid x) = f_\theta(x)_c$. Training a network with cross-entropy *is* maximum likelihood, which is the bridge stated in the introductory lecture: choosing $\ell = -\log p_\theta$ turns maximum likelihood into empirical risk minimization.

Part II. Optimization and the chain rule

The objective

Training minimizes over θ the regularized empirical risk

$$F(\theta) = \frac{1}{n}\sum_{i \le n} \ell(y_i, f_\theta(x_i)) + \Omega(\theta), \qquad \nabla_\theta F(\theta) = \frac{1}{n}\sum_{i \le n} \nabla_\theta \ell(y_i, f_\theta(x_i)) + \nabla_\theta \Omega(\theta).$$

Gradient descent.

- start at θ_0
- while a stopping criterion is not met, $\theta_t \leftarrow \theta_{t-1} - \gamma_t \nabla_\theta F(\theta_{t-1})$

The step γ_t is the **learning rate** at iteration t.

Two things are missing and are the subject of the rest of Part II: the gradient is a sum of n terms and n is large, and each term is a derivative through L compositions.

Stochastic gradient descent

When n is large, computing $\nabla_\theta F$ is too expensive. Replace it by an estimator computed on a subset of the data. For $B \subseteq \{1, \dots, n\}$,

$$\nabla_\theta^{(B)} F(\theta) = \frac{1}{|B|} \sum_{b \in B} \nabla_\theta \ell(y_b, f_\theta(x_b)) + \nabla_\theta \Omega(\theta).$$

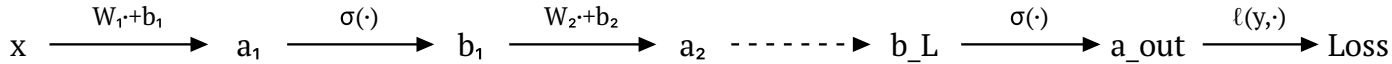
Stochastic gradient descent.

- start at θ_0
- while a stopping criterion is not met:
  - select $B \subseteq \{1, \dots, n\}$ by the chosen rule, random or deterministic
  - $\theta_t \leftarrow \theta_{t-1} - \gamma_t \nabla_\theta^{(B)} F(\theta_{t-1})$

The set B is a **batch**.

If B is drawn uniformly then $\mathbb{E}_B[\nabla_\theta^{(B)} F(\theta)] = \nabla_\theta F(\theta)$: the direction is unbiased, and only its variance is paid for.

1) Backpropagation, the problem



The network encodes a function $\theta \mapsto \ell(y, f_\theta(x))$ built from $2L$ or so elementary maps. How is its gradient in θ computed?

Solution. The chain rule. If f and g are differentiable, $d(g \circ f) = (dg \circ f) \circ df$.

The two identities

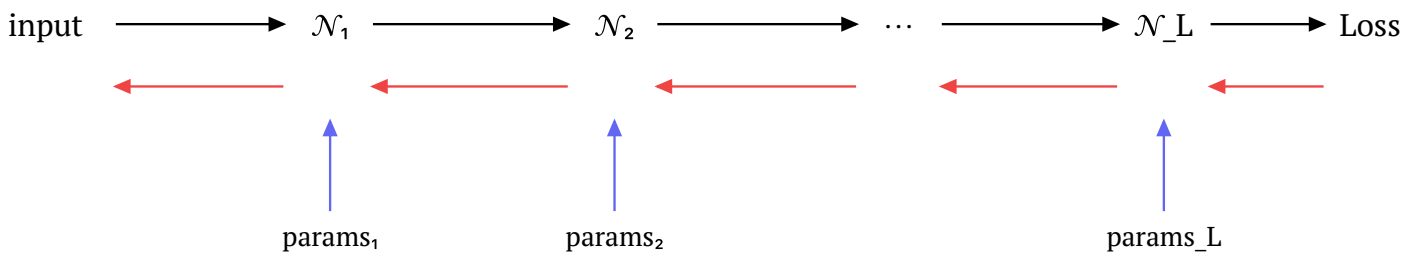
In informal notation, if $\mathcal{N} = \mathcal{N}(\text{input}, \text{params})$ is one node and $\text{Loss} = \text{Loss}(\mathcal{N})$ is everything downstream of it,

$$\frac{\partial \text{Loss}}{\partial \text{params}} = \frac{\partial \text{Loss}}{\partial \mathcal{N}} \frac{\partial \mathcal{N}}{\partial \text{params}} \quad (*), \qquad \frac{\partial \text{Loss}}{\partial \text{input}} = \frac{\partial \text{Loss}}{\partial \mathcal{N}} \frac{\partial \mathcal{N}}{\partial \text{input}} \quad (**).$$

Both are the chain rule, and both need the same object $\partial \text{Loss} / \partial \mathcal{N}$, the derivative of the loss with respect to the *output* of the node.

That object is what travels backwards: $(**)$ turns the one at a node into the one at the node before it, and $(*)$ reads off the gradient in the parameters of that node along the way. One traversal, every gradient.

The forward backward algorithm



Forward. Compute $f_\theta(x)$ and the loss, keeping the activations at every node.

Backward. Starting from the loss and going back, at each node $\mathcal{N}$: compute $\partial \text{Loss}/\partial \text{params}(\mathcal{N})$ with $(*)$, then $\partial \text{Loss}/\partial \text{input}(\mathcal{N})$ with $(**)$ and pass it to the previous node.

What backpropagation costs

Time. The backward pass applies one Jacobian per node, each of the same size as the corresponding forward operation. The total cost is a small constant times the cost of one forward pass, independent of P.

Memory. Every activation of the forward pass has to be kept until the backward pass reaches it. Memory, not time, is what limits depth and batch size in practice.

Why not forward differentiation. Propagating $\partial \mathcal{N} / \partial \theta$ forwards is also the chain rule, but it carries an object of size $\dim(\mathcal{N}) \times P$ at each node instead of a vector of size $\dim(\mathcal{N})$. It costs one pass per parameter. With P in the millions this is the whole difference.

2) Gradients by hand, ReLU and affine

ReLU, applied coordinatewise, $\mathrm{ReLU}(x) = ((x_1)_+, \dots, (x_c)_+)$:

$$\frac{\partial \mathrm{ReLU}(x)_j}{\partial x_i} = \delta_{ij} \mathbf{1}\{x_i > 0\}.$$

At 0 we have arbitrarily chosen a subgradient, since $u \mapsto (u)_+$ is not differentiable there.

Affine, $\mathrm{aff}(x) = Ax + b$:

$$\frac{\partial \mathrm{aff}(x)_j}{\partial x_i} = A_{ji}, \quad \frac{\partial \mathrm{aff}(x)_j}{\partial b_i} = \delta_{ij}, \quad \frac{\partial \mathrm{aff}(x)_k}{\partial A_{ij}} = \delta_{ik} x_j.$$

The Jacobian in x is A, so the backward pass through an affine layer is a multiplication by $A^\top$. That single line is most of what a linear layer does in a deep learning framework.

2) Gradients by hand, softmax

Proposition. For $\mathrm{sm}(x)_j = e^{x_j} / \sum_k e^{x_k}$,

$$\frac{\partial \mathrm{sm}(x)_j}{\partial x_i} = \mathrm{sm}(x)_j(\delta_{ij} - \mathrm{sm}(x)_i).$$

Proof. Write $Z = \sum_k e^{x_k}$, so $\mathrm{sm}(x)_j = e^{x_j}/Z$ and $\partial Z/\partial x_i = e^{x_i}$. Then

$$\frac{\partial}{\partial x_i}\frac{e^{x_j}}{Z} = \frac{\delta_{ij}e^{x_j}Z - e^{x_j}e^{x_i}}{Z^2} = \delta_{ij}\frac{e^{x_j}}{Z} - \frac{e^{x_j}}{Z}\frac{e^{x_i}}{Z}. \quad \blacksquare$$

Consequence. With cross-entropy on a one-hot label e_c and logits x, the gradient of $-\log \mathrm{sm}(x)_c$ in x is $\mathrm{sm}(x) - e_c$: the predicted distribution minus the truth. Softmax and cross-entropy are always implemented together for exactly this reason.

A warning about the optimization

Nothing above says that gradient descent finds a minimizer. F is not convex in θ as soon as $L \geq 2$, and it has many critical points.

Symmetries. Permuting the units of a hidden layer, together with the corresponding rows and columns of the neighbouring matrices, leaves f_θ unchanged. Every minimizer therefore comes in orbits of size at least $\prod_l (\mathrm{width}_l)!$, so there is no hope of a unique solution.

What is guaranteed. Under a step size condition, gradient descent converges to a critical point of F. Nothing more is proved in general.

What this costs us. The excess risk decomposition of the introductory lecture has an optimization term, and here it is not zero and not controlled. Part IV is built so that this does not matter.

Part III. Universal approximation

The statement

We prove that neural networks can approximate continuous functions as closely as we want. The class used is the smallest interesting one, **networks with one hidden layer**,

$$f(x) = \sum_{j \leq m} \eta_j (w_j^\top x + b_j)_+.$$

Theorem. Let $g \in C^0([a,b], \mathbb{R})$ and $\epsilon > 0$. There is a one hidden layer ReLU network f with finitely many units such that

$$\sup_{x \in [a,b]} |g(x) - f(x)| \leq \epsilon.$$

The proof is in two steps: continuous functions are approximated by continuous piecewise affine ones, and continuous piecewise affine functions are one hidden layer networks *exactly*.

Step 1, uniform continuity

Lemma (Heine). A continuous g on a compact $[a, b]$ is uniformly continuous.

Proof. Suppose not. Then there is $\epsilon > 0$ such that for every $\eta > 0$ some pair $x, y \in [a, b]$ has $|x - y| < \eta$ and $|g(x) - g(y)| \geq \epsilon$. Taking $\eta = 1/k$ gives sequences (a_k) and (b_k) in $[a, b]$ with

$$|a_k - b_k| < \frac{1}{k}, \quad |g(a_k) - g(b_k)| \geq \epsilon.$$

$[a, b]$ is compact, so extract $a_{\varphi(k)} \to c \in [a, b]$. Then $b_{\varphi(k)} = a_{\varphi(k)} + (b_{\varphi(k)} - a_{\varphi(k)}) \to c$ as well. By continuity of g at c,

$$|g(a_{\varphi(k)}) - g(b_{\varphi(k)})| \leq |g(a_{\varphi(k)}) - g(c)| + |g(c) - g(b_{\varphi(k)})| \to 0,$$

which contradicts $\geq \epsilon$. ∎

Step 1, the piecewise affine interpolant

Let $\epsilon > 0$. Uniform continuity gives $\eta > 0$ with $|x - y| \le \eta \Rightarrow |g(x) - g(y)| \le \epsilon$. Take knots

$$a = x_1 < x_2 < \cdots < x_N = b, \quad x_{k+1} - x_k \le \eta,$$

and let g_η be the continuous function that equals g at every knot and is affine on every $[x_k, x_{k+1}]$.

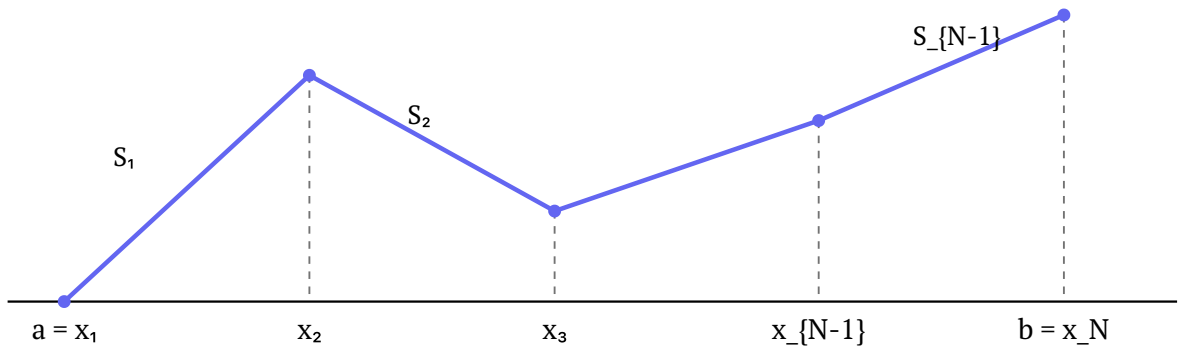
> **Claim.** $\sup_{x \in [a,b]} |g(x) - g_\eta(x)| \le \epsilon$.

Proof. Fix $x \in [x_k, x_{k+1}]$ and write $g_\eta(x) = \lambda g(x_k) + (1 - \lambda) g(x_{k+1})$ with $\lambda \in [0, 1]$. Both $|x - x_k|$ and $|x - x_{k+1}|$ are at most η, so

$$|g_\eta(x) - g(x)| = |\lambda (g(x_k) - g(x)) + (1 - \lambda)(g(x_{k+1}) - g(x))| \le \lambda \epsilon + (1 - \lambda)\epsilon = \epsilon.$$

■

Step 2, piecewise affine equals one hidden layer



Let f be continuous and affine on each $[x_k, x_{k+1}]$, with slope S_k there.

Step 2, the case $f(a) = 0$

Claim. With the convention $S_0 = 0$,

$$f(x) = \sum_{i \geq 1} (S_i - S_{i-1})(x - x_i)_+ \quad \text{on } [a, b].$$

Proof. Fix k and $x \in [x_k, x_{k+1}]$. Then $(x - x_i)_+ = x - x_i$ for $i \leq k$ and 0 for $i > k$, so the right hand side equals $\sum_{i \leq k} (S_i - S_{i-1})(x - x_i)$, which is affine on that interval with slope

$$\sum_{i \leq k} (S_i - S_{i-1}) = S_k - S_0 = S_k.$$

Both sides are continuous, have the same slope on every piece, and agree at $x = x_1 = a$, where the right hand side is $0 = f(a)$. Hence they agree on $[a, b]$. $\blacksquare$

Step 2, the general case

If $f(a) \neq 0$ the sum above starts at 0 and cannot match. There is no output bias in the class, so the constant has to be produced by a unit.

View f on $[a, b]$ as the restriction of a continuous piecewise affine f' on $[a-1, b]$, with the extra knot $x_0 = a - 1$ and the slope $S_0 = f(a)$ on $[x_0, x_1]$, so that $f'(a-1) = 0$. With $S_{-1} = 0$ the previous case gives

$$f(x) = \sum_{i \geq 0} (S_i - S_{i-1})(x - x_i)_+ \quad \text{on } [a, b].$$

On $[a, b]$ the unit $i = 0$ is always active and contributes $f(a)(x - a + 1)$, whose value at a is $f(a)$ and whose slope is already counted by the telescoping. The network has one unit per knot, $w_j = 1$ and $b_j = -x_j$.

■

The theorem, and what it does not say

Combining the two steps proves the theorem: given g and ϵ, take η from uniform continuity, interpolate on knots of spacing η, and write the interpolant as a network with one unit per knot.

The width is not free. The number of units is the number of knots, of order $(b-a)/\eta$. For a G-Lipschitz g one may take $\eta = \epsilon/G$, so the width is of order $G(b-a)/\epsilon$. In dimension k the same construction needs of order ϵ^{-k} units.

It is an existence statement. It says a good f is in $\mathcal{F}$, not that any algorithm finds it, and not that finitely many samples identify it. In the language of the introductory lecture, it makes the approximation error small and says nothing about the estimation error.

And the two fight. Shrinking ϵ inflates the width, hence P, hence the estimation error of Part IV. Universal approximation is one half of a trade-off, never a conclusion on its own.

The multivariate statement

Theorem (Cybenko, Hornik). Let σ be continuous, not polynomial. For every compact $K \subset \mathbb{R}^k$, every $g \in C^0(K, \mathbb{R})$ and every $\epsilon > 0$, there are m, η_j, w_j, b_j with

$$\sup_{x \in K} \left| g(x) - \sum_{j \le m} \eta_j \sigma(w_j^\top x + b_j) \right| \le \epsilon.$$

Proof omitted, see Cybenko (1989), Hornik (1991) and Leshno et al. (1993) for the "not polynomial" form.

The one dimensional ReLU proof above is the special case that can be written on a slide, and it is the one to remember: the units are the kinks, and the coefficients are the jumps of the slope.

Part IV. Generalization

What is left to bound

Universal approximation removed the approximation error. The excess risk of the introductory lecture,

$$R(\hat{f}_S) - R^\star = \underbrace{\{R(\hat{f}_S) - \inf_{f \in \mathcal{F}} R(f)\}}_{\textbf{estimation error}} + \underbrace{\{\inf_{f \in \mathcal{F}} R(f) - R^\star\}}_{\textbf{approximation error}},$$

is now entirely in its first term.

The class is parametric. $\mathcal{F}$ is indexed by $\theta \in \mathbb{R}^P$, so every tool of the last three lectures applies with $d = P$: covering numbers, Dudley, VC dimension.

Two regimes to separate. With a minimizer in hand the estimation error is bounded. Without one, and there is never one, the *generalization gap* is still bounded, and that is the statement that survives.

The class we can control

Fix an architecture of depth L with σ 1-Lipschitz and $\sigma(0) = 0$, which holds for ReLU and tanh. Assume the inputs and the weights are bounded,

$$\|x\|_2 \le R, \quad \|W_l\|_{\mathrm{op}} \le B \ \text{ for all } l, \quad \theta \in \Theta \subset \mathbb{R}^P.$$

Lemma. For $\theta, \theta' \in \Theta$ and every such x,

$$\|f_\theta(x) - f_{\theta'}(x)\|_2 \le L B^{L-1} R \max_{l \le L} \|W_l - W_l'\|_{\mathrm{op}}.$$

The network is Lipschitz in its parameters, with a constant that is exponential in the depth. Both halves are used below: the first makes the covering argument work, the second makes the resulting bound useless in practice.

Proof of the lemma

Let h_l and h'_l be the activations at layer l under θ and θ', with $h_0 = h'_0 = x$, and put $\Delta = \max_l \|W_l - W'_l\|_{\mathrm{op}}$.

$$
\begin{aligned}
\|h_l\| &\le B\|h_{l-1}\| \le B^l R && \sigma \text{ is 1-Lipschitz}, \sigma(0)=0 \\
\|h_l - h'_l\| &\le \|W_l - W'_l\|\|\sigma(h_{l-1})\| + \|W'_l\|\|\sigma(h_{l-1}) - \sigma(h'_{l-1})\| && \text{add and subtract } W'_l\sigma(h_{l-1}) \\
&\le \Delta B^{l-1} R + B\|h_{l-1} - h'_{l-1}\|
\end{aligned}
$$

Write $u_l = \|h_l - h'_l\|$. Then $u_0 = 0$ and $u_l \le \Delta B^{l-1} R + B u_{l-1}$, so $u_1 \le \Delta R$, $u_2 \le 2\Delta B R$, and by induction $u_l \le l \Delta B^{l-1} R$. Take $l = L$. $\blacksquare$

Covering numbers and the rate

An $\epsilon/(LB^{L-1}R)$-net of Θ gives an ϵ-net of $\mathcal{F}$ in supremum norm, hence in d_n. Θ is a ball of radius of order B in $\mathbb{R}^P$, so the volumetric bound of the entropy lecture gives

$$\log N(\epsilon, \mathcal{F}, d_n) \le P \log \frac{3LB^L R}{\epsilon}.$$

This is the **parametric shape** $d \log(A/\epsilon)$ with $d = P$ and $A = 3LB^L R$. Dudley's integral, in the parametric regime of the entropy lecture, gives

$$\mathfrak{R}_n(\mathcal{F}) \le 6\sqrt{\pi} A \sqrt{\frac{P}{n}}.$$

The rate in n is $n^{-1/2}$ and the dimension is the number of parameters. The constant A is exponential in the depth.

If we could minimize

Theorem. Let ℓ be G-Lipschitz in its second argument and bounded by ℓ_∞, and let $\hat{f}_S \in \arg\min_{f \in \mathcal{F}} \widehat{R}_S(f)$. Then

$$\mathbb{E}_{S \sim \mathbb{P}^{\otimes n}}[R(\hat{f}_S)] \le \inf_{f \in \mathcal{F}} R(f) + 4G\mathfrak{R}_n(\mathcal{F}) \le \inf_{f \in \mathcal{F}} R(f) + CGLB^L R\sqrt{\frac{P}{n}}.$$

Proof. The estimation error slide of the Rademacher lecture, then contraction to strip the loss, then the previous slide. ∎

The shape of the bound. The excess risk of a network trained by exact empirical risk minimization decays as $\sqrt{P/n}$. A parametric class gives a parametric rate, exactly as for linear models, and the depth enters only through the constant.

What survives without a minimizer

Theorem. Under the same assumptions, for every $\delta \in (0,1)$, with probability at least $1-\delta$ over S, **every** $f \in \mathcal{F}$ satisfies

$$R(f) \le \widehat{R}_S(f) + 2G\mathfrak{R}_n(\mathcal{F}) + \ell_\infty \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Proof. $R(f) - \widehat{R}_S(f) \le Z(S) := \sup_{f' \in \mathcal{F}} (R(f') - \widehat{R}_S(f'))$. Changing one sample point moves Z by at most ℓ_∞ / n, so McDiarmid gives $Z(S) \le \mathbb{E}_S[Z(S)] + \ell_\infty \sqrt{\log(1/\delta)/(2n)}$ with probability $1-\delta$, and symmetrization then contraction give $\mathbb{E}_S[Z(S)] \le 2G\mathfrak{R}_n(\mathcal{F})$. $\blacksquare$

Why this is useful. The bound is uniform over $\mathcal{F}$, so it holds for whatever weights stochastic gradient descent returns, minimizer or not. The optimization failure moves $\widehat{R}_S(f)$, which is measured on the training set, and leaves the gap untouched.

The same rate from the combinatorial side

For classification, sign of a network, the covering argument can be replaced by the VC route of the previous lecture, and the boundedness of the weights is no longer needed.

Theorem (Bartlett, Harvey, Liaw, Mehrabian). For a ReLU network with P parameters and L layers,

$$\mathrm{VC}(\mathcal{F}) = \Theta\left(PL \log \frac{P}{L}\right).$$

Proof omitted, see Bartlett et al. (2019).

With Haussler's entropy bound and Dudley, $\mathfrak{R}_n(\mathcal{F}) \lesssim \sqrt{PL \log(P/L)/n}$, we can control the excess risk.

Numerically

Setting. CIFAR-10, so $n = 5 \times 10^4$; a ResNet-18, so $P \approx 1.1 \times 10^7$ and $L = 18$; a loss with values in $[0, 1]$. Training error near zero and test error about 0.05, so the observed gap is about 0.05.

Two numbers to judge a bound against. The gap cannot exceed 1, and it is observed to be 0.05.

complexity term	its value at $n = 5 \times 10^4$	n it would need to reach 0.05
parameter counting, $\sqrt{P/n}$	≈ 15	4×10^9
VC, $\sqrt{PL \log(P/L)/n}$	$\approx 2 \times 10^2$	1×10^{12}

They are not loose, they are vacuous. A bound of 2×10^2 on a quantity that cannot exceed 1 asserts nothing, and constants such as $A = 3LB^L R$ only make it larger. The VC term does not fall below 1 until $n = 3 \times 10^9$.

Why no bound over $\mathcal{F}$ can work

The experiment (Zhang, Bengio, Hardt, Recht, Vinyals). Take the same architecture and the same n images, replace every label by an independent uniform one, and train. The network reaches zero training error.

What it proves. The class realizes every labelling of those n points, so it shatters them and $\mathrm{VC}(\mathcal{F}) \geq n$. Every bound of the form $\sqrt{\mathrm{VC}(\mathcal{F})/n}$ is then at least 1, for the architecture actually used, with no room left for a sharper constant.

What it does not prove. That generalization is impossible. The same network on the *true* labels generalizes. What fails is the attempt to explain it by a property of $\mathcal{F}$ alone.

Where the explanation must be. In the algorithm. Stochastic gradient descent does not explore $\mathcal{F}$: it reaches a small, data dependent subset of it, and the bound should be taken over that subset.

What to remember

A network is a composition. Affine layers alternating with a fixed coordinatewise non-linearity. Depth, not layer complexity, is where expressivity comes from, and composing affine maps alone gives an affine map.

Cross-entropy is a divergence. $\ell(p, q) \geq \ell(p, p)$ with equality only at $q = p$, the difference being $\mathrm{KL}(p \| q)$. On one-hot labels it is the negative log-likelihood, so training is maximum likelihood.

Backpropagation is the chain rule, ordered. One backward traversal computes every gradient, at the cost of one forward pass and of storing the activations. Forward differentiation costs one pass per parameter.

Universal approximation. Any continuous function on a compact can be approximated by neural networks.

The rate is parametric. Of the form $R(f) \leq \widehat{R}_S(f) + O(\sqrt{P/n})$.

The constants are wrong. On any real network the bound exceeds the trivial bound 1 by orders of magnitude, and the random label experiment shows no bound over the architecture can do better. Why deep networks generalize is mostly open.