BGSM/CRM
AL&DNN

Training dynamics: Lazy regime and NTK

S. Xambó

UPC & IMTech

19/10/2021

Abstract

Training dynamics: Neural Tangent Kernel (NTK). Kernel gradient. Lazy training. Convergence of the SGD.

References. The focus is on the topics studied in the papers [1] and [2], but we will follow an adaptation of the exposition in [21], mainly §4.2.

Other serviceable materials: the extensive treatise [3] (441 p), the survey [4] (78 p), and the papers [5] (29 p) and [6].

See also [7, Ch. 8]

Topics

Neurons

Neural networks

Training

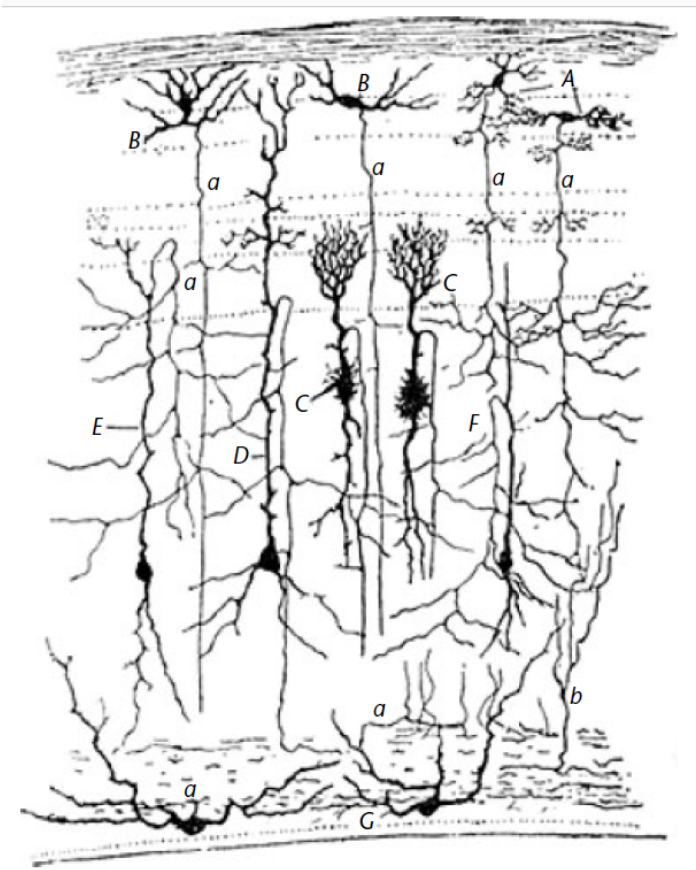
Training techniques

Neural tangent kernel

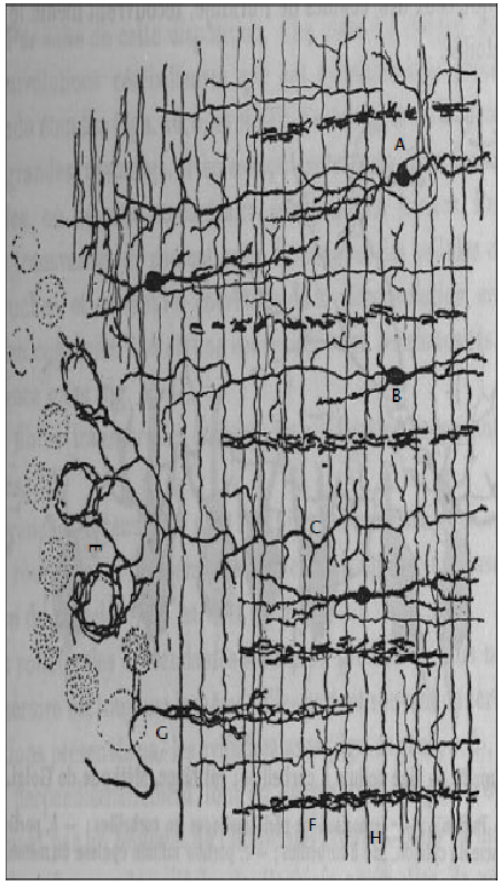
Lazy regime for wide NNs

Neurons

Cajal. Biological model. Artificial model. Activation functions

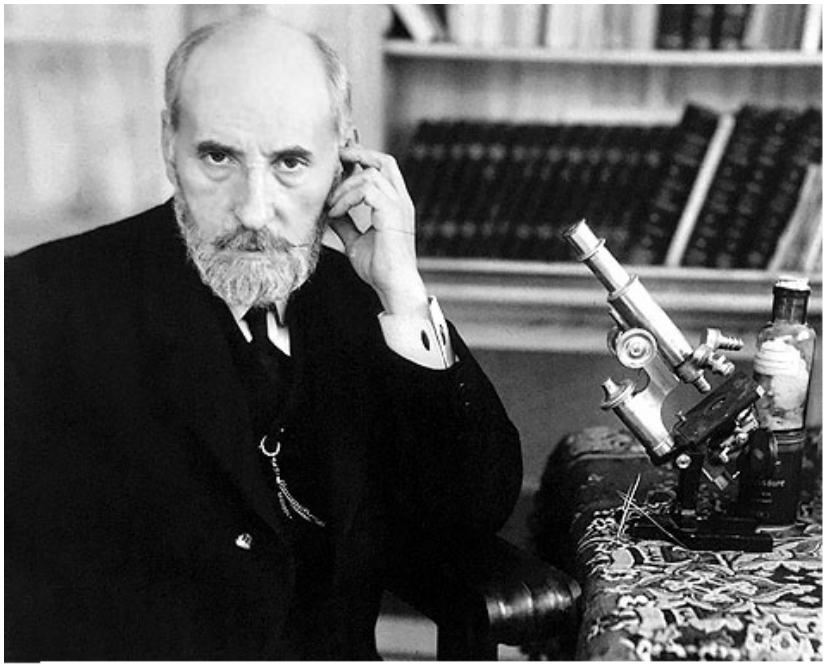


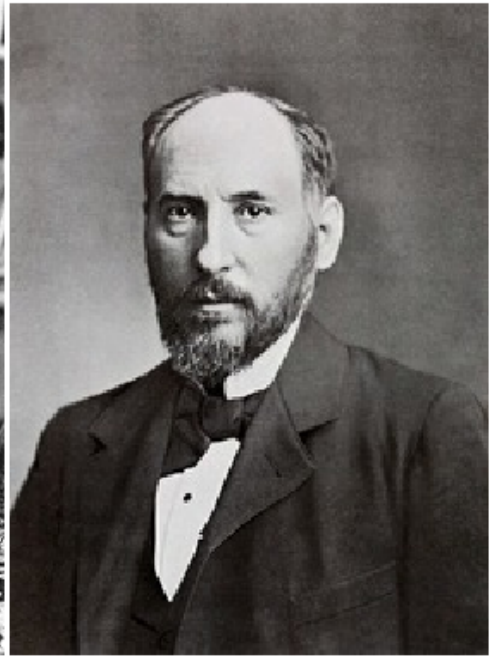
Santiago Ramón y Cajal: different types of neurons in the optic tectum of a bird (Cajal Institute, CSIC)



Cajal: circuitry of the cerebellum. The cell F is the dendrite of a Purkinje cell.

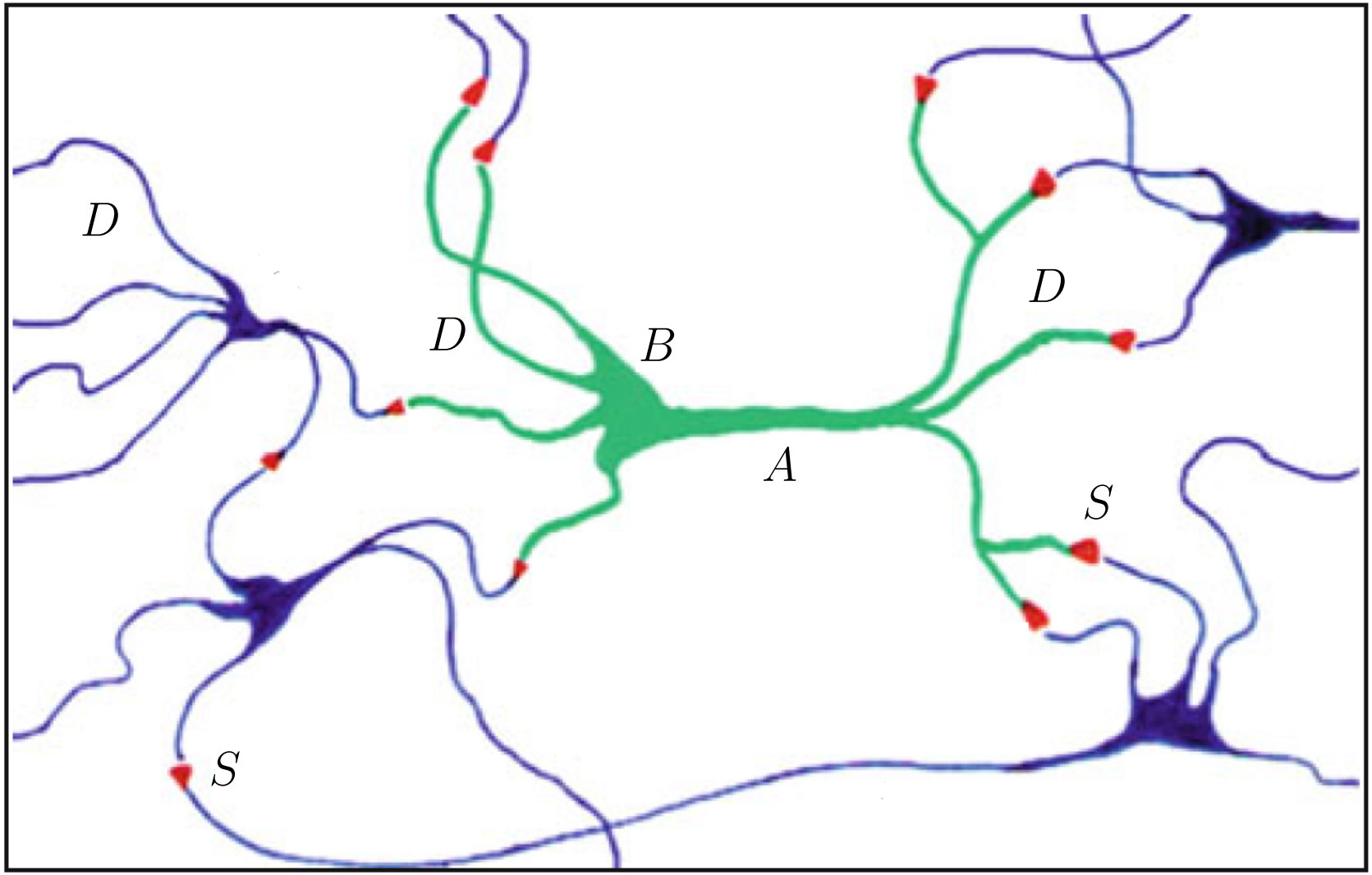
Cajal: Single Purkinje cell





Santiago Ramón y Cajal (1882-1934). Nobel Prize of Physiology and Anatomy (1906, shared with Camillo Golgi) for his discoveries about the structure of the nervous system and the role of the neuron.

In 1887, he moved to **Barcelona** to occupy the chair of Histology created at the Faculty of Medicine of the University of Barcelona. It was in **1888**, defined by Cajal himself as his '**peak year**', when he discovered the mechanisms that govern the morphology and connective processes of gray matter nerve cells of the cerebrospinal nervous system.



B: Body of a neuron. A: Axon. D: dendrites. S: synapsis.

Adapted from Fig. 9.1 in [8] (ertel-2017).

In AL, a useful model of a *neuron* is depicted in Fig. 2.1:

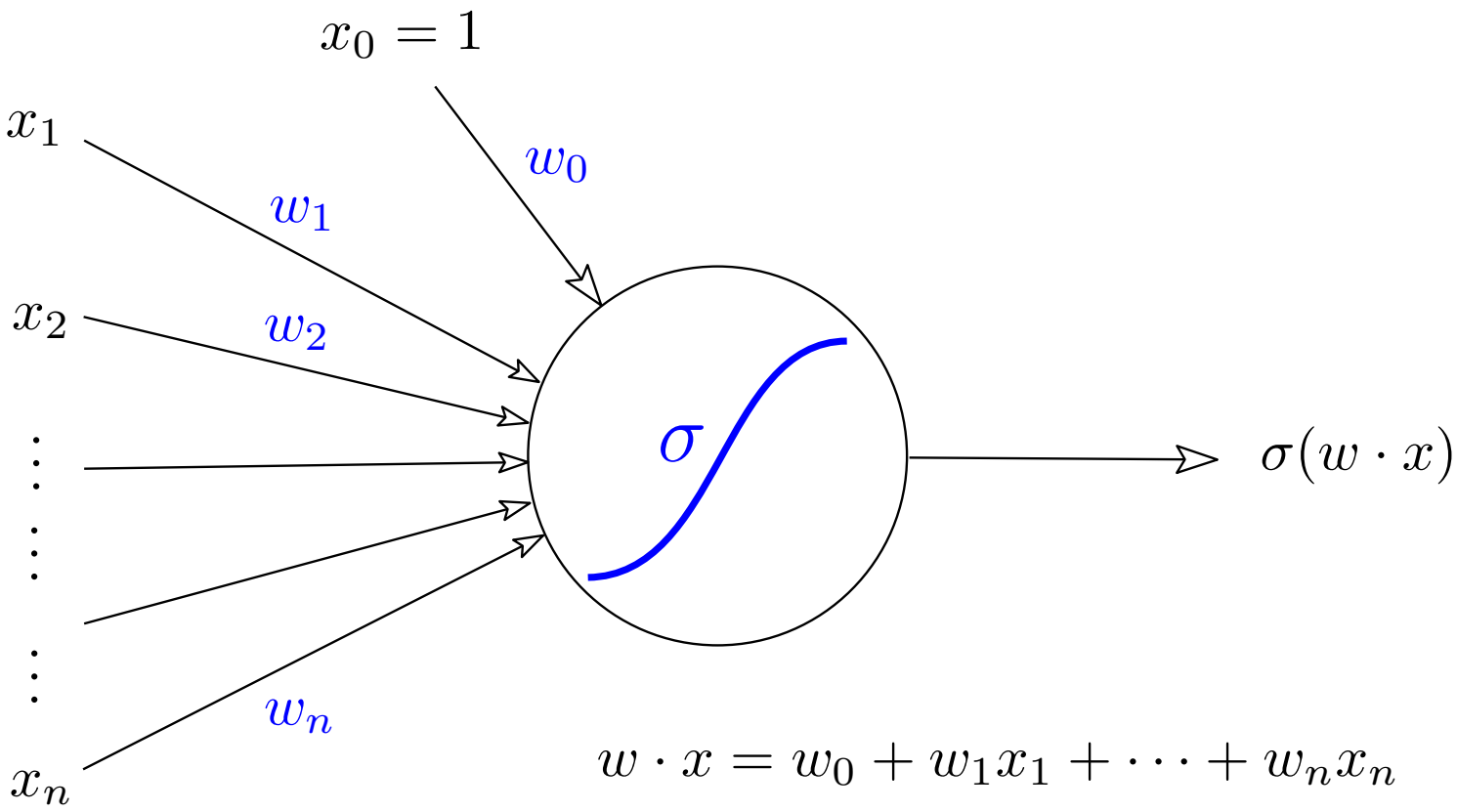


Figure 2.1: Scheme of a neuron. The neuron's output depends on the weights w and on σ (*activation function*), and this functionality is represented by the decorated circle.

To recap in mathematical terms: a neuron is a function

$$x \mapsto f_w(x) = \sigma(x \cdot w), \tag{1}$$

where $w \in \mathbf{R}^n$ (*weights* or *parameters*) and σ is a *sigmoid* function (called *activation function*), like for instance the *logistic function* $\sigma(t) = (1 + e^{-t})^{-1}$, in which case the neuron computes a *logistic regression*.

Augmenting x with $x_0 = 1$ and providing an extra weight w_0 (called the *bias*), the neuron computes $\sigma(w_0 + w_1 x_1 + \cdots + x_n w_n)$.

To display separately the bias and the other weights, we may write f_{w,w_0} or some similar notation.

Linear: $\sigma(x) = x$. Range: $(-\infty, +\infty)$.

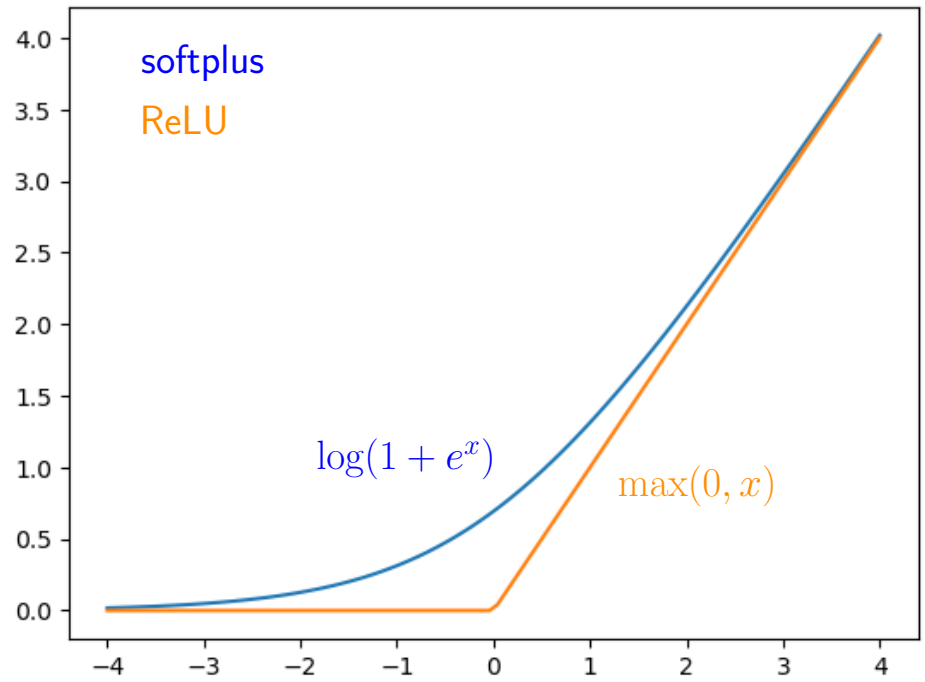
Perceptron: $\sigma(x) = 0$ if $x < \tau$, 1 if $x \geqslant \tau$ (τ: *threshold*).

ReLU: $\max(0, x)$. Range: $[0, +\infty)$.

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$. Range: $(0, 1)$.

Tanh: $\sigma(x) = \tanh(x)$. Range: $(-1, 1)$.

softplus: $\sigma(x) = \ln(1 + e^{x})$



Neural networks

Ground notions. The array model. Training. Conventional training. Overparameterized training. Training techniques


"WHAT DID I INVENT? MY INVENTION IS EVEN MORE REMARKABLE THAN YOURS. IT IS THE SIMPLE DECLARATIVE SENTENCE."

A *neural network* (NN) can be construed as a *composition of neurons* according to a graph of connections called the *architecture* of the net.

Here we will consider the case of *directed graphs* and thus leaving aside nets based on undirected graphs such as those of Hopfield networks and Boltzmann machines. Nor will we discuss networks with feedback (those having closed paths).

The standard architecture of a NN is a directed graph structured in *layers* L_j, as illustrated in Figure 4.1, and its *functional signature* can be condensed as a chain:

$$\mathcal{N}: \quad \text{Input} \to L_0 \overset{f_1}{\longrightarrow} L_1 \overset{f_2}{\longrightarrow} \cdots \to L_{d-1} \overset{f_d}{\longrightarrow} L_d \to \text{Output} \tag{2}$$

The integer d is the *depth* of the net. Conventionally, the net is *deep* if $d > 2$, and *shallow* otherwise. The layers $L_1, \ldots, L_{d-1}$ are considered to be *hidden*, while the input and output layers (L_0 and L_d), are *visible*.

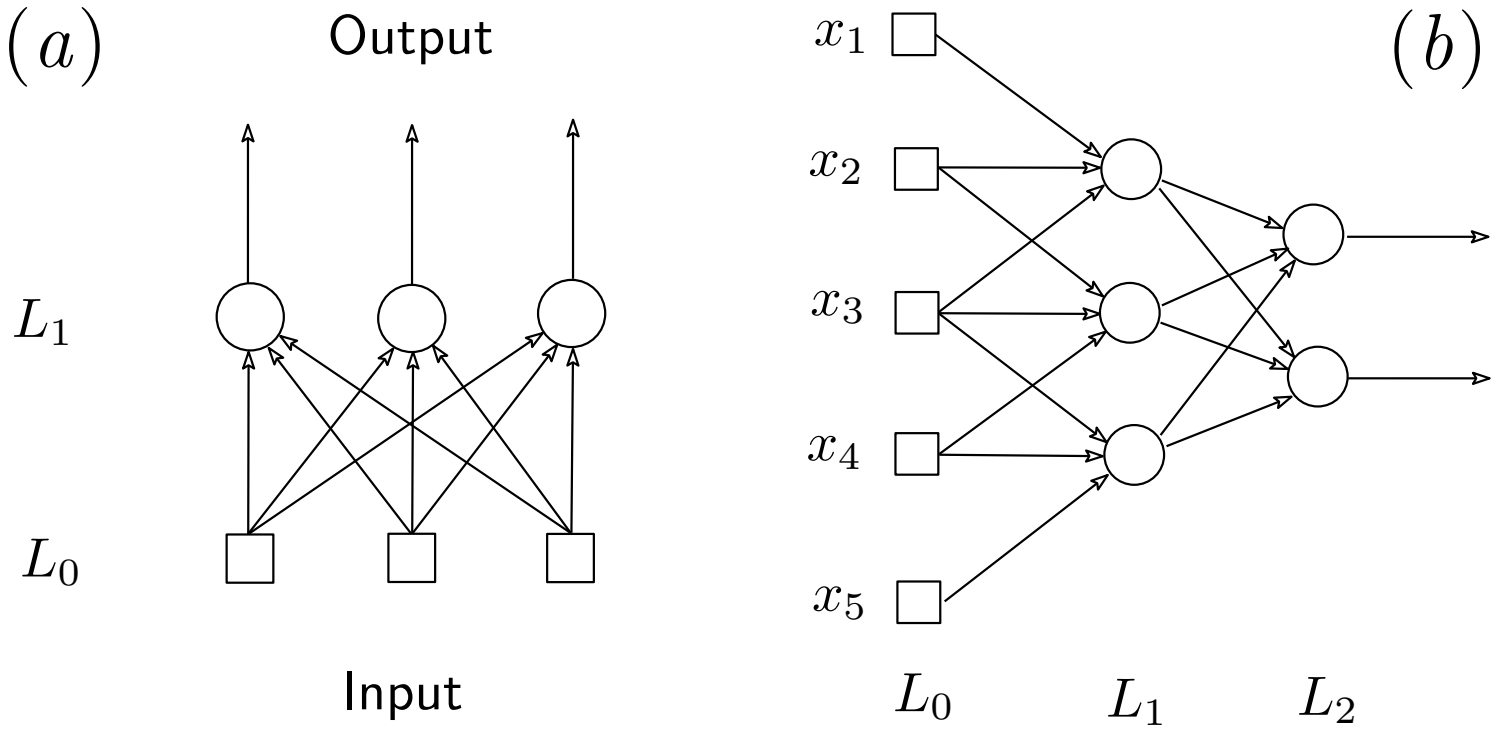


Figure 4.1: (*a*) Neural network with no hidden neurons and *fully connected*. (*b*) Network with a hidden layer L_1 of three neurons fully connected to the two output neurons of L_2. The input layer, L_0, is only partially connected to L_1.

Remark. In Fig. 4.1, each hidden neuron receives the outputs of three of the five input neurons. So the connections from L_0 to L_1 require 9 weights.

There is the possibility that the three hidden neurons share the same weights, thus reducing the number of weights from 9 to 3. This possibility prefigures the convolutional neural networks (CNN, or ConvNets) described later.

Notice that if the three shared weights are $w = (w_1, w_2, w_3)$, the inputs of the hidden neurons are $y_j = \sum_{i=1}^{3} w_i x_{i+j-1}$, $j = 1, 2, 3$, which we recognize as the cross correlation $w \star x$ of w with the input vector x.

Width of L_j, n_j: is the number of its neurons (also called *nodes*).

- Functionally, the layer L_j takes an input x (the output of L_{j-1}) and yields an output x'.
- The map $f_j : x \mapsto x'$ depends on the weights connecting L_{j-1} to L_j, and on the σ. Its particulars define the kind of the layer L_j (the main kinds are described later).
- The input x^0 to L_0 is the signal to be processed (a sound or an image, for example).
- The output of L_d (*output layer*) is the transformation produced by the net on x^0. It is the result of applying progressively the maps $f_1, \ldots, f_d$ (that is, the composition $f_d \circ f_{d-1} \circ \cdots \circ f_1$) to the input.
- The role of L_0 is akin to the sensory organs of living beings. The hidden layers, to the brain structure, and the output, to the signals sent by the brain to the various organs involved in the behavior of the being (like locomotion and phonation, for example).

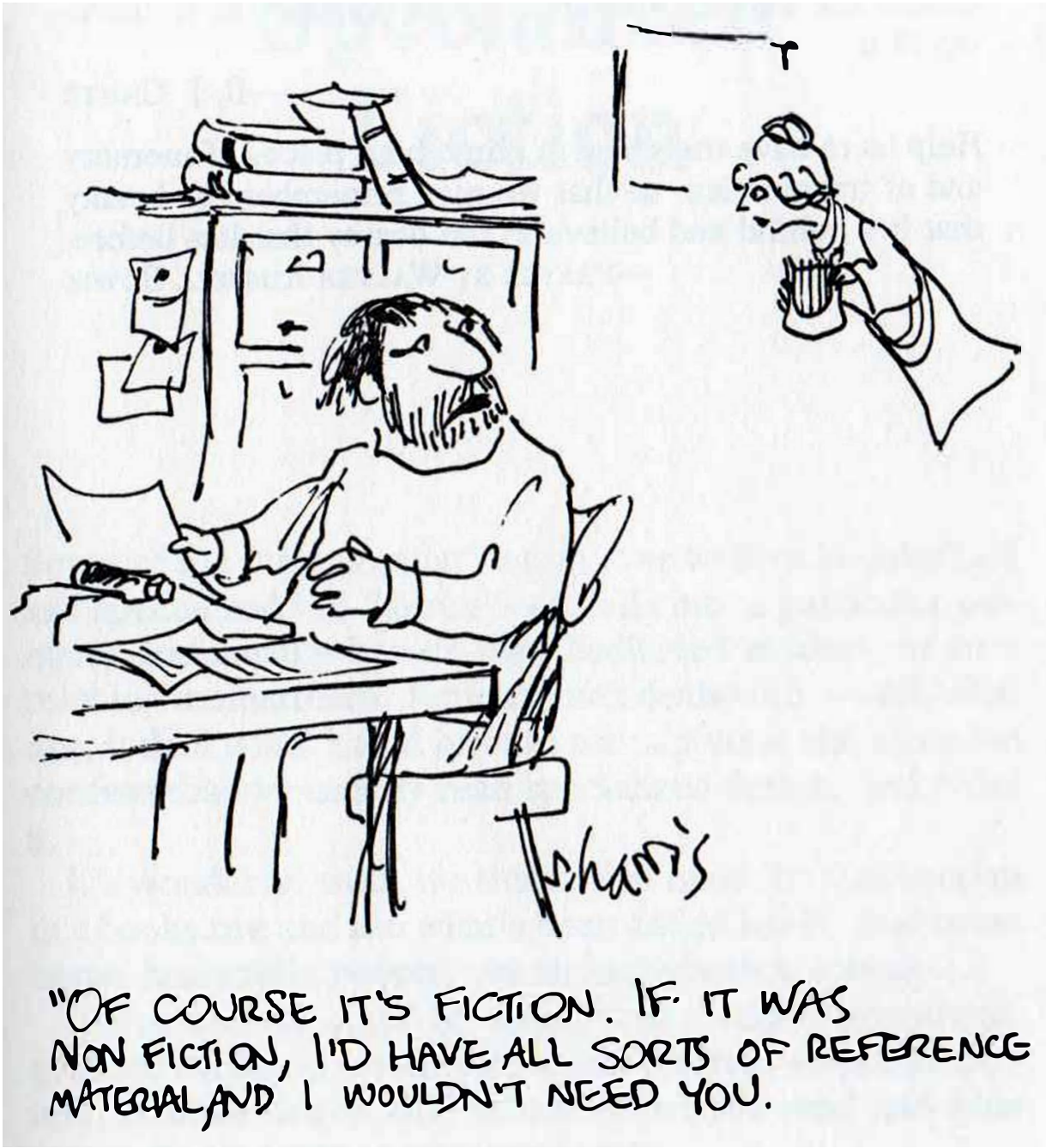
The map $f_j : x \mapsto x'$ is parameterized by the set W_j of weights of the connections of L_{j-1}-neurons to L_j-neurons, so that we can write $f_j = f_{W_j}$.

Consequently, the map computed by the NN is parameterized by the set $W = \cup_j W_j$: $f_W = f_{W_d} \circ \cdots \circ f_{W_1}$.

Since the activation functions of the neurons are (most often) non-linear, f_W may be a highly *non-linear* map.

The number of parameters is generally large or very large, the more so the wider and deeper the net is. At present the number of parameters of the largest NNs are approaching 10^{12}.

In biological terms, these weights play the role of the *synaptic potentials* of the neocortex, but these still outnumber by more than two orders of magnitude the biggest number of artificial connections.


"OF COURSE IT'S FICTION. IF IT WAS NON FICTION, I'D HAVE ALL SORTS OF REFERENCE MATERIAL AND I WOULDN'T NEED YOU.

In general, x and x' in $x' = f_j(x)$, and the *layer parameters* W_j, b_j (*weights* and *biases*), are *multidimensional arrays* whose nature is chosen according to the processing that has to be achieved.

Write $[n_1, n_2, \ldots, n_d]$ to denote the type of a d-*axial* (real) array with axis dimensions $n_1, \ldots, n_d$.

Thus $[n]$ is the type of n-dimensional vectors and $[n_1, n_2]$ the type of matrices with n_1 rows and n_2 columns.

Matrices are useful to represent monochrome images, but for RGB images we need arrays of type $[n_1, n_2, 3]$, or $[n_1, n_2, n_3]$ if it is required that the image be represented by n_3 *channels*.

The parameters associated to *fully connected* and *convolutional* layers are encoded by an *array of weights*, W, and a bias array, b.

In these cases, the expression of f has the form

$$f(x) = \sigma\left(x \star_\pi W + b\right) \tag{3}$$

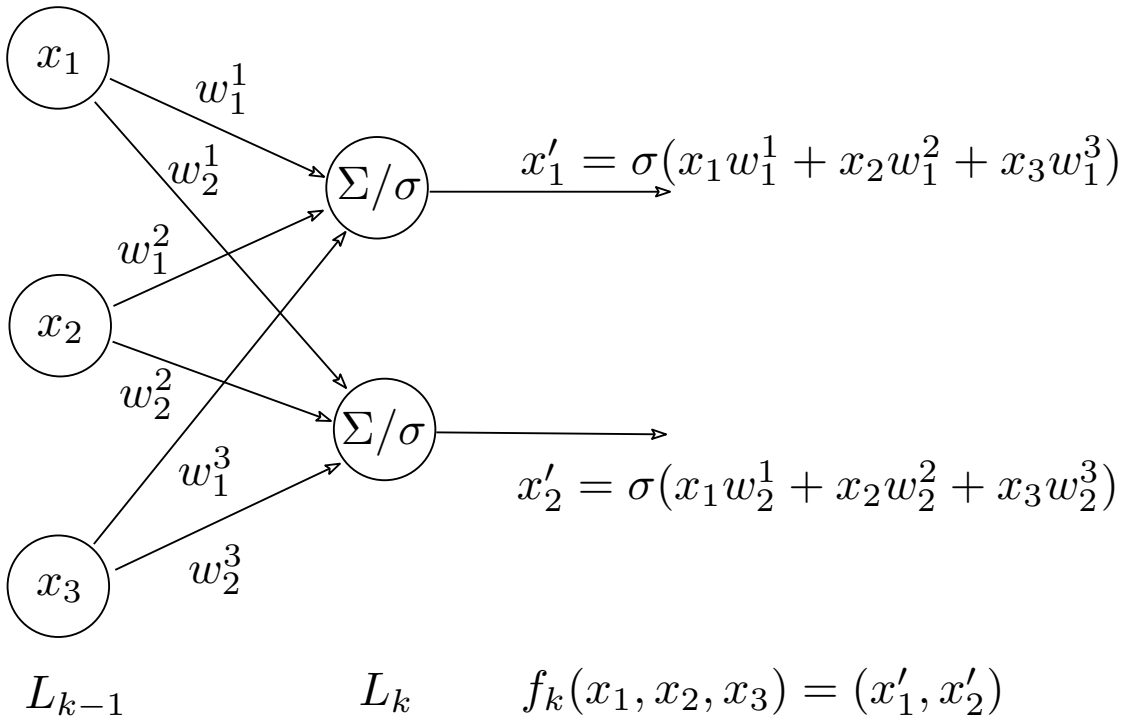
where $\star_\pi$ is a pairing specific of the layer and σ an activation function that is *applied component-wise* to arrays. The expression $\tilde{f}_j(x) = x \star_\pi W + b$ is called the *preactivation* of the layer. Clearly, $f_j(x) = \sigma(\tilde{f}_j(x))$.

Remark: Instead of a sigmoid activation, it has become practical to use a *rectified linear unit* (ReLU), $\max(0, x)$. Its advantages are that it is continuous and piecewise linear, that it is not bounded above, and that it works fine when its derivative is needed (the *jump function* $x \mapsto 0$ if $x < 0$, 1 if $x \geqslant 0$ (met earlier, on page 10, as *peceptron activation*).

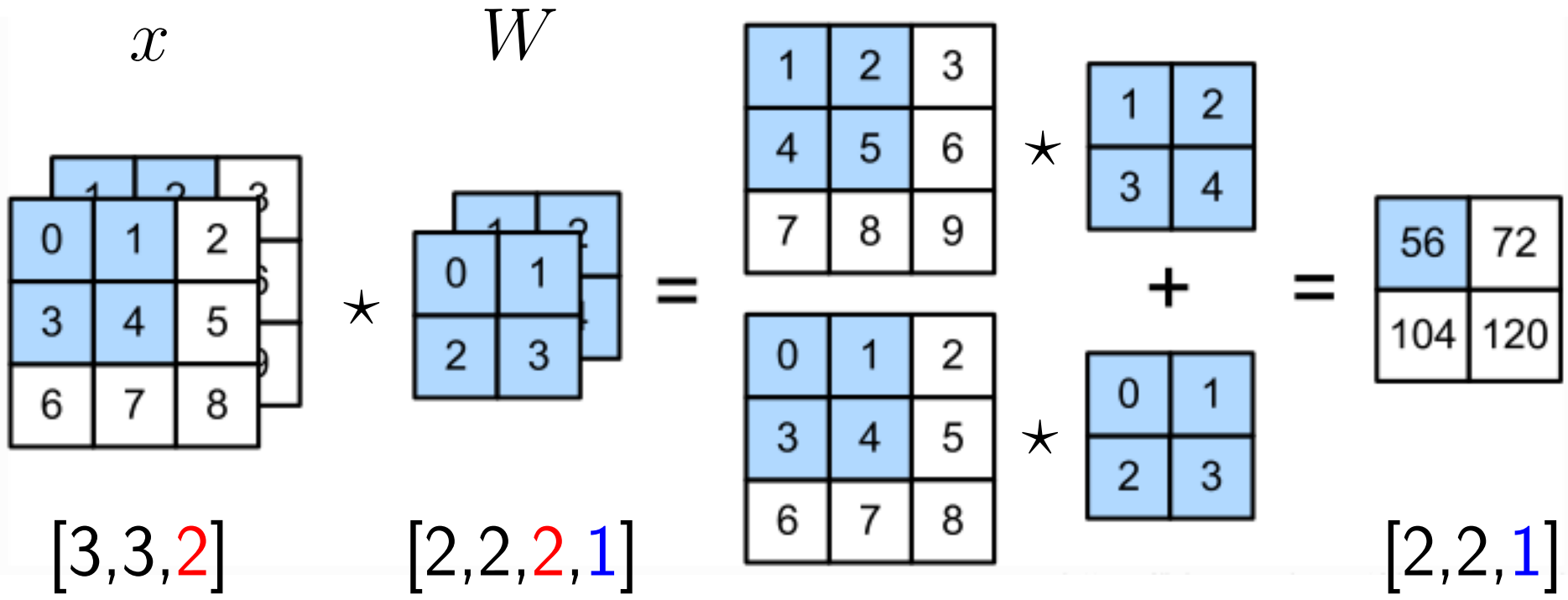
For fully connected layers, x and x' are vectors, W is a matrix and $\star_\pi$ is *matrix product* (xW).

Explicitly: if $x \in \mathbf{R}^{n_{k-1}}$ is the output of the layer L_{k-1}, then $W \in \mathbf{R}^{n_{k-1}}_{n_k}$, and $x' \in \mathbf{R}^{n_k}$ is the output of the layer L_k, $x' = \sigma(xW)$. In detail,

$$x'_j = \sigma(\textstyle\sum_{i=1}^{n_{k-1}} x_i W^i_j),\ j \in [n_k].$$



For convolutional layers, $\star_\pi = \star$ is *array cross-correlation*.



In the *cross-correlation* product $y = x \star W$, x is an array of type $[n_1, n_2, n_3]$; W (the *filter*, or *mask*) an array of type $[w_1, w_2, n_3, m_3]$.

The pair (n_1, n_2) is the shape of the geometric dimensions of x and n_3 the number of channels. The pair (w_1, w_2) denotes the *window dimensions* of the filter and m_3 the number of channels of the output array y. The type of y is $[n_1 - w_1 + 1, n_2 - w_2 + 1, m_3]$.

$$y[i,j,k] = \sum_{m=0}^{w_1-1} \sum_{n=0}^{w_2-1} \sum_{r=0}^{n_3-1} x[i+m, j+n, r] W[m,n,r,k] \qquad (4)$$

which can be expressed more compactly as

$$y[i,j,k] = \sum_{r=0}^{n_3-1} x\,[i : i+w_1-1, j : j+w_2-1, r] * W[:,r,k] \qquad (5)$$

where we use the standard slicing conventions for arrays and $*$ denotes the ordinary *scalar product of matrices*.

There is a *downsampled cross-correlation* $y = x \star_s W$ by a *stride* s:

$$\begin{aligned} y[i,j,k] &= \sum_{r,m,n} x[is+m, js+n, r] W[m,n,r,k] \\ &= \sum_{r} x\left[is : is+w_1-1, js : js+w_2-1, r\right] * W[:,:,r,k] \end{aligned}$$

The shape of the array $x \star_s W$ is $[n_1', n_2', m_3]$, where n_1' and n_2' are the greatest integers such that $n_1' \leq (n_1 - w_1)/s$ and $n_2' \leq (n_2 - w_2)/s$.

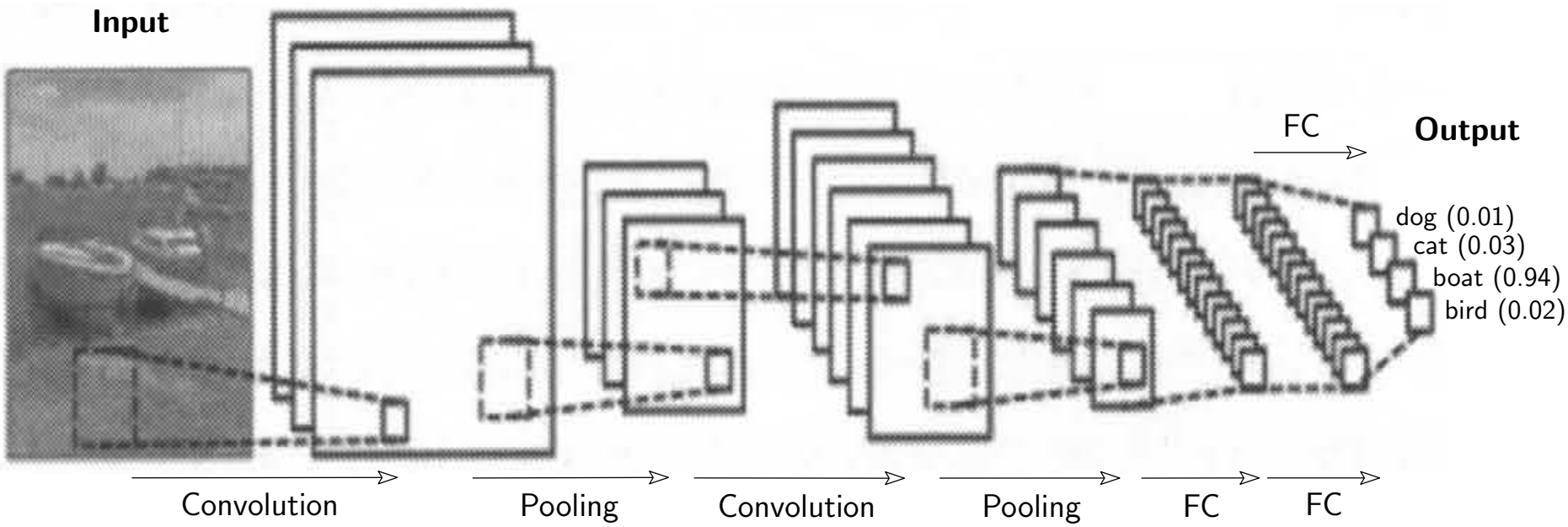
Remark. When a NN has at least one convolutional layer, we qualify it as a *convolutional* NN (CNN for short).

For a maximum pooling (*maxpool*) layer, the parameters are represented by a triple of positive integers $(w_1, w_2, s = 1)$, where (w_1, w_2) is the shape of the pooling window and s is the stride (1 *by default*).

In this case $\star_\pi = \star_{\mathrm{mp}}$ is given by the rule

$$(x \star_{\mathrm{mp}} W)[i, j, k] = \max(x[is : is + w_1 - 1, js : js + w_2 - 1, k]).$$

The shape of the array $x \star_{\mathrm{mp}} W$ is $[n_1', n_2', n_3]$, where n_1' and n_2' are the greatest integers such that $n_1' \leq (n_1 - w_1)/s$ and $n_2' \leq (n_2 - w_2)/s$.

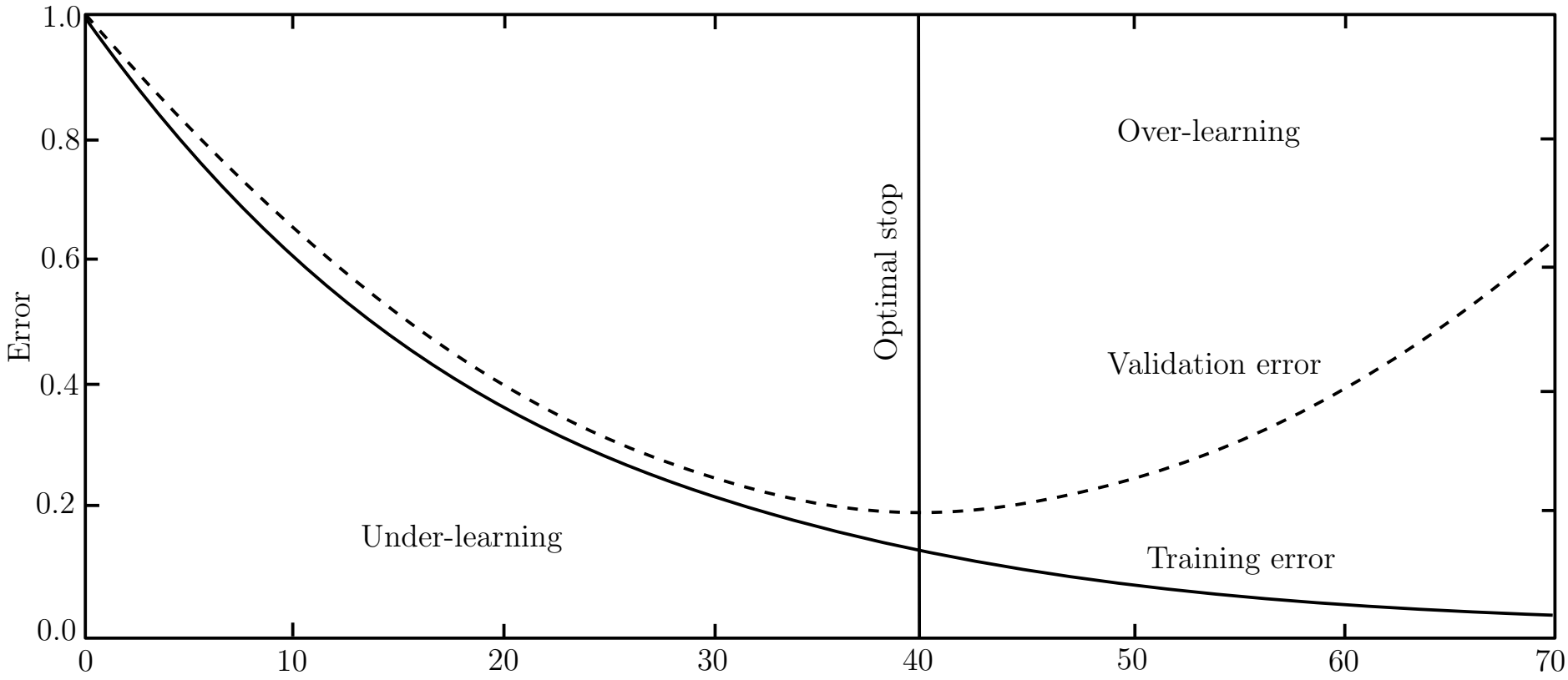

Input
FC
Output
dog (0.01)
cat (0.03)
boat (0.94)
bird (0.02)
Convolution
Pooling
Convolution
Pooling
FC
FC


...and in that second paragraph, there should be a comma after 'tomorrow'. In addition, 'rhythm' is not spelled correctly...

A *training algorithm* for the network (2) using a labeled dataset $\mathcal{D}^*$ is any procedure to adjust the weights W_j and biases b_j so that the function $f_{W_d,b_d} \circ \cdots \circ f_{W_1,b_1}$ computed by the net has *a good balance of the learning and generalization rates*.

This is usually done by iterating two steps (which together form an *epoch*):

- a *forward pass*, ending with a measure (*loss*) of how close the output is to what it should be, and
- a *backward pass*, to modify the parameters in order to *decrease* the loss incurred in the forward step. This is mostly achieved by (variations of) *gradient descent*, particularly SGD.



When the number of parameters is less than the number of data samples, there is an unavoidable *trade-off* : the *learning* (decreasing of the *learning error*) and the *generalization* (decreasing of the *validation error*) can both increase for a while, but there is a *turning point* beyond which the learning keeps improving but the generalization enters a steady degrading.

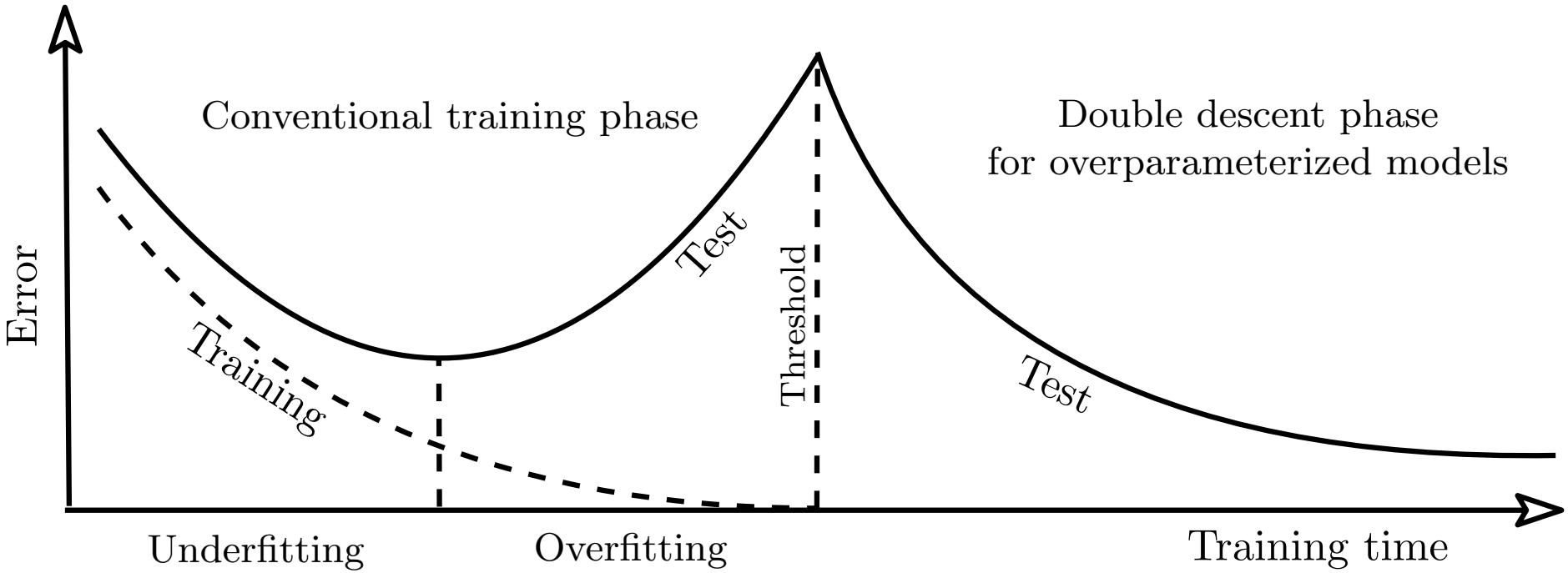
Before the turning point we have *underfitting*, in the sense that the learning and the generalization rates can still improve.

After the turning point we have *overfitting*, in the sense that we are forcing the net to have a higher learning rate at the expense of a poorer generalization rate (under these circumstances, rote learning of the data does not favor the generalization capacity).

This is the *underparemeterized scenario* (capacity, as measured by the number of weights, below the number of data samples) and it was the accepted wisdom until not long ago.

The question of what happens with overparameterized networks, a scenario favored by the increasing computing power, has been addressed in the last few years and the answers so far are *surprising breakthroughs*.

Prominent among such discoveries is the double descent phenomenon described in [9], which shows that for overparameterized NNs the training follows the pattern explained above until reaching zero training error, corresponding to a threshold of maximal testing error, and *then the test error starts decreasing steadily and becomes smaller than the relative minimum achieved before the threshold*.



Adapted from Fig. 1(b) in [9].

To learn more about this fascinating behavior, see [10] (on the *role of kernel learning in deep learning*), [11] (*two models of double descent*), and also [12] (an important *first step in understanding the phenomenon of double-decent*).

Further references for NNs: [13]* (*overview of DL in NNs*), [14] (DL with SVM), [15] (mathematical underpinnings of CNN), [16] (deep versus shallow performance of NNs), [17] (mathematics of DL), [18] (universality of deep CNNs). And a quotation:

[19] (donoho-2000): "The *blessings of dimensionality* are less widely noted, but they include the concentration of measure phenomenon (so-called in the geometry of Banach spaces), which means that certain random fluctuations are very well controlled in high dimensions and the success of asymptotic methods, used widely in mathematical statistics and statistical physics, which suggest that *statements about very high-dimensional settings may be made where moderate dimensions would be too complicated*."

The goal is to minimize the empirical loss $\hat{L}(w) = \frac{1}{m}\sum_{j=1}^{m} \hat{L}_j(w)$, where $\hat{L}_j(w)$ only depends on j-th (labeled) data item, (x^j, y^j). Typically, $\hat{L}_j(w) = (f_w(x^j) - y^j)^2$.

- **GD** on $\hat{L}(w)$ (uses the entire data set):

$$w = w - \eta \nabla \hat{L}(w) = w - \frac{\eta}{m} \sum_{j=1}^{m} \nabla \hat{L}_j(w).$$

- **SGD**: Relies on a *stochastic approximation* of the gradient. Typically by using a random subset of the data (*random minibatch*).

In On-line GD a single data item is used:

$$w = w - \eta \nabla \hat{L}_i(w).$$

Several passes (*epochs*) may be needed for convergence. A variable learning rate may speed up convergence.

On-line learning procedure (one epoch)

[Randomly shuffle items in the data set]

Input: Initial w

for $j \in [m]$:

$$w = w - \eta \nabla \hat{L}_j(w)$$

return w.

Mini-batch learning procedure (one epoch)

Input: Initial w

split $[m]$ into mini-batches $J_1, \ldots, J_s$

for $k = 1, \ldots, s$:

$$w = w - \eta \frac{1}{|J_k|} \sum_{j \in J_k} \nabla \hat{L}_j(w)$$

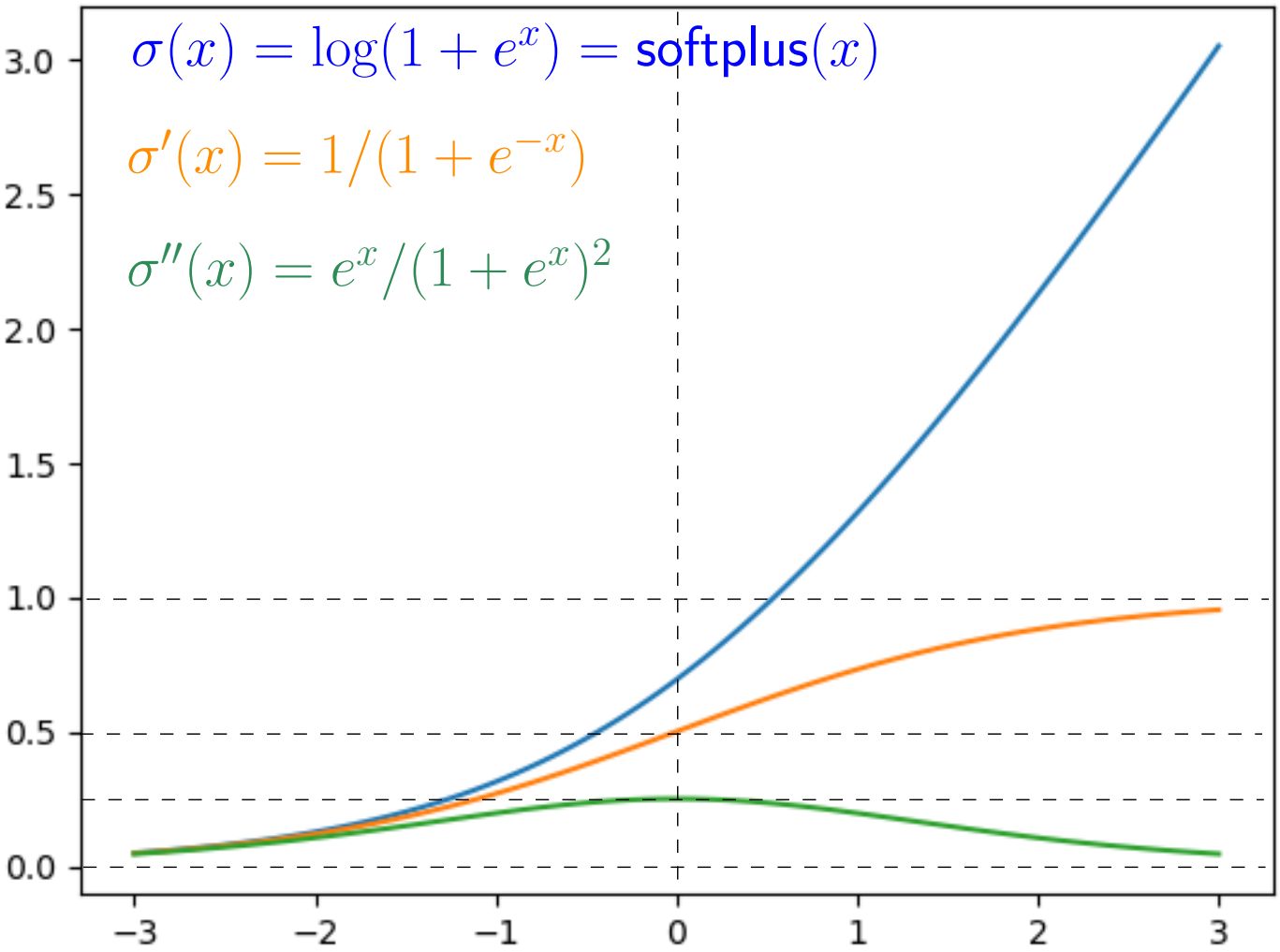
return w.

For *adaptative* policies on the learning rate, see [20].

Neural Tangent Kernel (NTK)

The setup. Relation to GD.
Gradient flux and the tangent kernel.
(After [1], as interpreted in [21, §4.2])

- FC NN of depth d and widths $n_0, \ldots, n_d$.
- Activation $\sigma(x)$: Twice differentiable with bounded second derivative.



- $N = \sum_{j=0}^{d-1}(n_j + 1)n_{j+1}$: Dimension of the *parameter space*, $\mathbf{R}^N$ (*weight space*).
- Instead of w, points in $\Theta = \mathbf{R}^N$ will be denoted θ. Note: $f_\theta : \mathbf{R}^{n_0} \to \mathbf{R}^{n_d}$.
- *Function space*, $\mathcal{F}$: Maps $f_\theta : \mathbf{R}^{n_0} \to \mathbf{R}^{n_d}$.
- $\phi : \Theta \to \mathcal{F}, \theta \mapsto f_\theta$: *parameterization of* $\mathcal{F}$.

$E : \mathcal{F} \to \mathbf{R}$: *Cost functional*. $E(f)$ only depends on the values of f on the data points. Here E stands for the empirical loss, previously denoted by $\hat{L}$.

In this setup, *training* amounts to optimizing f_θ in $\mathcal{F}$ with respect to the cost E.

$\tilde{E} = E \circ \phi : \Theta \to \mathbf{R}$. Generally *non-convex*, even if E is convex (cf. [22]).

GD for $\tilde{E}$, starting at θ_0, reads:

$$\theta_{k+1} = \theta_k - \eta \nabla \tilde{E}(\theta_k). \tag{6}$$

Eq. (6) can be reinterpreted geometrically from the linear approximation of $\tilde{E}$ around θ_k. Assuming that $\tilde{E}$ is β-regular and $\eta < \beta^{-1}$, we have

$$\tilde{E}(\theta) \leq \tilde{E}(\theta_k) + \nabla \tilde{E}(\theta_k) \cdot (\theta - \theta_k) + \eta^{-1} \|\theta - \theta_k\|^2 \ ,$$

which allows us to rewrite (6) in *variational form*:

$$\theta_{k+1} = \operatorname{argmin}_\theta \ \tilde{E}(\theta_k) + \nabla \tilde{E}(\theta_k) \cdot (\theta - \theta_k) + \eta^{-1} \|\theta - \theta_k\|^2. \quad (7)$$

This method allows to find local minima of $\tilde{E}$ with *no curse of dimensionality*: for an error $\epsilon > 0$, $\tilde{O}(\epsilon^{-2})$ iterations suffice to find an ϵ-approximation of a local minimum (see [23]). Here $\tilde{O}(\cdot)$ is like $O(\cdot)$, but ignoring logarithmic factors.

The GD (6) can be regarded as an Euler discretization of the ODE

$$\dot{\theta} = -\nabla \tilde{E}(\theta) \ ,$$

with a random initial condition $\theta(0)$ sampled from a certain probability distribution on the parameter space Θ.

This equation, known as *gradient flux*, defines a continuous dynamic $\theta(t) \in \Theta$, $t \geq 0$. The corresponing dynamics on the function space $\mathcal{F}$ is $h(t) = \phi(\theta(t))$. Now the *chain rule* implies that

$$\dot{h} = -\mathcal{K}(t) \cdot \nabla E(h(t)) \ , \tag{8}$$

with $\mathcal{K}(t) = D\phi(\theta(t))^\top D\phi(\theta(t))$ (*tangent kernel*), where $D\phi(\theta)$ is the differential of ϕ at θ (a linear map from $T_\theta \Theta = \mathbf{R}^N$ to $T_{\phi(\theta)}\mathcal{F}$).

The difficulty of the mathematical analysis of Eq. (8) is understanding the dependency of the tangent kernel $\mathcal{K}(t)$ with respect to t.

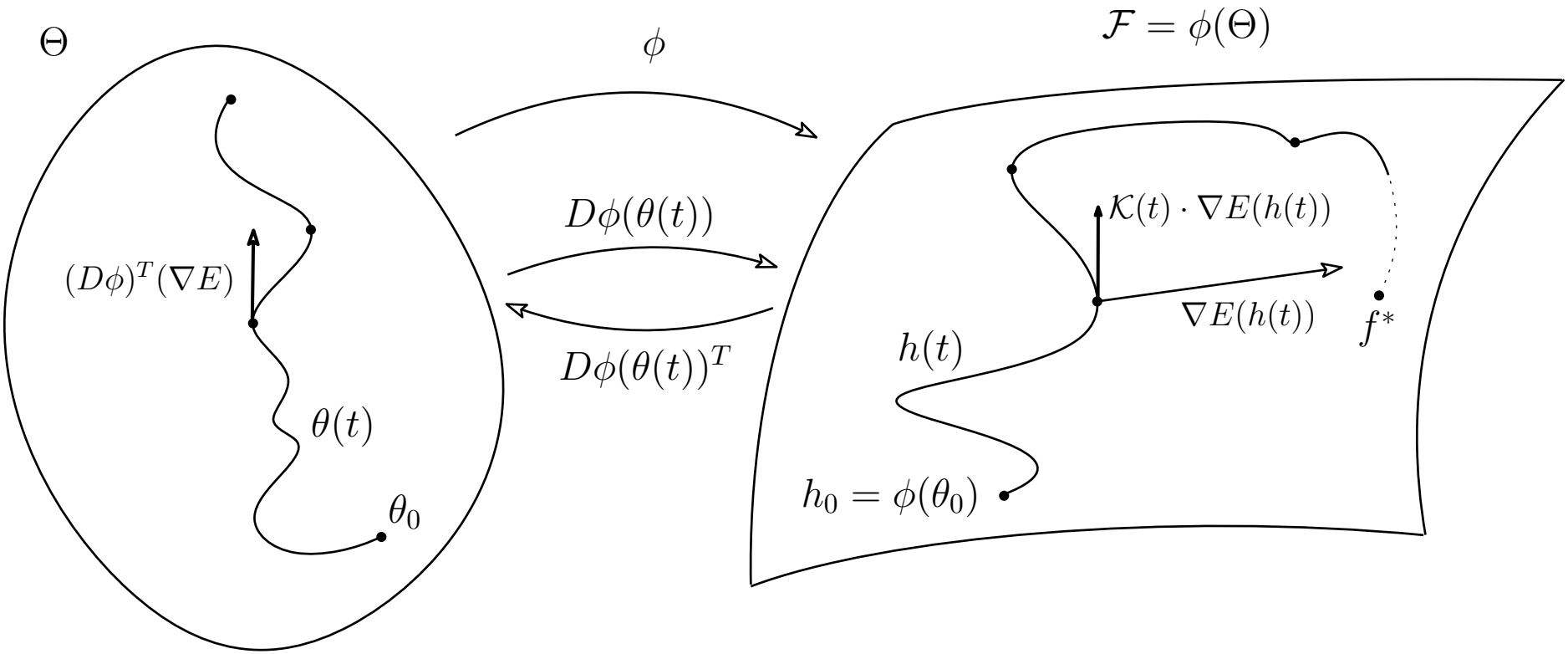


Figure 6.1: Illustration on the GD and the tangent kernel: Geometric and analytic relations between the weight space Θ and the function space $\mathcal{F}$. In general, the expert or supervisor f^* does not belong to $\mathcal{F}$.

In fact, the temporal variation of $\mathcal{K}(t)$ can be understood geometrically as the curvature of the function space $\mathcal{F}$ at the point $h(t)$. In the case of a FC NN as in Eq. (2), an important result is (cf. [1, 2, 24, 25]) that this curvature tends to 0 when the net widths tend to ∞, with a suitable weight normalization, and $\mathcal{K}(t) \to \mathcal{K}(0)$ uniformly in finite time (cf. [1]).

For example, in the case of a shallow net, consider

$$\phi(\theta, x) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \chi(\theta_j, x) \ ,$$

where $\chi(\theta, x)$ is the *neuron* defined by (1). If the parameters θ_j are sampled iid according to a distribution μ_0, the tangent kernel turns out to be

$$\begin{aligned} \mathcal{K}(t)[x, x'] &= \frac{1}{m} \sum_{j=1}^{m} \nabla_\theta \chi(\theta_j(t), x) \nabla_\theta \chi(\theta_j(t), x') \\ &\to \mathbb{E}_{\theta \sim \mu_0} [\nabla_\theta \chi(\theta, x) \nabla_\theta \chi(\theta, x')] \\ &:= \overline{\mathcal{K}}(x, x') \quad (\text{width} \to \infty) \ . \end{aligned}$$

Under fairly general conditions (cf. [26]), the tangent kernel of a finite width net concentrates uniformly toward $\overline{\mathcal{K}}$, with fluctuacions of order $\sim \frac{1}{\sqrt{m}}$. In the asymptotic regime of width tending to ∞, and considering for simplicity the loss $E(h) = \frac{1}{2}\|h - f^*\|^2$, the training dynamics gets simplified to

$$\dot{f} = -\overline{\mathcal{K}}\nabla E(f(t)) = -\overline{\mathcal{K}}(f(t) - f^*) \ ,$$

which corresponds to the linear dynamics associated to a linear regression model in the RKHS associated to the $\overline{\mathcal{K}}$. This space allows a more precise study of questions concerning approximation, generalization and optimization.

With this parameterization, the wide NNs behave as linear models, characterized by a tangent kernel that remains constant after initialization (*lazy training*). If it is true that this phenomenon allows to understand the reason of the good behavior of GD, it does not explain mathematically the advantages of the non-linear models defined by NNs. This will be studied in next session.


"I'VE GOTTEN QUITE A BIT OUT OF THAT CREATIVE WRITING CLASS. I THINK I'LL BECOME A LITERARY AGENT."

References I

[1] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," 2020.

arXiv:1806.07572, v4.

[2] L. Chizat, E. Oyallon, and F. Bach, "On lazy training in differentiable programming," 2020.

arXiv:1812.07956,v5.

[3] D. A. Roberts, S. Yaida, and B. Hanin, *The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks*.

Cambridge University Press, 2021.

Forthcoming. Draft: https://arxiv.org/pdf/2106.10165.pdf.

References II

[4] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen, "The modern mathematics of deep learning," 2021.

`https://arxiv.org/pdf/2105.04026.pdf`.

[5] M. Geiger, L. Petrini, and M. Wyart, "Perspective: A phase diagram for deep learning unifying jamming, feature learning and lazy training," 2020.

`https://arxiv.org/pdf/2012.15110.pdf`.

References III

[6] S. d'Ascoli, M. Refinetti, G. Biroli, and F. Krzakala, "Double trouble in double descent: Bias and variance (s) in the lazy regime," in *International Conference on Machine Learning*, pp. 2280–2290, PMLR, 2020.

`http://proceedings.mlr.press/v119/d-ascoli20a/d-ascoli20a.pdf`. The preprint `https://arxiv.org/pdf/2003.01054.pdf` contains supplementary materials not included in the published paper.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*.

MIT press, 2016.

`http://www.deeplearningbook.org`.

References IV

[8] W. Ertel, *Introduction to artificial intelligence*.

Springer, 2017.

[9] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning and the bias-variance trade-off," *stat*, vol. 1050, p. 28, 2018.

`https://arxiv.org/pdf/1812.11118.pdf`. V2: 2019.

[10] M. Belkin, S. Ma, and S. Mandal, "To understand deep learning we need to understand kernel learning," in *International Conference on Machine Learning*, pp. 541–549, PMLR, 2018.

`http://proceedings.mlr.press/v80/belkin18a/belkin18a.pdf`.

References V

[11] M. Belkin, D. Hsu, and J. Xu, "Two models of double descent for weak features," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 4, pp. 1167–1180, 2020.

[12] S. Mei and A. Montanari, "The generalization error of random features regression: Precise asymptotics and double descent curve," 2019.

`https://arxiv.org/pdf/1908.05355.pdf`.

[13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[14] Y. Tang, "Deep learning using linear support vector machines," *https://arxiv.org/pdf/1306.0239.pdf, v4*, 2015.

References VI

[15] S. Mallat, "Understanding deep convolutional networks," *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, p. 20150203, 2016.

[16] H. Mhaskar, Q. Liao, and T. Poggio, "Learning functions: when is deep better than shallow," *arXiv: 1603. 00988*, 2016.

[17] R. Vidal, J. Bruna, R. Giryes, and S. Soatto, "Mathematics of deep learning," *arXiv: 1712. 04741*, 2017.

[18] D.-X. Zhou, "Universality of deep convolutional neural networks," *Applied and Computational Harmonic Analysis*, 2019.

References VII

[19] D. L. Donoho *et al.*, "High-dimensional data analysis: The curses and blessings of dimensionality," *AMS math challenges lecture*, vol. 1, no. 2000, p. 32, 2000.

`https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.329.3392&rep=rep1&type=pdf`.

[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[21] J. Bruna and S. Xambó-Descamps, "Aprenentatge algorísmic i xarxes neuronals profundes," Butlletí de la SCM, vol. 36, no. 1, pp. 5–67, 2021.

References VIII

[22] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Artificial intelligence and statistics*, pp. 192–204, PMLR, 2015.

[23] C. Jin, P. Netrapalli, R. Ge, S. M. Kakade, and M. I. Jordan, "On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points," *Journal of the ACM (JACM)*, vol. 68, no. 2, pp. 1–29, 2021.

[24] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," 2019.

Published as a conference paper at ICLR 2019: `https://arxiv.org/pdf/1810.02054.pdf`.

References IX

[25] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *International Conference on Machine Learning*, pp. 1675–1685, PMLR, 2019.

http://proceedings.mlr.press/v97/du19c/du19c.pdf.

[26] A. Rahimi, B. Recht, *et al.*, "Random features for large-scale kernel machines.," in *NIPS*, vol. 3, p. 5, 2007.

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.8736&rep=rep1&type=pdf.