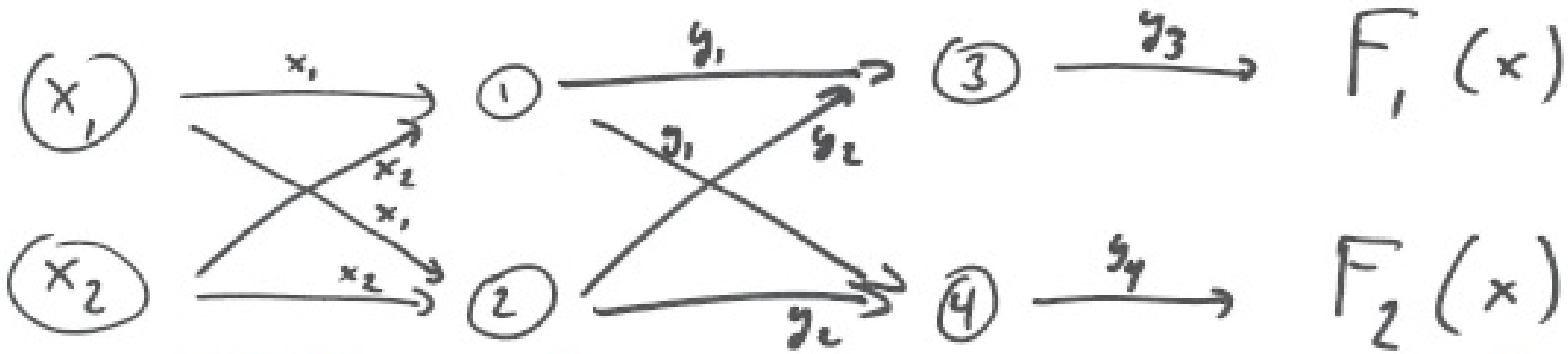
Example



① $w^1 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$, $b^1 = 1$ $\longrightarrow$ $y_1 = \sigma(2x_1 + (-1)x_2 + 1)$

② $w^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $b^2 = -1$ $\longrightarrow$ $y_2 = \sigma(1\,x_1 + 1\,x_2 + (-1))$

③ $w^3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, $b^3 = 0$ $\longrightarrow$ $y_3 = \sigma((-1)y_1 + 1\,y_2 + 0)$

④ $w^4 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$, $b^4 = 2$ $\longrightarrow$ $y_4 = \sigma((-2)y_1 + 1\,y_2 + 2)$

$$\sigma(x): \text{ReLU} \Rightarrow \sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Let $x = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix} \Rightarrow$

$$y_1 = \sigma(0.8) = 0.8$$

$$y_2 = \sigma(-0.2) = 0$$

$$y_3 = \sigma(-0.8) = 0$$

$$y_4 = \sigma(0.4) = 0.4$$

$$F(x) = \begin{bmatrix} y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.4 \end{bmatrix} \Rightarrow k = 2$$

(because $\max\{0, 0.4\} = 0.4$)

parameters : $w^1, b^1, w^2, b^2, w^3, b^3, w^4, b^4$

Training a NN is the process of finding good values for the parameters w^i, b^i of each neuron.

A **cost function** is a function of all the parameters that has a low value if NN gives a good classification of the pre-labeled data (**training set**).

Typical cost function:

Given a training set x^j labeled k^j for $j = 1, \ldots, N$,

$$\mathrm{cost}(w^1, b^1, w^2, b^2, \ldots) = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \| e_{k^j} - F(x^j) \|^2$$

e_{k^j}: entry $k^j \rightarrow \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

Training a NN:

solve

$$\min_{w^1, b^1, w^2, b^2, \ldots} \mathrm{cost}(w^1, b^1, \ldots)$$

$$= \min_{w^1, b^1, w^2, b^2, \dots} \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \| e_{k^j} - F(x^j) \|^2 \quad (T)$$

(T) is an unconstrained nonlinear optimization problem.

→ find local minimizers with gradient descent.

Issues: how to compute cost?
(N is big, number of variables is big)
How to compute ∇cost?
In general how to compute ∇f for any f?

a) manual differentiation
if f has a simple expression,
for example $f(x) = 3x^2 - 2x + 1 \Rightarrow \nabla f(x) = 6x - 2$

b) automatic differentiation
Same as above, but performed by a computer, directly on the code that implements f.

c) numerical differentiation

$$\frac{\partial f}{\partial x_j}(\bar{x}) = \lim_{\Delta \to 0} \frac{f(\bar{x} + \Delta e_j) - f(\bar{x})}{\Delta}$$

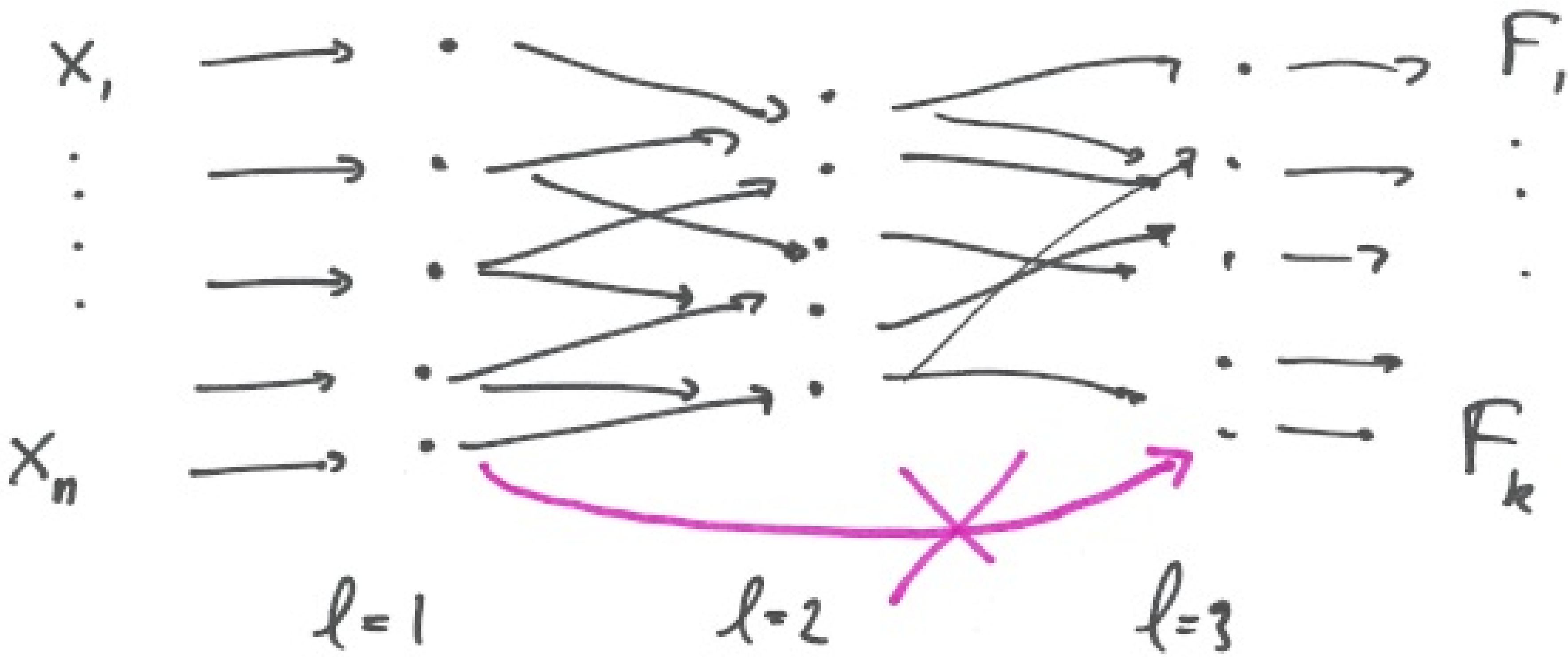
$$\simeq \frac{f(\bar{x} + \Delta e_j) - f(\bar{x})}{\Delta}$$

for some small $\Delta \neq 0$

- always works
- sometimes inaccurate (numerically unstable)
- <u>costly</u>!

Keys to NN efficacy

K1) If a NN is **layered**, and the only inputs for a neuron of layer ℓ are the outputs of layer $\ell - 1$, then we can differentiate $F(x)$ **manually**.



This is called **backpropagation**.

Deep neural network (DNN) = many layers.

K2) $\nabla \operatorname{cost}(w^1, b^1, \dots) = \nabla \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \| e_{k^j} - F(x^j) \|^2$

$$= \frac{1}{N} \sum_{j=1}^{N} \nabla \frac{1}{2} \| e_{k^j} - F(x^j) \|^2$$

N is large $\Rightarrow$

$$\nabla \operatorname{cost}(w^1, b^1, \dots) \simeq \frac{1}{|S|} \sum_{j \in S} \nabla \frac{1}{2} \| e_{k^j} - F(x^j) \|^2$$

where S is a small subset of the training set $\{1, \dots, N\}$.

$\Rightarrow$ **stochastic** gradient descent

(note: S changes at every iteration)

K3) step size α^k is constant for all k (where k is the iteration of gradient descent). It is called learning rate (one of many "hyperparameters", i.e. constants that must be chosen by NN designer).