Example

\[(x, y_1, y_2, y_3, y_4) \rightarrow F_1(x)\]

\[(x_1, x_2, y_1, y_2, y_3, y_4) \rightarrow F_2(x)\]

1. \( \mathbf{w}^1 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad b^1 = 1 \quad \rightarrow \quad y_1 = \sigma(2x_1 + (-1)x_2 + 1) \)
2. \( \mathbf{w}^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^2 = -1 \quad \rightarrow \quad y_2 = \sigma(1x_1 + 1x_2 + (-1)) \)
3. \( \mathbf{w}^3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad b^3 = 0 \quad \rightarrow \quad y_3 = \sigma((-1)y_1 + 1y_2 + 0) \)
4. \( \mathbf{w}^4 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \quad b^4 = 2 \quad \rightarrow \quad y_4 = \sigma((-2)y_1 + 1y_2 + 2) \)

\(\sigma(x) = \text{ReLU} \Rightarrow \sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}\)
Let $x = \left[ \begin{array}{c} 0.2 \\ 0.6 \end{array} \right] \Rightarrow \begin{align*}
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
\end{align*}

\[
F(x) = \left[ \begin{array}{c} y_3 \\ y_4 \end{array} \right] = \left[ \begin{array}{c} 0 \\ 0.4 \end{array} \right] \Rightarrow k = 2
\]

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

parameters: $w', b', w^2, b^2, w^3, b^3, w^4, b^4$
Training a NN is the process of finding good values for the parameters $w, b$ 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 $j = 1, \ldots, N$,
\[
\text{cost}(w', b', w^2, b^2, \ldots) = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \|e^j_{k^j} - F(x^j)\|^2
\]

Training a NN: solve
\[
\min_{w', b', w^2, b^2, \ldots} \text{cost}(w', b', \ldots)
\]
\[
\min_{w', b', u^2, b^2, \ldots} \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \| e_{k}^{j} - F(x^{j}) \|^2
\]

\((T)\) is an unconstrained nonlinear optimization problem.

To find local minimizers with gradient descent.
Issues: how to compute cost?
(N is big, number of variables is big)

How to compute $\nabla$ cost?
In general how to compute $\nabla f$ 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

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

\[ \approx \frac{f(\bar{x} + \Delta e_j) - f(\bar{x})}{\Delta} \]

for some small \( \Delta \neq 0 \)

- always works
- sometimes inaccurate (numerically unstable)
- costly!
Keys to NN efficacy

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

\[ x_1 \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots \rightarrow F_1 \]
\[ \vdots \]
\[ x_n \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots \rightarrow F_k \]
\[ l=1 \quad l=2 \quad l=3 \]
This is called backpropagation.

Deep neural network (DNN) = many layers.

\[ \nabla \text{cost}(w', b', \ldots) = \nabla \frac{1}{N} \sum_{j=1}^{N} \frac{1}{2} \| e_{k,j} \! - \! F(x^i) \|^2 \\
= \frac{1}{N} \sum_{j=1}^{N} \nabla \frac{1}{2} \| e_{k,j} \! - \! F(x^i) \|^2 \\
N \text{ is large} \Rightarrow \\
\nabla \text{cost}(w', b', \ldots) \approx \frac{1}{|S|} \sum_{j \in S} \nabla \frac{1}{2} \| e_{k,j} \! - \! F(x^i) \|^2 \\
\text{where } S \text{ is a small subset of the training set } \{1, \ldots, N\}. \\
\Rightarrow \text{ stochastic gradient descent} \]
(note: $S$ changes at every iteration)

$k3)$ step size $\lambda^k$ 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).