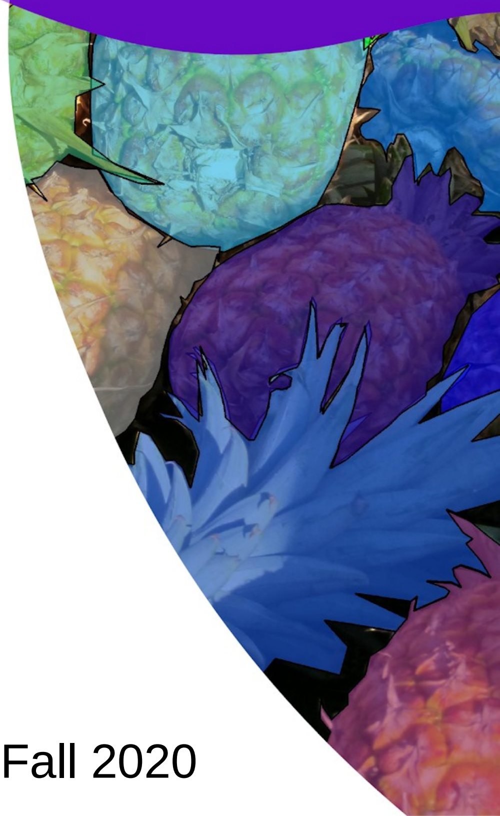
NEW YORK UNIVERSITY

Deep Learning

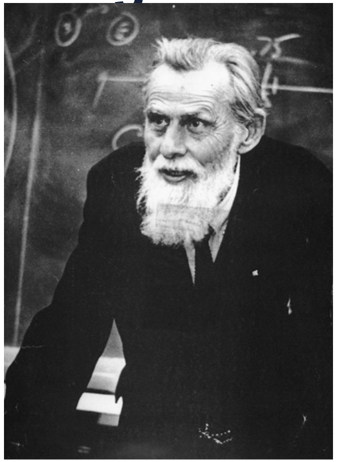
Kyunghyun Cho, Alfredo Canziani, Yann LeCun
NYU - Courant Institute & Center for Data Science

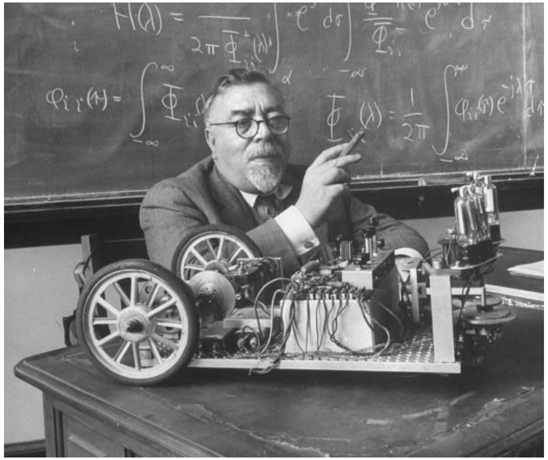


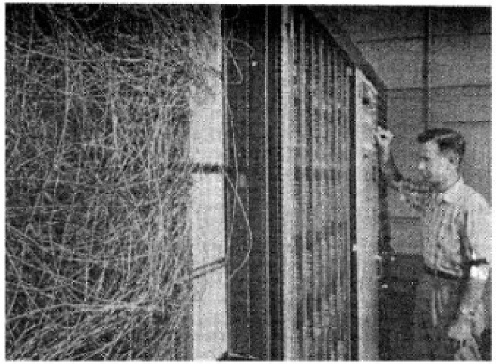
Y. LeCun

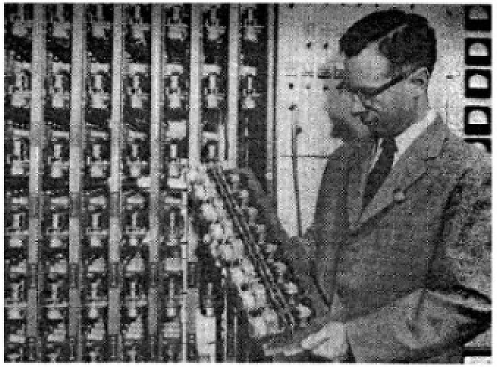
Inspiration for Deep Learning: The Brain!

- **McCulloch & Pitts (1943): networks of binary neurons can do logic**
- **Donald Hebb (1947): Hebbian synaptic plasticity**
- **Norbert Wiener (1948): cybernetics, optimal filter, feedback, autopoïesis, auto-organization.**
- **Frank Rosenblatt (1957): Perceptron**
- **Hubel & Wiesel (1960s): visual cortex architecture**



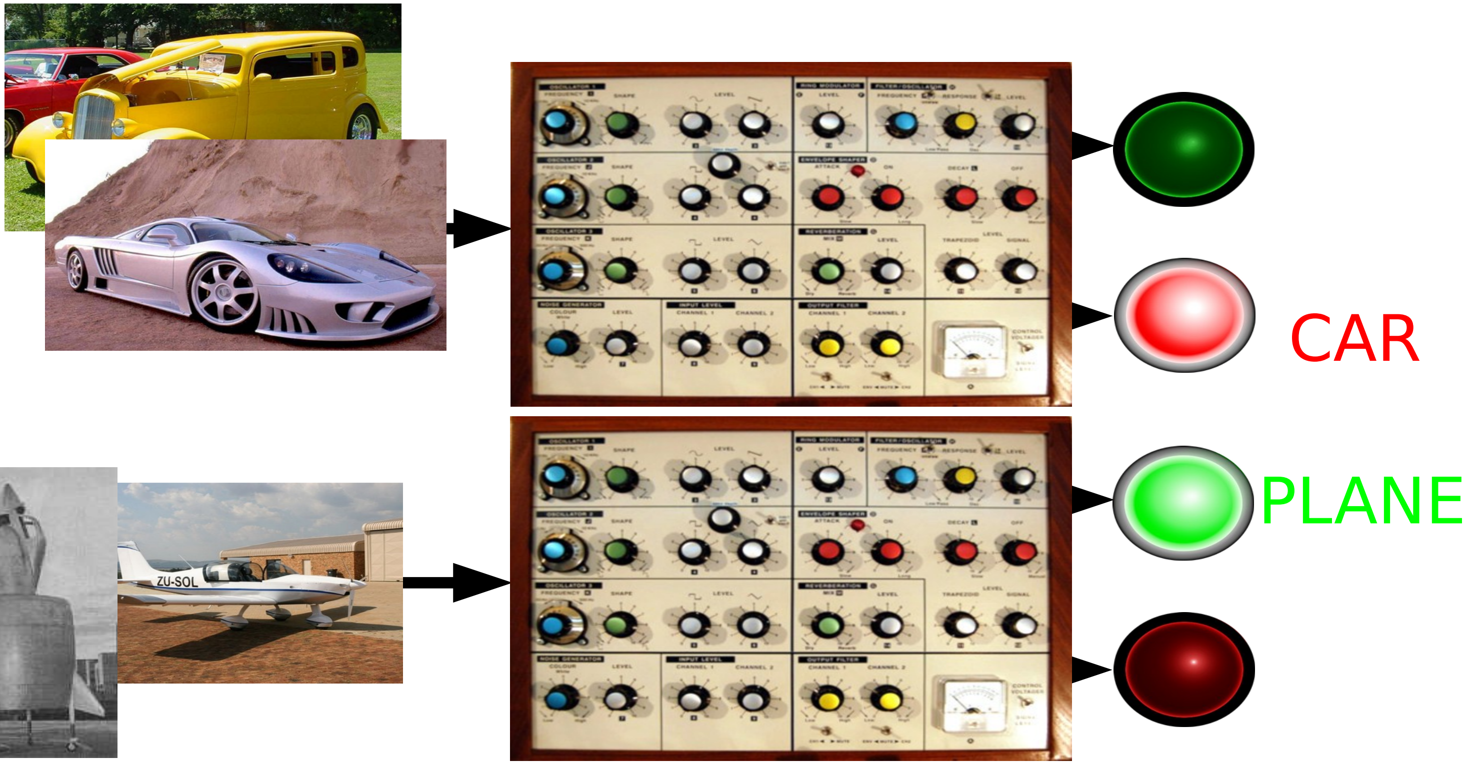


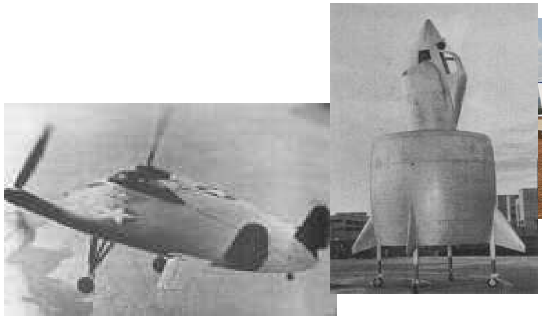




Supervised Learning

- **Training a machine by showing examples instead of programming it**
- **When the output is wrong, tweak the parameters of the machine**
- **Works well for:**
  - Speech → words
  - Image → categories
  - Portrait → name
  - Photo → caption
  - Text → topic
  - ….

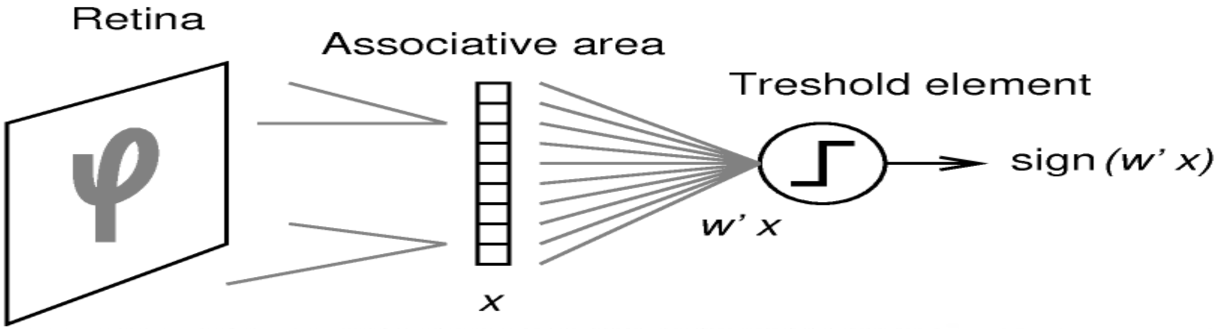




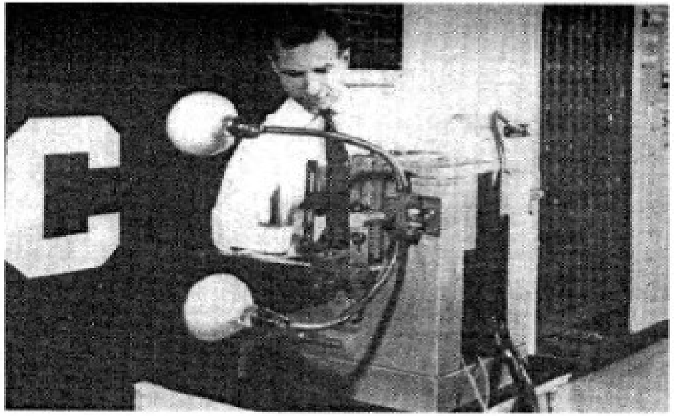
Supervised Learning goes back to the Perceptron & Adaline

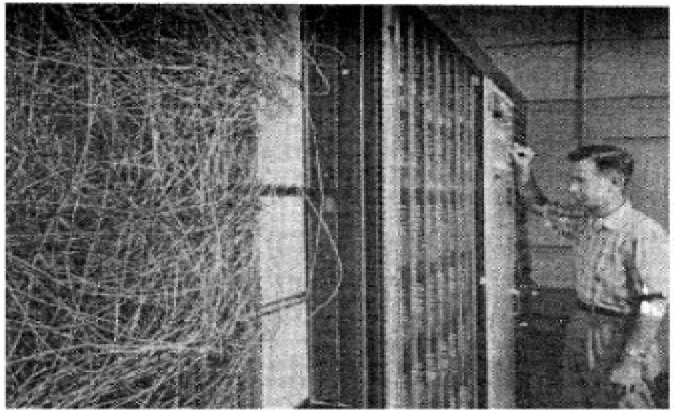
- **The McCulloch-Pitts Binary Neuron**
  - Perceptron: weights are motorized potentiometers
  - Adaline: Weights are electrochemical “memistors”

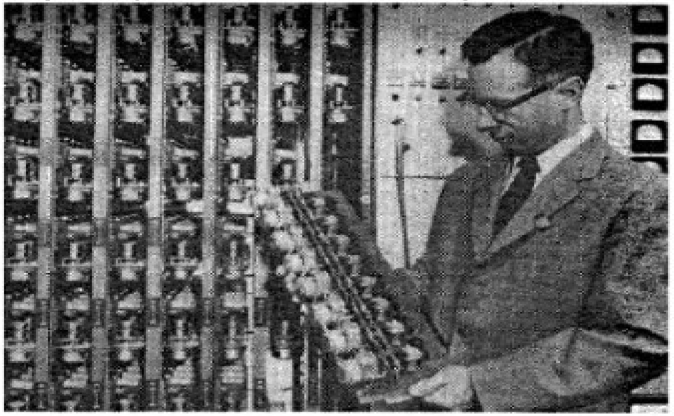
$$y = sign\left(\sum_{i=1}^{N} W_i X_i + b\right)$$



https://youtu.be/X1G2g3SiCwU

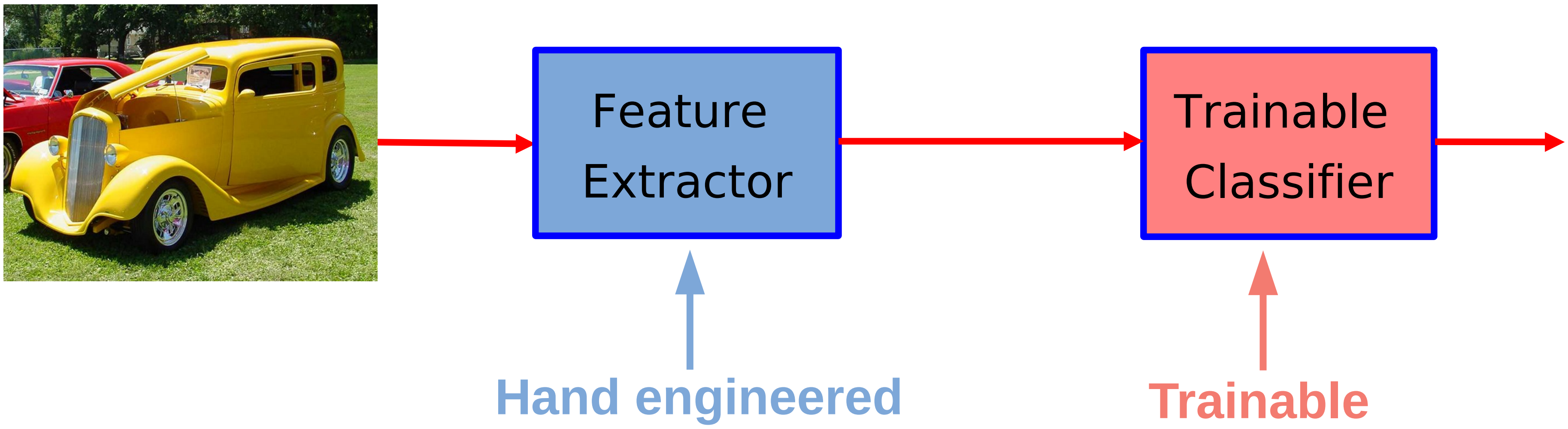






The Standard Paradigm of Pattern Recognition

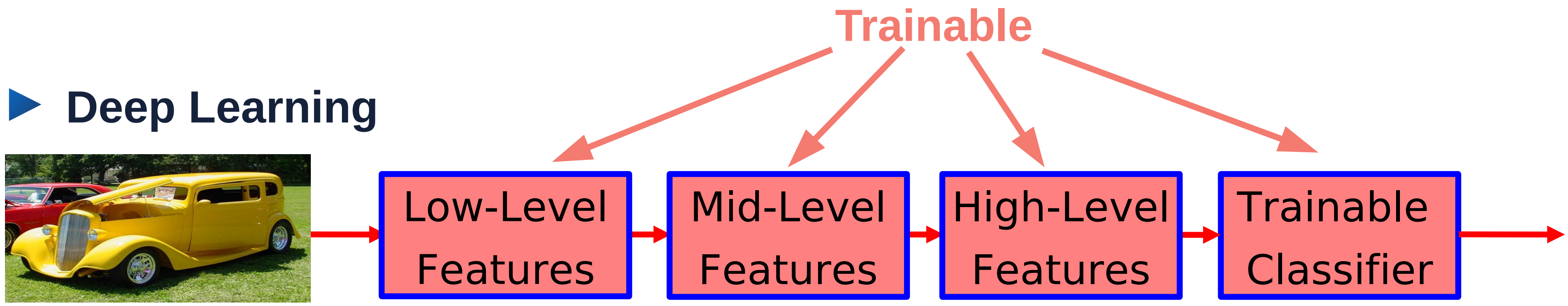
- **...since the 1960s**
- **...and “traditional” Machine Learning**
  - until the “Deep Learning Revolution” (circa 2012)



Multilayer Neural Nets and Deep Learning

- **Traditional Machine Learning**

- **Deep Learning**



Parameterized Model

- **Parameterized model**

$$\bar{y} = G(x, w)$$

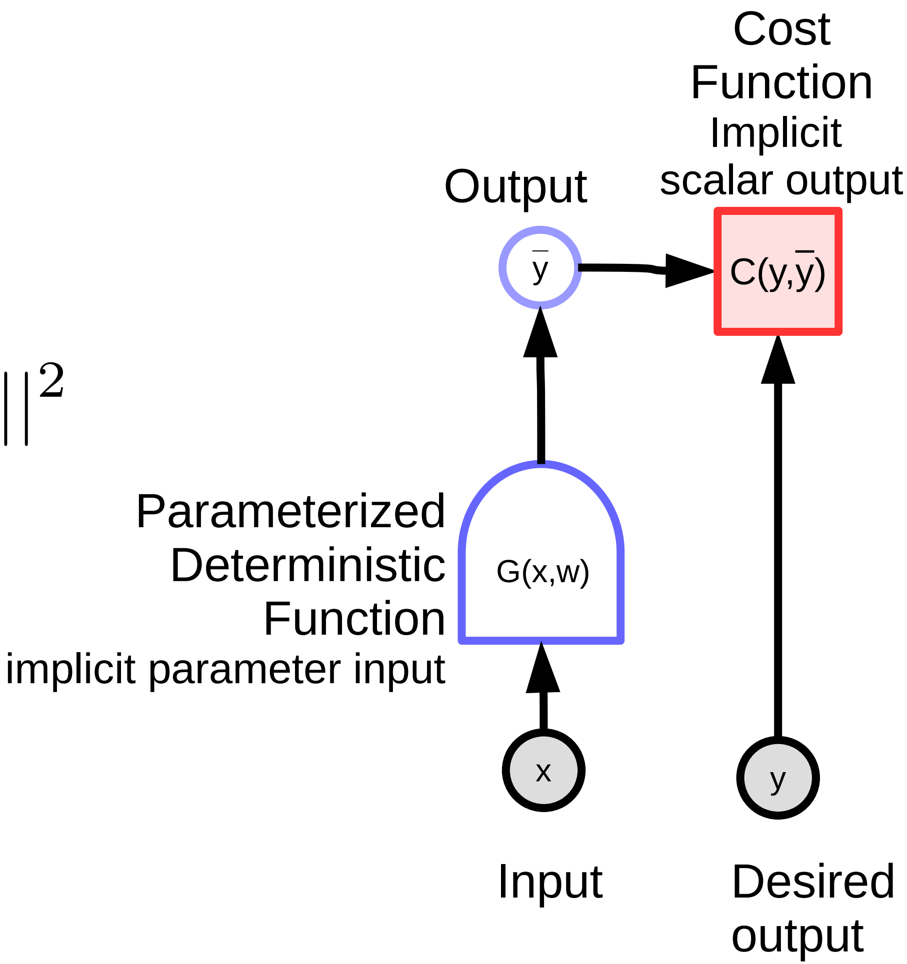
  - Example: linear regression

$$\bar{y} = \sum_i w_i x_i \quad C(y, \bar{y}) = ||y - \bar{y}||^2$$

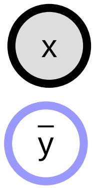
  - Example: Nearest neighbor:

$$\bar{y} = \text{argmin}_k ||x - w_{k,.}||^2$$

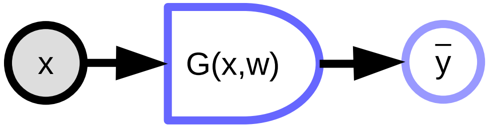
  - Computing function G may involve complicated algorithms



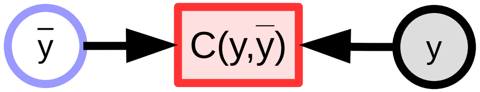
Block diagram notations for computation graphs



- **Variables (tensor, scalar, continuous, discrete...)**
  - Observed: input, desired output…
  - Computed variable: outputs of deterministic functions



- **Deterministic function**
  - Multiple inputs and outputs (tensors, scalars,….)
  - Implicit parameter variable (here: w)



- **Scalar-valued function (implicit output)**
  - Single scalar output (implicit)
  - used mostly for cost functions

Loss function, average loss.

- **Simple per-sample loss function**

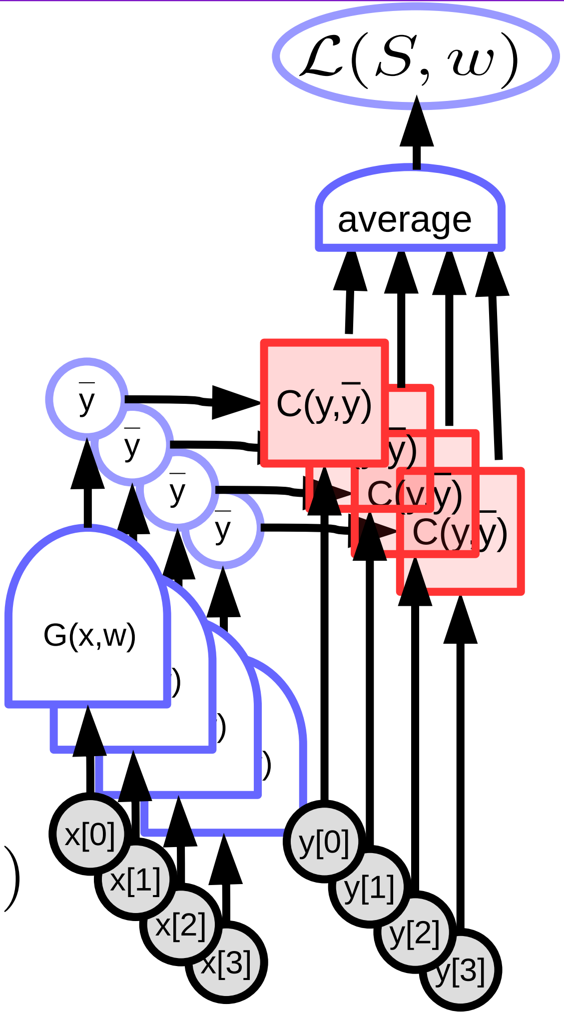
$$L(x, y, w) = C(y, G(x, w))$$

- **A set of samples**

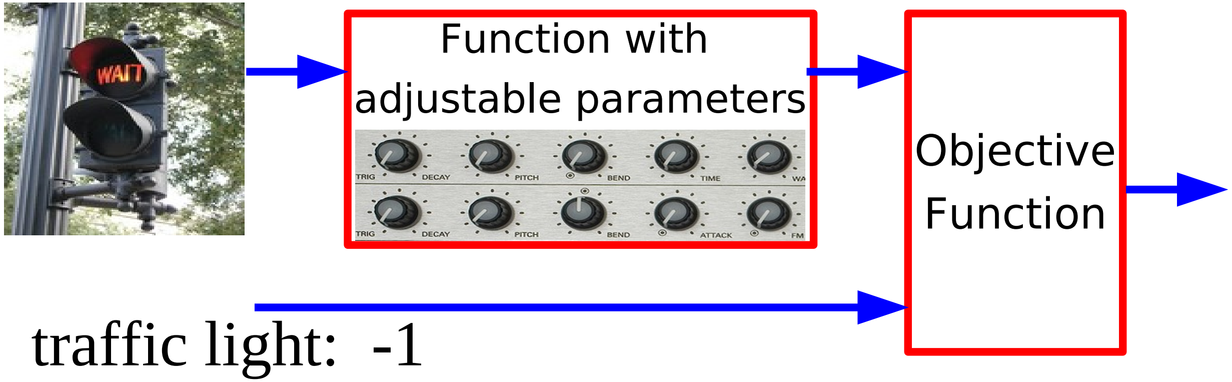
$$S = \{(x[p], y[p]) \ / \ p = 0 \dots P-1\}$$

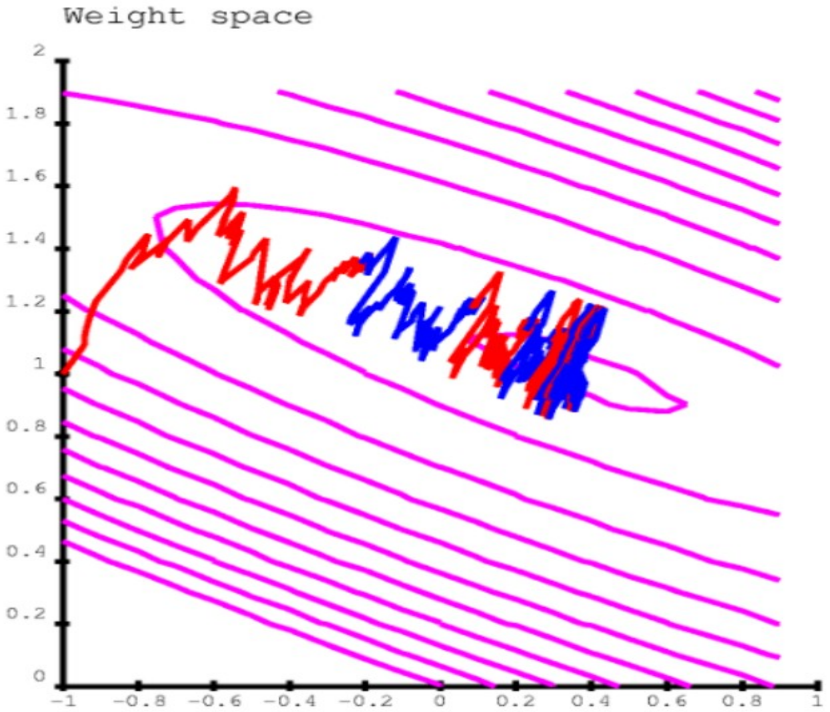
- **Average loss over the set**

$$\mathcal{L}(S, w) = \frac{1}{P} \sum_{(x,y)} L(x, y, w) = \frac{1}{P} \sum_{p=0}^{P-1} L(x[p], y[p], w)$$



Supervised Machine Learning = Function Optimization





- **It's like walking in the mountains in a fog and following the direction of steepest descent to reach the village in the valley**
- **But each sample gives us a noisy estimate of the direction. So our path is a bit random.**

$$W_i \leftarrow W_i - \eta \frac{\partial L(W, X)}{\partial W_i}$$

Gradient Descent

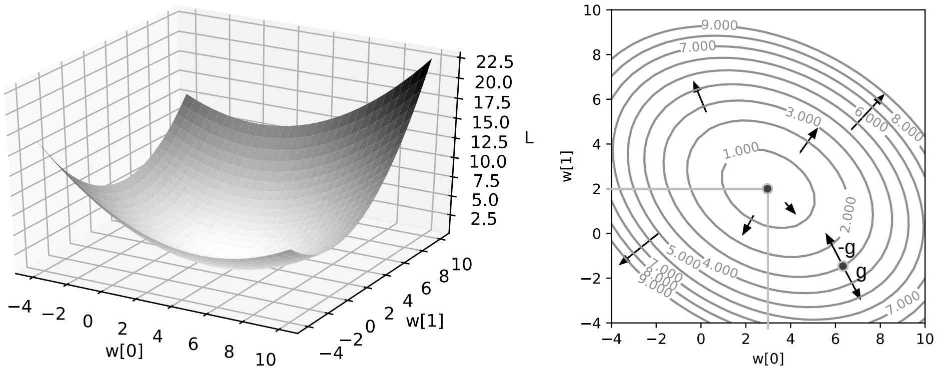
- **Full (batch) gradient**

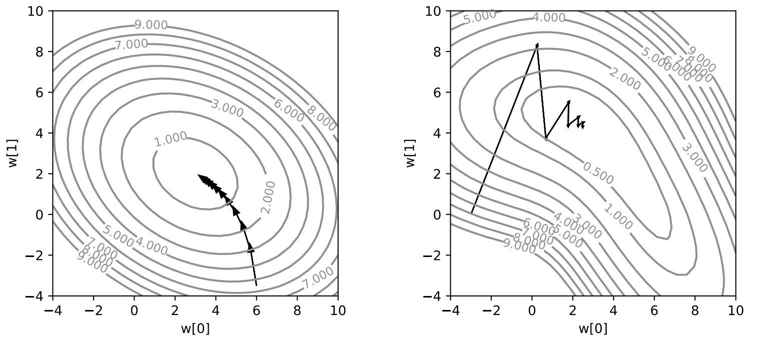
$$w \leftarrow w - \eta \frac{\partial \mathcal{L}(S, w)}{\partial w}$$

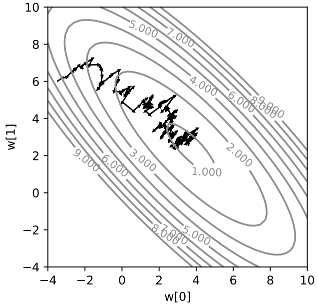
- **Stochastic Gradient (SGD)**
  - Pick a p in 0...P-1, then update w:

$$w \leftarrow w - \eta \frac{\partial L(x[p], y[p], w)}{\partial w}$$

- **SGD exploits the redundancy in the samples**
  - It goes faster than full gradient in most cases
  - In practice, we use mini-batches for parallelization.



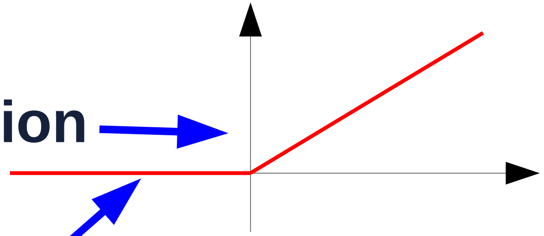




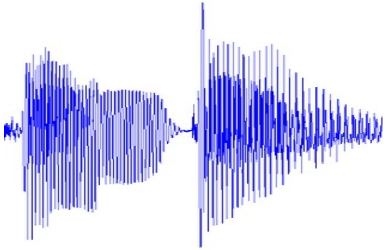
(Deep) Multi-Layer Neural Nets

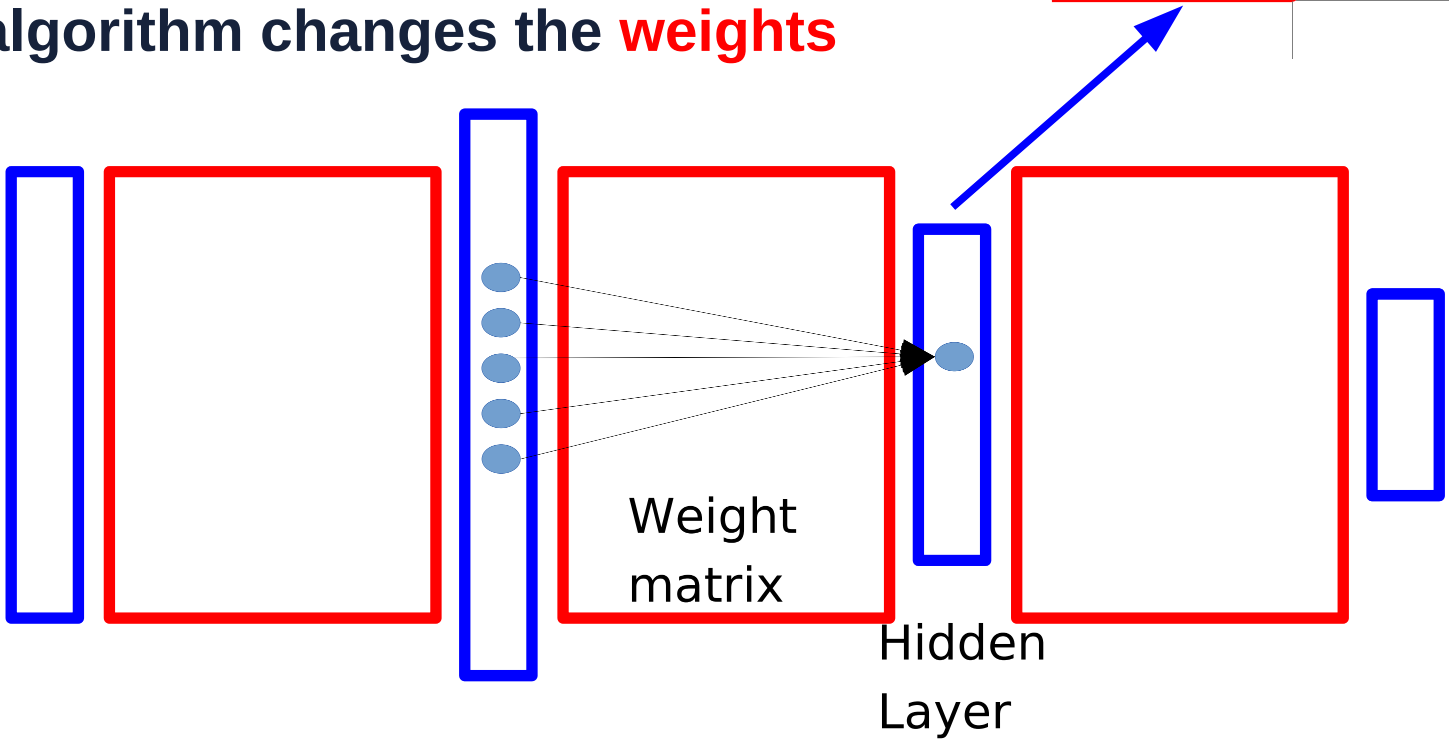
- **Multiple Layers of simple units**
- **Each units computes a weighted sum of its inputs**
- **Weighted sum is passed through a non-linear function**
- **The learning algorithm changes the weights**

$$ReLU(x) = max(x, 0)$$



Ceci est une voiture

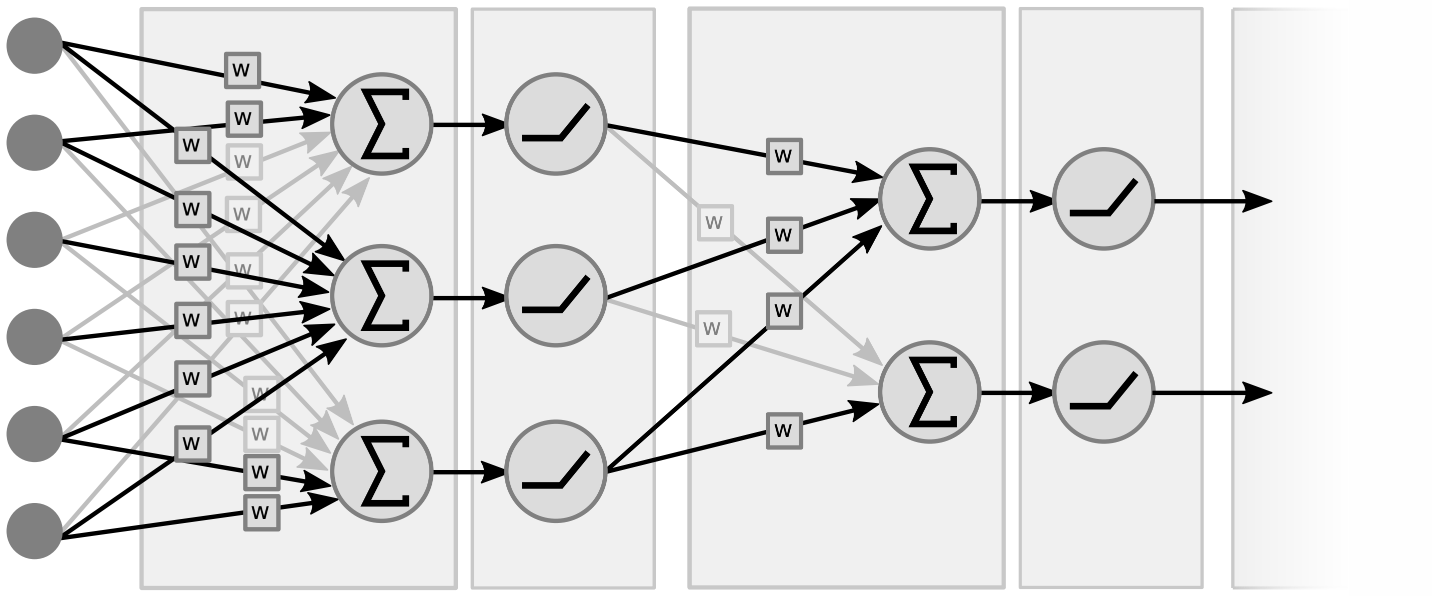




Y. LeCun

Traditional Neural Net

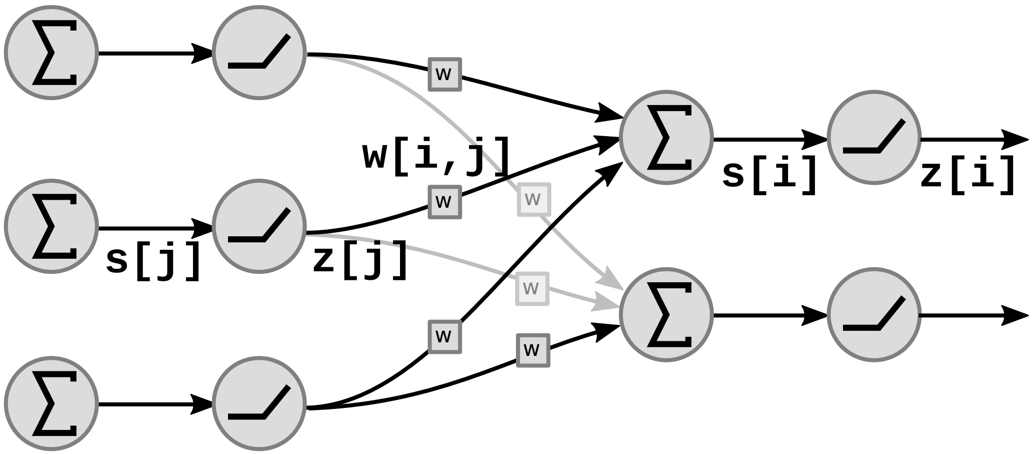
- **Stacked linear and non-linear functional blocks**
  - Weighted sums, matrix-vector product
  - Point-wise non-linearities (e.g. ReLu, tanh, ….)



Traditional Neural Net

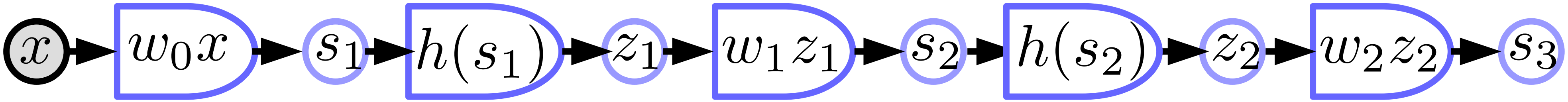
- **Stacked linear and non-linear functional blocks**

$$s[i] = \sum_{j \in \mathrm{UP}(i)} w[i,j] \cdot z[j] \qquad z[i] = f(s[i])$$

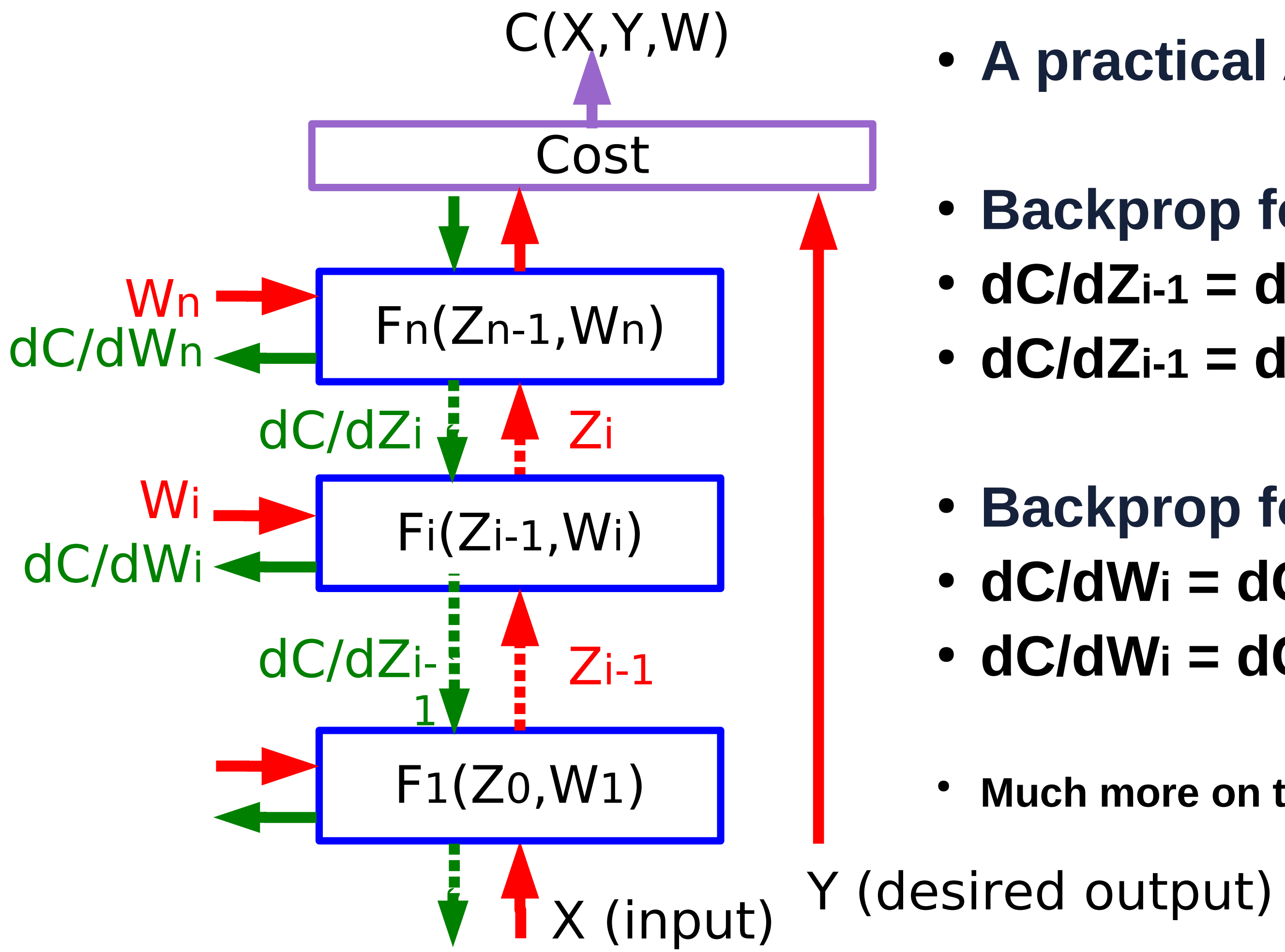


Block Diagram of a Traditional Neural Net

- **linear blocks** $$s_{k+1} = w_k z_k$$
- **Non-linear blocks** $$z_k = h(s_k)$$



Computing Gradients by Back-Propagation



- **A practical Application of Chain Rule**

- **Backprop for the state gradients:**
- **$dC/dZ_{i-1} = dC/dZ_i \, . \, dZ_i/dZ_{i-1}$**
- **$dC/dZ_{i-1} = dC/dZ_i \, . \, dF_i(Z_{i-1},W_i)/dZ_{i-1}$**

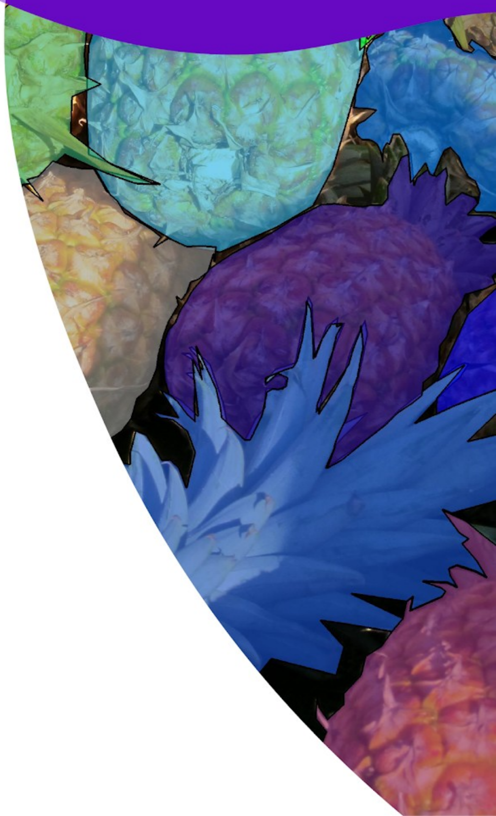
- **Backprop for the weight gradients:**
- **$dC/dW_i = dC/dZ_i \, . \, dZ_i/dW_i$**
- **$dC/dW_i = dC/dZ_i \, . \, dF_i(Z_{i-1},W_i)/dW_i$**

- **Much more on this later……**

NEW YORK UNIVERSITY

Learning Representations

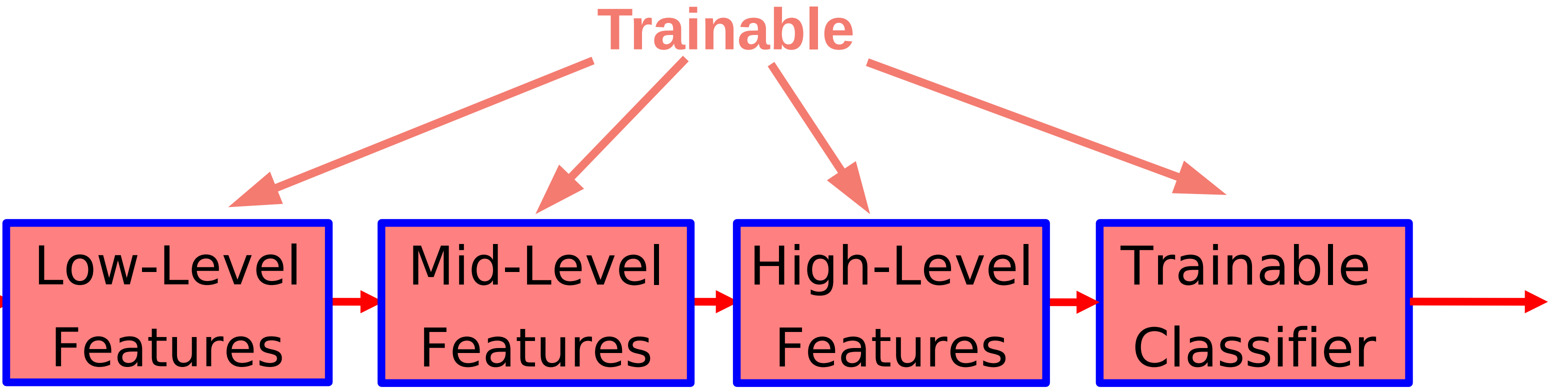
What are good representations?
Why do networks need to be deep?



Deep Learning is about Learning Representations

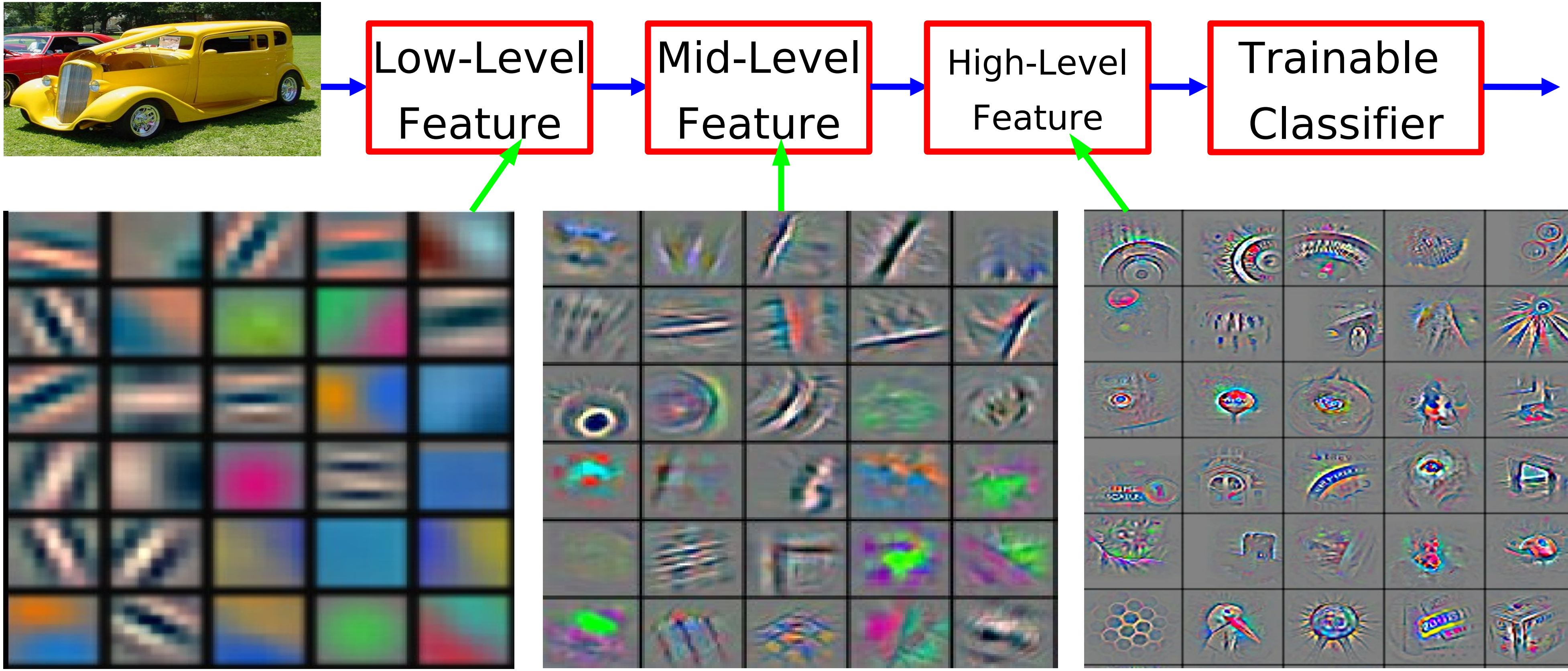
- **Traditional Machine Learning**

- **Deep Learning**



Multilayer Architectures == Compositional Structure of Data

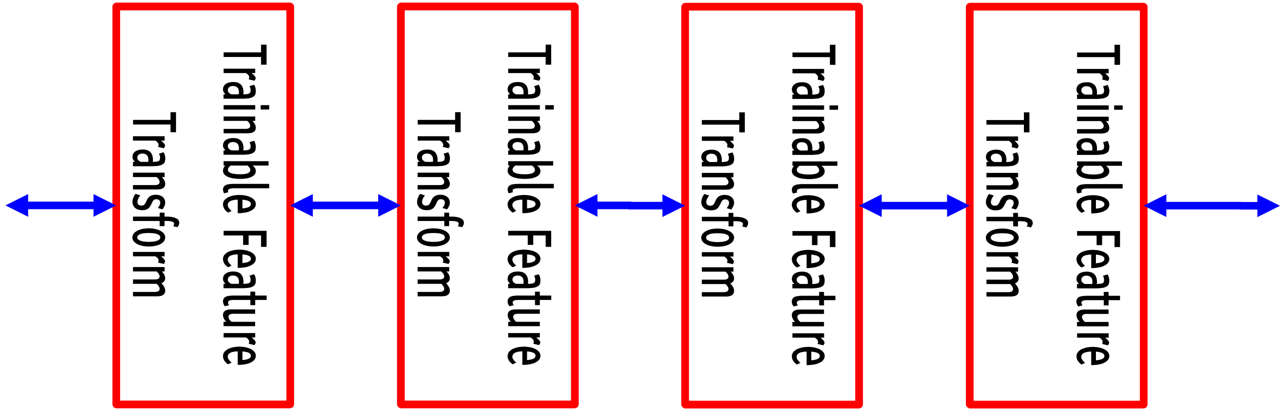
- **Natural is data is compositional => it is efficiently representable hierarchically**



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Multilayer Architecture == Hierarchical representation

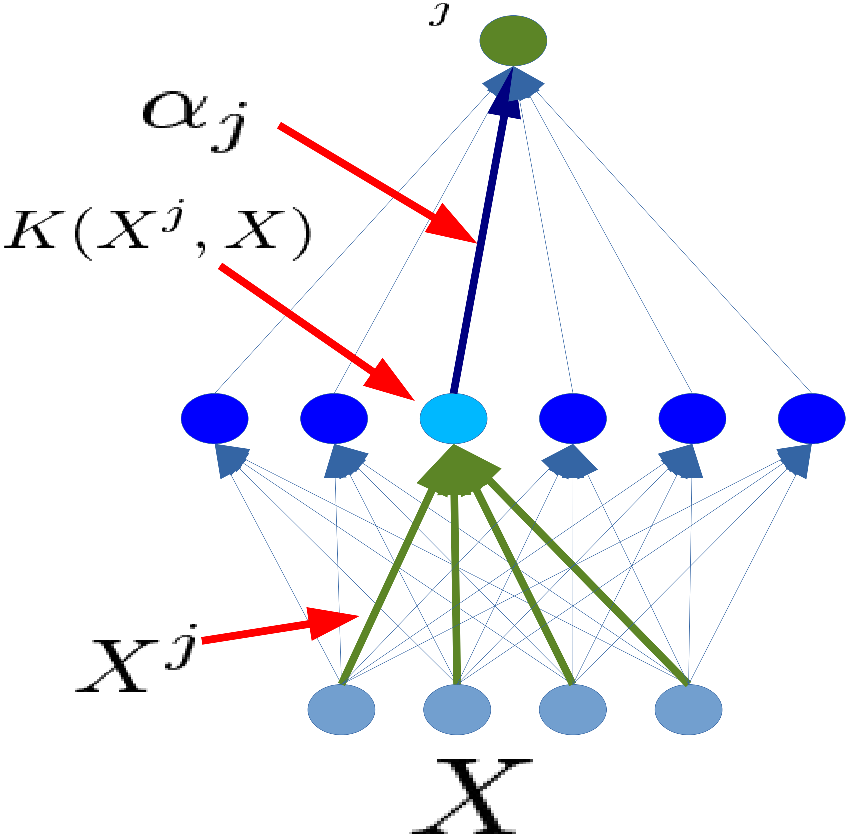
- **Hierarchy of representations with increasing level of abstraction**
- **Each stage is a kind of trainable feature transform**
- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word



Shallow networks are universal approximators!

- **SVMs and Kernel methods**
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- **2-layer neural nets**
  - Layer1: dot products + non-linear function; Layer2: linear
- **But few useful functions can be efficiently represented with only two layers of reasonable size.**

$$G(X,\alpha) = \sum_j \alpha_j K(X^j, X)$$



Ideas for “generic” feature extraction

- **Basic principle:**
  - expanding the dimension of the representation so that things are more likely to become linearly separable.

- **- space tiling**
- **- random projections**
- **- polynomial classifier (feature cross-products)**
- **- radial basis functions**
- **- kernel machines**

Do we really need deep architectures?

- **Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"**

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

  - kernel machines (and 2-layer neural nets) are "universal".
- **Deep learning machines**

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - they can represent more complex functions with less "hardware"
- **We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)**

Why would deep architectures be more efficient?

[Bengio & LeCun 2007 “Scaling Learning Algorithms Towards AI”]

- **A deep architecture trades space for time (or breadth for depth)**
  - more layers (more sequential computation),
  - but less hardware (less parallel computation).
- **Example1: N-bit parity**
  - requires N-1 XOR gates in a tree of depth log(N).
  - Even easier if we use threshold gates
  - requires an exponential number of gates of we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).
- **Example2: circuit for addition of 2 N-bit binary numbers**
  - Requires O(N) gates, and O(N) layers using N one-bit adders with ripple carry propagation.
  - Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
  - Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms O(2^N).....