# Master TRIED
# Reconnaissance des formes et méthodes neuronales (US330X) - Neural Networks and Deep Learning

**Nicolas Thome**
Conservatoire Nationnal des Arts et Métiers (Cnam)
Laboratoire CEDRIC - équipe Vertigo

le cnam

Cédric

# Outline

1 Context

2 Neural Nets

3 Convolutionnal Neural Networks

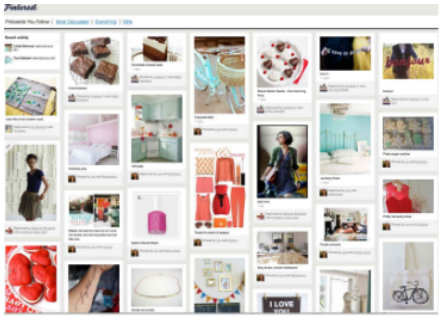
4 Case Study: LeNet Model

# Context

## Big Data

- Superabundance of data: images, videos, audio, text, use traces, *etc*

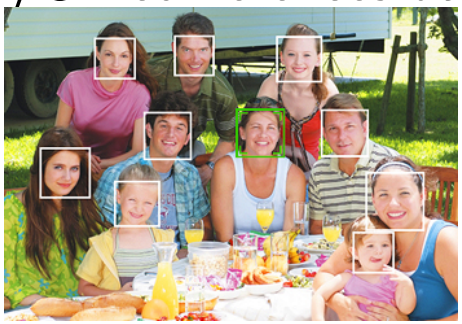
BBC: 2.4M videos

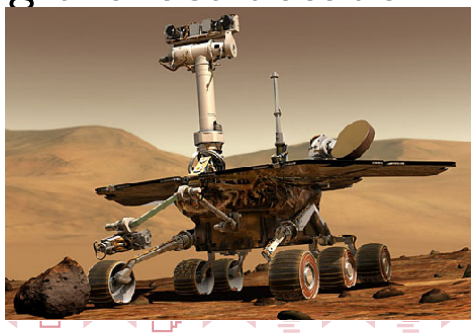


Facebook: 350B images
1B each day

100M monitoring cameras

- Obvious need to access, search, or classify these data: **Recognition**
- Huge number of applications: mobile visual search, robotics, autonomous driving, augmented reality, medical imaging *etc*
- Leading track in major ML/CV conferences during the last decade





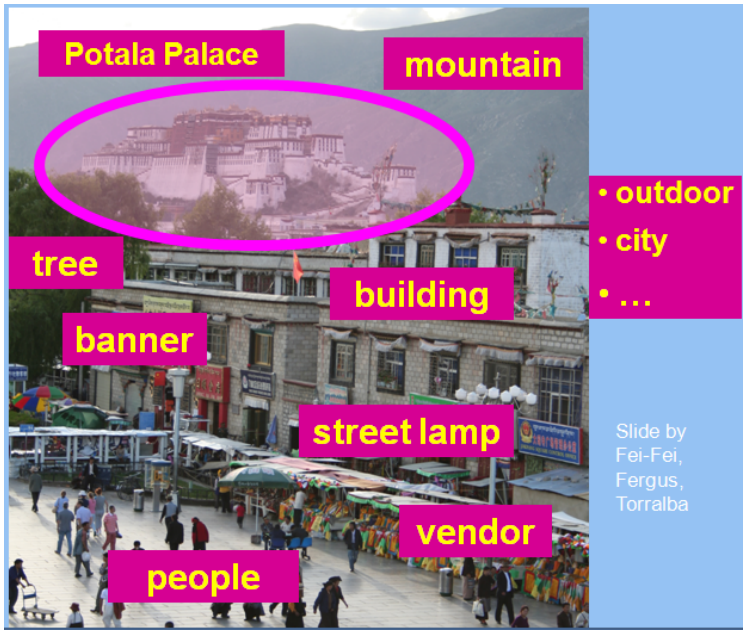
# Recognition and classification

- Classification : assign a given data to a given set of pre-defined classes
- Recognition much more general than classification, *e.g.*
  - Ranking for document indexing
  - Localization, segmentation for image understanding
  - Sequence prediction for text, speech, audio, *etc*
- Many tasks can be cast as classification problems
  $\Rightarrow$ **importance of classification**

# Focus on Visual Recognition: Perceiving Visual World

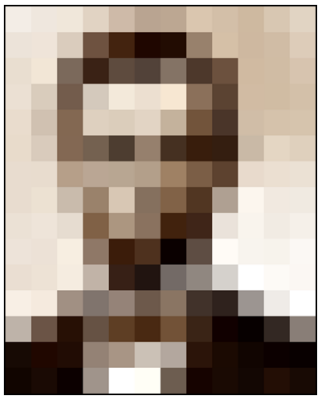
- Visual Recognition: archetype of low-level signal understanding
- Supposed to be a master class problem in the early 80's
- Certainly the most impacted topic by deep learning

- Scene categorization
- Object localization
- Context & Attribute recognition
- Rough 3D layout, depth ordering
- Rich description of scene, *e.g.* sentences



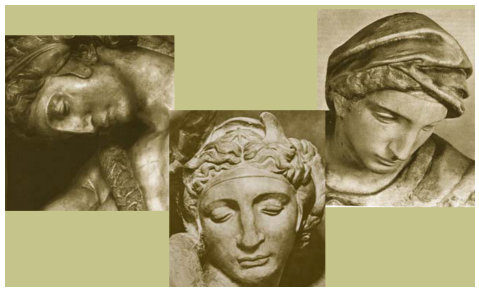
# Recognition of low-level signals

## Challenge: filling the semantic gap



What we perceive *vs*
What a computer sees

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 243 | 239 | 240 | 225 | 206 | 185 | 188 | 218 | 211 | 206 | 216 | 225 |
| 242 | 239 | 218 | 110 | 67 | 31 | 34 | 152 | 213 | 206 | 208 | 221 |
| 243 | 242 | 123 | 58 | 94 | 82 | 132 | 77 | 108 | 208 | 208 | 215 |
| 235 | 217 | 115 | 212 | 243 | 236 | 247 | 139 | 91 | 209 | 208 | 211 |
| 233 | 208 | 131 | 222 | 219 | 226 | 196 | 114 | 74 | 208 | 213 | 214 |
| 232 | 217 | 131 | 116 | 77 | 150 | 69 | 56 | 52 | 201 | 228 | 223 |
| 232 | 232 | 182 | 186 | 184 | 179 | 159 | 123 | 93 | 232 | 235 | 235 |
| 232 | 236 | 201 | 154 | 216 | 133 | 129 | 81 | 175 | 252 | 241 | 240 |
| 235 | 238 | 230 | 128 | 172 | 138 | 65 | 63 | 234 | 249 | 241 | 245 |
| 237 | 236 | 247 | 143 | 59 | 78 | 10 | 94 | 255 | 248 | 247 | 251 |
| 234 | 237 | 245 | 193 | 55 | 33 | 115 | 144 | 213 | 255 | 253 | 251 |
| 248 | 245 | 161 | 128 | 149 | 109 | 138 | 65 | 47 | 156 | 239 | 255 |
| 190 | 107 | 39 | 102 | 94 | 73 | 114 | 58 | 17 | 7 | 51 | 137 |
| 23 | 32 | 33 | 148 | 168 | 203 | 179 | 43 | 27 | 17 | 12 | 8 |
| 17 | 26 | 12 | 160 | 255 | 255 | 109 | 22 | 26 | 19 | 35 | 24 |

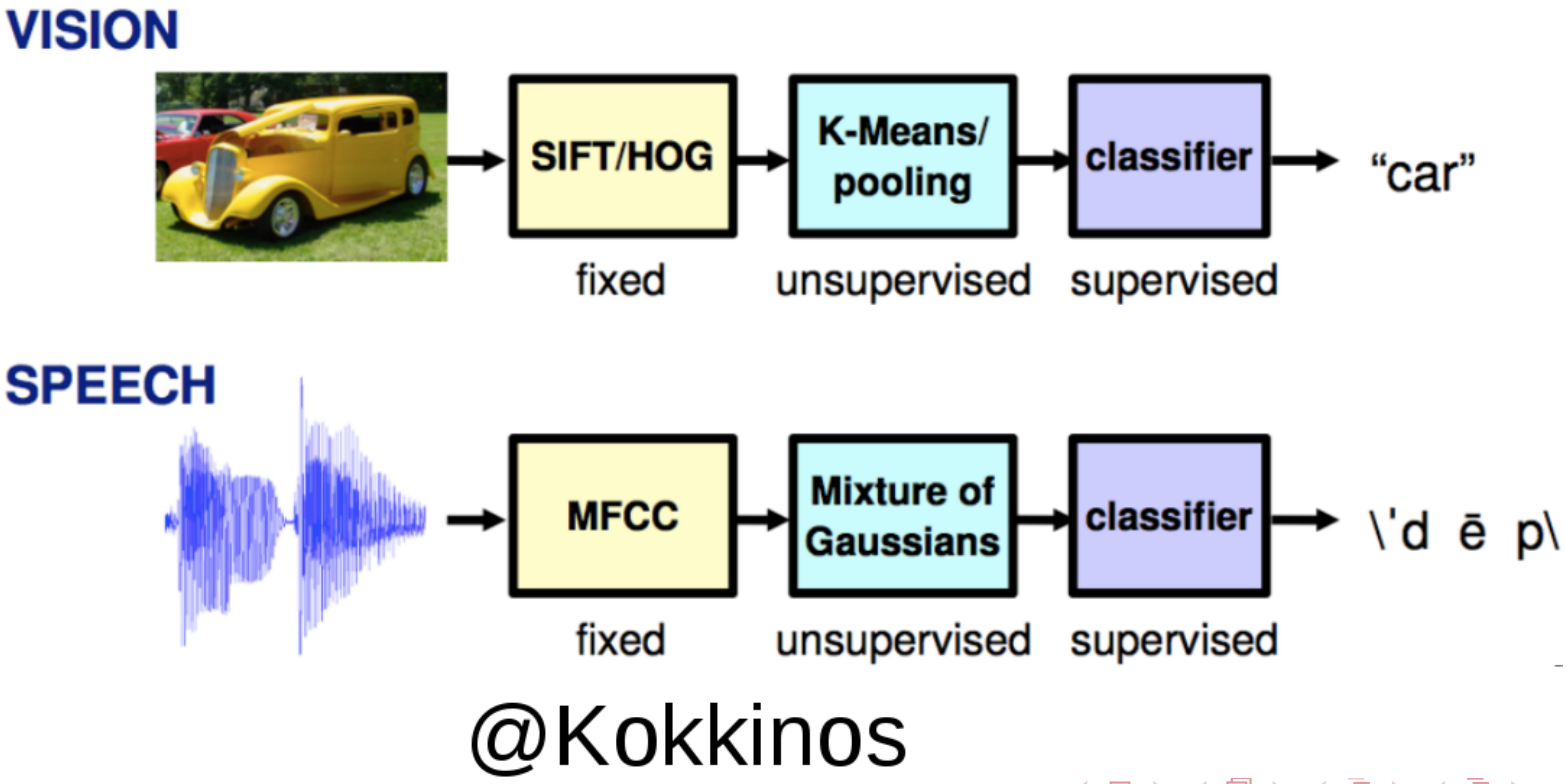


- Illumination variations
- View-point variations
- Deformable objects
- intra-class variance
- *etc*

⇒ **How to design "good" intermediate representation ?**

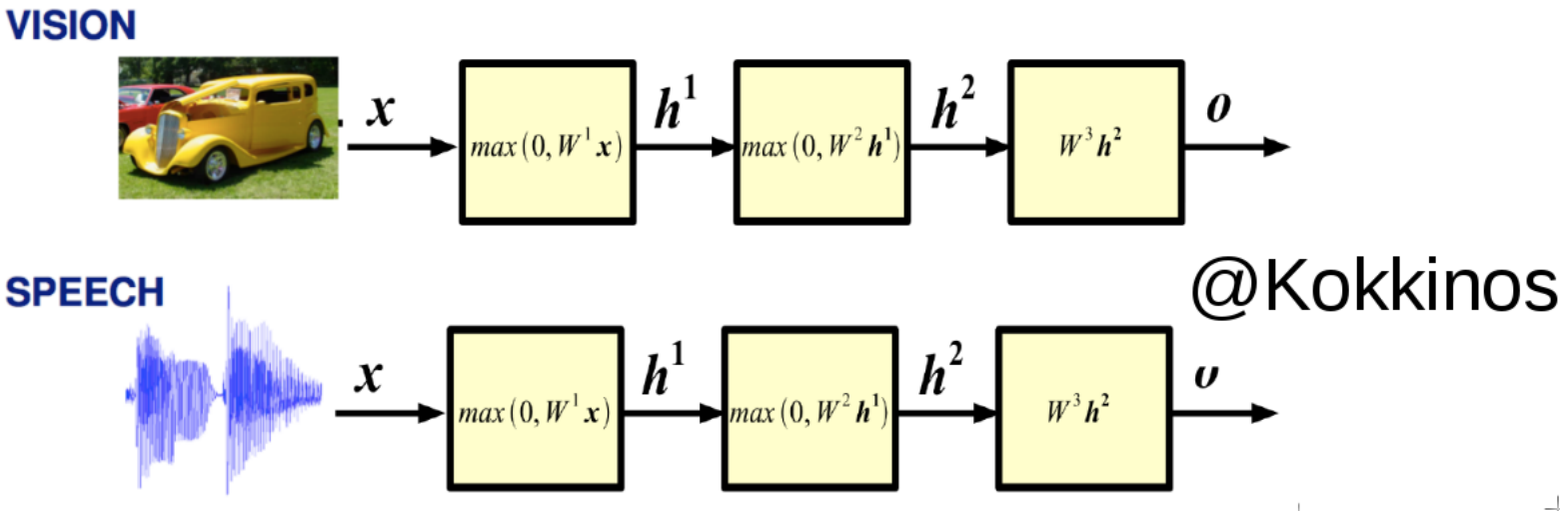
# Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Before DL: handcrafted intermediate representations for each task
  - ⊖ Needs expertise (PhD level) in each field
  - ⊖ Weak level of semantics in the representation



# Deep Learning (DL) & Recognition of low-level signals

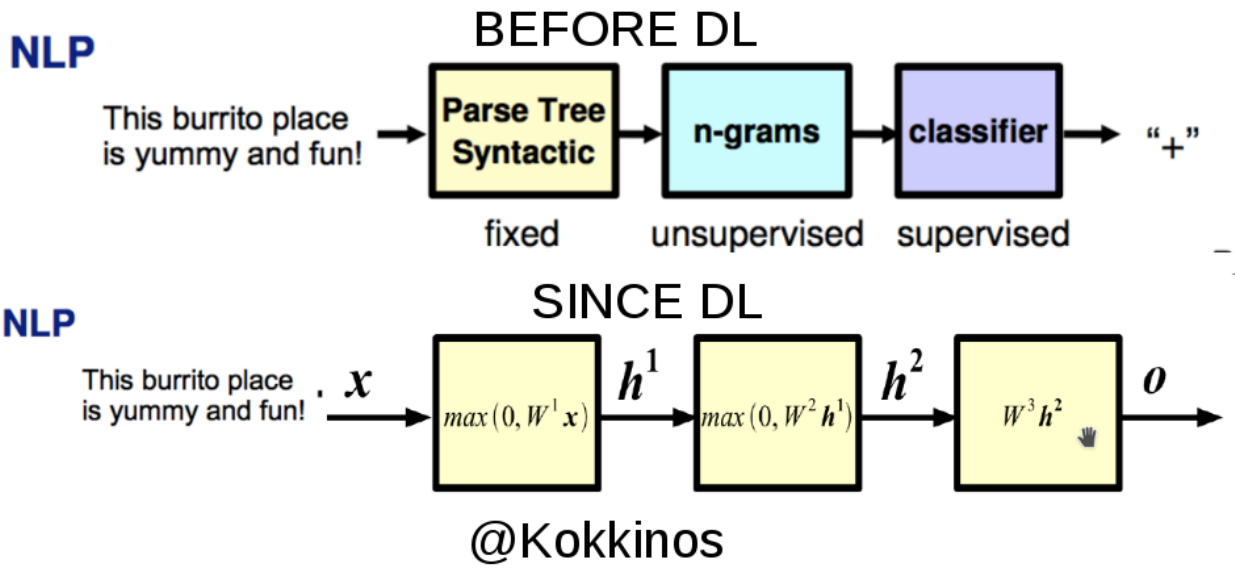
- DL: breakthrough for the recognition of low-level signal data
- Since DL: automatically **learning intermediate representations**
  - ⊕ Outstanding experimental performances >> handcrafted features
  - ⊕ Able to learn high level intermediate representations
  - ⊕ Common learning methodology ⇒ field independent, no expertise

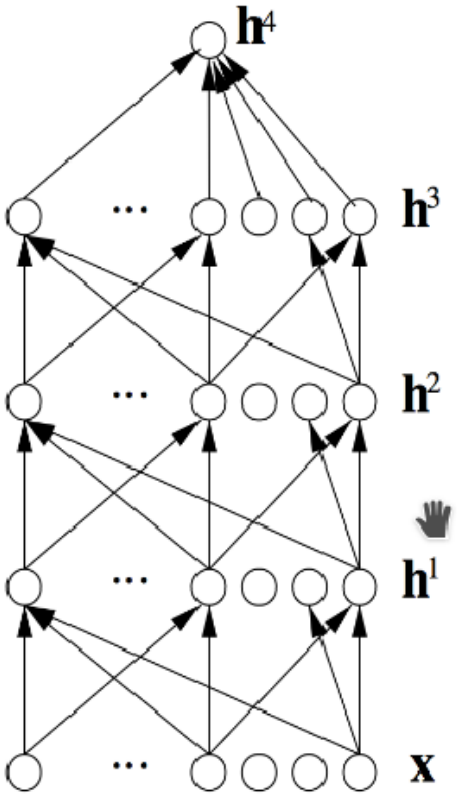


# Deep Learning (DL) & Representation Learning

- DL: breakthrough for representation learning
  - Automatically learning intermediate levels of representation

- Ex: Natural language Processing (NLP)



# Deep Learning (DL) & Representation Learning

**This course: five weeks on deep learning**

**Ressources:** `http://cedric.cnam.fr/~thomen/cours/US330X/index.html`

- **Week 1: Deep learning and ConvNets**
  - TP1: Back-prop "a la mano"
- **Week 2: Deep Revolution**
  - TP2: Keras and manifold untangling
- **Week 3: Architecture, optimization and transfer**
  - TP3: Deep Features and fine-tuning
- **Week 4: Embeddings and recurrent neural networks (RNNs)**
  - TP4: RNNs for sequential data
- **Week 5: Open Issues and perspectives**
  - TP5: Vision and language

# Outline

1 Context

2 Neural Nets

3 Convolutionnal Neural Networks

4 Case Study: LeNet Model

# The origins

## History

- **1943:** The formal neuron [McCulloch and Pitts, 1943]
- **1958:** First perceptron [Rosenblatt, 1958]
- **1974:** Backpropagation algorithm [Werbos, 1974]
- **1980:** First deep feedforward network [Fukushima, 1980]
- **1989:** First convolutional neural network [LeCun et al., 1989]

## The formal neuron, basis of the neural networks

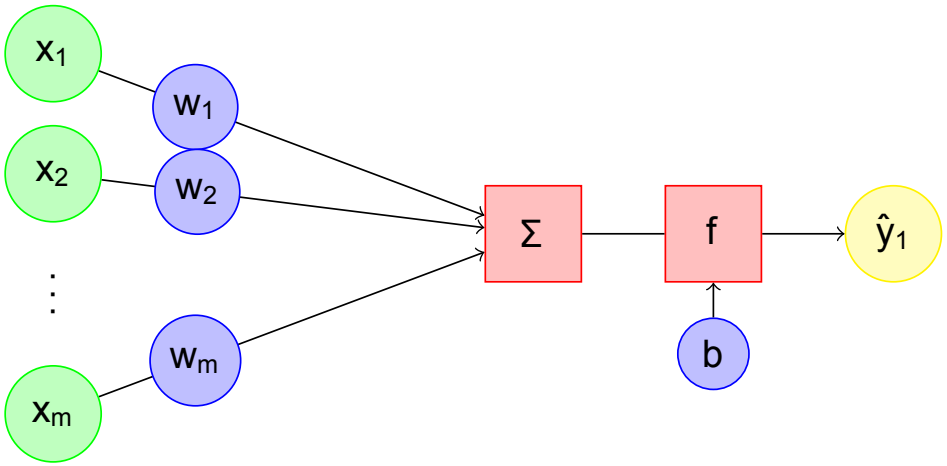


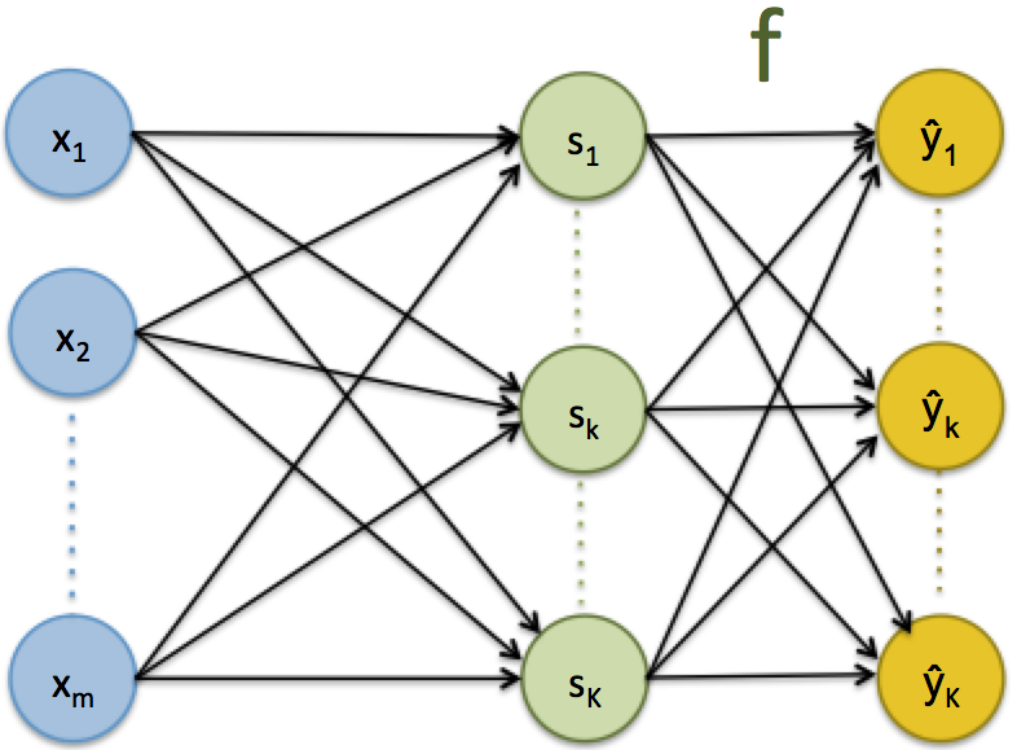

Figure: The formal neuron – Credits: R. Herault

$x_i$: inputs
$w_i, b$: weights
$f$: activation function
$y$: output of the neuron

$$y = f(w^\top x + b)$$

# Neural Networks

- Stacking several formal neurons ⇒ **Perceptron**



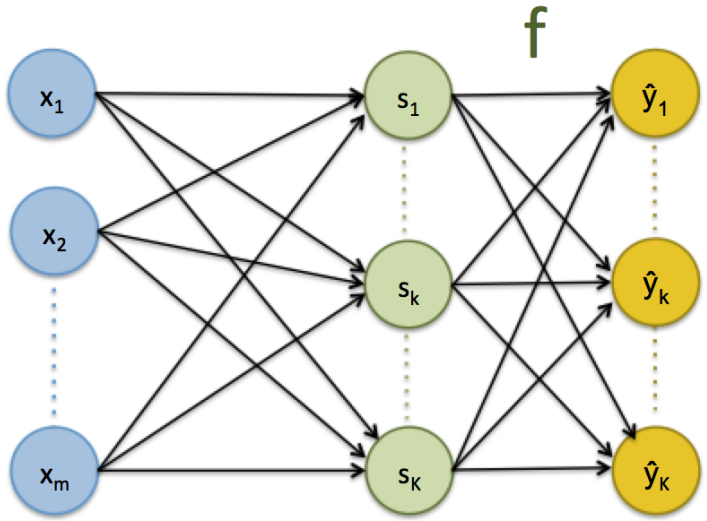
# Perceptron and Multi-Class Classification

- **Soft-max Activation**:

$$\hat{y}_k = f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}}$$

- **Probabilistic interpretation for multi-class classification**:
  - Each output neuron $\Leftrightarrow$ class
  - $\hat{y}_k \sim P(k/\mathrm{x}, \mathrm{w})$

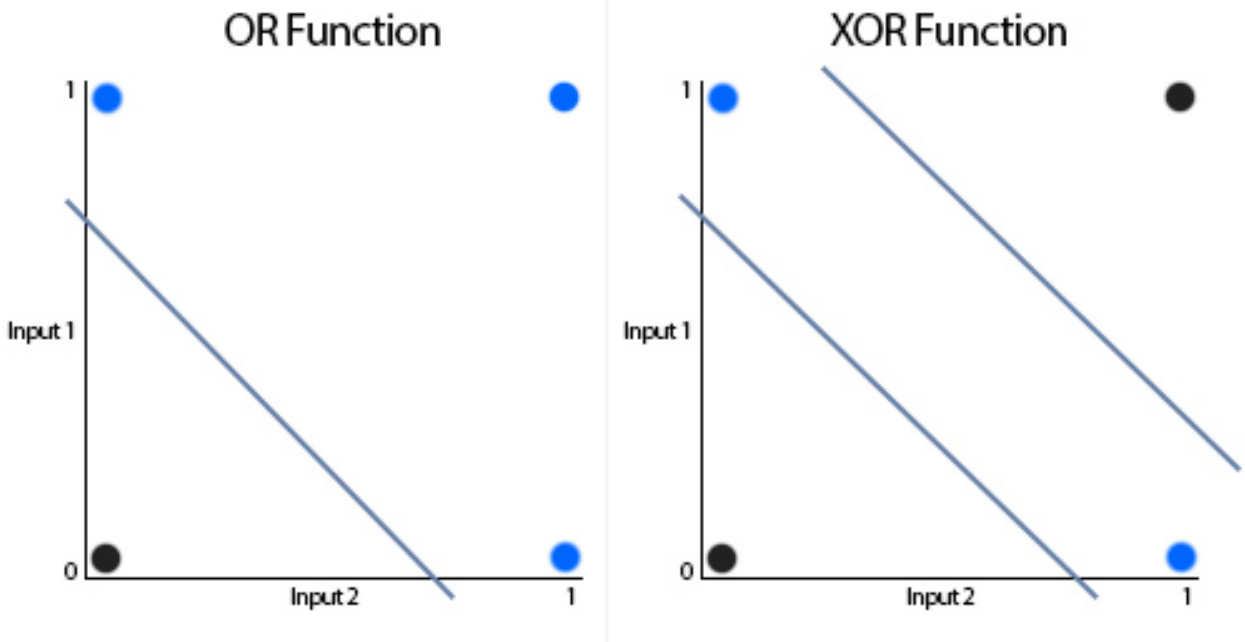
  $\Rightarrow$ **Logistic Regression (LR) Model !**



# Beyond Linear Prediction

## X-OR Problem

- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR:** NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR: Non linear decision function**



# The Multi-Layer Perceptron (MLP)

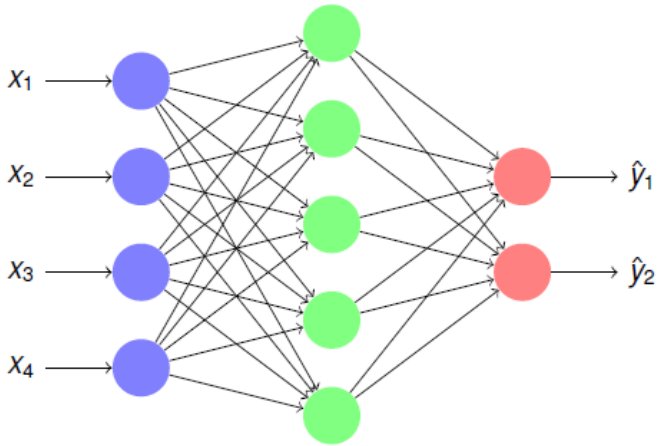



Figure: Perceptron with 1 hidden layer – Credits: R. Herault

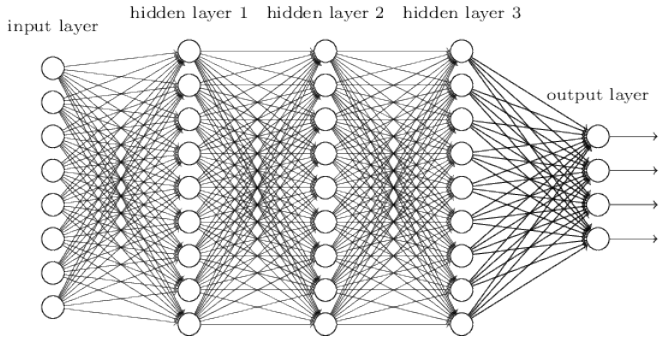
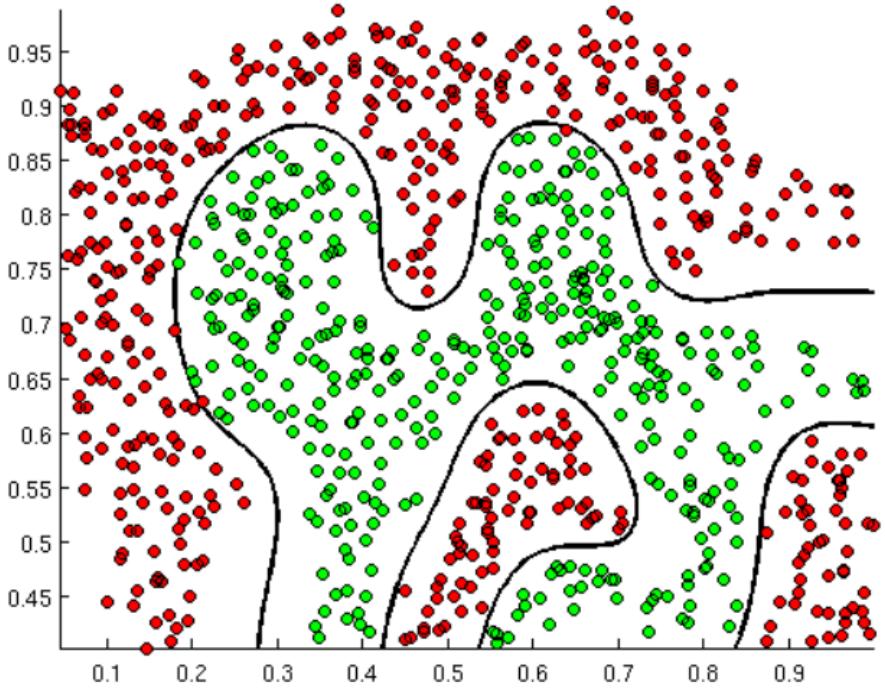



Figure: Stacking more layers, toward "deep learning" – Credits: M. Nielsen

- Basis of the "deep learning" field
- Principle: Stacking layers of neural networks to allow more complex and rich functions
- With a hidden layer, **can approximate any function** given enough hidden units [Cybenko, 1989]
- Can be seen as **different levels of abstraction** from low-level features to the high-level ones

# The Multi-Layer Perceptron (MLP)

- **Neural network with one single hidden layer ⇒ universal approximator [Cybenko, 1989]**
  - Ex for classification: any decision boundaries can be expressed

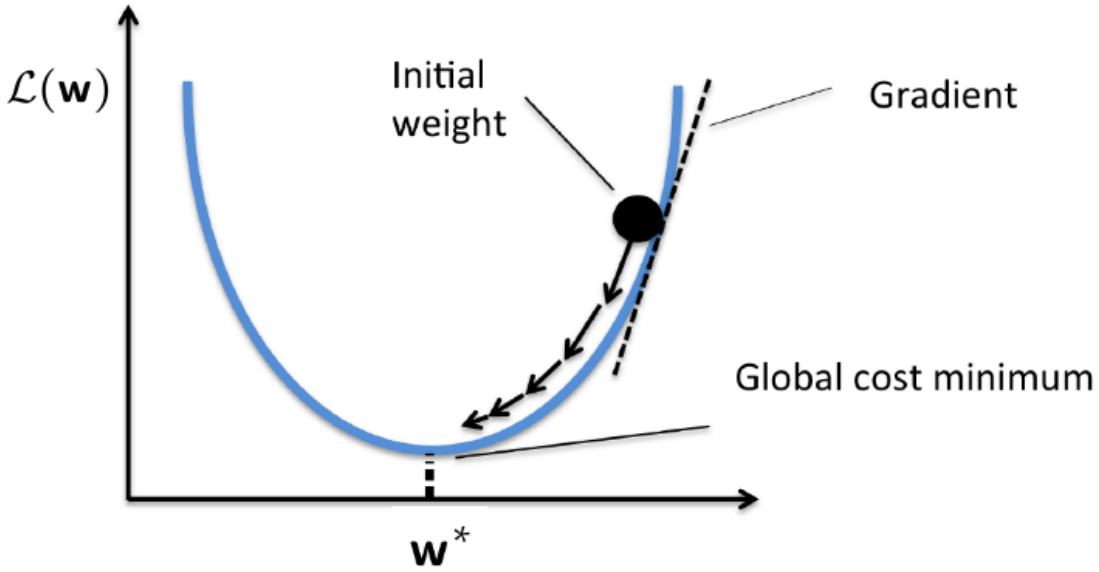


# Training Multi-Layer Perceptron (MLP)

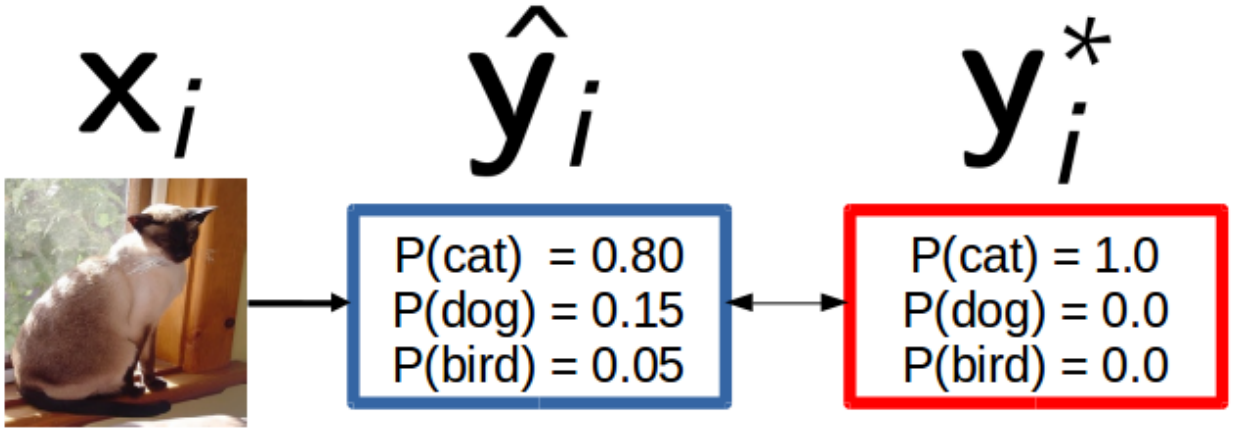
- Input x, output y
- A parametrized model $x \Rightarrow y$: $f_w(x_i) = \hat{y}_i$
- Supervised context: training set $\mathcal{A} = \{(x_i, y_i^*)\}_{i \in \{1,2,\ldots,N\}}$
  - A loss function $\ell(\hat{y}_i, y_i^*)$ for each annotated pair $(x_i, y_i^*)$
- Assumptions: parameters $w \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_w = \frac{\partial \mathcal{L}}{\partial w}$: steepest direction to decrease loss $\mathcal{L}$

# MLP Training

- Gradient descent algorithm:
  - Initialyze parameters w
  - Update: $\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathrm{w}}$
  - Until convergence, *e.g.* $\|\nabla_{\mathrm{w}}\|^2 \approx 0$



# MLP Training: loss function




- **Input $x_i$, ground truth output supervision $y_i^*$**
- One hot-encoding for $y_i^*$:

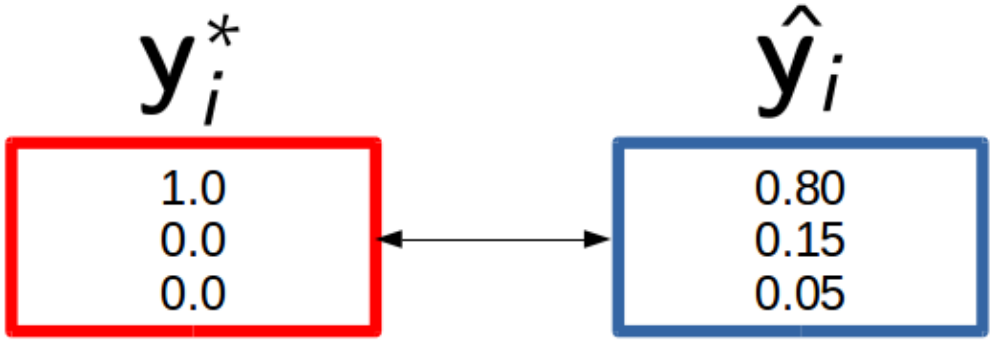
$$y_{c,i}^* = \begin{cases} 1 & \text{if c is the groud truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$

# MLP Training

- Loss function: multi-class Cross-Entropy (CE) $\ell_{CE}$
- $\ell_{CE}$: Kullback-Leiber divergence between $y_i^*$ and $\hat{y}_i$

$$\ell_{CE}(\hat{y}_i, y_i^*) = KL(y_i^*, \hat{y}_i) = -\sum_{c=1}^{K} y_{c,i}^* log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i})$$

- KL asymmetric: $KL(\hat{y}_i, y_i^*) \neq KL(y_i^*, \hat{y}_i)$

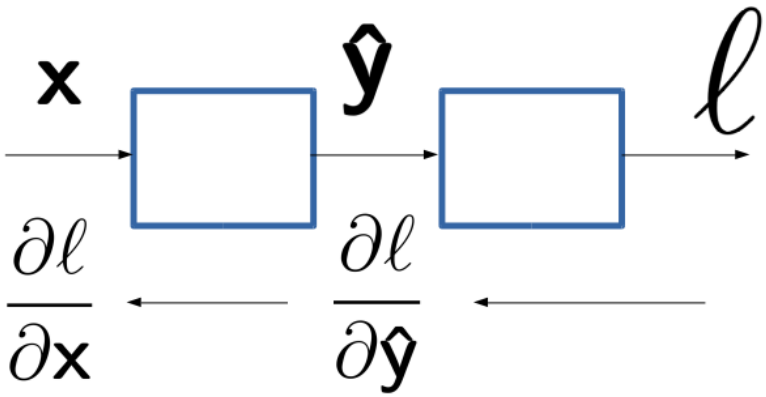



$$KL(y_i^*, \hat{y}_i) = -log(\hat{y}_{c^*,i}) = -log(0.8) \approx 0.22$$

# MLP Training: Backpropagation

- $\mathcal{L}_{CE}(\mathrm{W}, \mathrm{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- $\ell_{CE}$ smooth convex upper bound of $\ell_{0/1}$
  $\Rightarrow$ **gradient descent optimization**
- Gradient descent: $\mathrm{W}^{(t+1)} = \mathrm{W}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{W}}$
  $(\mathrm{b}^{(t+1)} = \mathrm{b}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{b}})$
- Computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{W}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathrm{W}}$ ?
  $\Rightarrow$ **Backpropagation of gradient error!**
  $\Rightarrow$ **Key Property:** chain rule $\frac{\partial \mathrm{x}}{\partial z} = \frac{\partial \mathrm{x}}{\partial \mathrm{y}} \frac{\partial \mathrm{y}}{\partial z}$

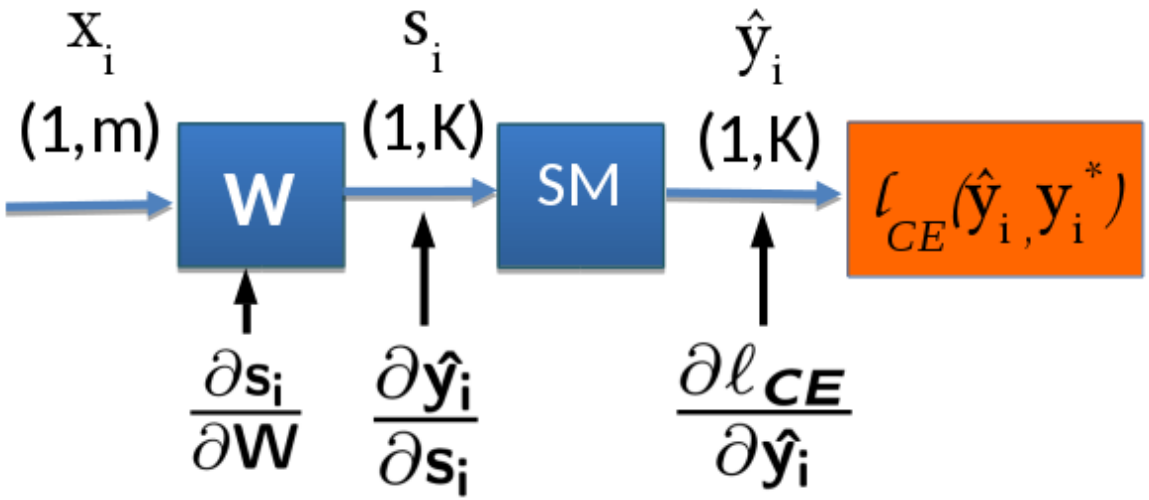
# Chain Rule




$$\frac{\partial \ell}{\partial x} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x}$$

- Logistic regression:

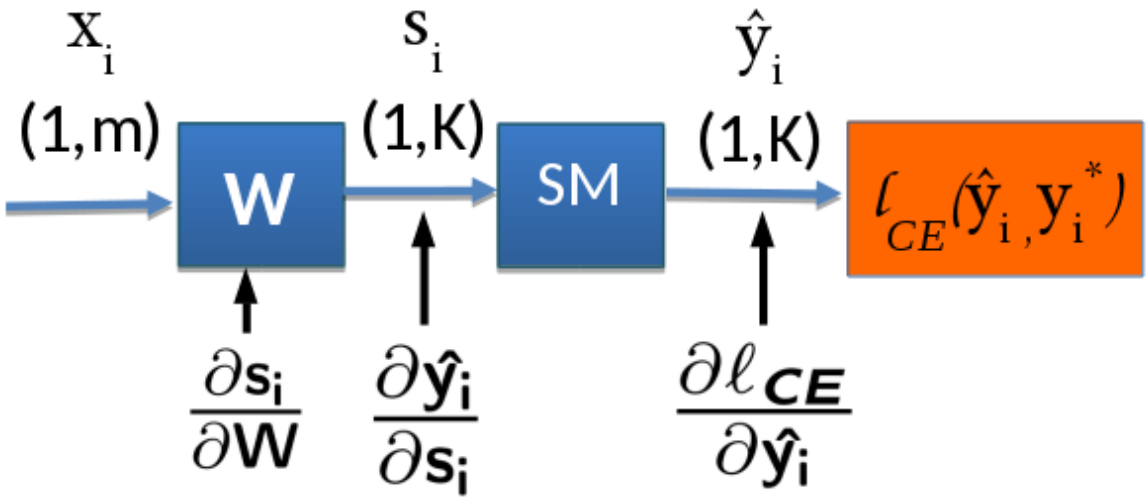
$$\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}$$



# Logistic Regression Training: Backpropagation

$\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}$, $\ell_{CE}(\hat{y}_i, y_i^*) = -log(\hat{y}_{c^*,i}) \Rightarrow$ Update for 1 example:

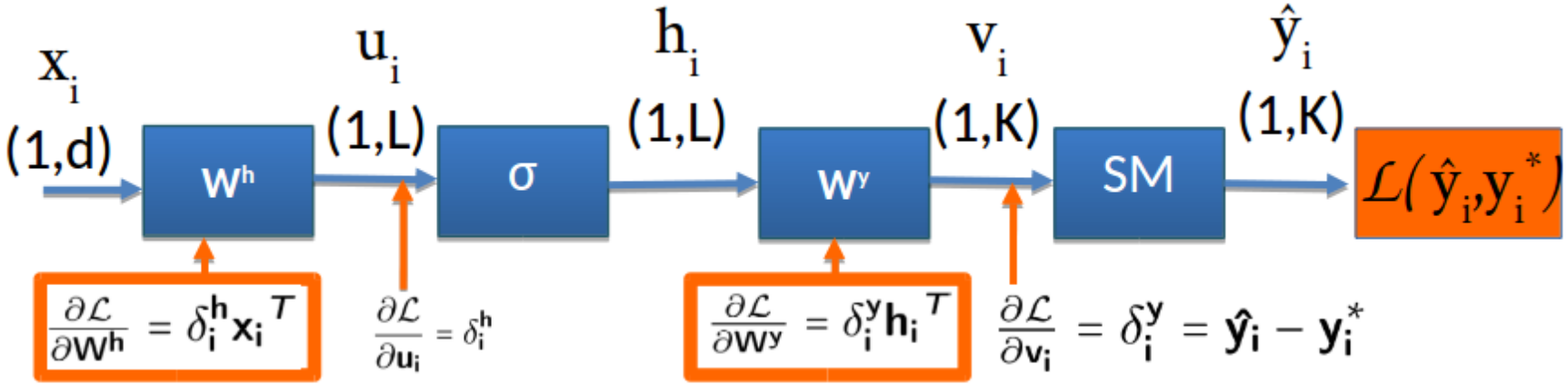
- $\frac{\partial \ell_{CE}}{\partial \hat{y}_i} = \frac{-1}{\hat{y}_{c^*,i}} = \frac{-1}{\hat{y}_i} \odot \delta_{c,c^*}$
- $\frac{\partial \ell_{CE}}{\partial s_i} = \hat{y}_i - y_i^* = \delta_i^y$
- $\frac{\partial \ell_{CE}}{\partial W} = x_i{}^T \delta_i^y$



# Perceptron Training: Backpropagation

- Perceptron *vs* Logistic Regression: adding hidden layer (sigmoid)
- **Goal:** Train parameters $W^y$ and $W^h$ (+bias) with Backpropagation

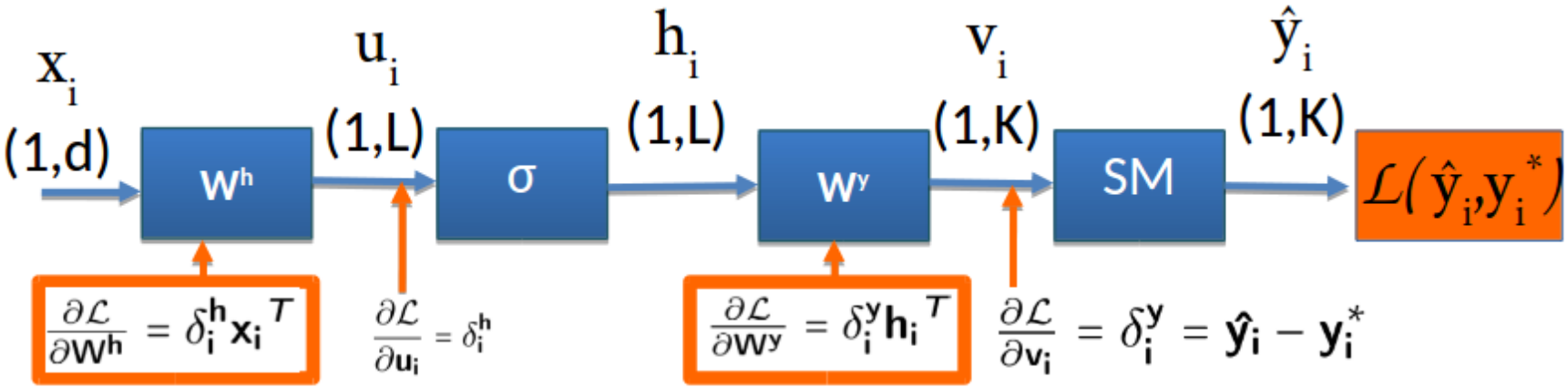
$\Rightarrow$ computing $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^y} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^y}}$ and $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^h} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^h}}$




- Last hidden layer ~ Logistic Regression
- First hidden layer: $\frac{\partial \ell_{CE}}{\partial W^h} = x_i^T \frac{\partial \ell_{CE}}{\partial u_i} \Rightarrow$ **computing** $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h$

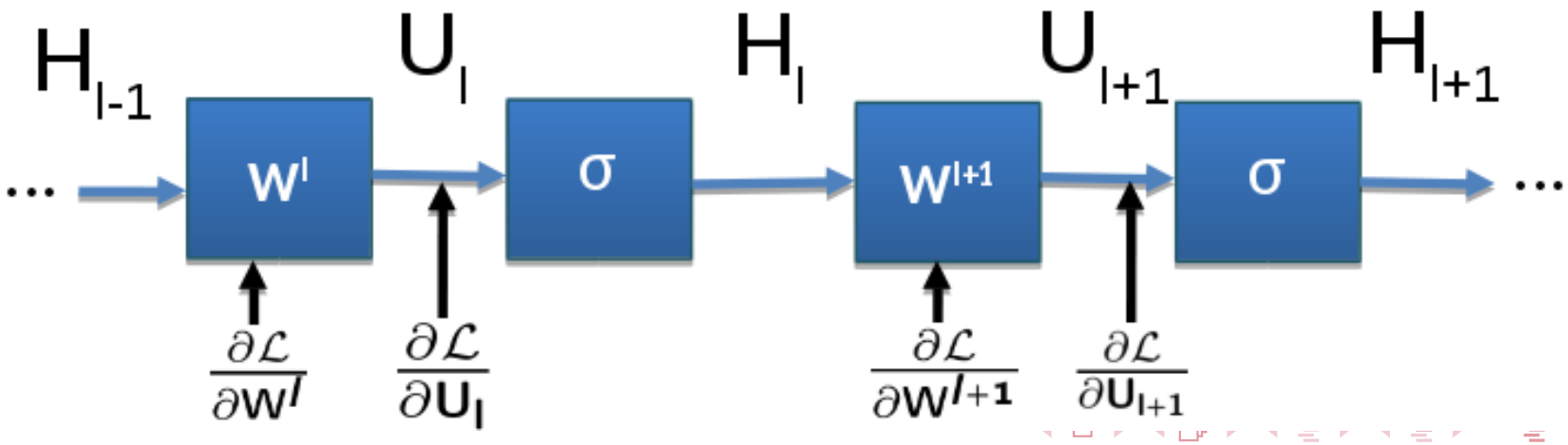
# Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h \Rightarrow$ use chain rule: $\frac{\partial \ell_{CE}}{\partial u_i} = \frac{\partial \ell_{CE}}{\partial v_i} \frac{\partial v_i}{\partial h_i} \frac{\partial h_i}{\partial u_i}$
- ... Leading to:
  $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h = \delta_i^{y\,T} W^y \odot \sigma'(h_i) = \delta_i^{y\,T} W^y \odot (h_i \odot (1 - h_i))$



# Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update ~ Perceptron: **assuming** $\frac{\partial \mathcal{L}}{\partial U_{l+1}} = \Delta^{l+1}$ **known**
  - $\boxed{\frac{\partial \mathcal{L}}{\partial W^{l+1}} = H_l{}^T \Delta^{l+1}}$
  - $\boxed{\text{Computing } \frac{\partial \mathcal{L}}{\partial U_l} = \Delta^l}$ $(= \Delta^{l+1}{}^T W^{l+1} \odot H_l \odot (1 - H_l)$ sigmoid$)$
  - $\boxed{\frac{\partial \mathcal{L}}{\partial W^l} = H_{l-1}{}^T \Delta^{h_l}}$



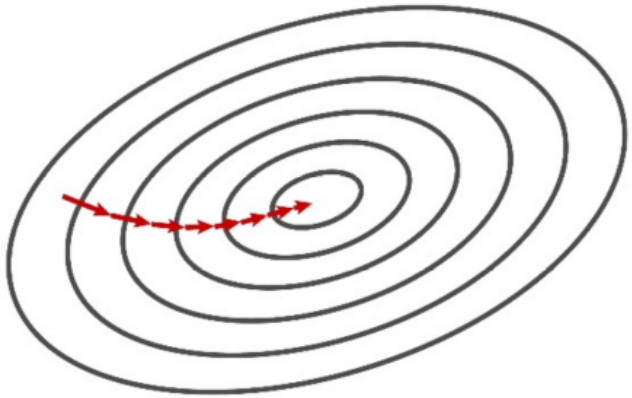
# Neural Network Training: Optimization Issues

- Classification loss over training set (w):

$$\mathcal{L}_{CE}(\mathrm{w}) = \frac{1}{N}\sum_{i=1}^{N}\ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$$

- Gradient descent optimization:

$$\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} - \eta\frac{\partial\mathcal{L}_{CE}}{\partial \mathrm{w}}\left(\mathrm{w}^{(t)}\right) = \mathrm{w}^{(t)} - \eta\nabla_{\mathrm{w}}^{(}$$



- Gradient $\nabla_{\mathrm{w}}^{(t)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial\ell_{CE}(\hat{\mathrm{y}}_i,\mathrm{y}_i^*)}{\partial \mathrm{w}}\left(\mathrm{w}^{(t)}\right)$ linea..,

  scales wrt:
  - w dimension
  - Training set size

$\Rightarrow$ **Too slow even for moderate dimensionality & dataset size!**

# Stochastic Gradient Descent

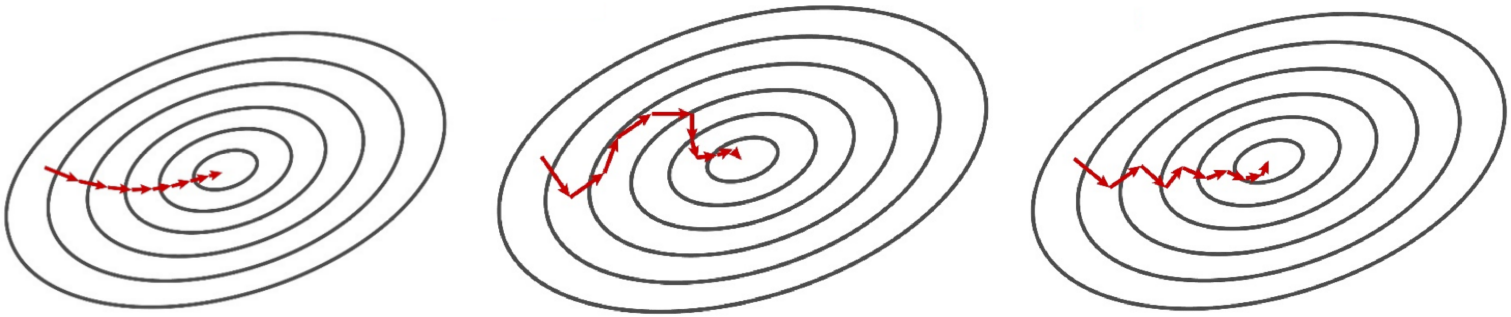
- **Solution:** approximate $\nabla_{\mathrm{w}}^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$ with subset

  $\Rightarrow$ **Stochastic Gradient Descent (SGD)**
  - Use a single example (online):

$$\nabla_{\mathrm{w}}^{(t)} \approx \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$$

  - Mini-batch: use $B < N$ examples:

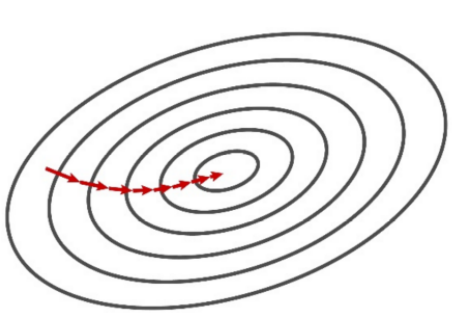
$$\nabla_{\mathrm{w}}^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$$



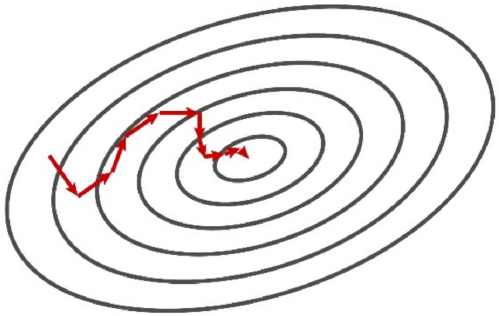
Full gradient SGD (online) SGD (mini-batch)

# Stochastic Gradient Descent

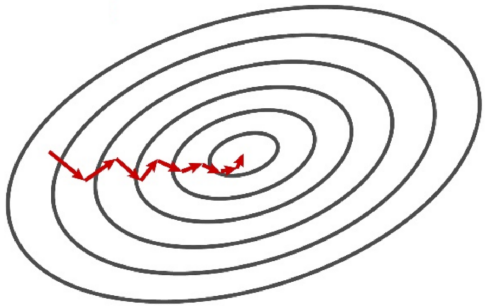
- **SGD: approximation of the true Gradient $\nabla_w$ !**
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets



Full gradient



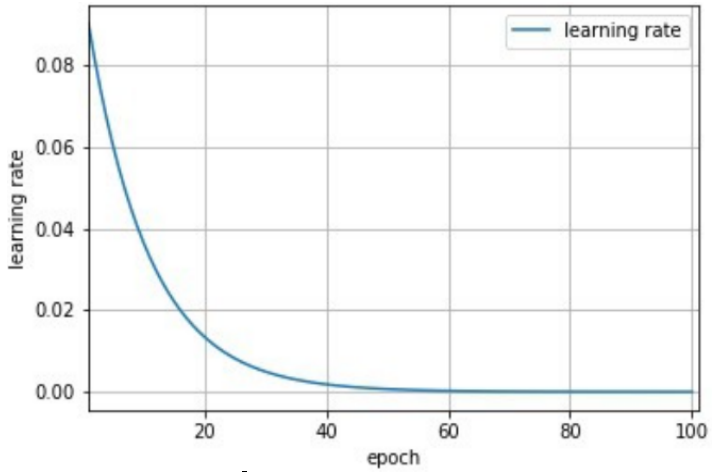
SGD (online)



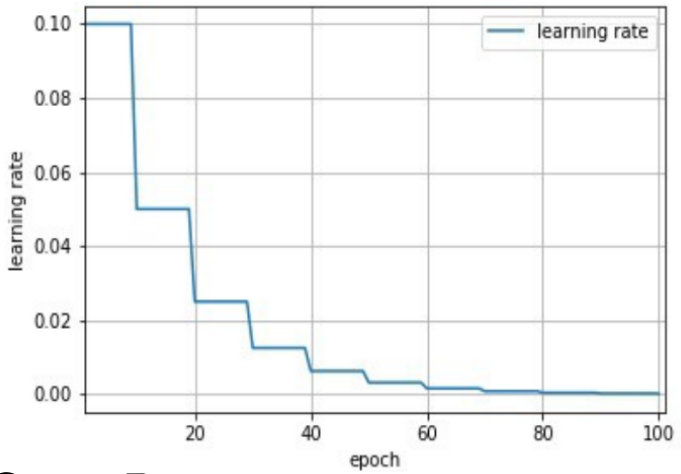
SGD (mini-batch)

# Optimization: Learning Rate Decay

- Gradient descent optimization: $w^{(t+1)} = w^{(t)} - \eta \nabla_w^{(t)}$
- $\eta$ setup ? $\Rightarrow$ open question
- Learning Rate Decay: decrease $\eta$ during training progress
  - Inverse (time-based) decay: $\eta_t = \frac{\eta_0}{1+r \cdot t}$, $r$ decay rate
  - Exponential decay: $\eta_t = \eta_0 \cdot e^{-\lambda t}$
  - Step Decay $\eta_t = \eta_0 \cdot r^{\frac{t}{t_u}}$ ...



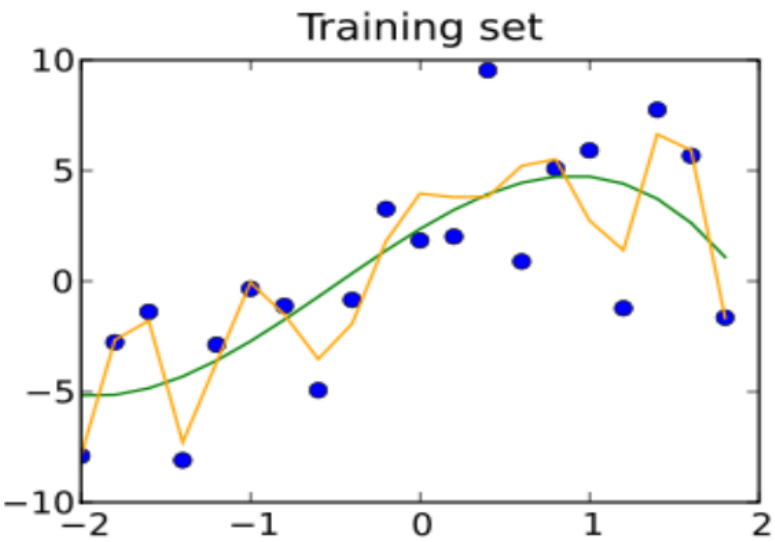

Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1$s)

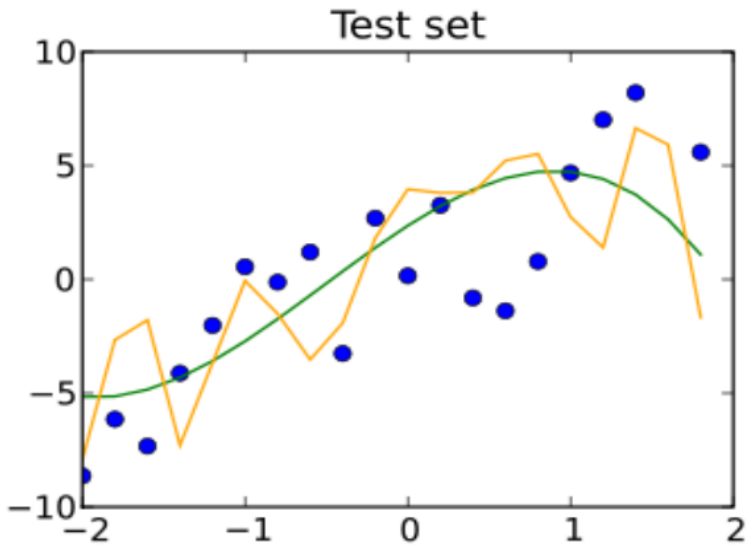



Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)

# Generalization and Overfitting

- **Learning:** minimizing classification loss $\mathcal{L}_{CE}$ over training set
  - Training set: sample representing data *vs* labels distributions
  - **Ultimate goal:** train a prediction function with low prediction error on the **true (unknown) data distribution**




$\mathcal{L}_{train} = 4$, $\mathcal{L}_{train} = 9$ $\mathcal{L}_{test} = 15$, $\mathcal{L}_{test} = 13$

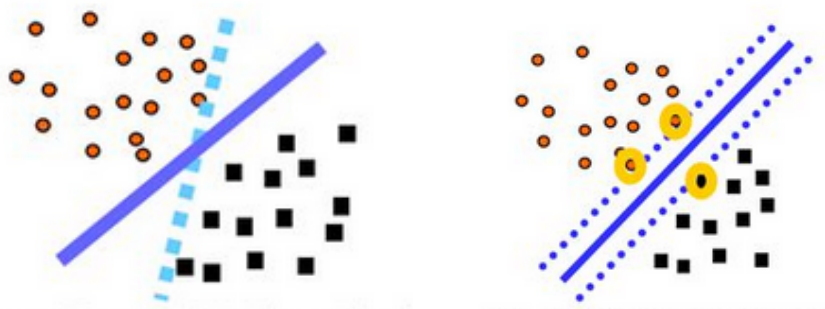
$\Rightarrow$ **Optimization $\neq$ Machine Learning!**
$\Rightarrow$ **Generalization / Overfitting!**

# Regularization

- **Regularization:** improving generalization, *i.e.* test ($\neq$ *train*) performances
- Structural regularization: add ***Prior*** $R(\mathrm{w})$ in training objective:

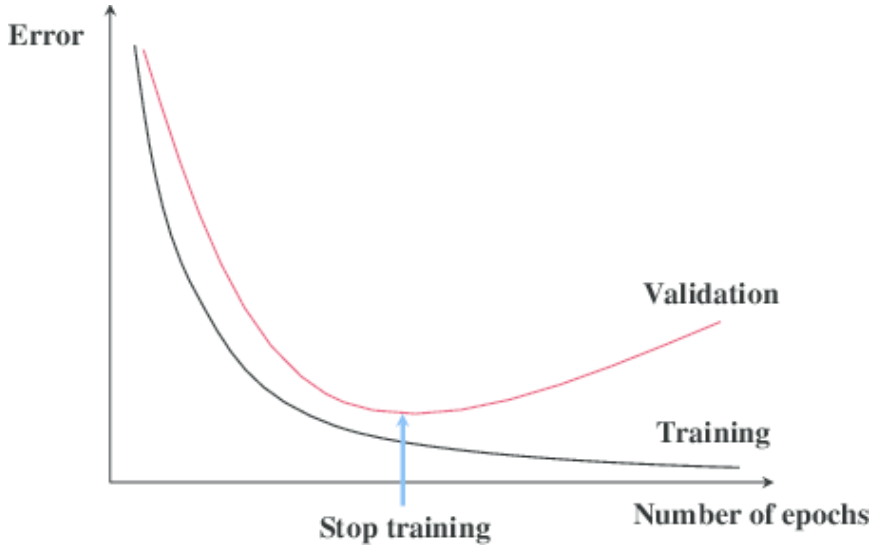
$$\mathcal{L}(\mathrm{w}) = \mathcal{L}_{CE}(\mathrm{w}) + \alpha R(\mathrm{w})$$

- $L^2$ **regularization: weight decay**, $R(\mathrm{w}) = ||\mathrm{w}||^2$
  - Commonly used in neural networks
  - Theoretical justifications, generalization bounds (SVM)
- Other possible $R(\mathrm{w})$: $L^1$ regularization, dropout, *etc*



# Regularization and hyper-parameters

- **Neural networks:** hyper-parameters to tune:
  - **Training parameters:** learning rate, weight decay, learning rate decay, # epochs, *etc*
  - **Architectural parameters:** number of layers, number neurones, non-linearity type, *etc*
- **Hyper-parameters tuning:** ⇒ improve generalization: estimate performances on a validation set



# Outline

1 Context

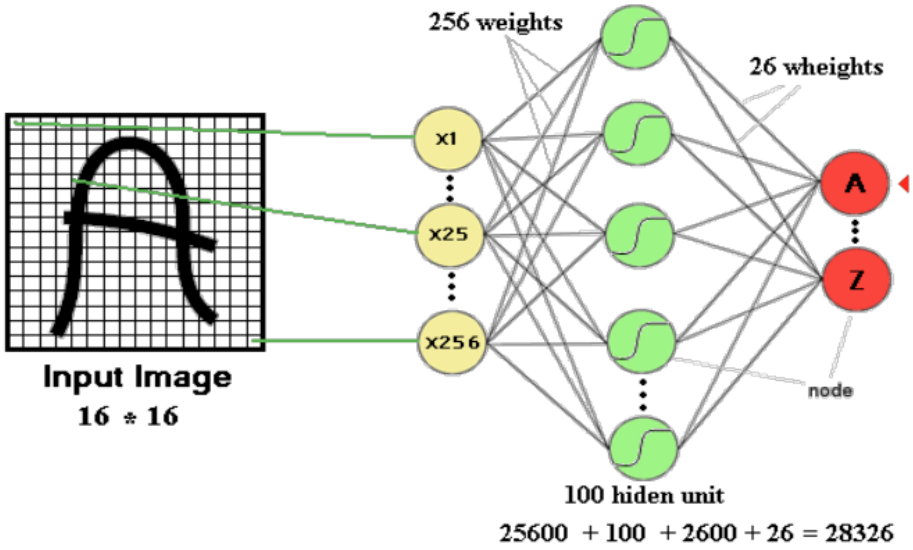
2 Neural Nets

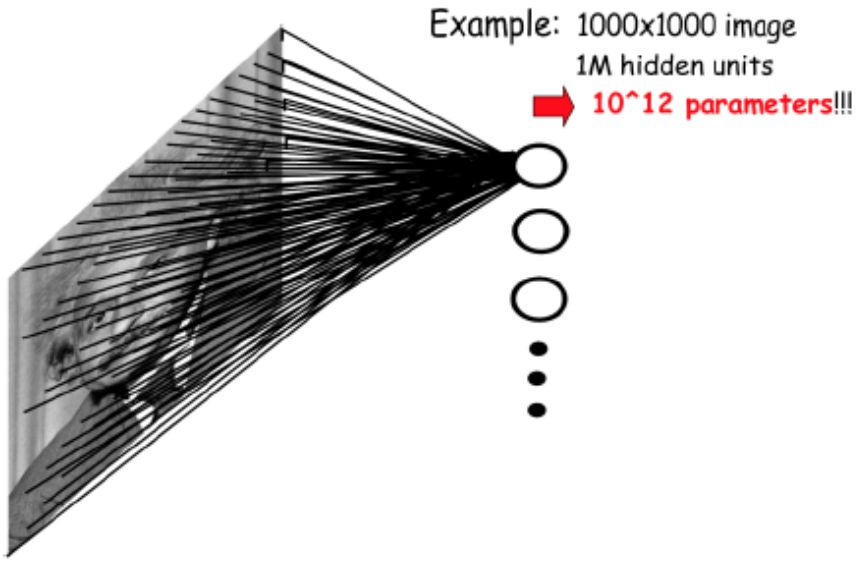
3 Convolutionnal Neural Networks

4 Case Study: LeNet Model

# Fully Connected Networks: Limitations

- Scalability issue with Fully Connected Networks (MLP)

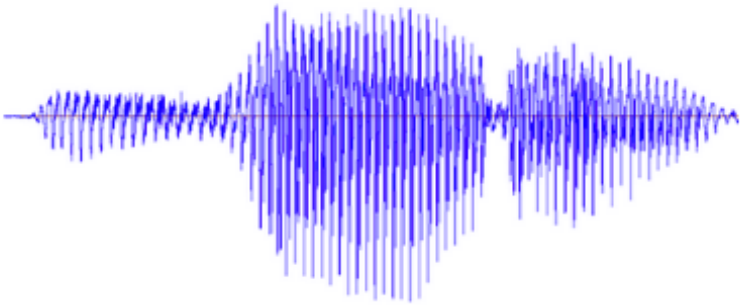


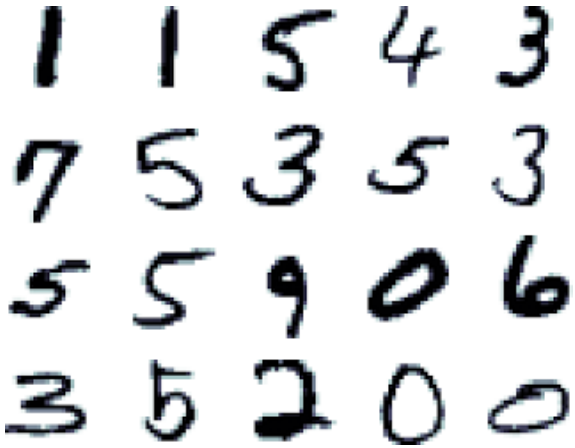




$\Rightarrow$ # Parameter explosion even for a single hidden layer !

# Limitations of Fully Connected Networks

- **Signal data: importance of local structure**
  - 1D signals: local temporal structure
  - 2D signal data: local spatial structure



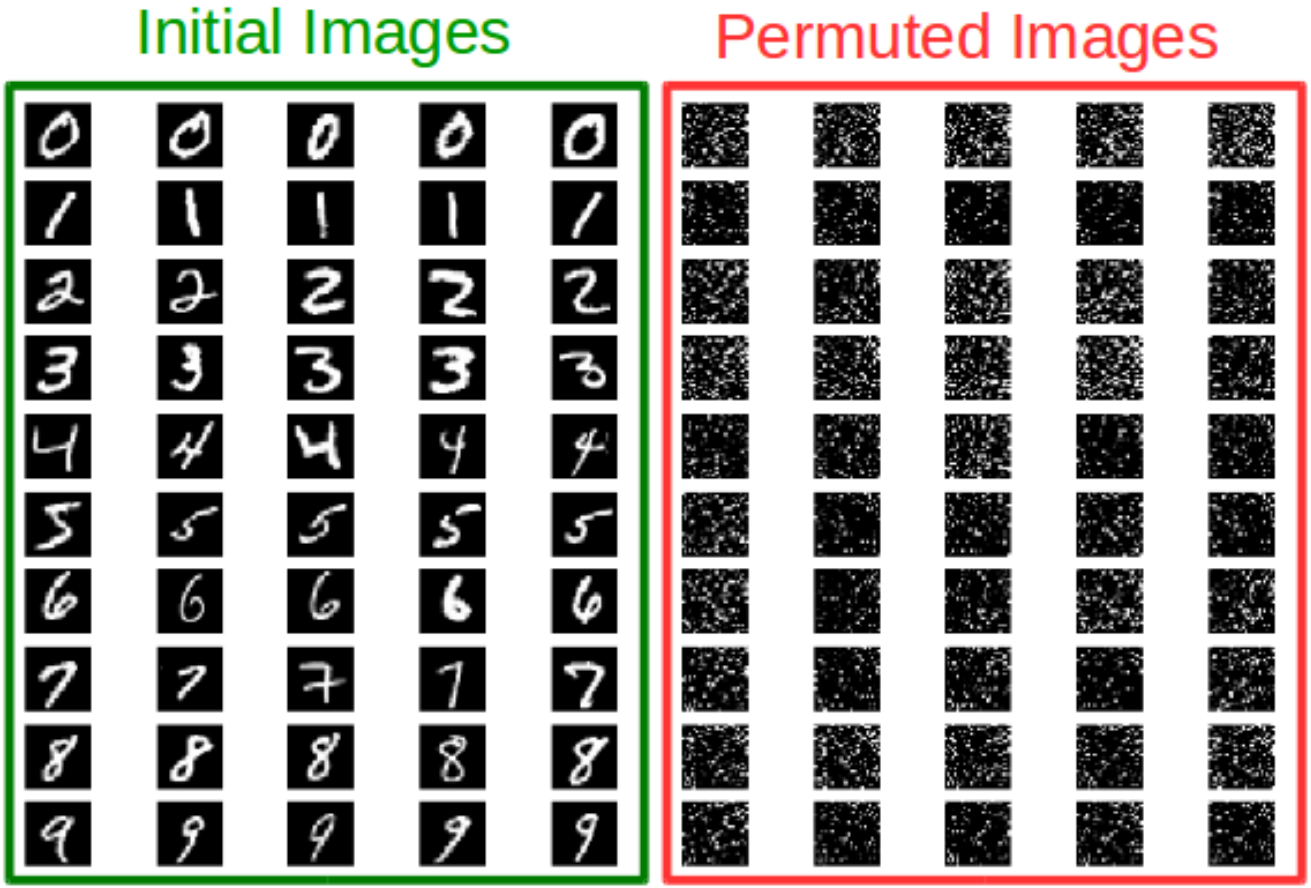



# Limitations of Fully Connected Networks

- BUT: vectorial representation of inputs: dimensions arbitrary!

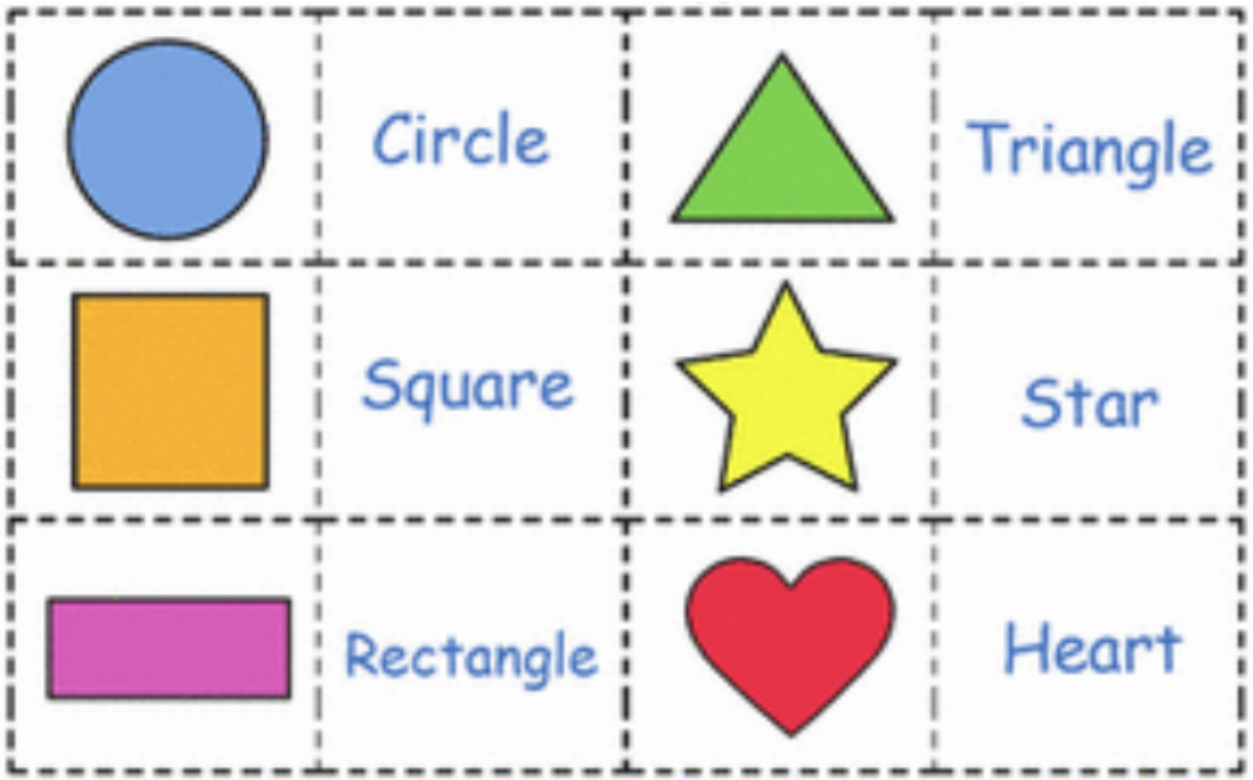
# Limitations of Fully Connected Networks

- MNIST ex: same performances with initial and permuted images!
  - However, local information obviously useful



# Fully Connected Networks: Limitations

- Fully connected networks: no assumption on data structure
  - Structure can be learned but need lots of annoatated data
  - Prior knowlege on data structure ⇒ useful
- Example: MLP training for shape recognition from color images

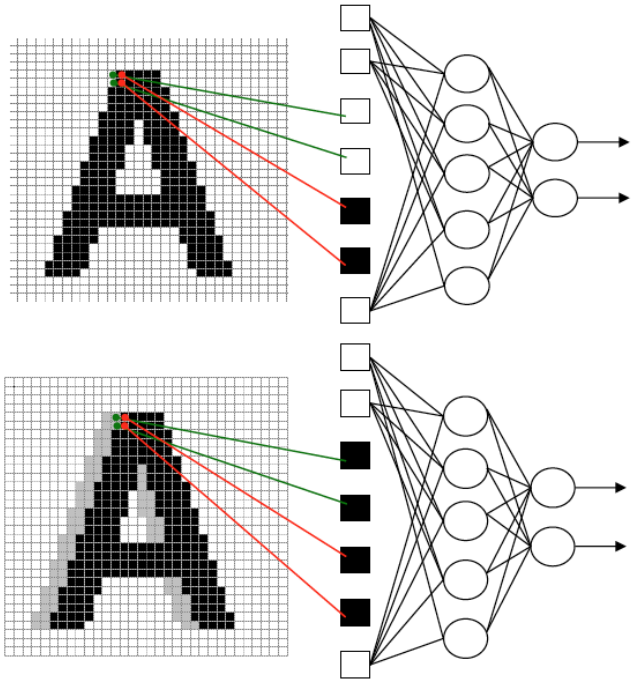


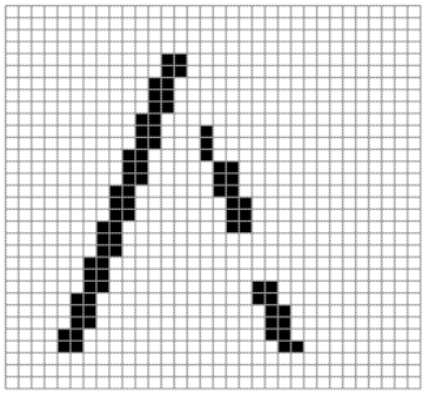

Input image encoding:

- Color (RBG) ?
- Grayscale $L = \frac{R+B+G}{3}$ ?

# Fully Connected Networks: Limitations

- Invariance and robustness to deformation (stability)
- What we expect:
  - Small deformation in input space ⇒ similar representations
  - Large transfo in input space ⇒ very dissimilar representations
- Example (image): impact of a 2 pixel shift





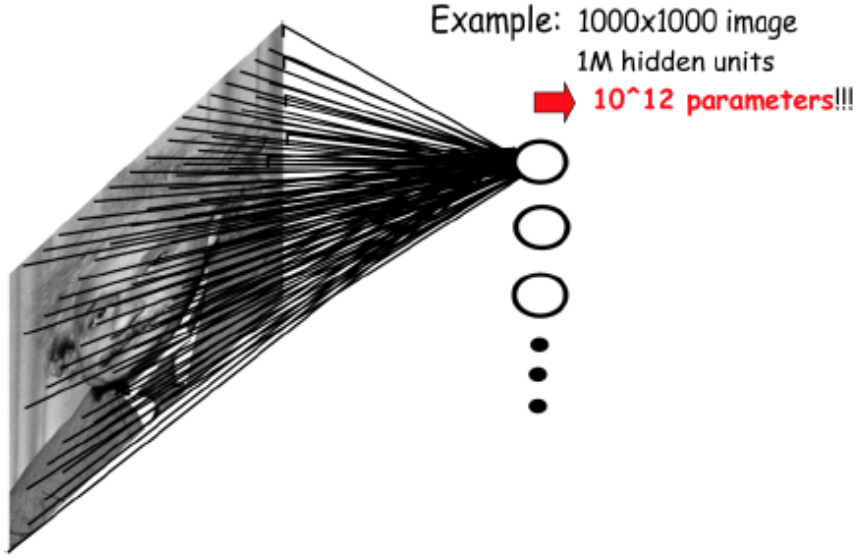
154 input change
from 2 shift left
77 : black to white
77 : white to black

@LeCun

# Fully Connected Networks: Limitations

## Conclusion of MLP on raw data

- Brute force connection of images as input of MLP NOT a good idea
  - No Invariance/Robustness of the representation because topology of the input data completely ignored
    $\Rightarrow$ *e.g.* indifferent to permutations of input pixel
  - Nb of weights grows largely with the size of the input image

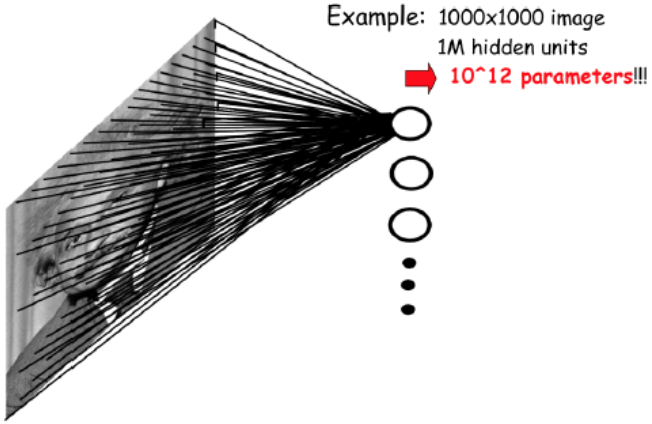


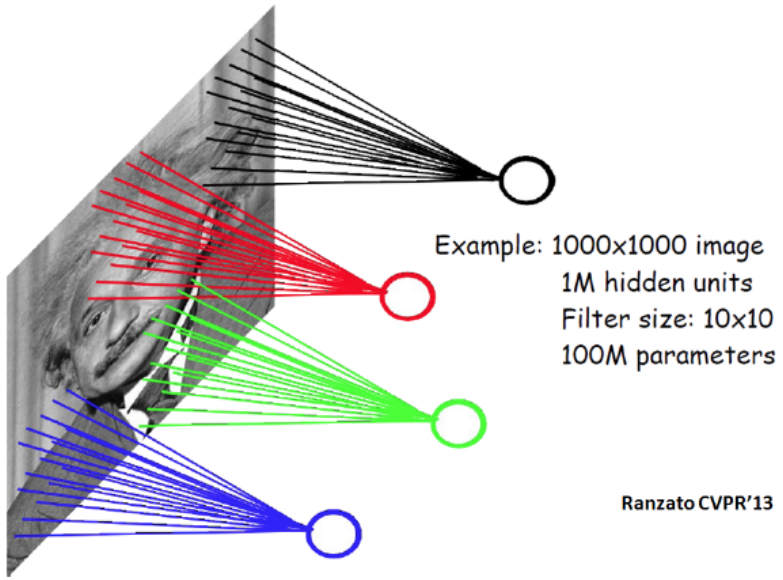

**$\Rightarrow$ How keep spatial topology?**
**$\Rightarrow$ How to limit the number of parameters?**

# Taking advantage of structure: Convolution

## How to limit the number of parameters?

1. Sparse connectivity: hidden unit only connected to a local patch
   - Weights connected to the patch: **filter** or **kernel**
   - Inspired by biological systems: cell only sensitive to a small sub-region of the input space (receptive field). Many cells tiled to cover the entire visual field

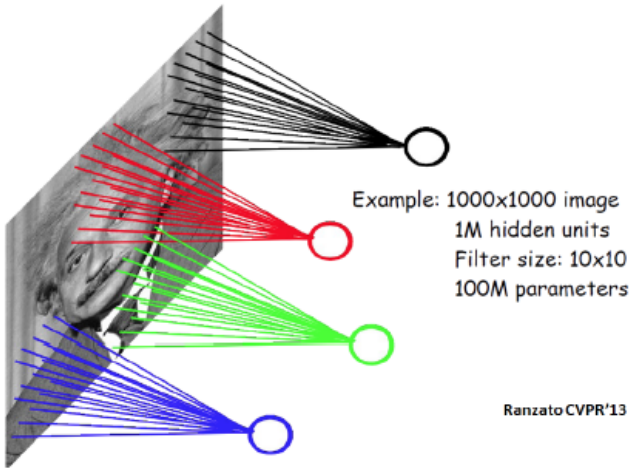


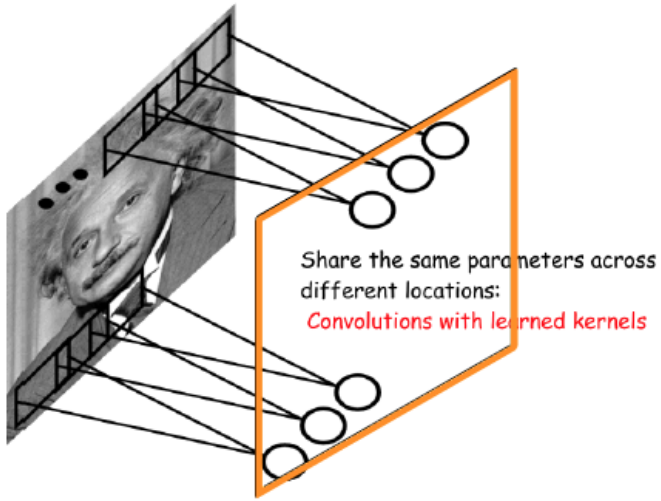



# Taking advantage of structure: Convolution

## How to limit the number of parameters?

2. Shared Weights
   - Hidden nodes at different locations share the same weights
     - Substantially reduces the number of parameters to learn
   - Keep spatial information in a 2D feature map (hidden layer map)




⇒ Computing responses at hidden nodes equivalent to convolving input image with a linear filter (learned)

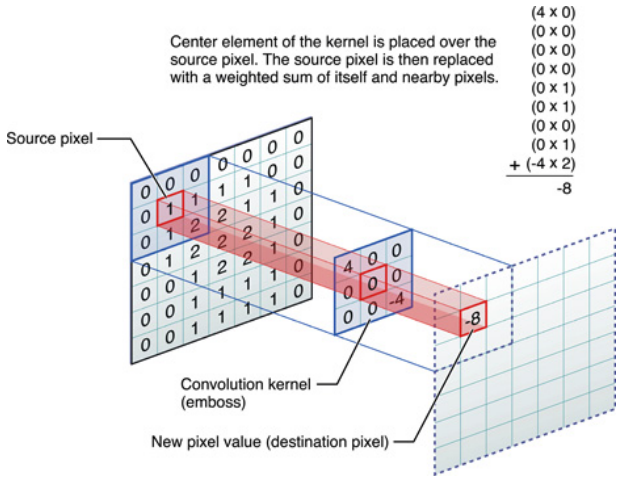
⇒ A learned filter as a feature detector

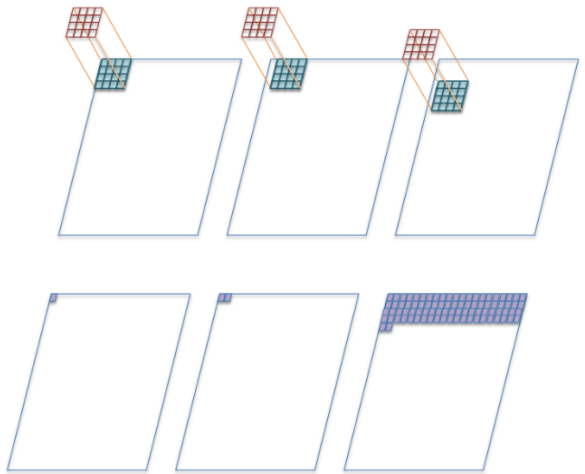
# Convolution: Scalar Images

- 2D convolution with a Finite Impulse Response (FIR) $h$ of size $d$ (odd):

$$f'(i,j) = (f \star h)(i,j) = \sum_{n=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{m=-\frac{d-1}{2}}^{\frac{d-1}{2}} f(i-n, j-m) h(n,m)$$

- Simply centering filer $h$ in pixel $(x, y)$ $\Rightarrow$ weighted sum



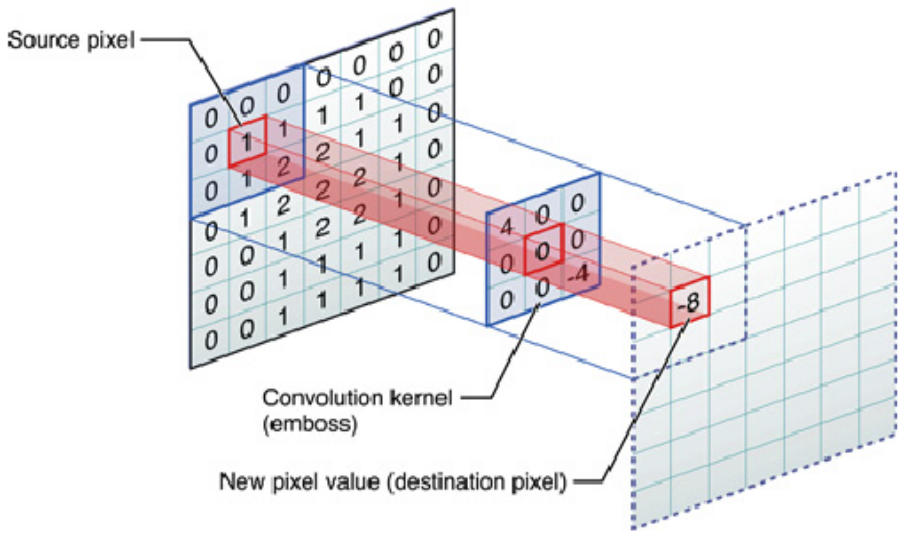



- Output for 1 filer (resp. $K$ filters): 1 2D map (resp. $K$ 2D maps)

# 2D Convolution *vs* Cross-Correlation

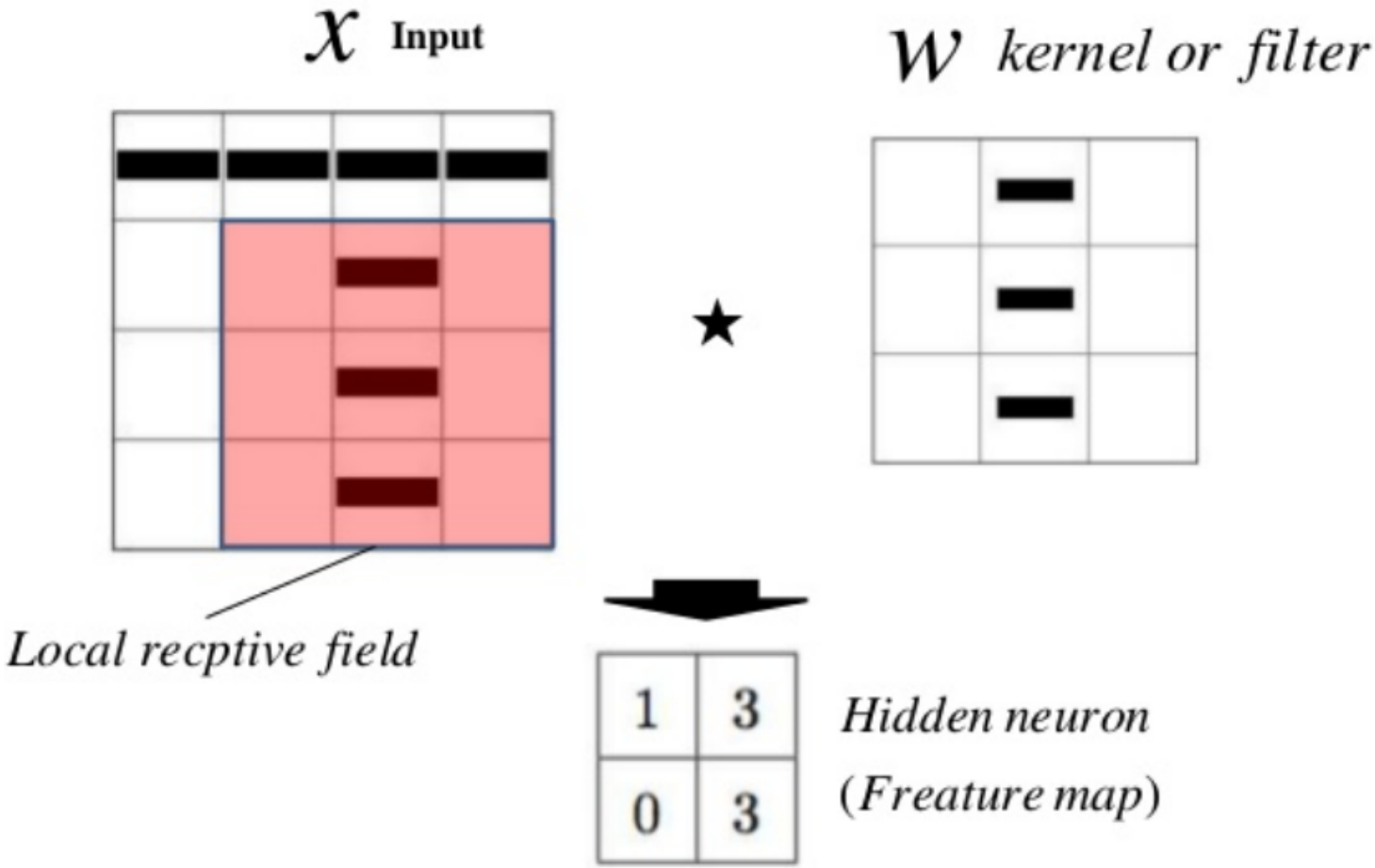
- 2D Convolution:
$$f'(i,j) = (f \star h)(i,j) = \sum_n \sum_m f(i-n, j-m) h(n,m)$$
- Cross-Correlation:
$$f'(i,j) = (f \otimes h)(i,j) = \sum_n \sum_m f(i+n, m+j) h(n,m)$$
  - Cross-Correlation $\sim$ Convolution without symmetrizing mask!

$$h = \begin{pmatrix} -4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix} \Rightarrow g = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{pmatrix}$$



# 2D Convolution / Cross-Correlation: Example

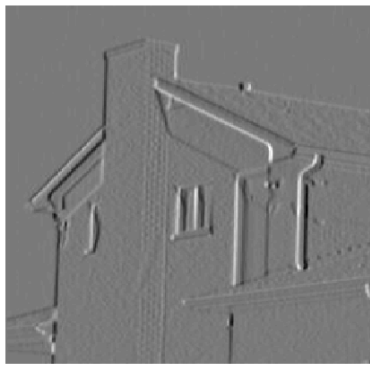
- **Cross-Correlation: output maps $\Leftrightarrow$ location in input image similar to mask**



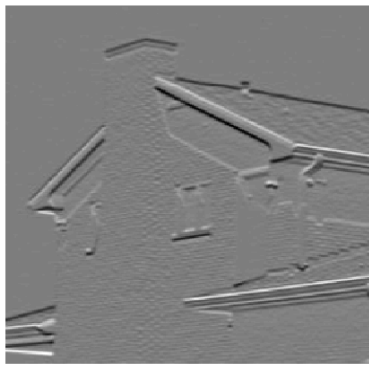
# Convolution: Example for Gradient Computation

- Gradient: $\overrightarrow{G}(x,y) = \begin{pmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{pmatrix}^T = \begin{pmatrix} Ix & Iy \end{pmatrix}^T$
- Convolution approximation: $I_x \approx I \star M_x$, $I_y \approx I \star M_y$

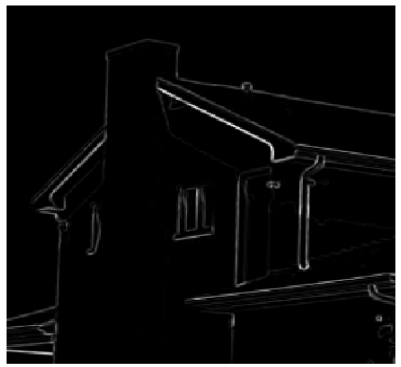
$$M_x = \frac{1}{4} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad M_y = \frac{1}{4} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



$I_x \approx I \star M_x$



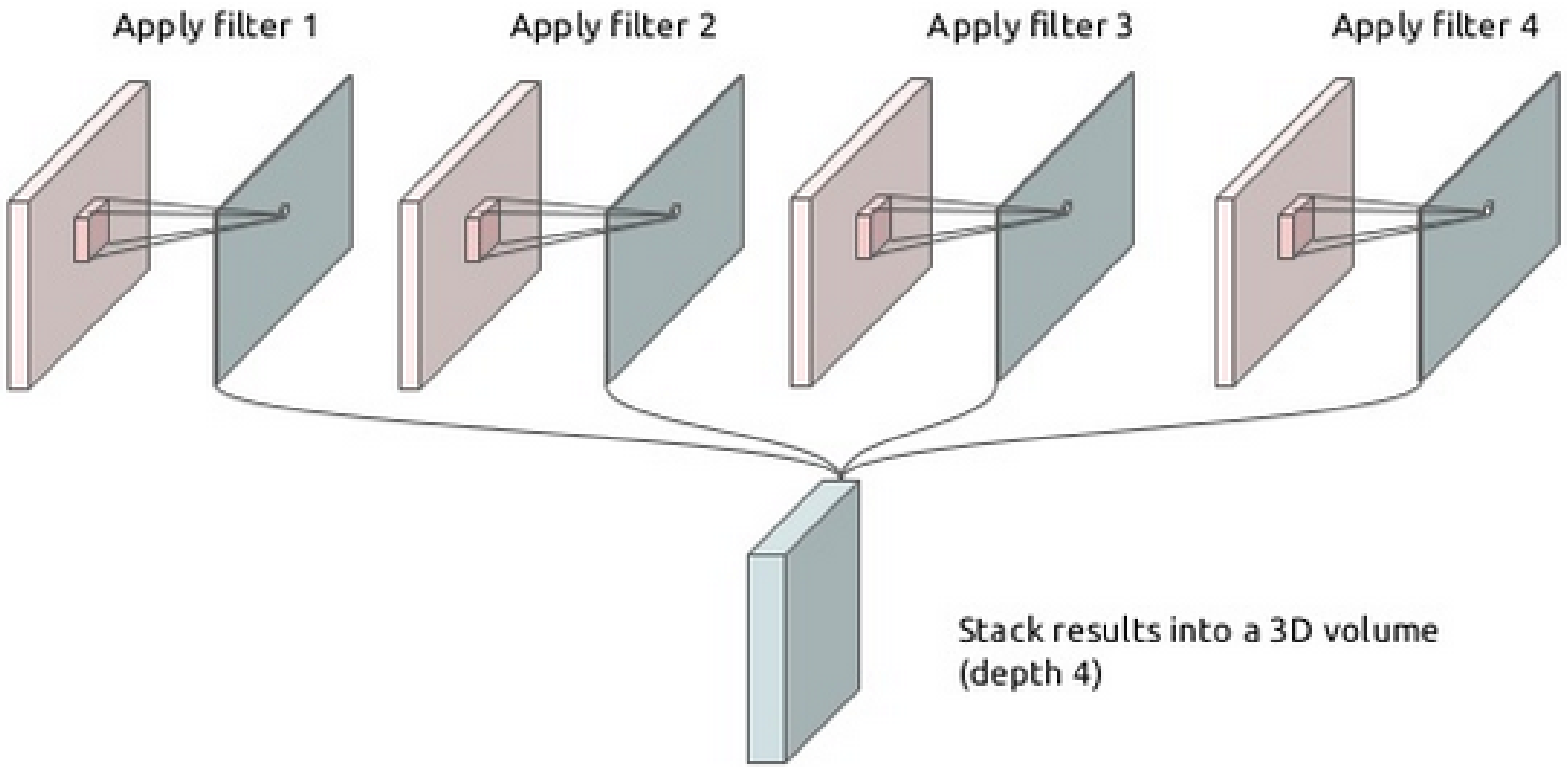
$I_y \approx I \star M_y$



$I_e = Ix^2 + Iy^2$

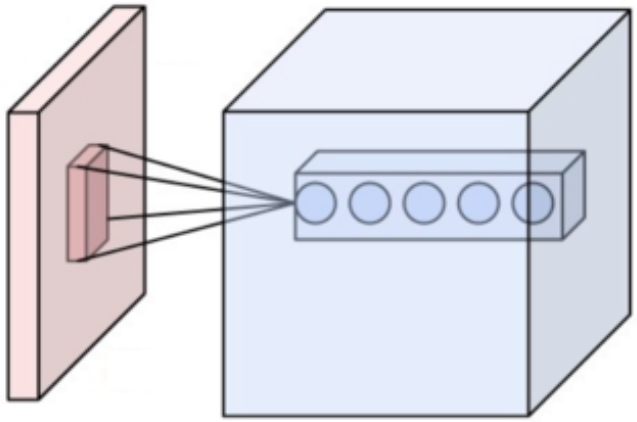
# Convolution Layer

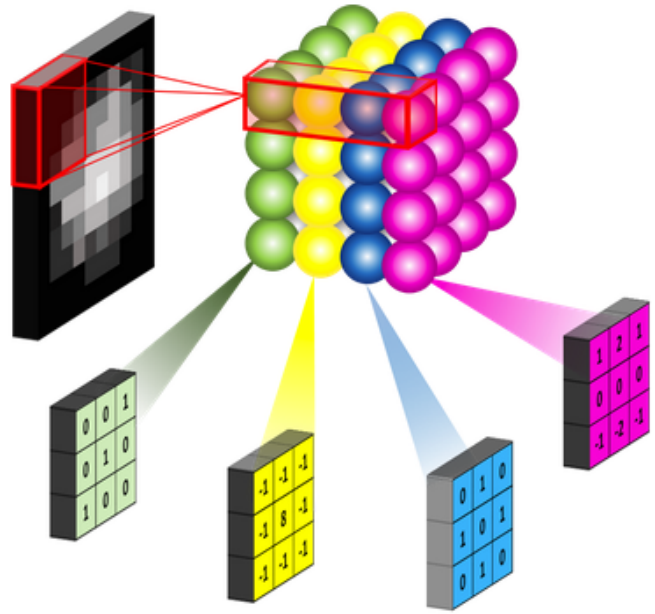
- 2D convolution: each filter ⇒ 2D map (image)
- Convolution Layer: staking maps from multiple Filters
  ⇒ **Tensor: multi-dimensional array**



# Convolution Layer

- Tensor: stacking several filters outputs
  - Depth $\Leftrightarrow$ # filters
  - Each spatial position: output for the different filters
- Ex: 2D conv with gray-scale images, input tensor depth=1
- **Convolution on color images / hierarchies:**
  - **Convolution on tensors!**
  - **Input Tensor $\Rightarrow$ output Tensor**

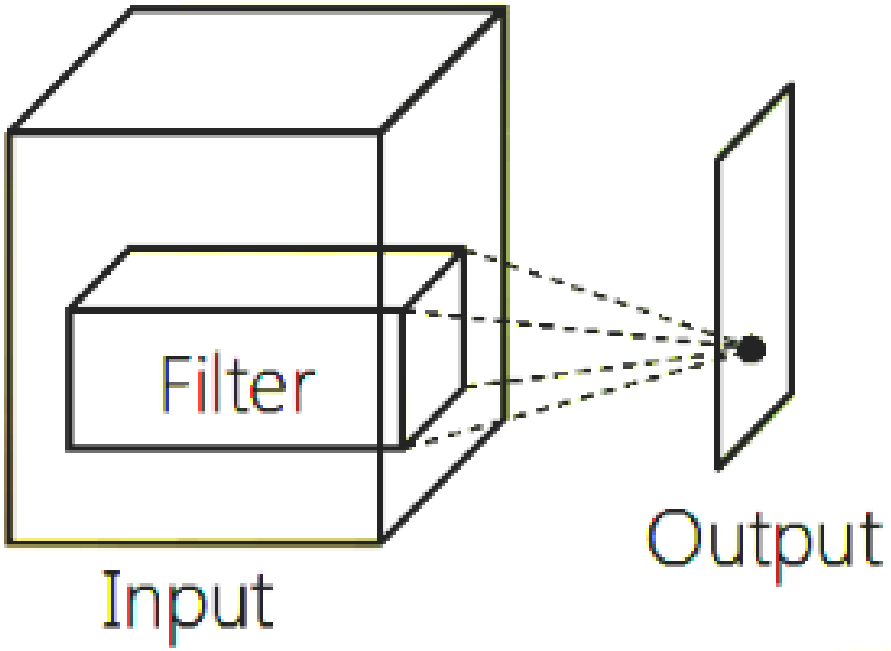


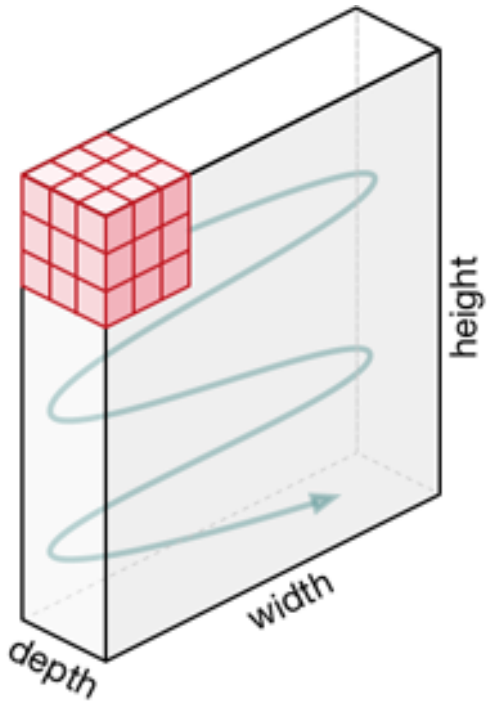


# Convolution Layer for Tensors

$$f'(i,j) = (f \star h)(i,j) = \sum_{k=1}^{K} \sum_{n=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{m=-\frac{d-1}{2}}^{\frac{d-1}{2}} f(i-n, m-j, k) h(n,m,k) \; + \; \mathrm{b}$$

- **Convolution: linear, bias b $\Rightarrow$ affine**

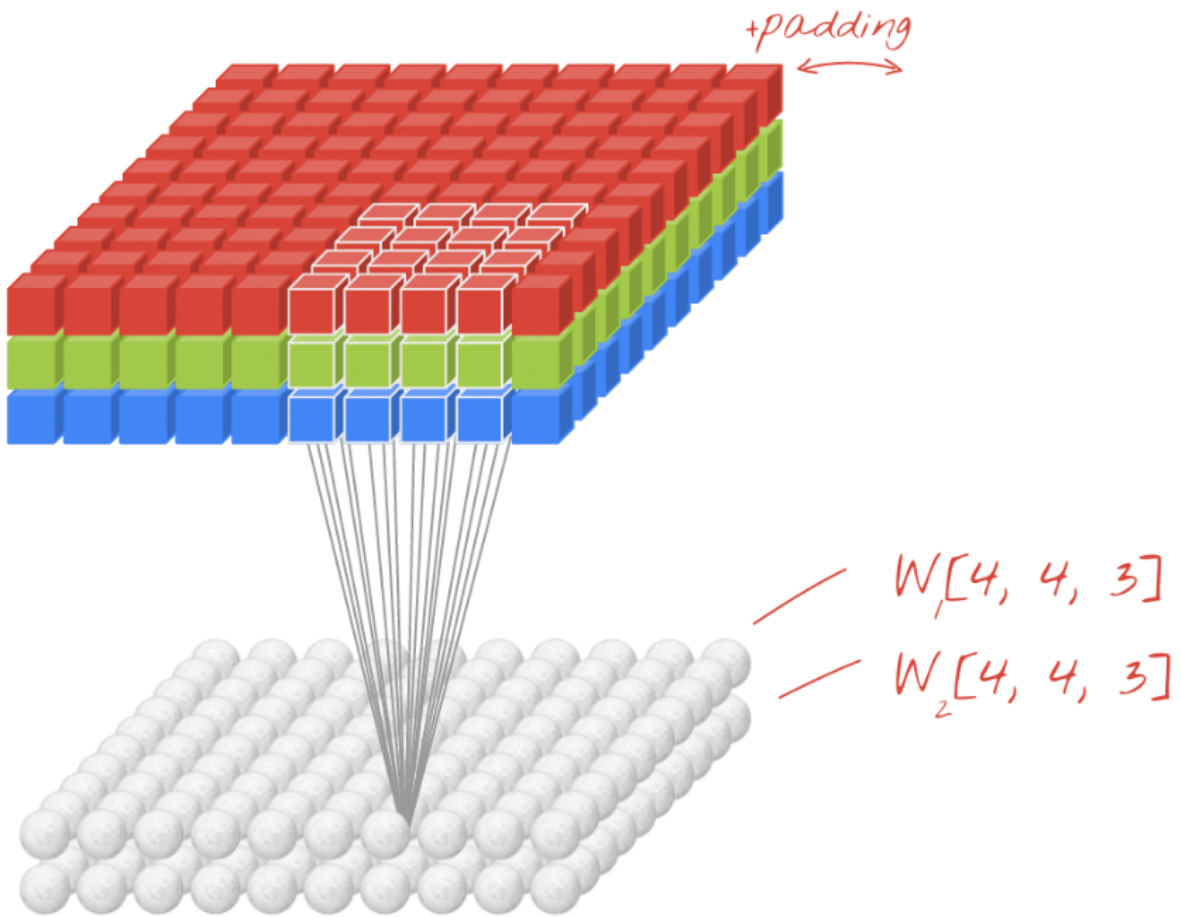



- **Filtering on depth:** correlation between feature maps

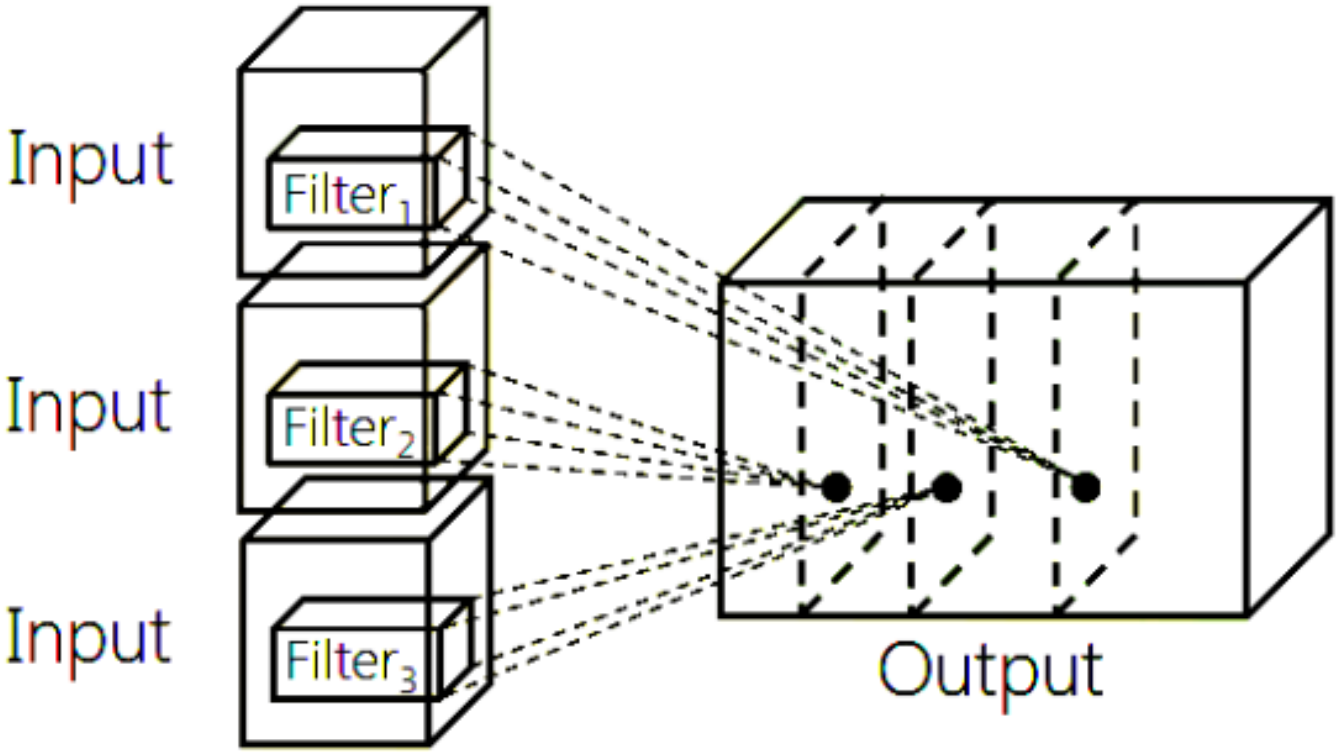
# Convolution Layer for Tensors

## Ex: input color image



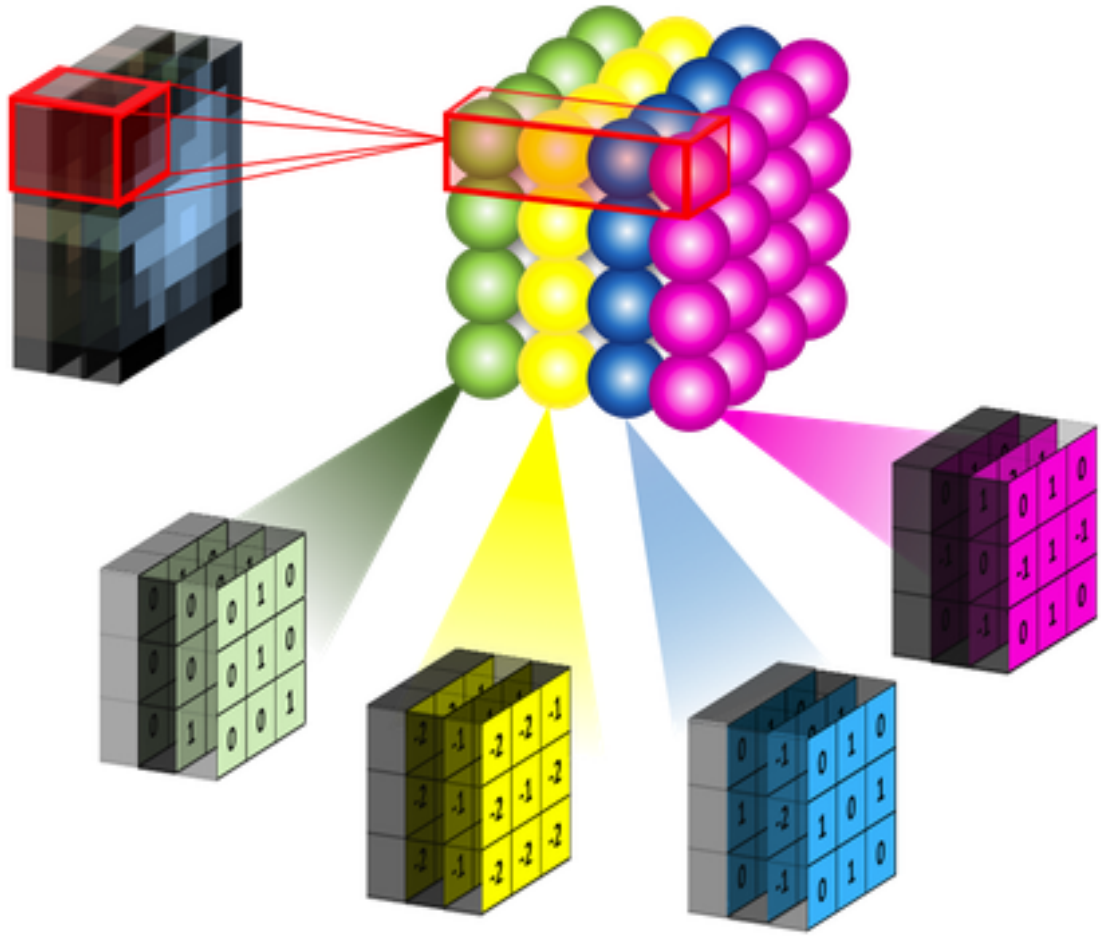
# Convolution Layer for Tensors

**Natural extension for multiple filters**



# Convolution Layer for Tensors

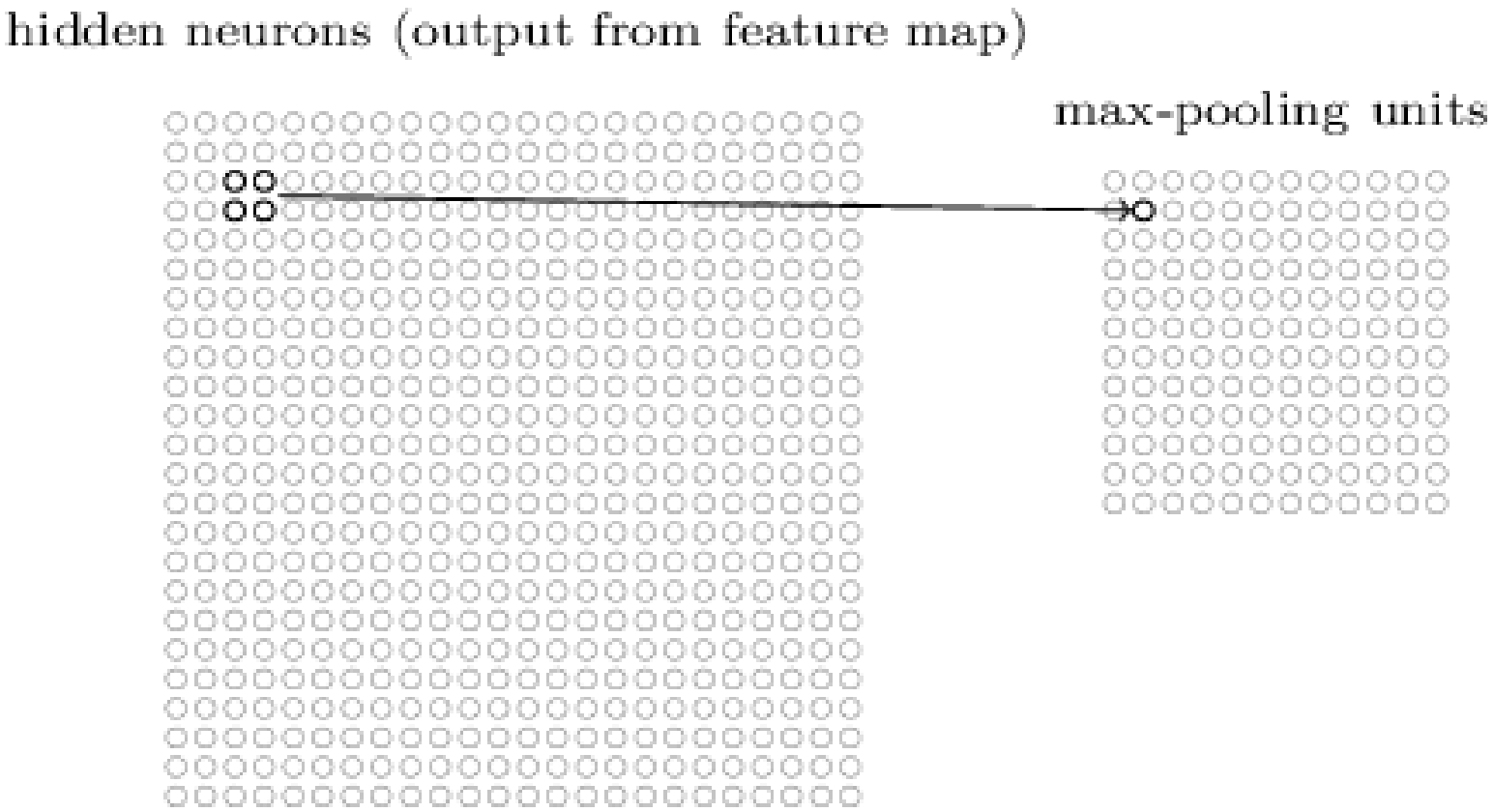
## Ex: input color image



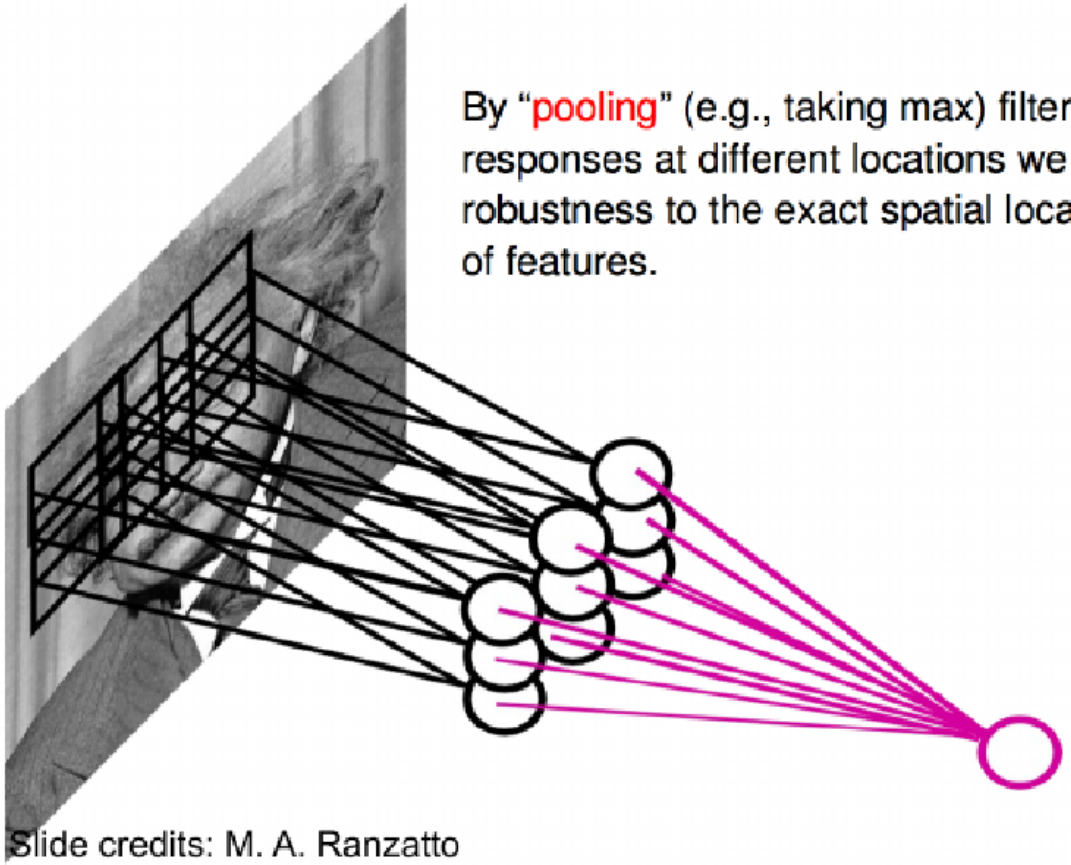
# Pooling Layers

**How to gain (local) shift invariance ?**

- Spatial agregation for each layer
- If stride $s > 1$, sptial resolution decreases (subsampling) $\Rightarrow$ gaining invariace to local translations



# Pooling Layers



# Pooling Layers: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\bar{x}\in N(x),\,\bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x}\in N(x),\,\bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

L2-pooling:

$$h_j^n(x,y) = \sqrt{\sum_{\bar{x}\in N(x),\,\bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x,y) = \sqrt{\sum_{k\in N(j)} h_k^{n-1}(x,y)^2}$$

Slide credits: M. A. Ranzatto

# Max Pooling & Translation Invariance

- Translation invariance wrt vector $\overrightarrow{T} = (t_x, t_y)^t$ if:
  - $\overrightarrow{T} \not\Rightarrow$ new largest element at pooling region edge
  - $\overrightarrow{T} \not\Rightarrow$ remove max from pooling region
- Ex: $5 \times 5$ conv map, $3 \times 3$ max pooling centered at 15: $max = 15$,
- **Invariance OK: $\forall$ translation $(t_x, t_y) \in \pm 1$ px**
  $\Rightarrow max = 15$

$$C = \begin{bmatrix} 11 & -5 & 1 & -2 & 0 \\ 1 & 3 & 0 & 0 & 5 \\ 8 & 4 & 15 & -10 & 4 \\ 8 & 6 & 5 & 3 & 7 \\ 3 & 0 & -2 & 9 & 3 \end{bmatrix}$$

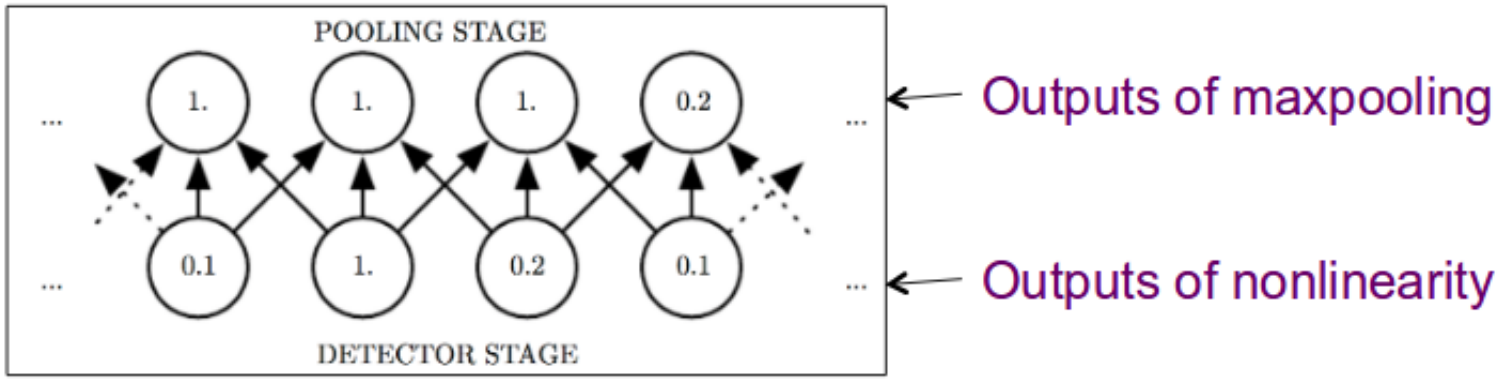
# Max Pooling & Translation Invariance

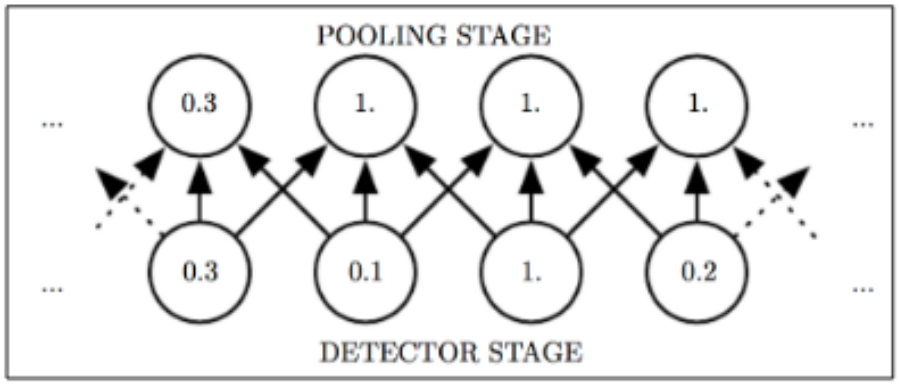
- Translation invariance wrt vector $\overrightarrow{T} = (t_x, t_y)^t$ if:
  - $\overrightarrow{T} \not\Rightarrow$ new largest element at pooling region edge
  - $\overrightarrow{T}$ remove max from pooling region
- Ex: $5 \times 5$ conv map, $3 \times 3$ max pooling centered at 15: $max = 15$,
- **Invariance KO: right translation $t_x = +1$ px**
  $\Rightarrow$ $max = 7$

$$C = \begin{bmatrix} 11 & -5 & 1 & -2 & 0 \\ 1 & 3 & 0 & 0 & 5 \\ 8 & 15 & 4 & -10 & 4 \\ 8 & 6 & 5 & 3 & 7 \\ 3 & 0 & -2 & 9 & 3 \end{bmatrix}$$

# Max Pooling & Translation Invariance

- Max pooling: partial translation invariance (under some conditions)
  - **At least local stability:** every value in bottom changed, only half values in top changed ⇒ Distance after pooling decreases




From [Goodfellow et al., 2016]

# Convolutional Neural Networks (ConvNets)

- An elementary block: Convolution + Non linearity + pooling
- Stack several blocks: Convolutional Neural Networks (ConvNets)

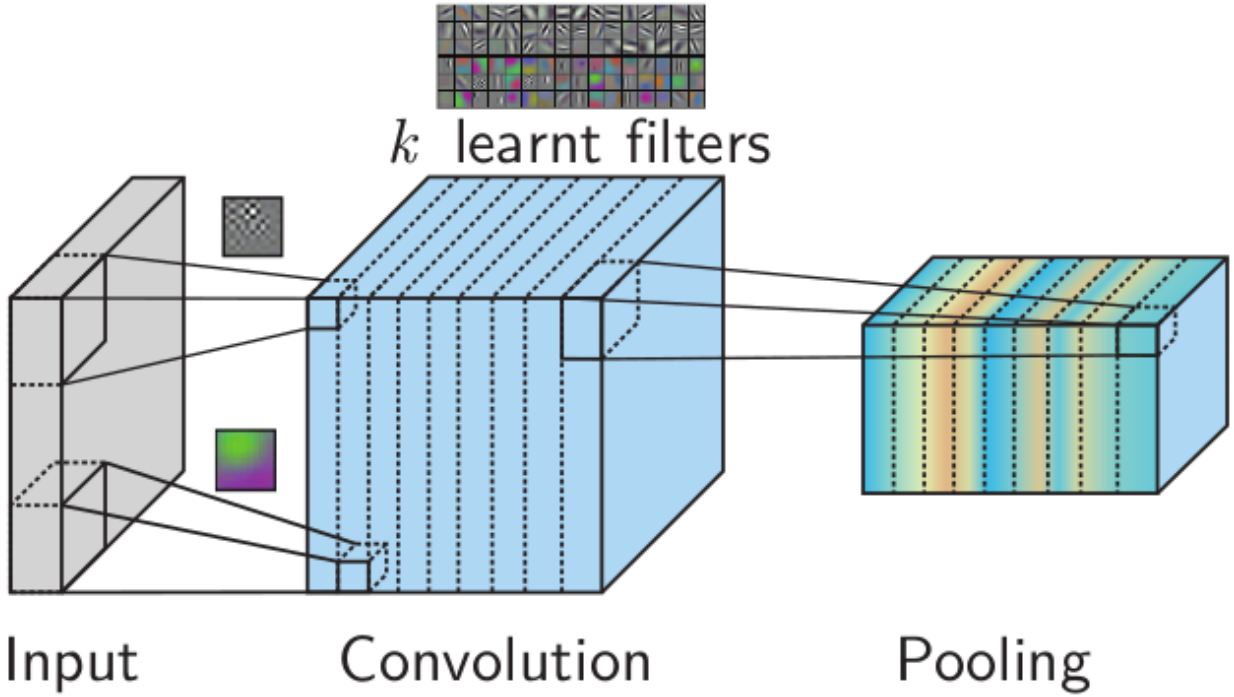
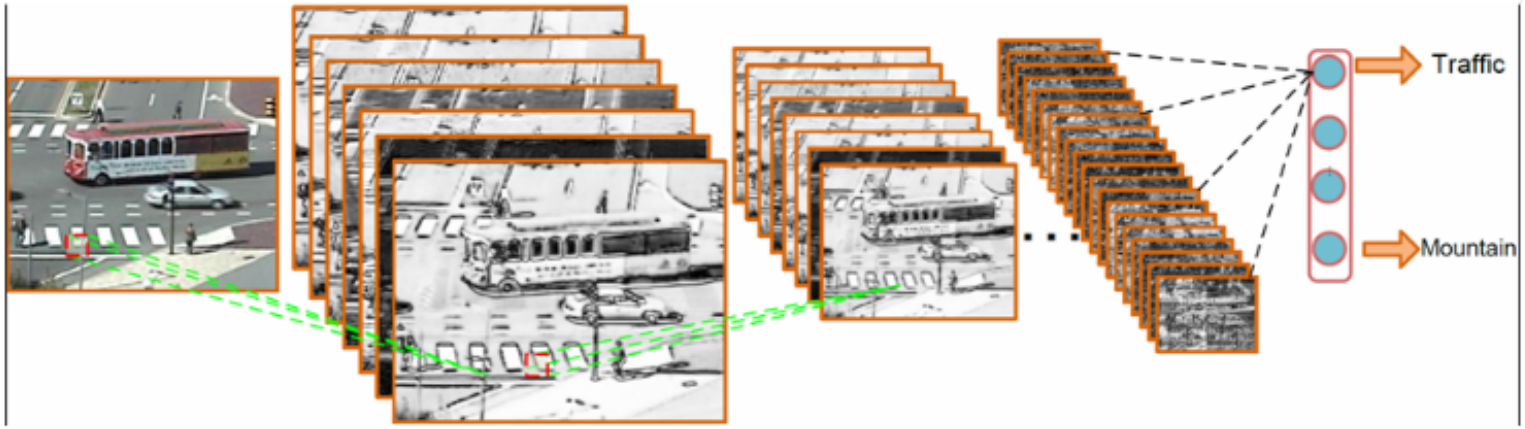



Figure: Important building blocks in CNN

# Convolutional Neural Networks (ConvNets)

- Generally, Feature maps stacked together at one point ⇒ fully connected layers



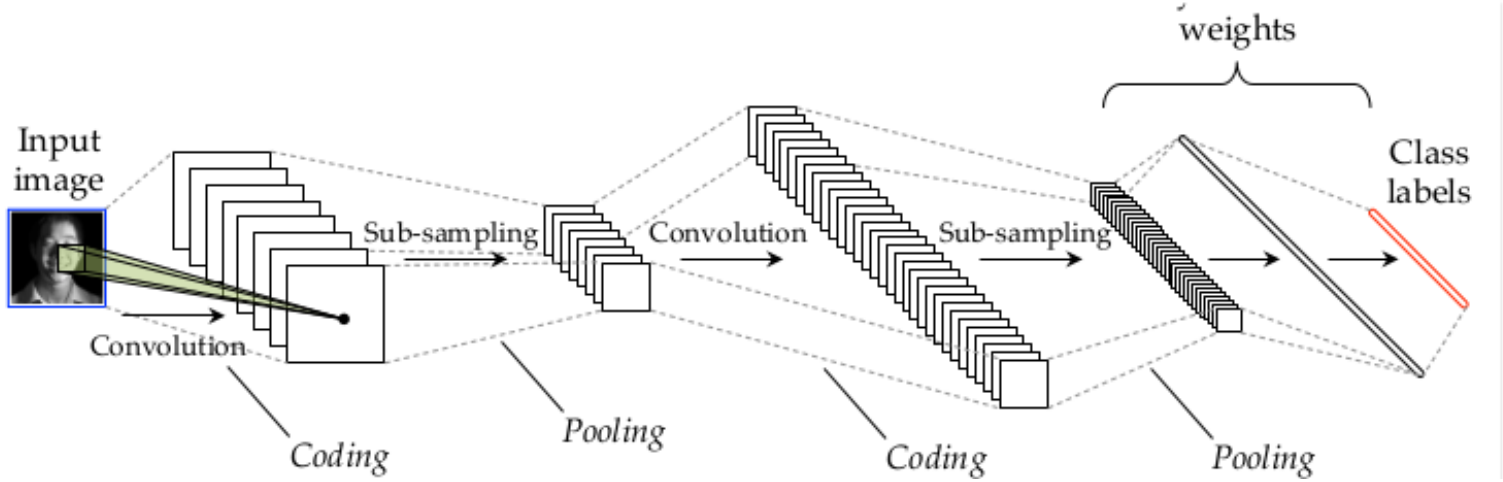
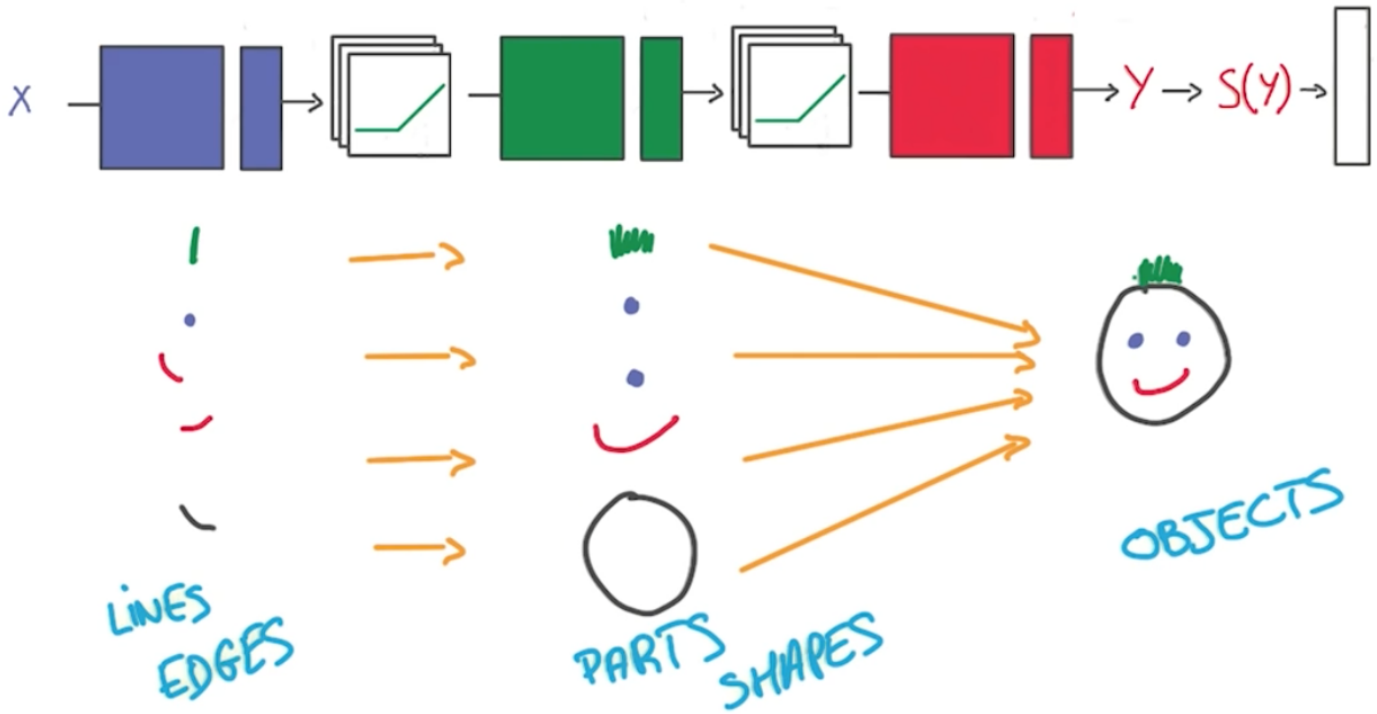




Figure: Important building blocks in CNN

# ConvNets: Conclusion

- Crucial step for tacking advantage of structure ⇒ local processing
- Useful for many data types and applications:
  - Low-level signal, *e.g.* image, audio (speech, music)
  - More semantic data, *e.g.* modern text embedding (`word2vec`) or RNN
- Block [Convolution + Non linearity + pooling ] intuitive for modeling hierarchical information extraction



# Outline

1 Context

2 Neural Nets

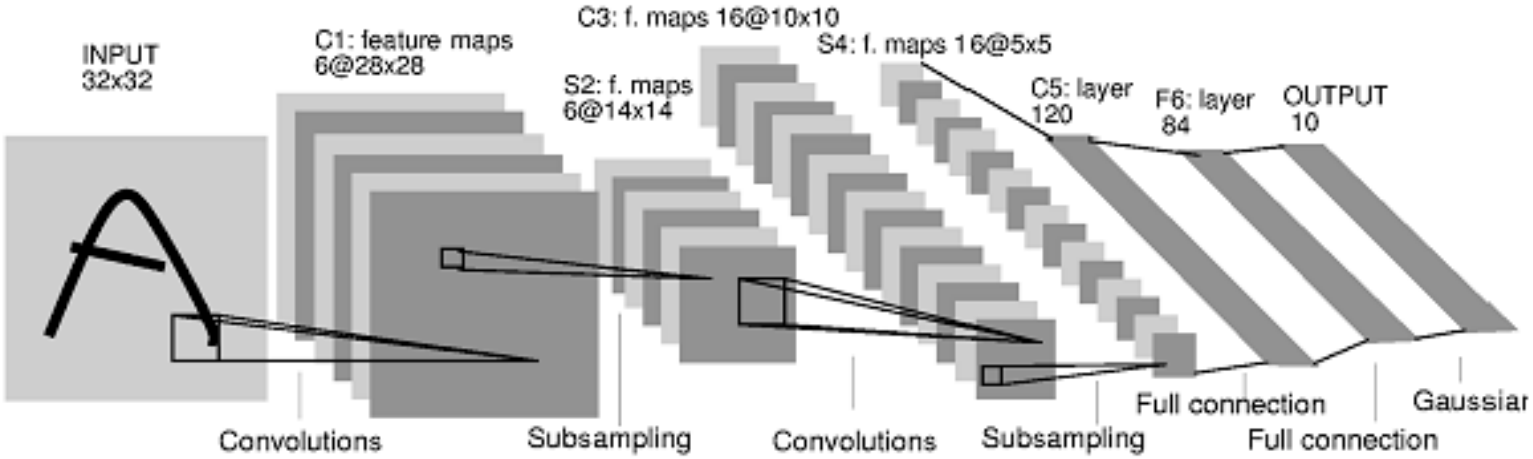
3 Convolutionnal Neural Networks

4 Case Study: LeNet Model

# Deep Learning: Trends and methods in the last four decades

## 80's: 1$^{st}$ Convolutionnal Neural Networks

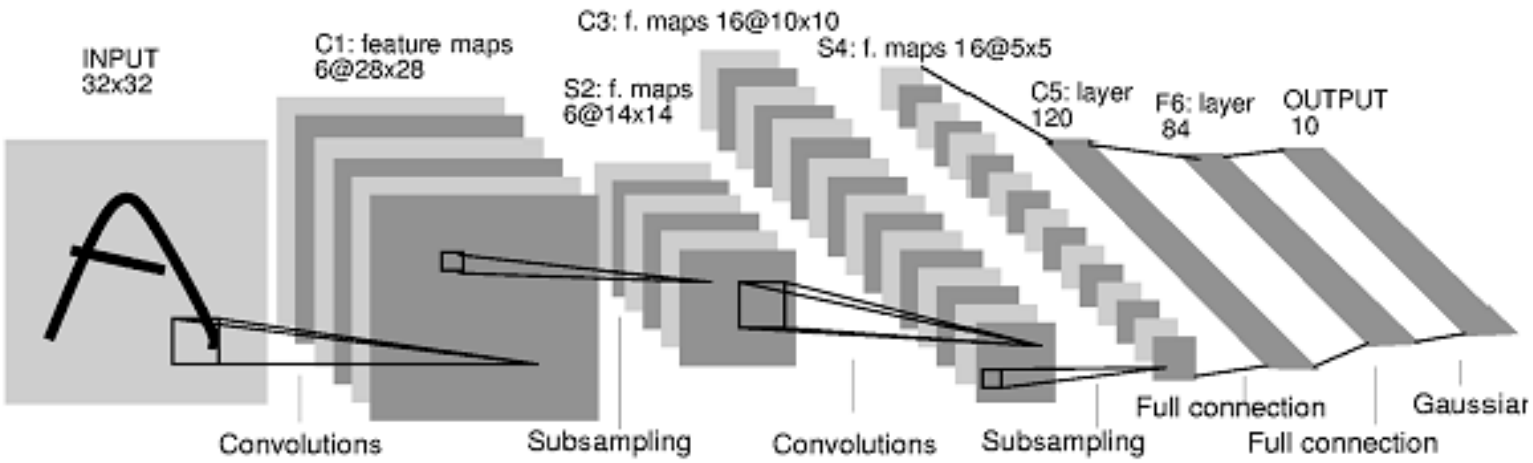
- LeNet 5 Model [LeCun et al., 1989], trained using back-prop




- Input: 32x32 pixel image. Largest character is 20x20
- 2 successive blocks [Convolution + Sigmoid + Pooling (+sigmoid)]
  Cx: Convolutional layer, Sx: Subsampling layer
- C5: convolution layer ~ fully connected
- 2 Fully connected layers Fx

# 80's: LeNet 5 Model

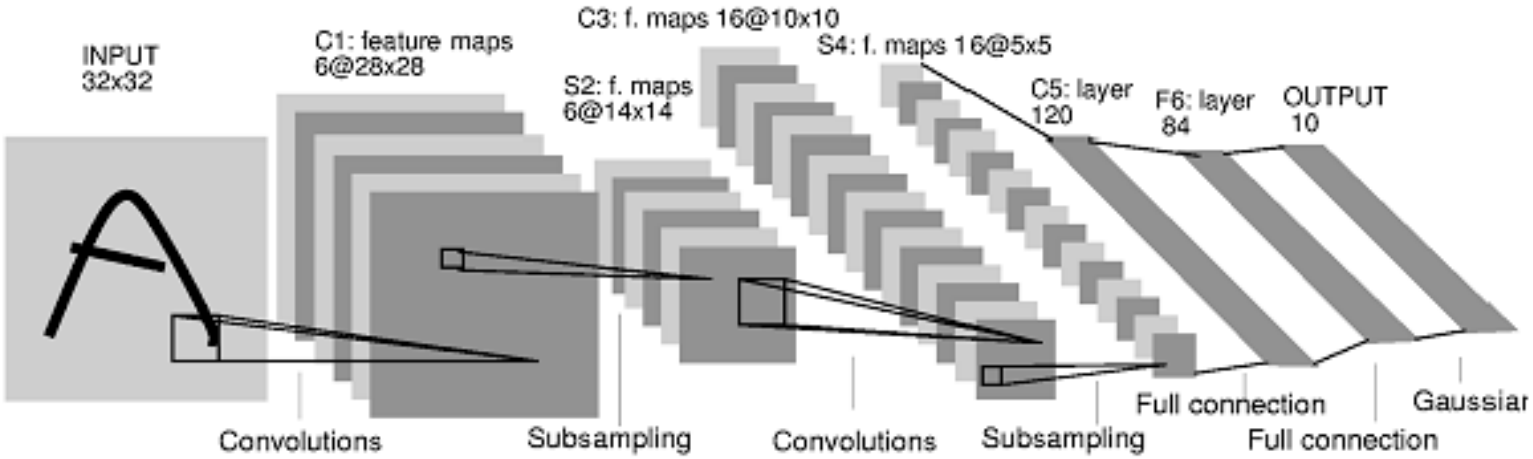
## C1 Layer




- Convolutional layer with 6 5x5 filters $\Rightarrow$ 6 feature maps of size 28x28 (no padding).
- # Parameters: $5^2$ per filer + bias $\Rightarrow (5 * 5 + 1) * 6 = 156$
  - If it was fully connected: (32*32+1)*(28*28)*6 parameters $5 \sim 10^6$ !

# 80's: LeNet 5 Model

## S2 Layer

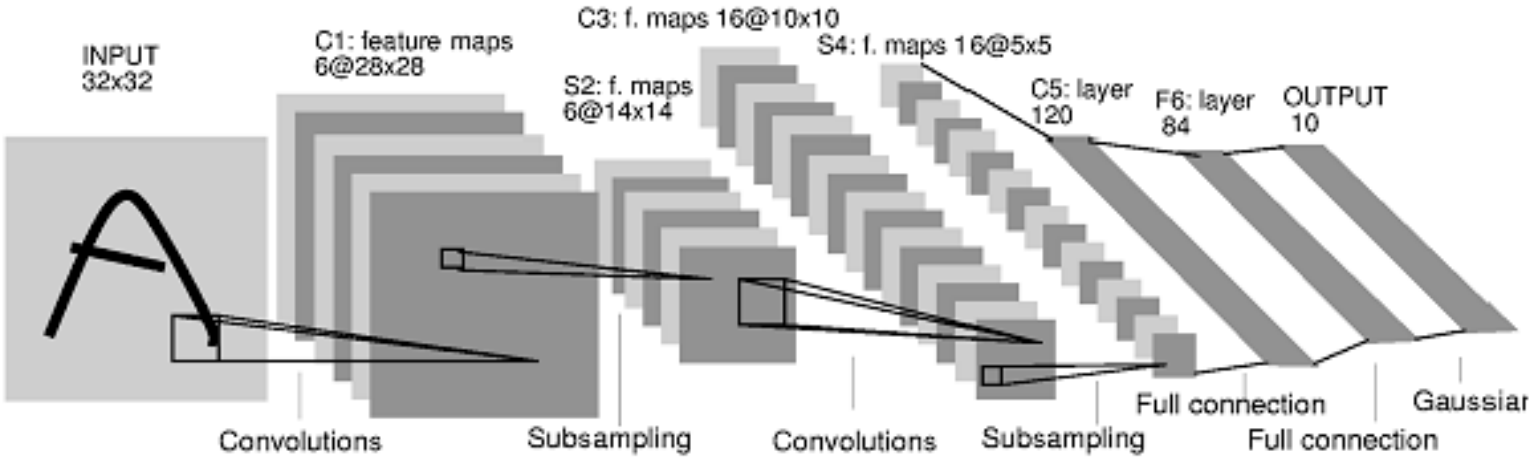



- Subsampling layer = pooling layer
- Pooling area : 2x2 in C1
- Pooling stride: 2 $\Rightarrow$ 6 features maps of size 14x14
- Pooling type : sum, multiplied by a trainable param + bias $\Rightarrow$ 2 parameters per channel
- Total # Parameters: $2 * 6 = 12$

# 80's: LeNet 5 Model

## C3 Layer: Convolutional

- C3: 16 filters ⇒ 16 feature maps of size 10x10 (no padding)



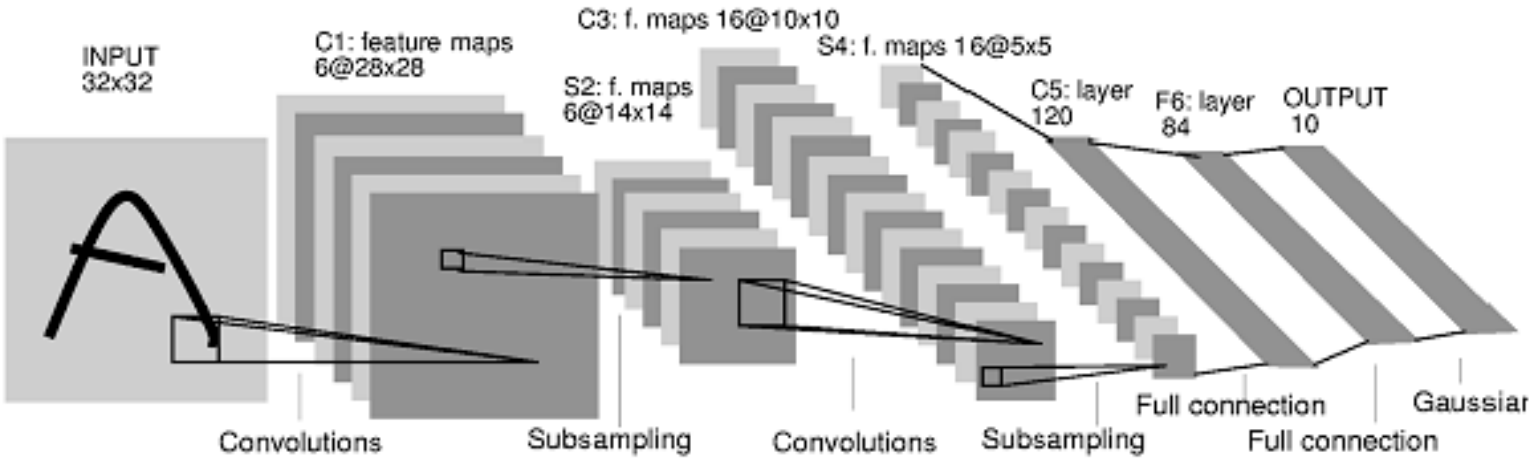

- 5x5 filters connected to a subset of S2 maps
  ⇒ 0-5 connected to 3, 6-14 to 4, 15 connected to 6

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | X | | | | X | X | X | | | X | X | X | X | | X | X |
| 1 | X | X | | | | X | X | X | | | X | X | X | X | | X |
| 2 | X | X | X | | | | X | X | X | | | X | | X | X | X |
| 3 | | X | X | X | | | X | X | X | X | | | X | | X | X |
| 4 | | | X | X | X | | | X | X | X | X | | X | X | | X |
| 5 | | | | X | X | X | | | X | X | X | X | | X | X | X |

  - # Parameters: 1516
    $(5 * 5 * 3 + 1) * 6 + (5 * 5 * 4 + 1) * 9 + (5 * 5 * 6 + 1) = 456 + 909 + 151$

# 80's: LeNet 5 Model

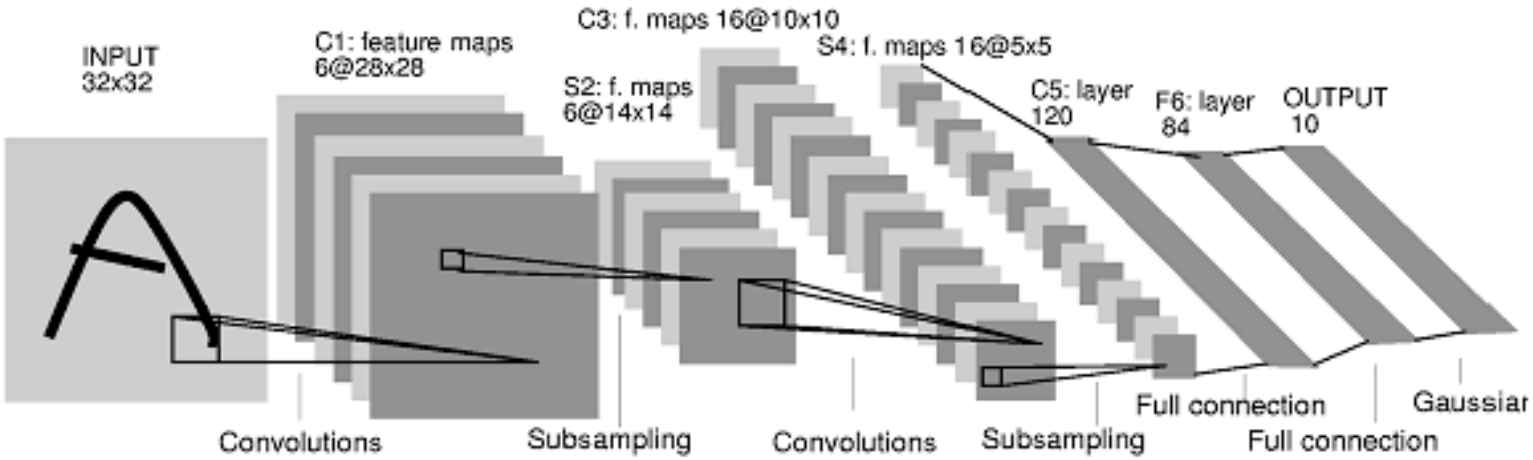
## S4 Layer




- Subsampling layer = pooling layer
- Pooling area : 2x2 in C3
- Pooling stride: 2 $\Rightarrow$ 16 features maps of size 5x5
- Pooling type : sum, multiplied by a trainable param + bias $\Rightarrow$ 2 parameters per channel
- Total # Parameters: $2 * 16 = 32$

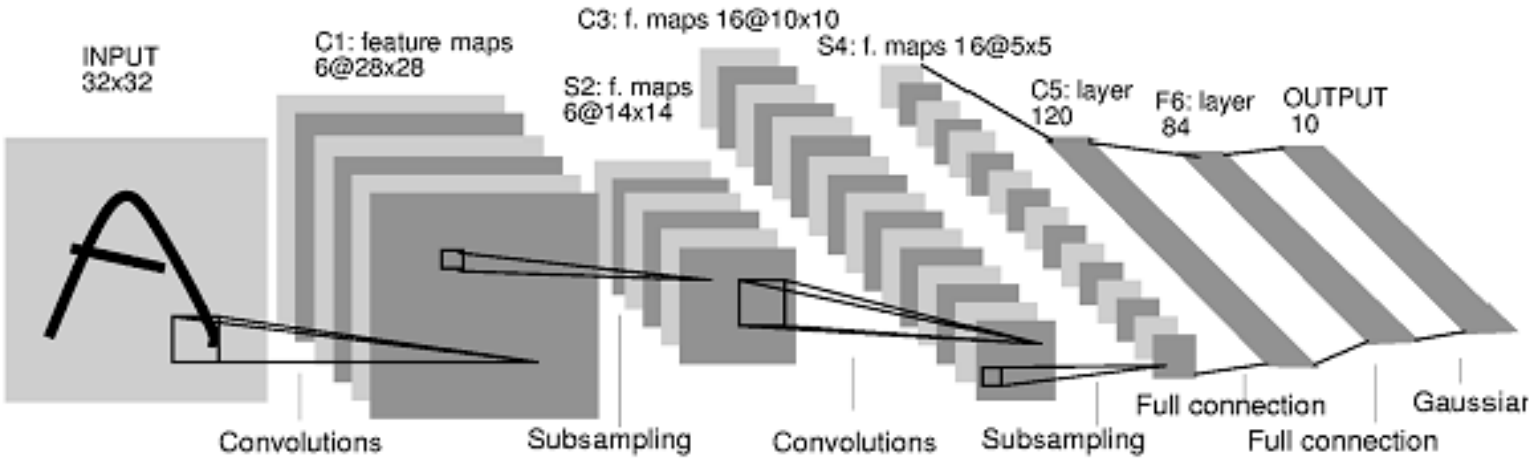
# 80's: LeNet 5 Model

## C5 Layer: Convolutionnal layer




- 120 5x5x16 filters $\Rightarrow$ whole depth of $S4$ ($\neq C3$)
- Each maps in $S4$ is 5x5 $\Rightarrow$ single value for each $C5$ maps
- $C5$ 120 features map of size 1x1 (vector of size 120)
  $\Rightarrow$ spatial information lost, $\sim$ to a fully connected layer
- Total # Parameters: $(5 * 5 * 16 + 1) * 120 = 48210$

# 80's: LeNet 5 Model




## F6 Layer: Fully Connected layer

- 84 fully connected units.
- # Parameters: 84*(120+1)=10164

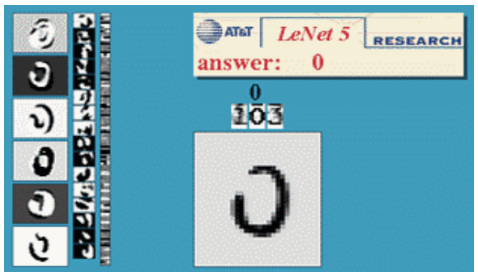
## F7 Layer (output): Fully Connected layer

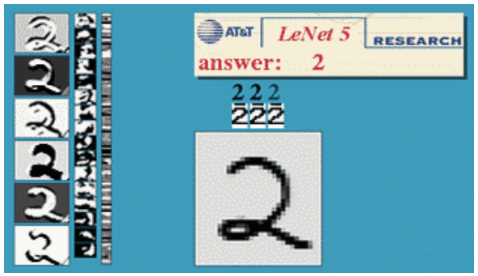
- 10 (# classes) fully connected units.
- # Parameters: 10*(84+1)=850

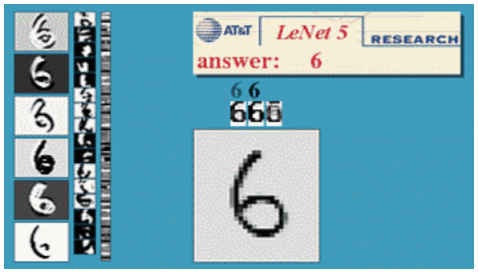
# 80's: LeNet 5 Model

- Evaluation on MNIST
- Total # parameters ~ 60000
  - 60,000 original datasets: test error: 0.95%
  - 540,000 artificial distortions + 60,000 original: Test error: 0.8%

- **Successful deployment for postal code reading in the US**



# References I


[Cybenko, 1989] Cybenko, G. (1989).
Approximation by superpositions of a sigmoidal function.
*Mathematics of control, signals and systems*, 2(4):303–314.

[Fukushima, 1980] Fukushima, K. (1980).
Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.
*Biological cybernetics*, 36(4):193–202.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016).
*Deep Learning*.
MIT Press.
http://www.deeplearningbook.org.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989).
Backpropagation applied to handwritten zip code recognition.
*Neural computation*, 1(4):541–551.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943).
A logical calculus of the ideas immanent in nervous activity.
*The bulletin of mathematical biophysics*, 5(4):115–133.

[Rosenblatt, 1958] Rosenblatt, F. (1958).
The perceptron: A probabilistic model for information storage and organization in the brain.
*Psychological Review*, 65(6):386–408.

[Werbos, 1974] Werbos, P. (1974).
*Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*.
PhD thesis, Harvard University.