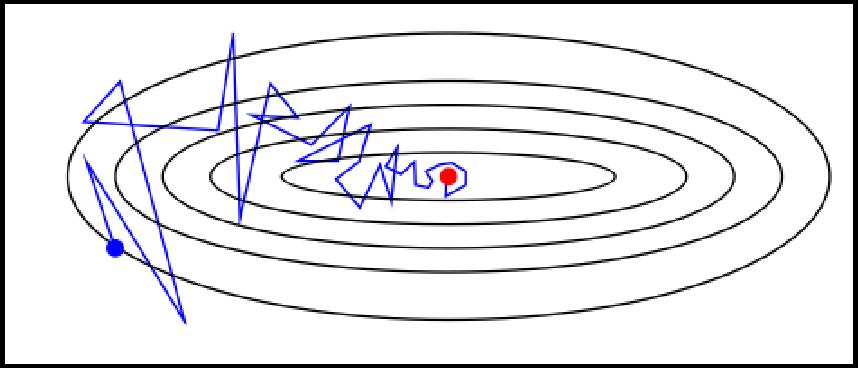
Introduction to Deep Learning

Back Propagation, Neural Networks



Dimitris Samaras

Most uncredited slides by I. Kokkinos

Last time: Image Classification

assume given set of discrete labels
{dog, cat, truck, plane, ...}

→ cat

Slide credit:Fei-Fei Li

k-Nearest Neighbor

Slide credit:Fei-Fei Li

Linear Classification

1. define a **score function**

class scores

Slide credit:Fei-Fei Li

Linear Classification

1. define a **score function**

data (image)

$$f(x_i, W, b) = Wx_i + b$$

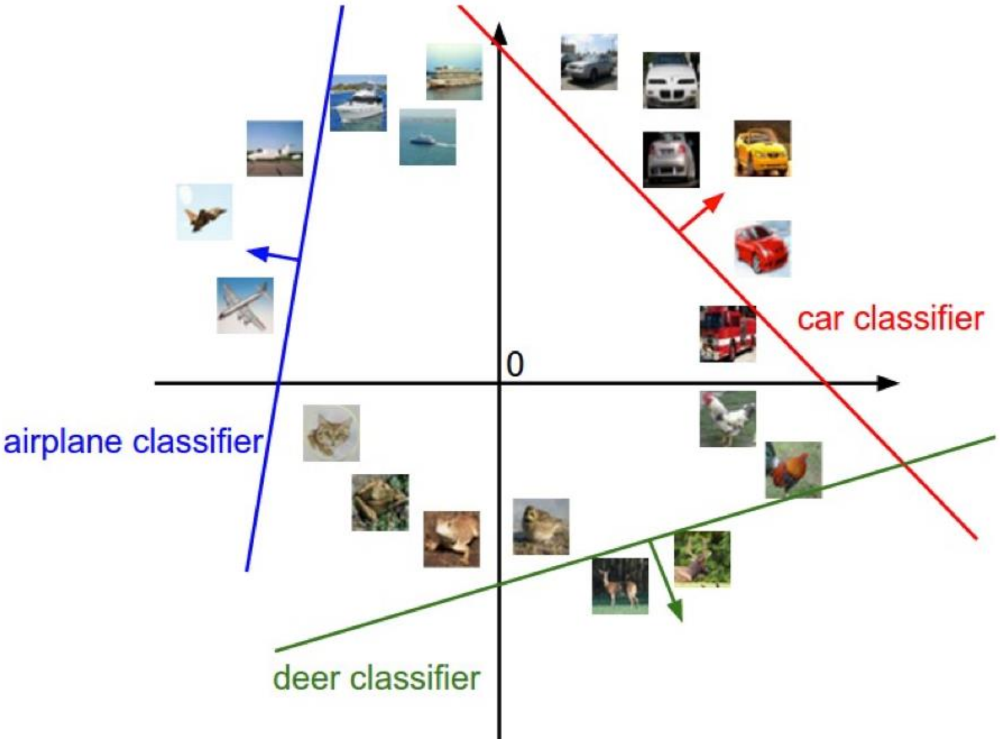
class scores

"weights"

"bias vector"

"parameters"

Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

Slide credit:Fei-Fei Li

Loss

2. Define a **loss function** (or cost function, or objective)

- scores, label ⟶ *loss*.

$f(x_i, W)$ y_i L_i

Example:

$$f(x_i, W) = [\boxed{13}, -7, 11]$$

$$y_i = 0$$

Question: if you were to assign a single number to how “unhappy” you are with these scores, what would you do?

Slide credit:Fei-Fei Li

Multiclass SVM Loss

2. Define a **loss function** (or cost function, or objective)
One (of many ways) to do it: **Multiclass SVM Loss**

$$\boxed{L_i} = \boxed{\sum_{j \neq y_i}} \max(0, \boxed{f(x_i, W)_j - f(x_i, W)_{y_i}} + \Delta)$$

loss due to example i

sum over all incorrect labels

difference between the correct class score and incorrect class score

Slide credit:Fei-Fei Li

$$\boxed{L_i} = \boxed{\sum_{j \neq y_i}} \max(0, \boxed{f(x_i, W)_j - f(x_i, W)_{y_i}} + \Delta)$$

loss due to example i

sum over all incorrect labels

difference between the correct class score and incorrect class score

Slide credit:Fei-Fei Li

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

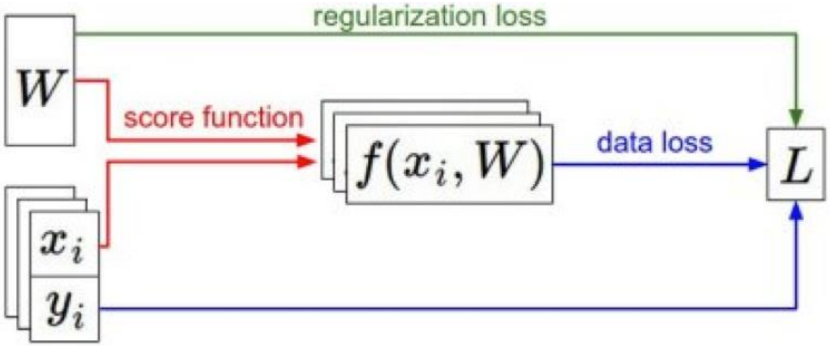
$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \boxed{\lambda R(W)}$$

Regularization strength

Slide credit:Fei-Fei Li

Putting it all together:

Linear Classification



SVM:

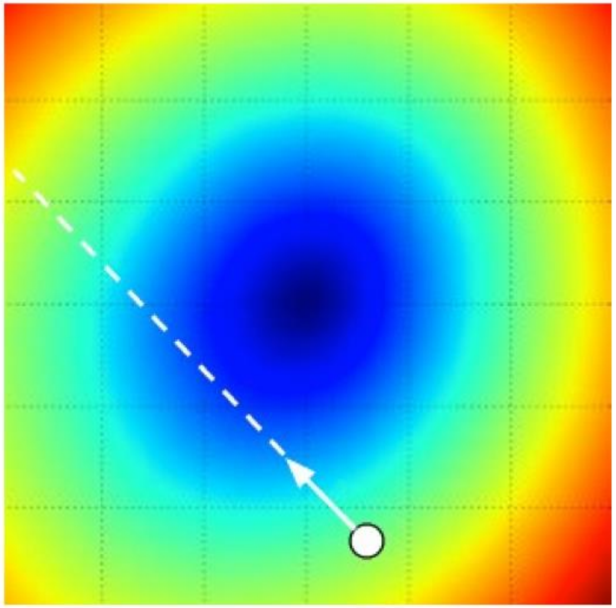
$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda \sum_k \sum_l W_{k,l}^2$$

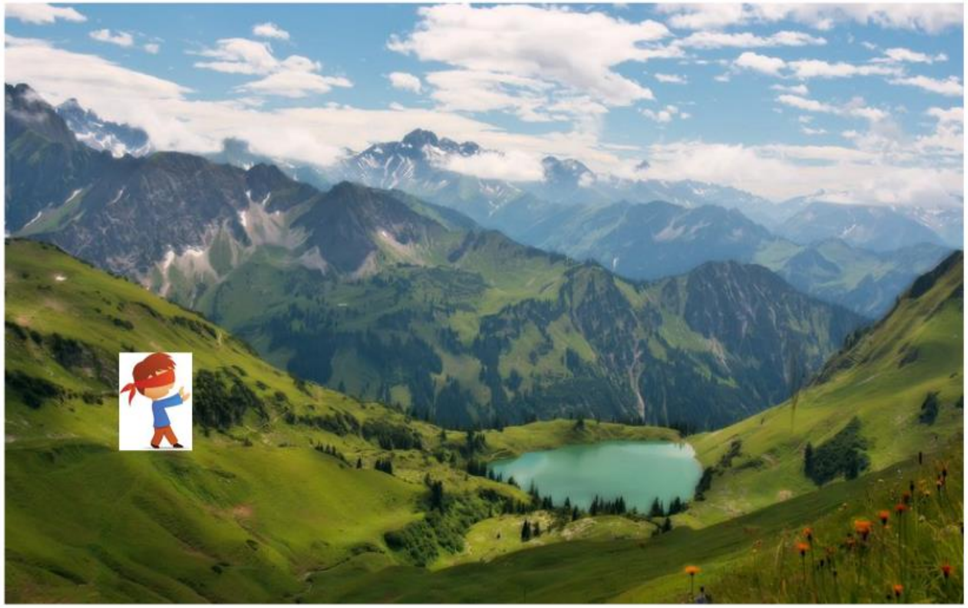
Softmax:

$$L = \frac{1}{N} \sum_i -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) + \lambda \sum_k \sum_l W_{k,l}^2$$

Slide credit:Fei-Fei Li

Optimization Landscape





Slide credit:Fei-Fei Li

Gradient Descent

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

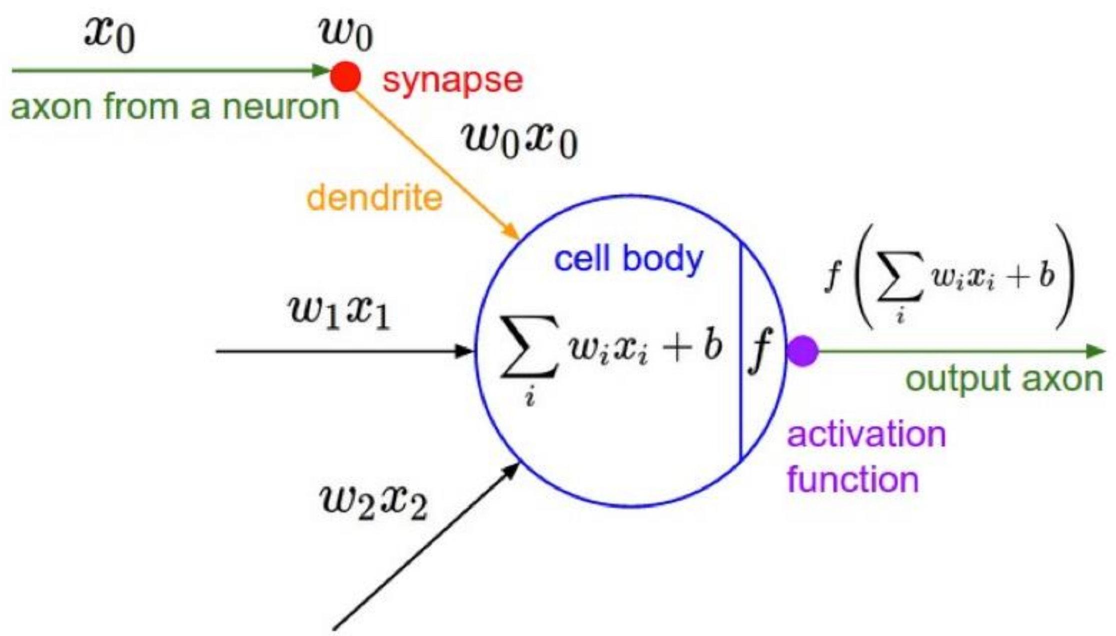
Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient

Slide credit:Fei-Fei Li

A Single Neuron

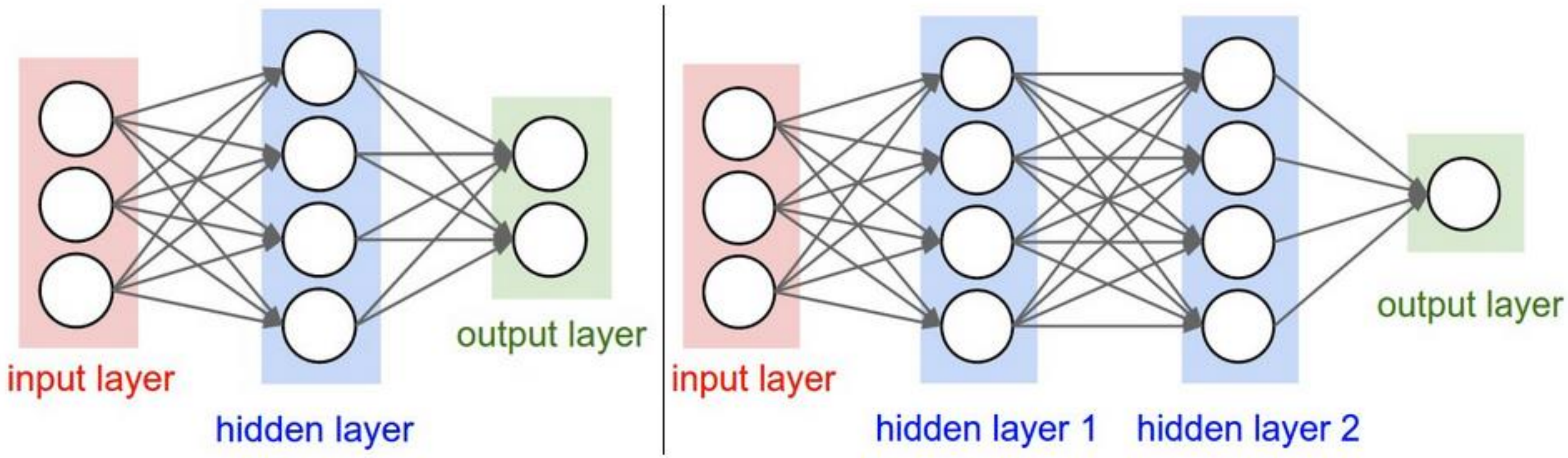
Activation Functions



Slide credit:Fei-Fei Li

Slide credit:Fei-Fei Li

Neural Network Structure



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Activation functions

Step

$$g(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

Sigmoidal

$$g(a) = \frac{1}{1 + \exp(-a)}$$

Hyperbolic tangent

$$g(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

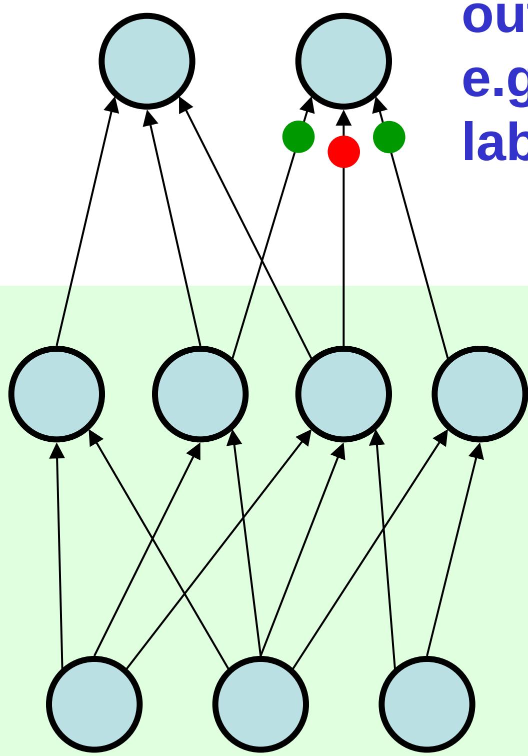
Rectified Linear Unit (RELU)

$$g(a) = \max(0, a)$$

Slide credit:Fei-Fei Li

Perceptron, ‘60s

Step function, single layer

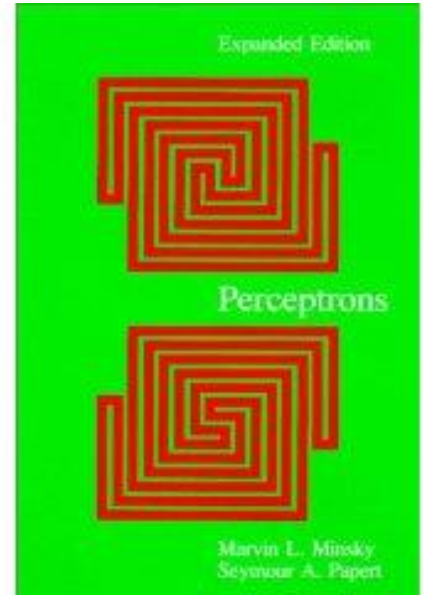


output units
e.g. class
labels

non-adaptive
hand-coded
features

**Fixed
mapping**

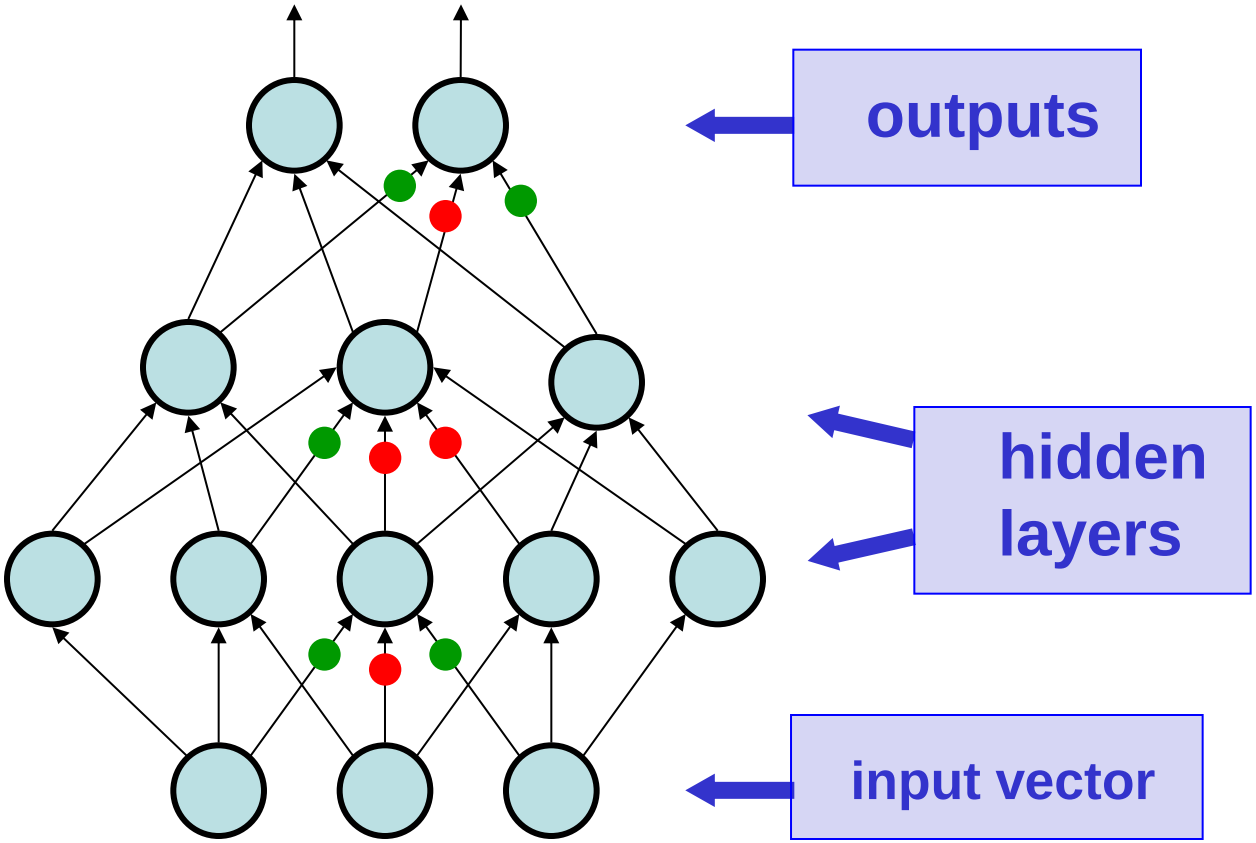
input units
e.g. pixels



Slide credits: G. Hinton

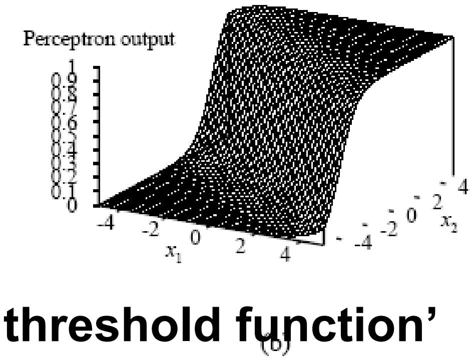
Multi-Layer Perceptrons (~1985)

$$u_i = g\left(\sum_{k\in\mathcal{N}(i)} w_{k,i}\, g\left(\sum_{m\in\mathcal{N}(k)} w_{m,k} u_m + b_k\right) + b_i\right)$$



Slide credits: G. Hinton

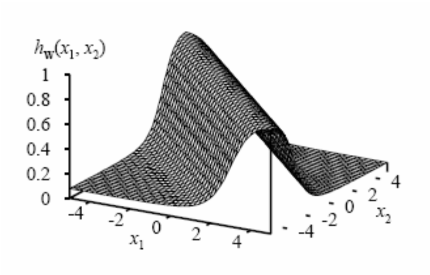
Expressiveness of perceptrons



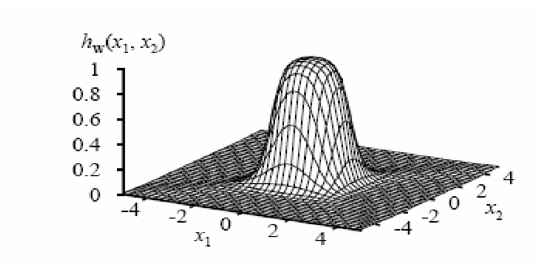
Single layer perceptron:
Linear classifier

`soft threshold function’

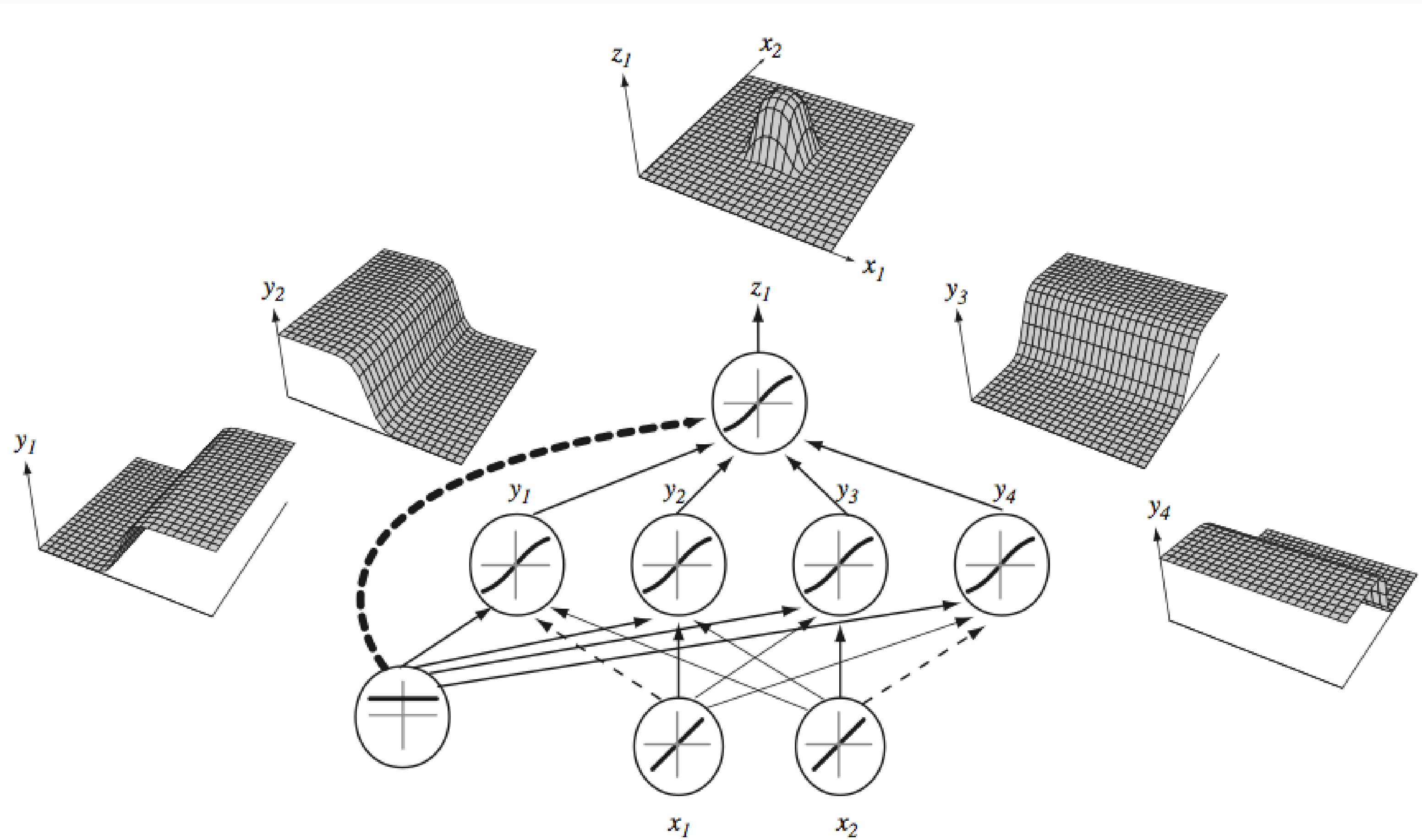
Two opposite `soft threshold’ functions: a ridge



Two ridges: a bump



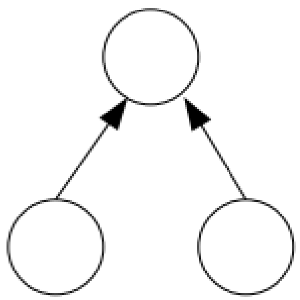
A network for a single bump

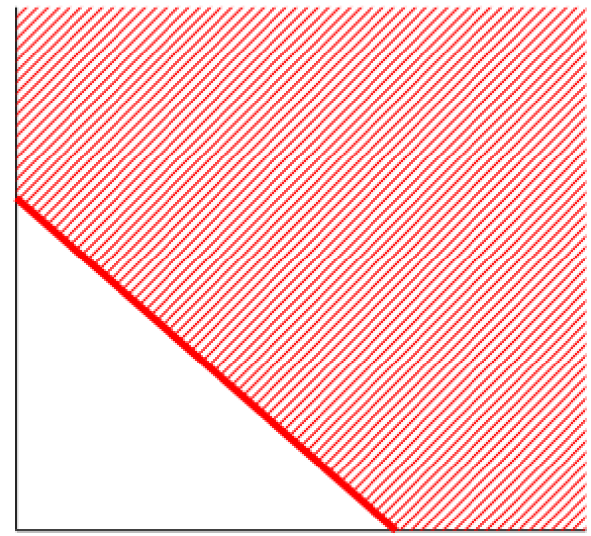


Any function: sum of bumps

From flat to deep

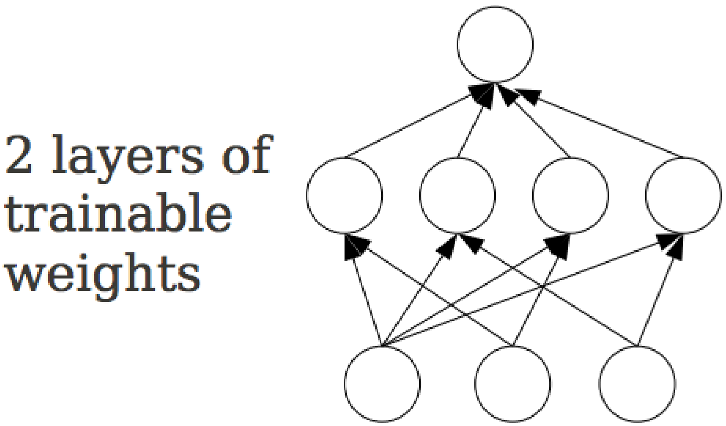
1 layer of trainable weights

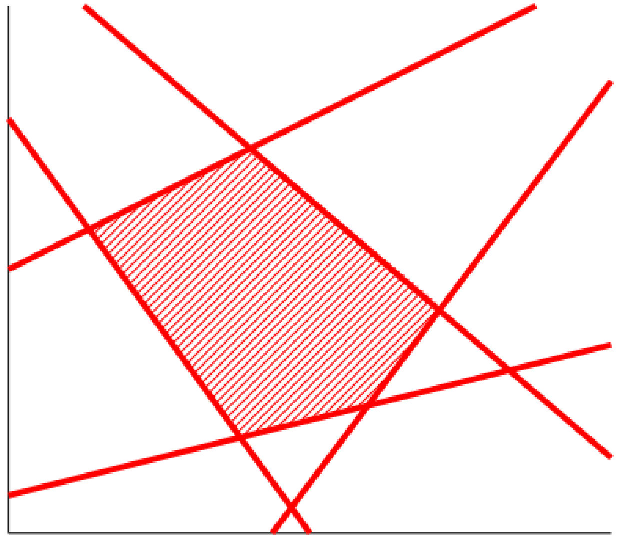




separating hyperplane

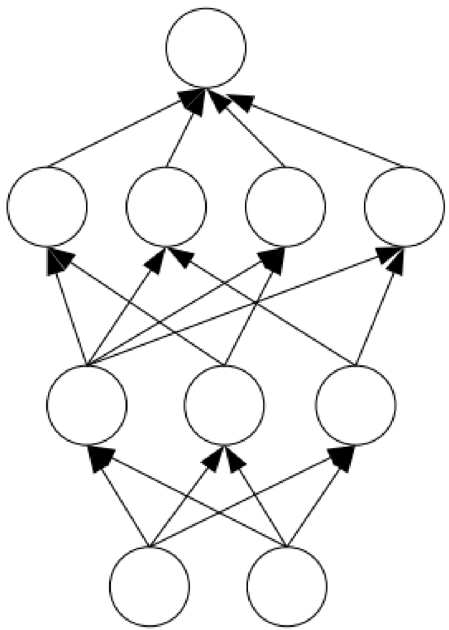
From flat to deep

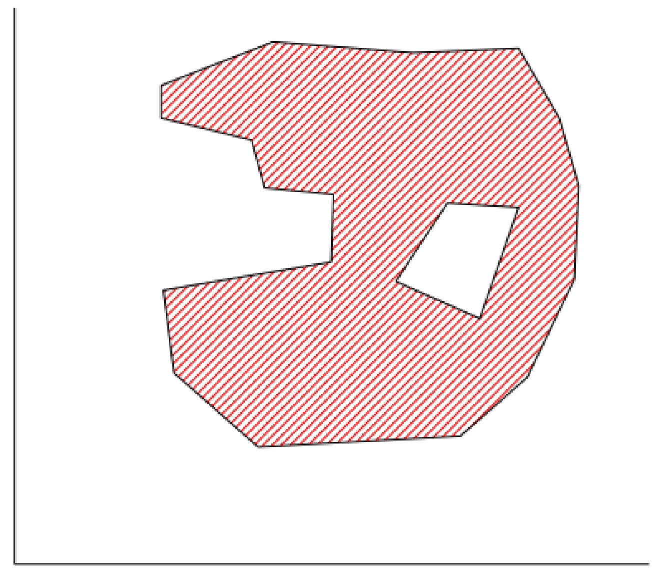




From flat to deep

3 layers of trainable weights

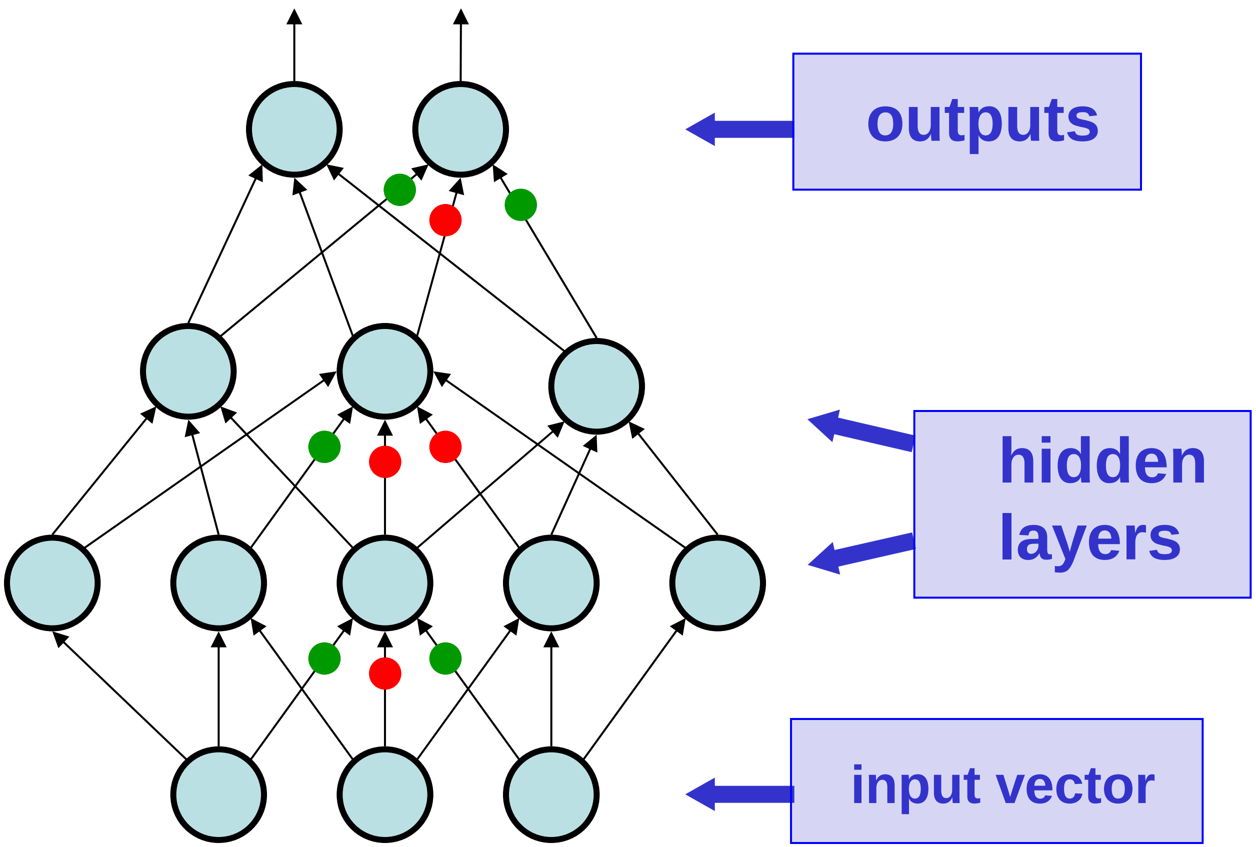




composition of polygons: convex regions

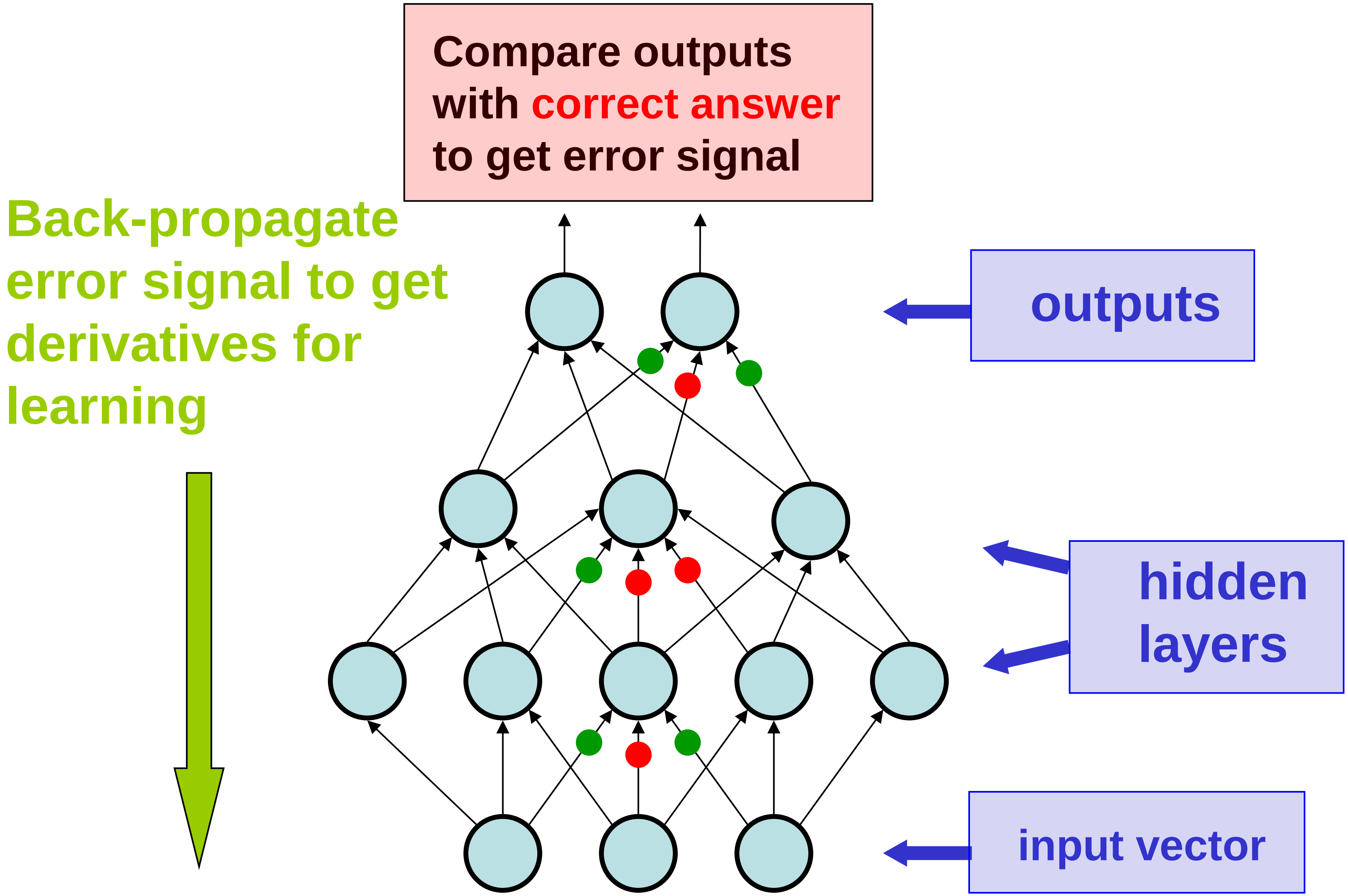
Multi-Layer Perceptrons (~1985)

$$u_i = g\left(\sum_{k\in\mathcal{N}(i)} w_{k,i}\, g\left(\sum_{m\in\mathcal{N}(k)} w_{m,k} u_m + b_k\right) + b_i\right)$$



Slide credits: G. Hinton

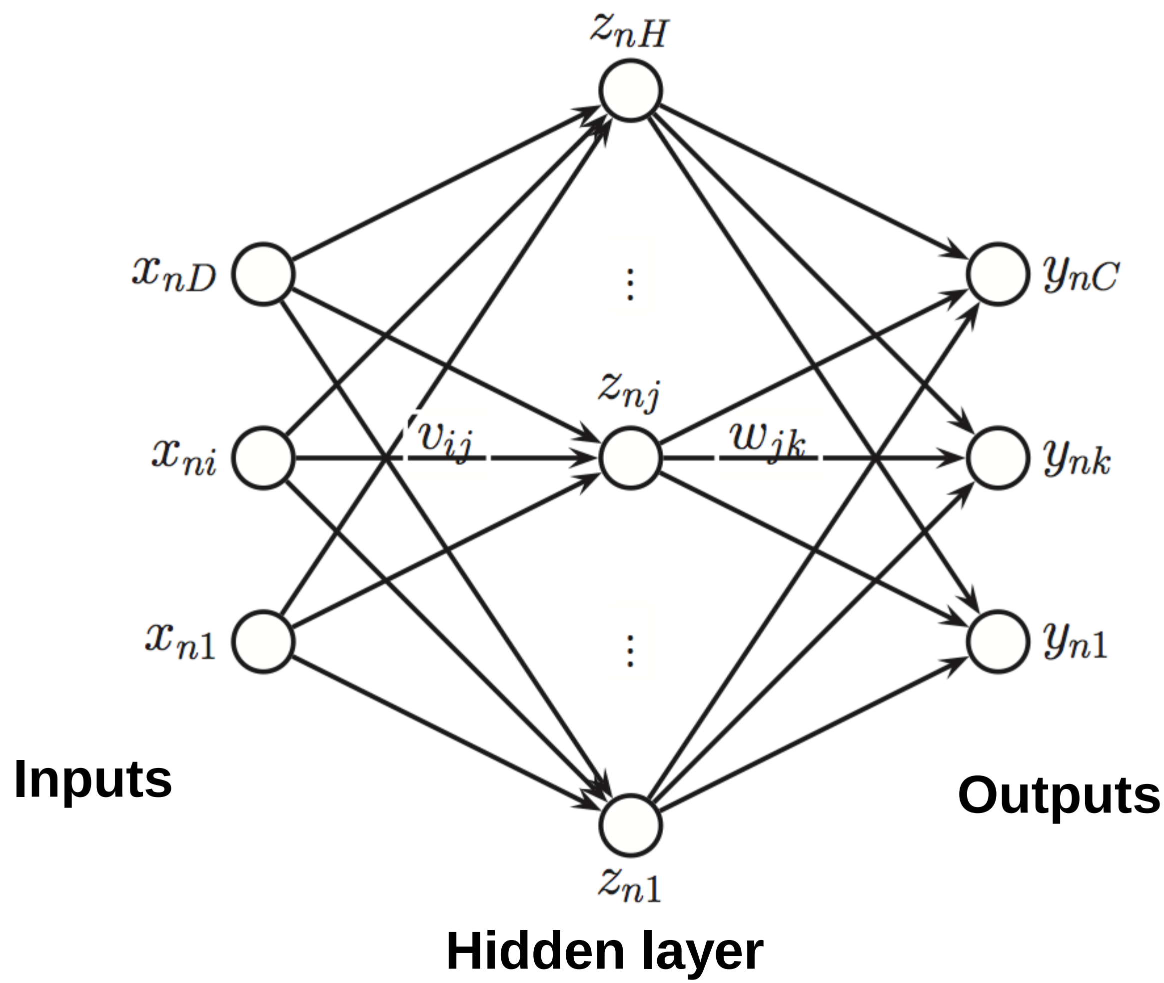
Training Multi-Layer Perceptrons



Slide credits: G. Hinton

A neural network for multi-way classification

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

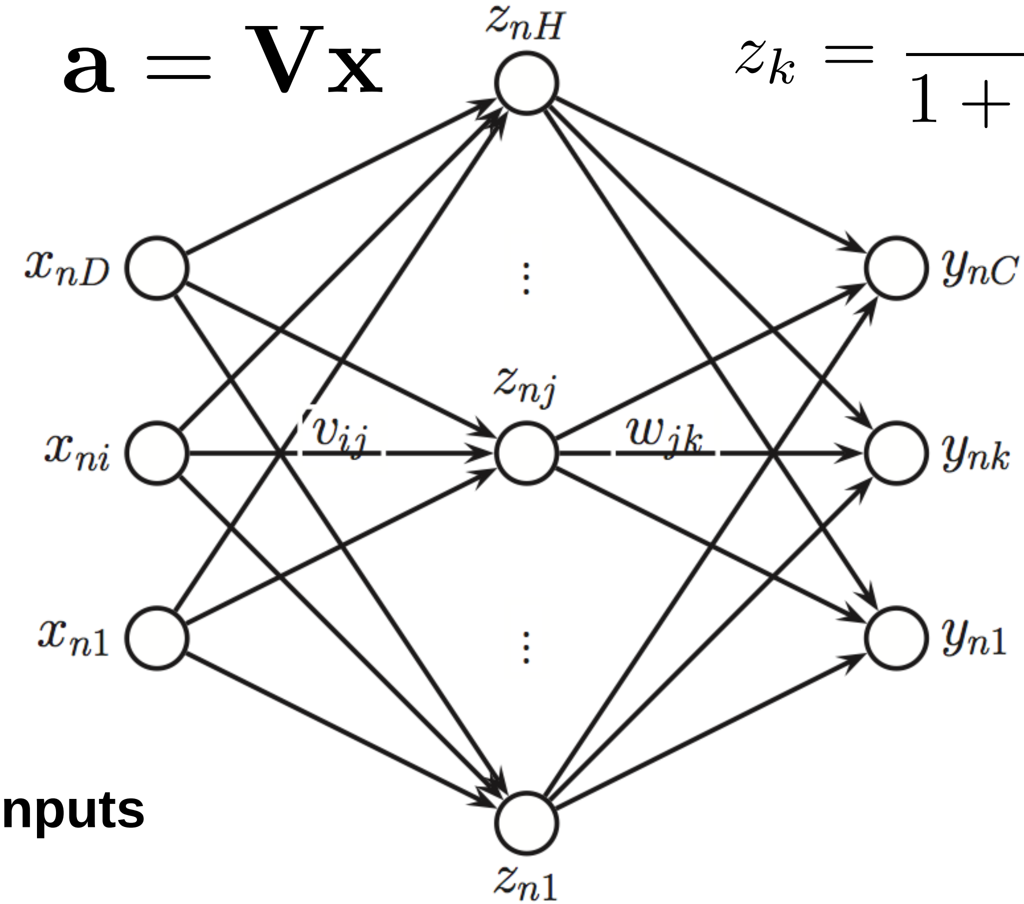


A neural network:

$$\mathbf{z} = g(\mathbf{a})$$

$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

$$z_k = \frac{1}{1 + \exp(-a_k)}$$

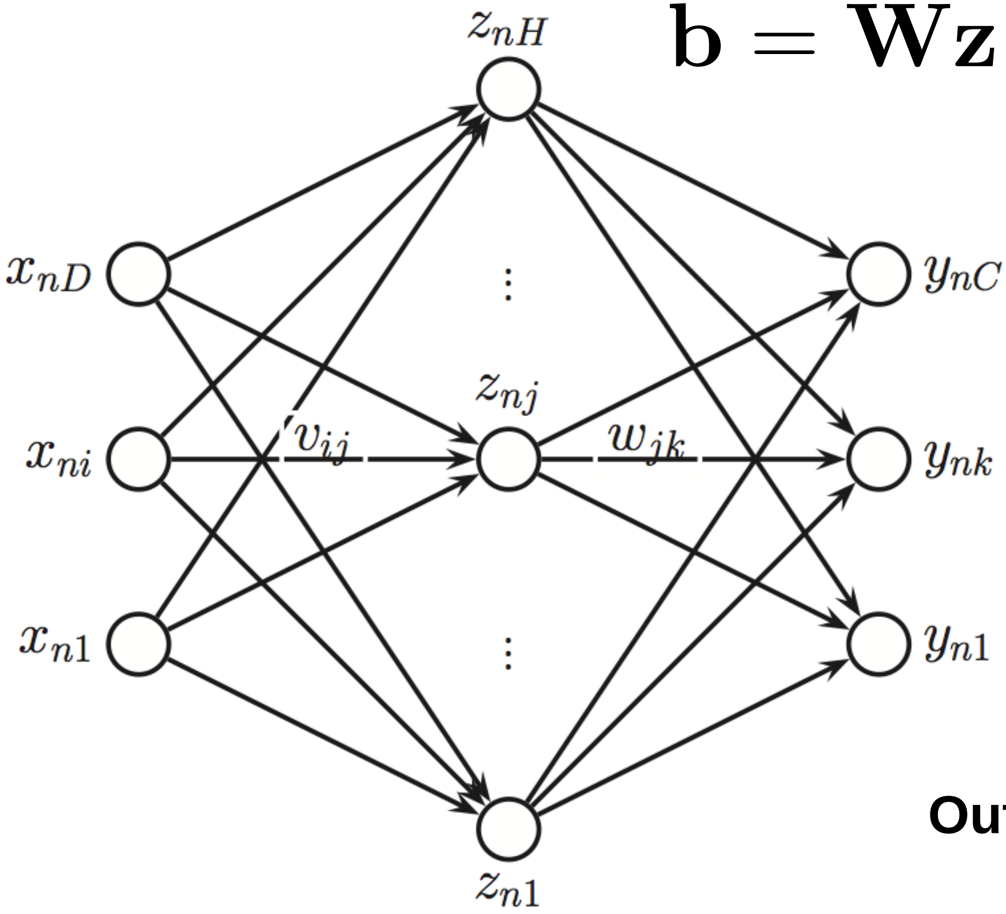


Inputs

Hidden layer

A neural network:

$$\mathbf{b} = \mathbf{W}\mathbf{z}$$



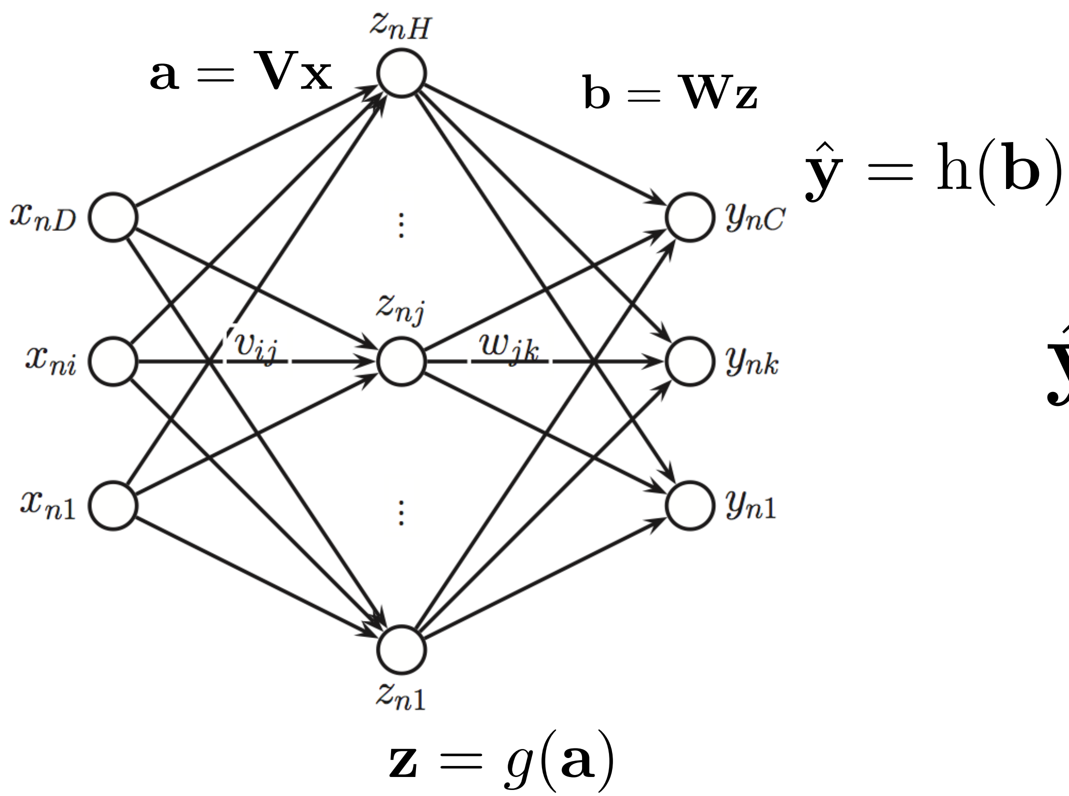
$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{c=1}^{C} \exp(b_c)}$$

Outputs

Hidden layer

Training a neural network

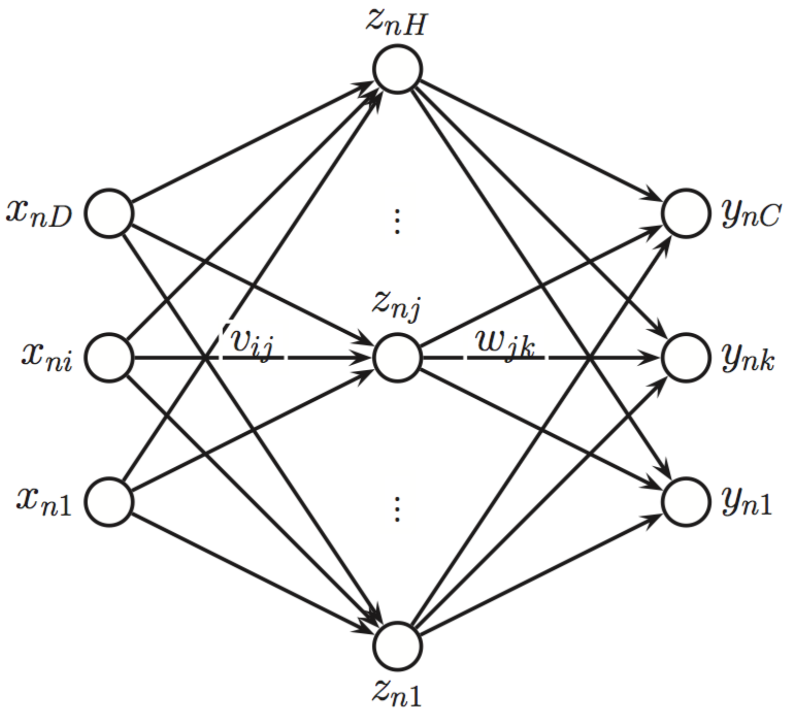


$$\hat{\mathbf{y}}(\theta) = f(\mathbf{x}; \theta)$$

$$l(\theta) = l(\mathbf{y}, \hat{\mathbf{y}}(\theta))$$

$$\theta' = \theta - c\nabla_{\theta} l(\theta)$$

Training a neural network

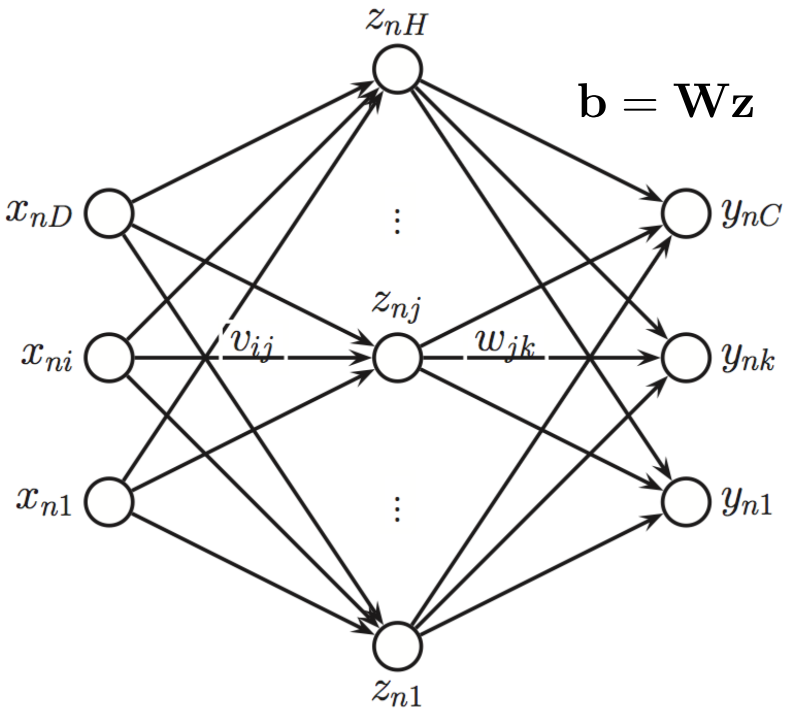


$$\hat{\mathbf{y}}(\theta) = f(\mathbf{x};\theta)$$

$$l(\theta) = l(\mathbf{y}, \hat{\mathbf{y}}(\theta))$$

$$\theta' = \theta - c\nabla_\theta l(\theta)$$

Training a neural network



$$\mathbf{b} = \mathbf{W}\mathbf{z}$$

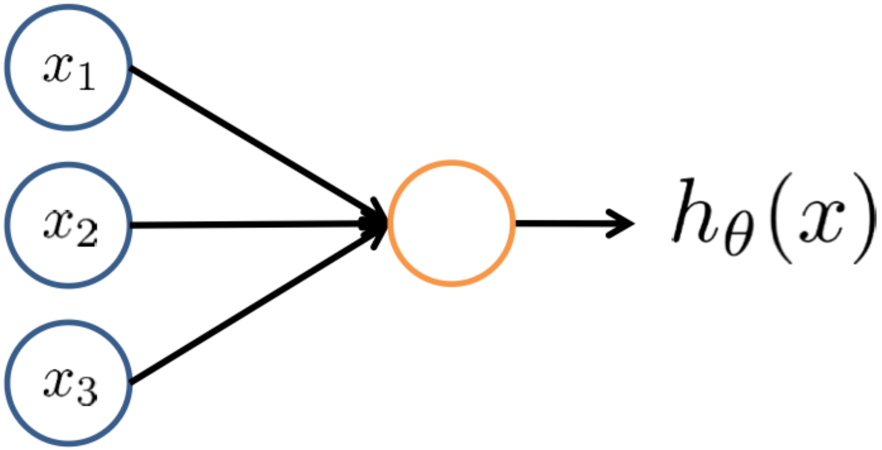
$$\mathbf{z} = g(\mathbf{a})$$

$$\hat{\mathbf{y}} \longleftrightarrow \mathbf{y}$$

$$\frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{y}_c} = \hat{y}_c - y_c$$

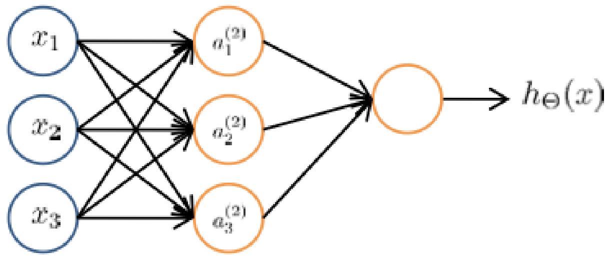
Neuron model: Logistic unit

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



Sigmoid (logistic) activation function.

Neural Network



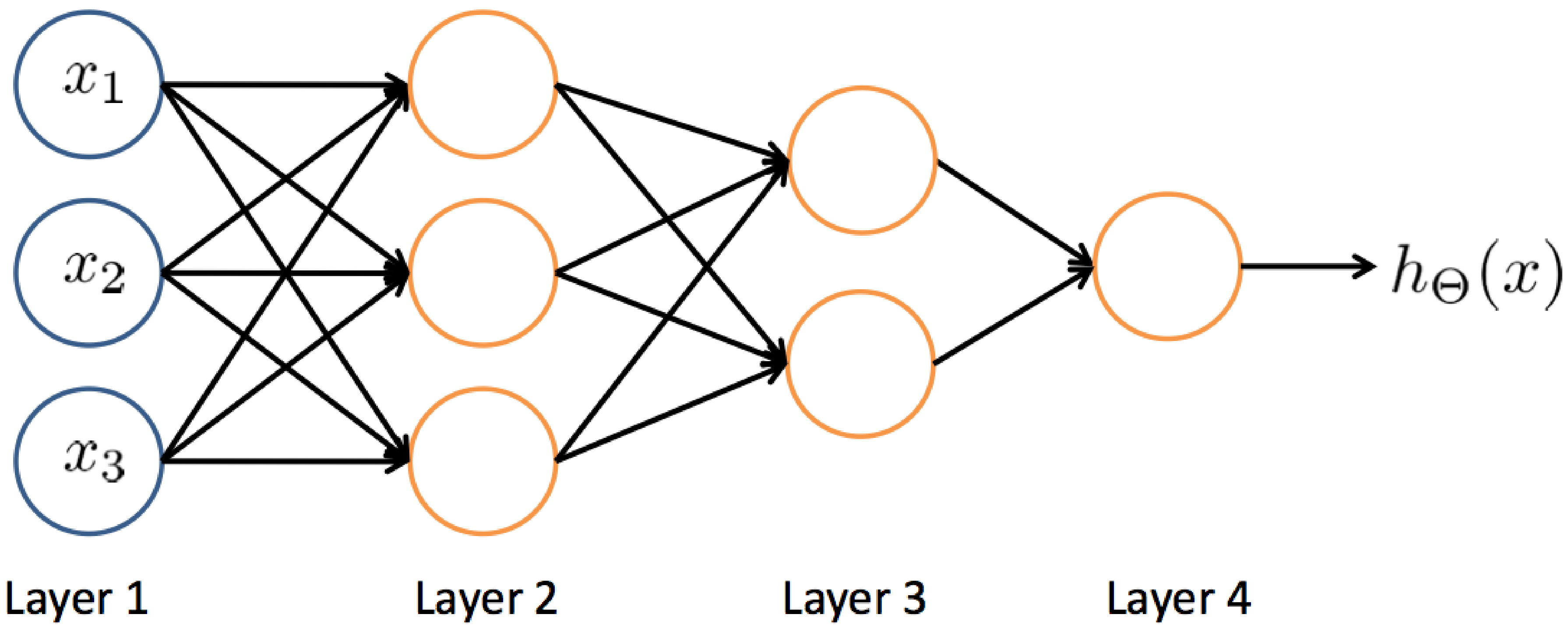
$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

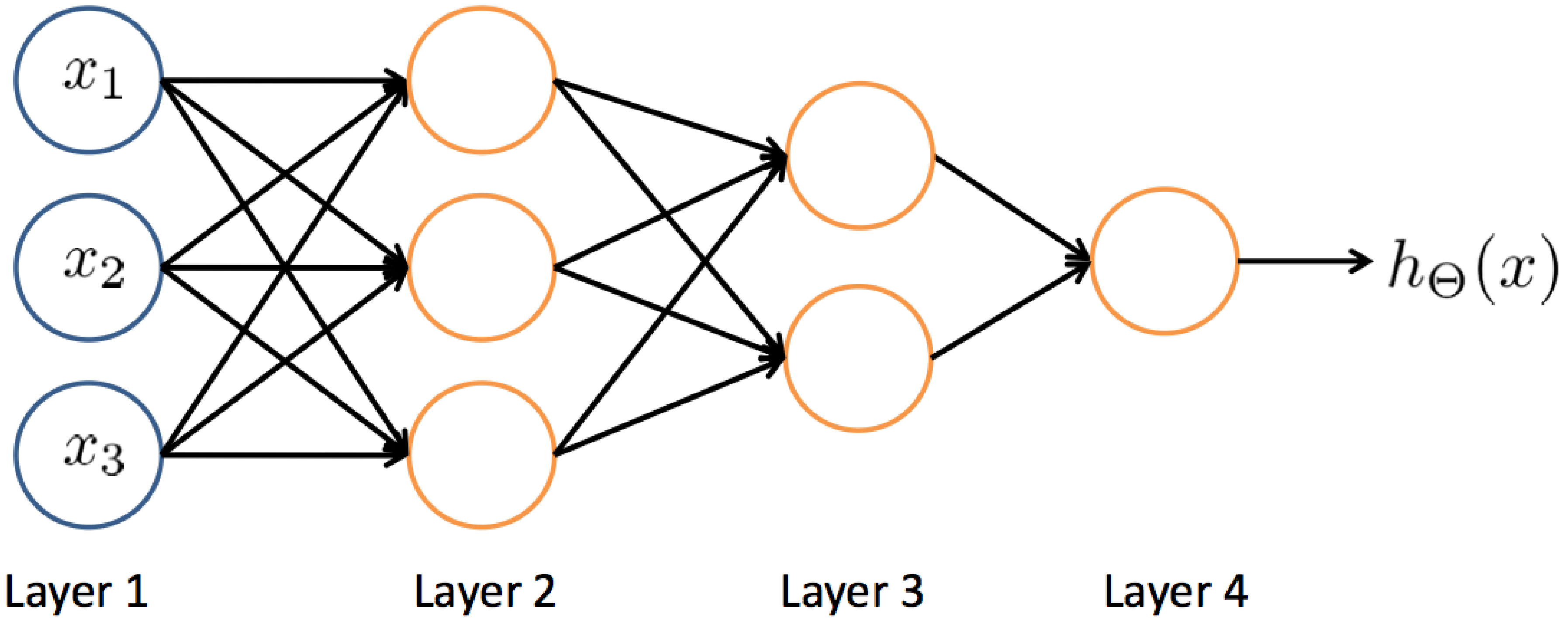
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

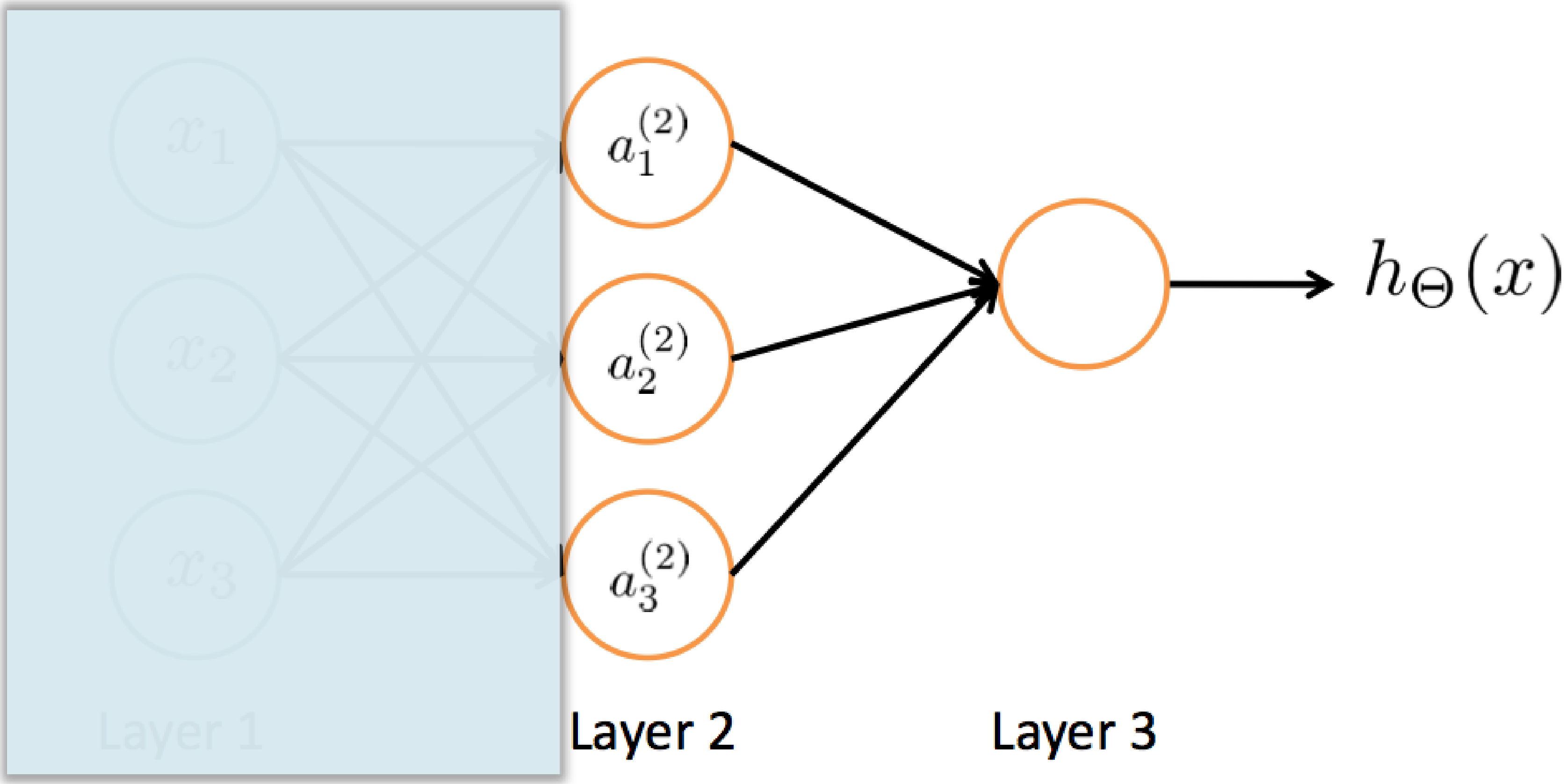
Slide credits: A. Ng



Slide credits: A. Ng

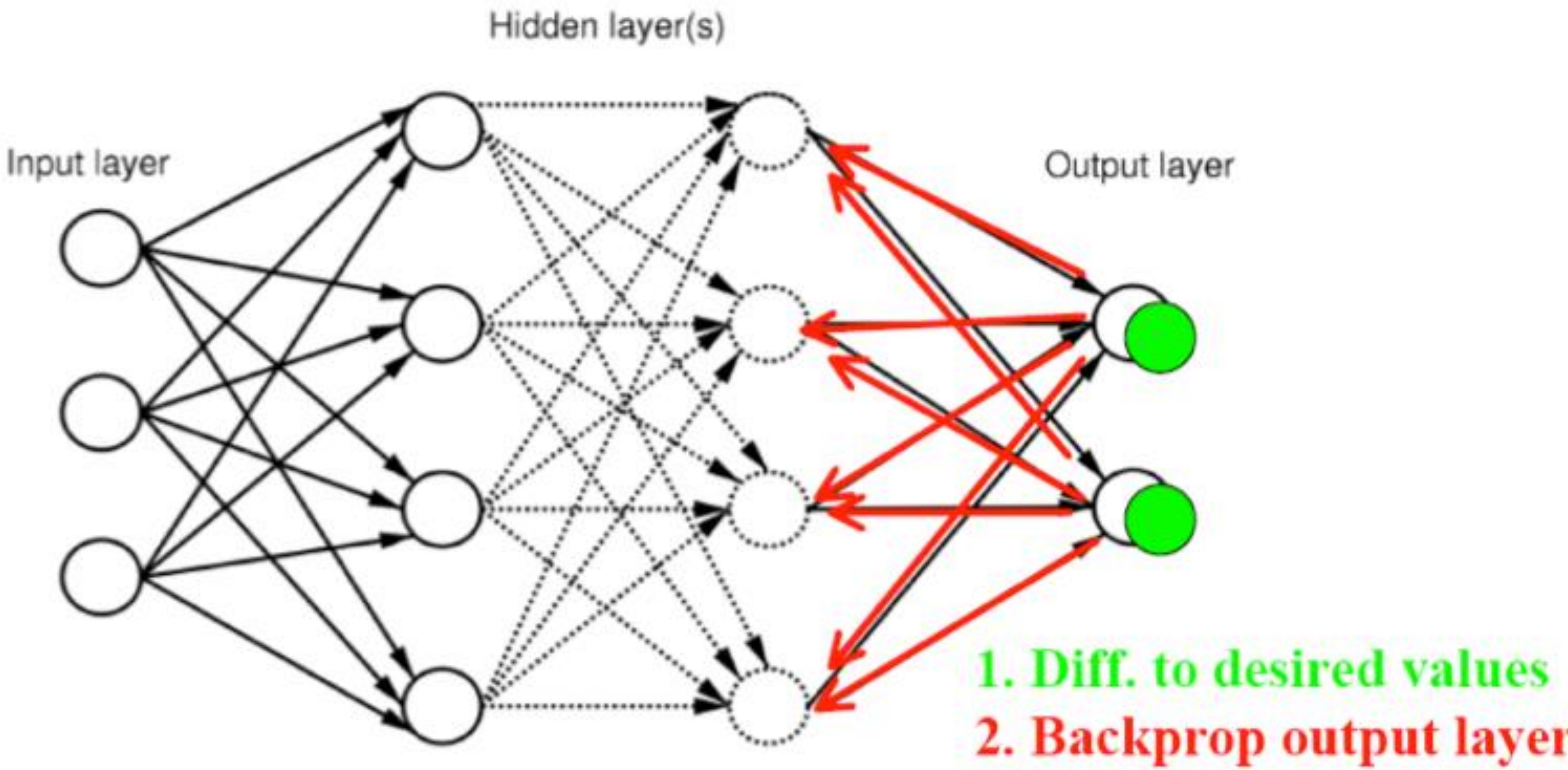


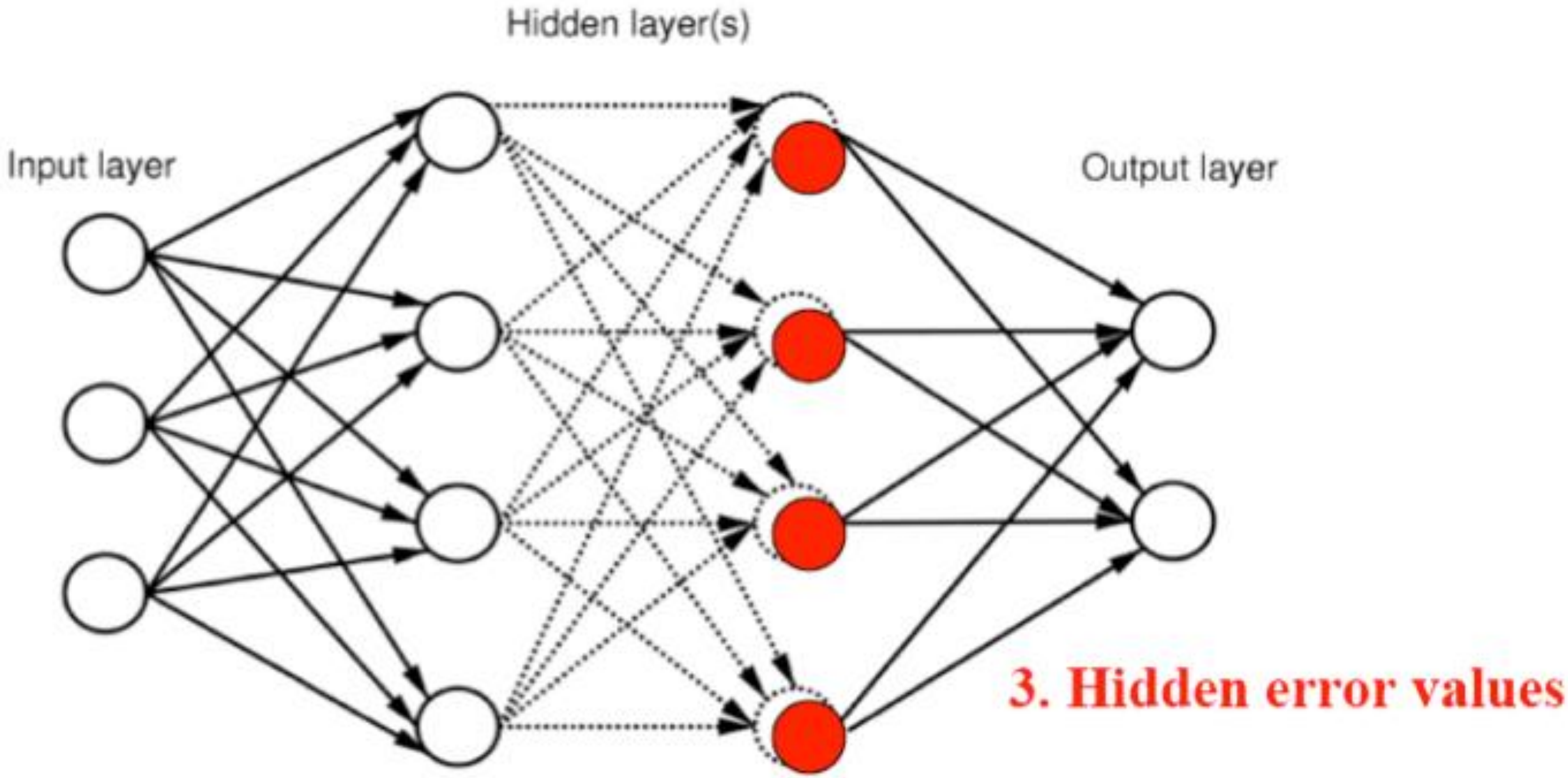
Slide credits: A. Ng

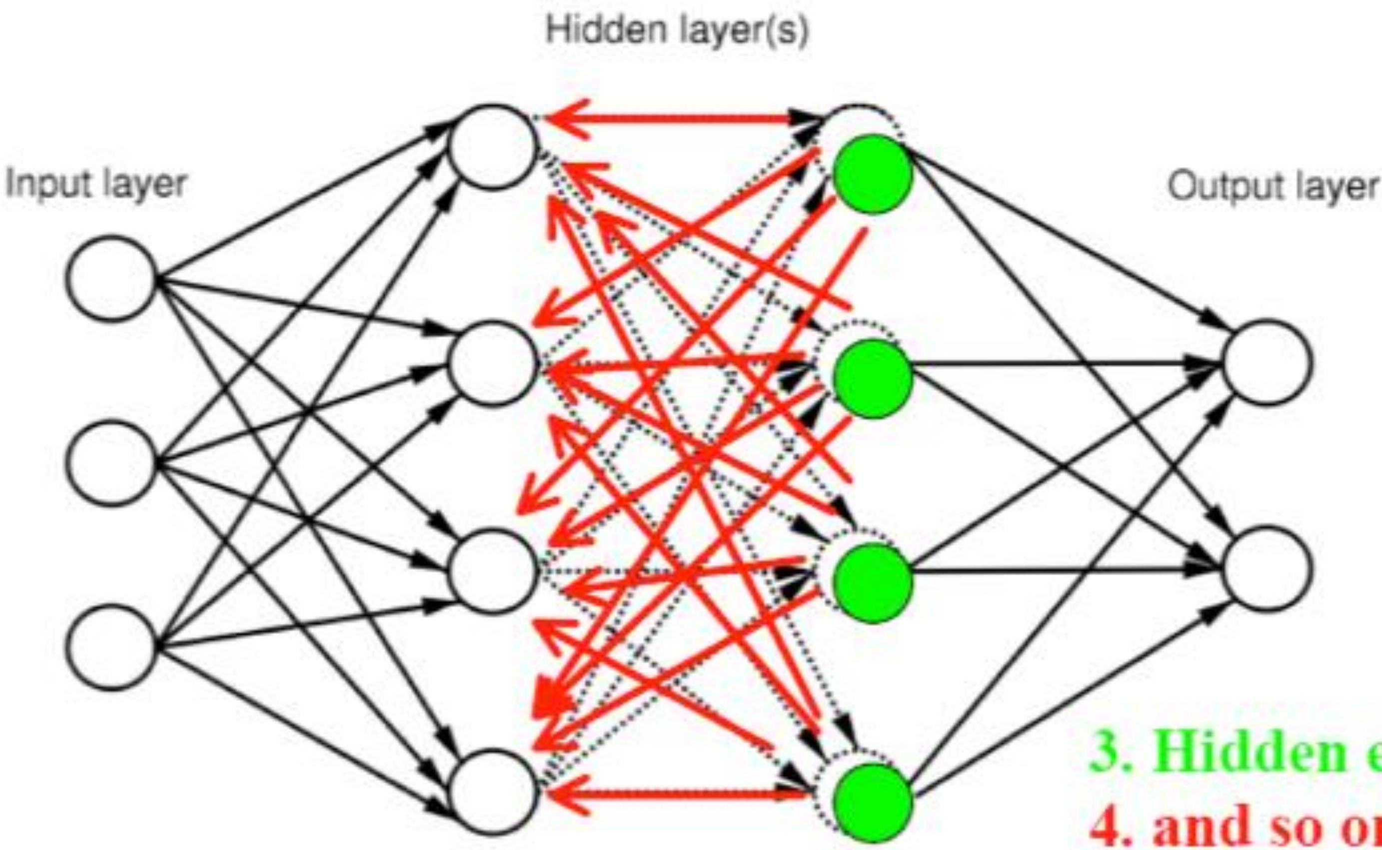


Learned features

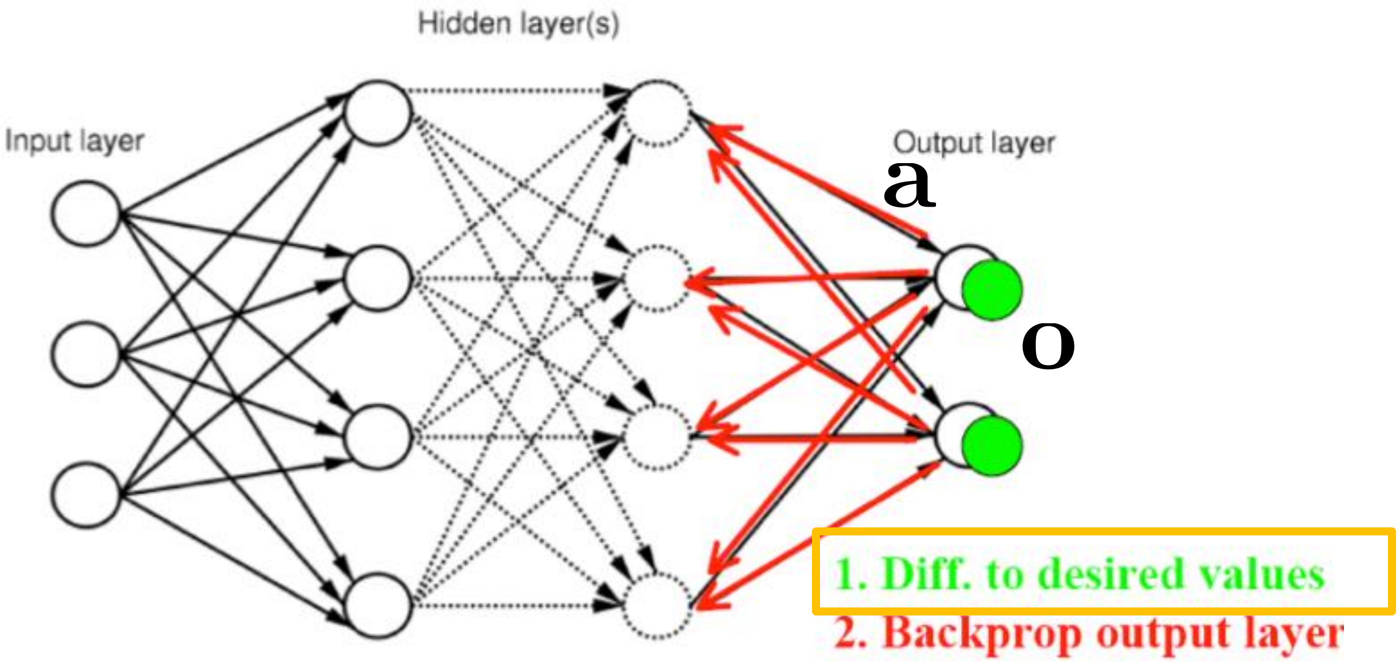
Slide credits: A. Ng


Hidden layer(s)
Input layer
Output layer
1. Diff. to desired values
2. Backprop output layer


Hidden layer(s)
Input layer
Output layer
3. Hidden error values

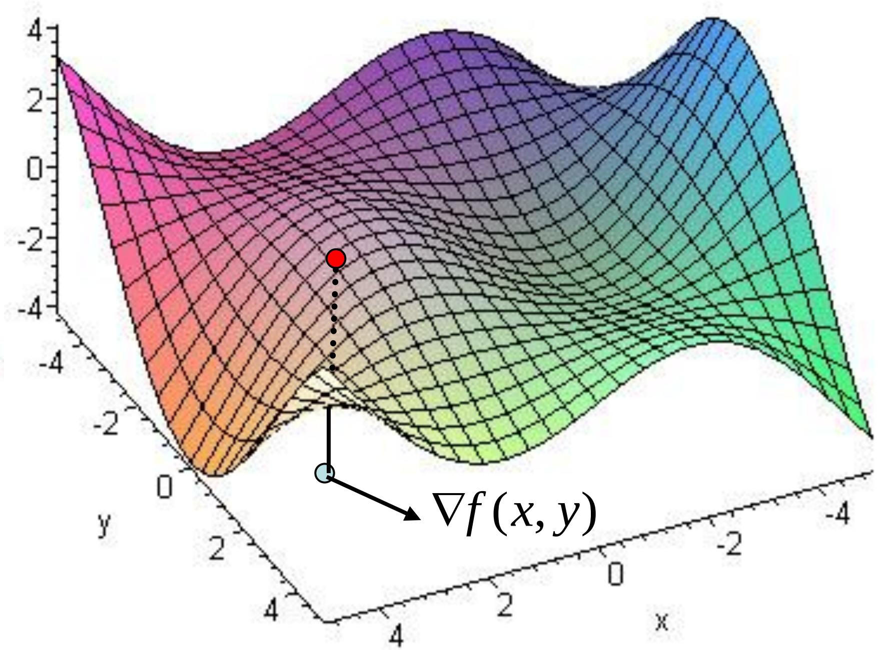


3. Hidden error values
4. and so on …


Hidden layer(s)
Input layer
Output layer
a
o
1. Diff. to desired values
2. Backprop output layer

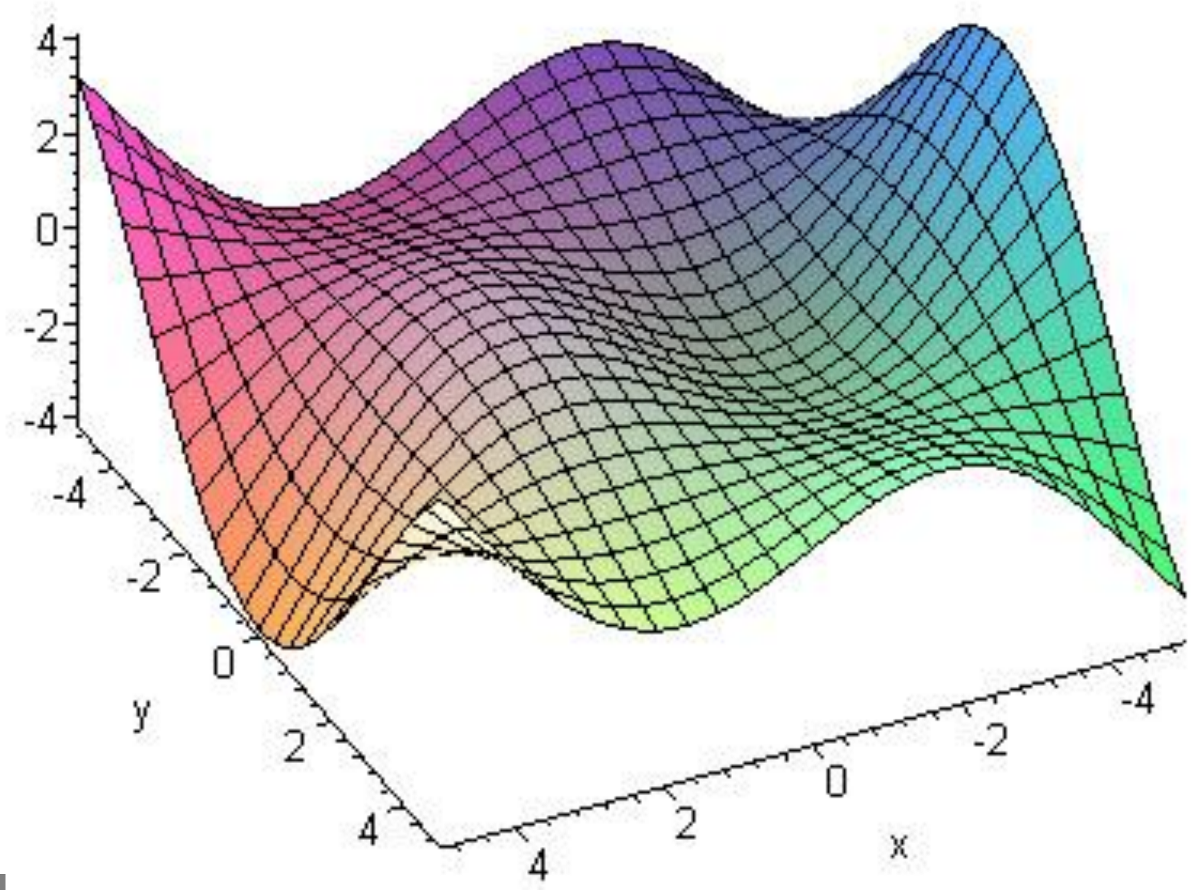
The Gradient: Definition in R^2

$$f : R^2 \to R \qquad \nabla f(x,y) := \begin{pmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \end{pmatrix}$$



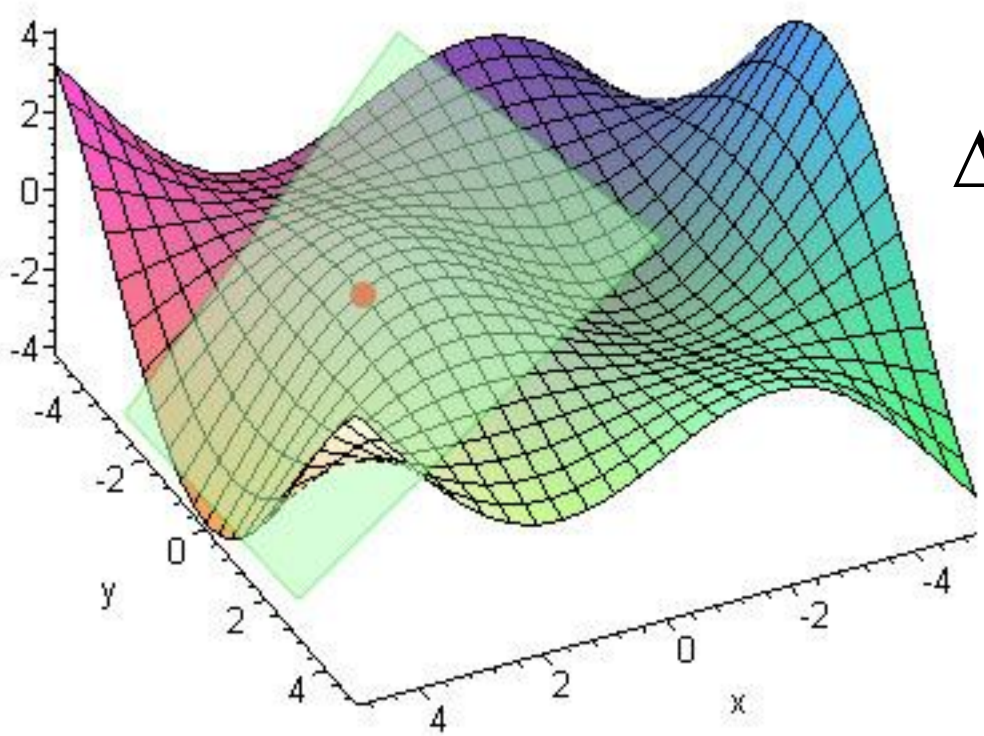
In the plane

$$f := (x, y) \rightarrow \cos\left(\frac{1}{2}x\right)\cos\left(\frac{1}{2}y\right)x$$



The Gradient Properties

- The gradient defines (hyper) plane approximating the function infinitesimally

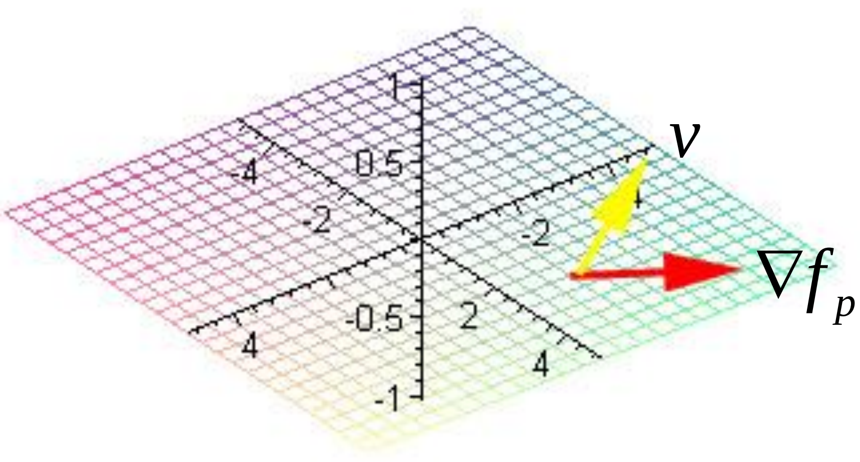


$$\Delta z = \frac{\partial f}{\partial x} \cdot \Delta x + \frac{\partial f}{\partial y} \cdot \Delta y$$

The Gradient properties

- By the chain rule: (important for later use)

$$\|v\| = 1 \qquad \frac{\partial f}{\partial v}(p) = \langle \nabla f_p, v \rangle$$



The Gradient properties

- **Proposition 1:**

  $\frac{\partial f}{\partial v}$ **is maximal choosing** $v = \frac{1}{\left\|\nabla f_p\right\|} \cdot \nabla f_p$

  $\frac{\partial f}{\partial v}$ **is minimal choosing** $v = \frac{-1}{\left\|\nabla f_p\right\|} \cdot \nabla f_p$

(intuitive: the gradient points at the greatest change direction)

Steepest Descent

- What it mean?
- We now use what we have learned to implement the most basic minimization technique.
- First we introduce the algorithm, which is a version of the model algorithm.
- The problem:

$$\min_{x} \ f(x)$$

Steepest Descent

- Steepest descent algorithm:

Data: $x_0 \in R^n$

Step 0: set i=0

Step 1: if $\nabla f(x_i) = 0$ stop,

else, compute ***search direction***

Step 2: compute the ***step-size*** $h_i = -\nabla f(x_i)$

Step 3: set $\lambda_i \in \arg\min_{\lambda \geq 0} f(x_i + \lambda \cdot h_i)$ go to step 1

$$x_{i+1} = x_i + \lambda_i \cdot h_i$$

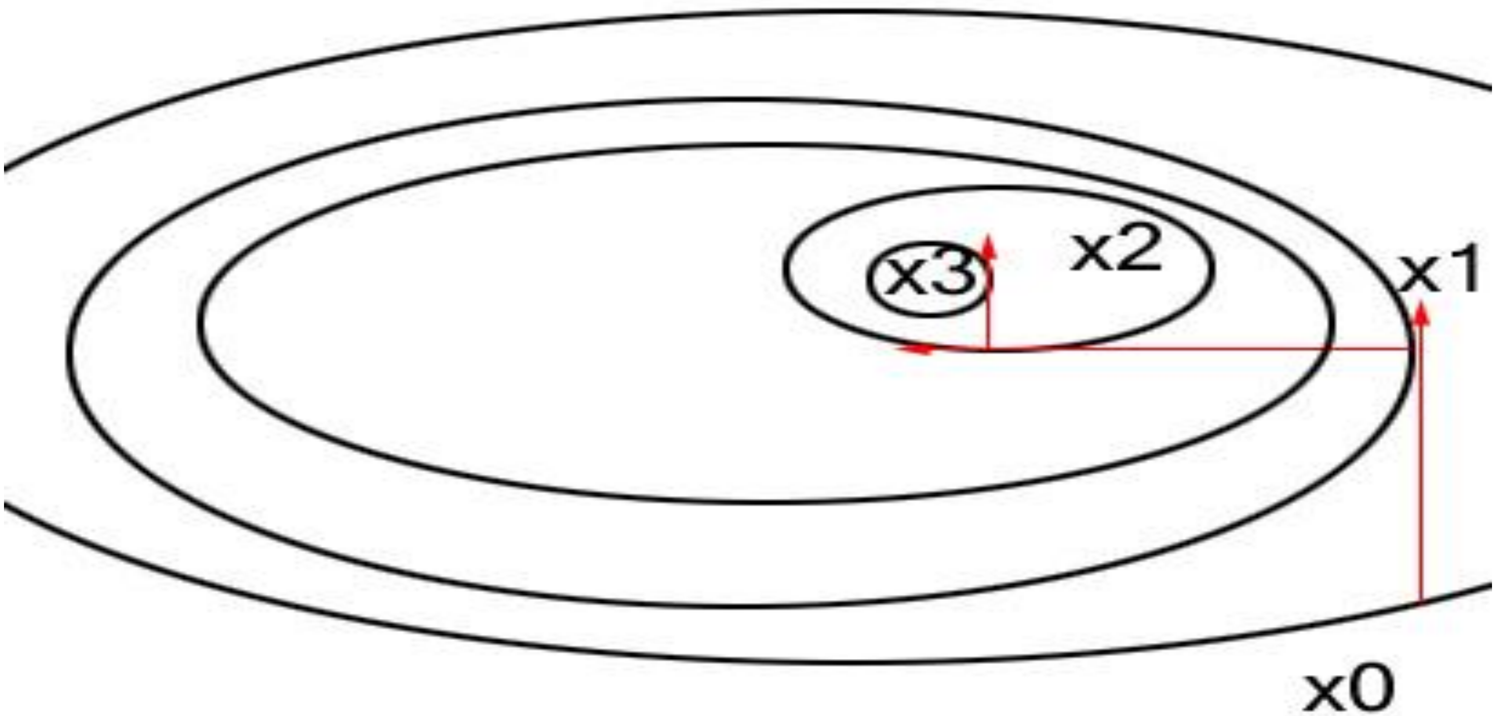
Steepest Descent

- From the chain rule:

$$\frac{d}{d\lambda} f(x_i + \lambda \cdot h_i) = \langle \nabla f(x_i + \lambda \cdot h_i), h_i \rangle = 0$$

- Therefore the method of steepest descent looks like this:

Steepest Descent



Steepest Descent

- The steepest descent find critical point and local minimum.
- Implicit step-size rule
- Actually we reduced the problem to finding minimum:

$$f : R \to R$$

- There are extensions that gives the step size rule in discrete sense. (Armijo)

Steepest Descent

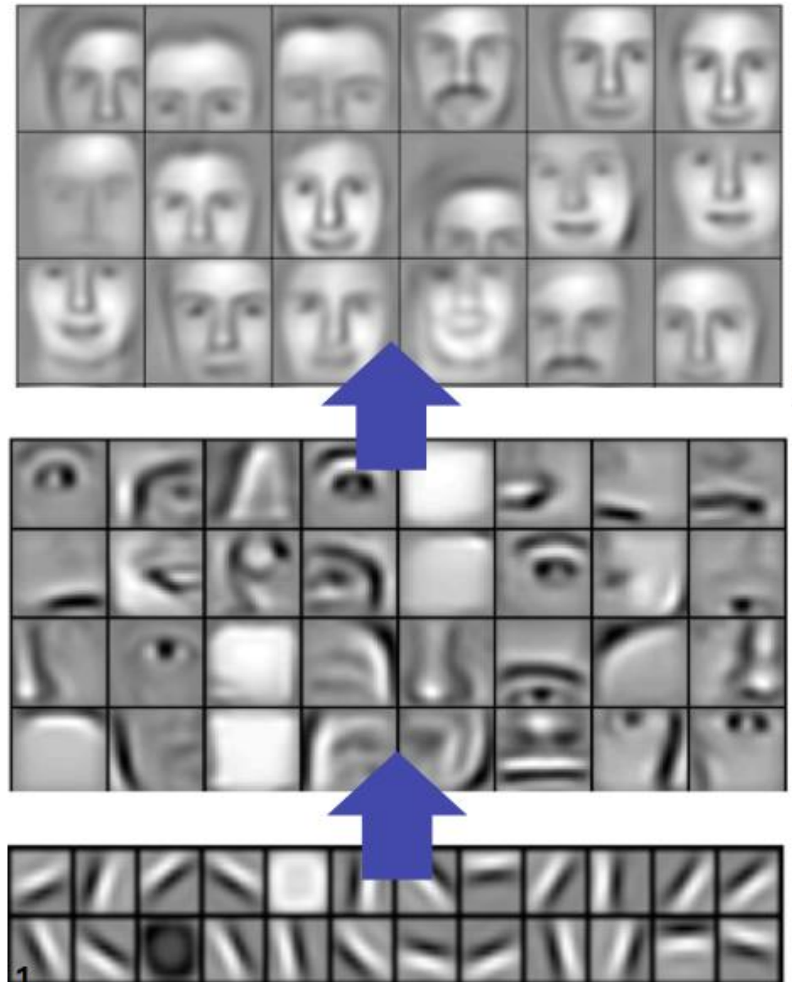
- Back with our connectivity shapes: the authors solve the 1-dimension problem analytically.

$$\lambda_i \in \arg\min_{\lambda \geq 0} f(x_i + \lambda \cdot h_i)$$

- They change the spring energy and get a quartic polynomial in x

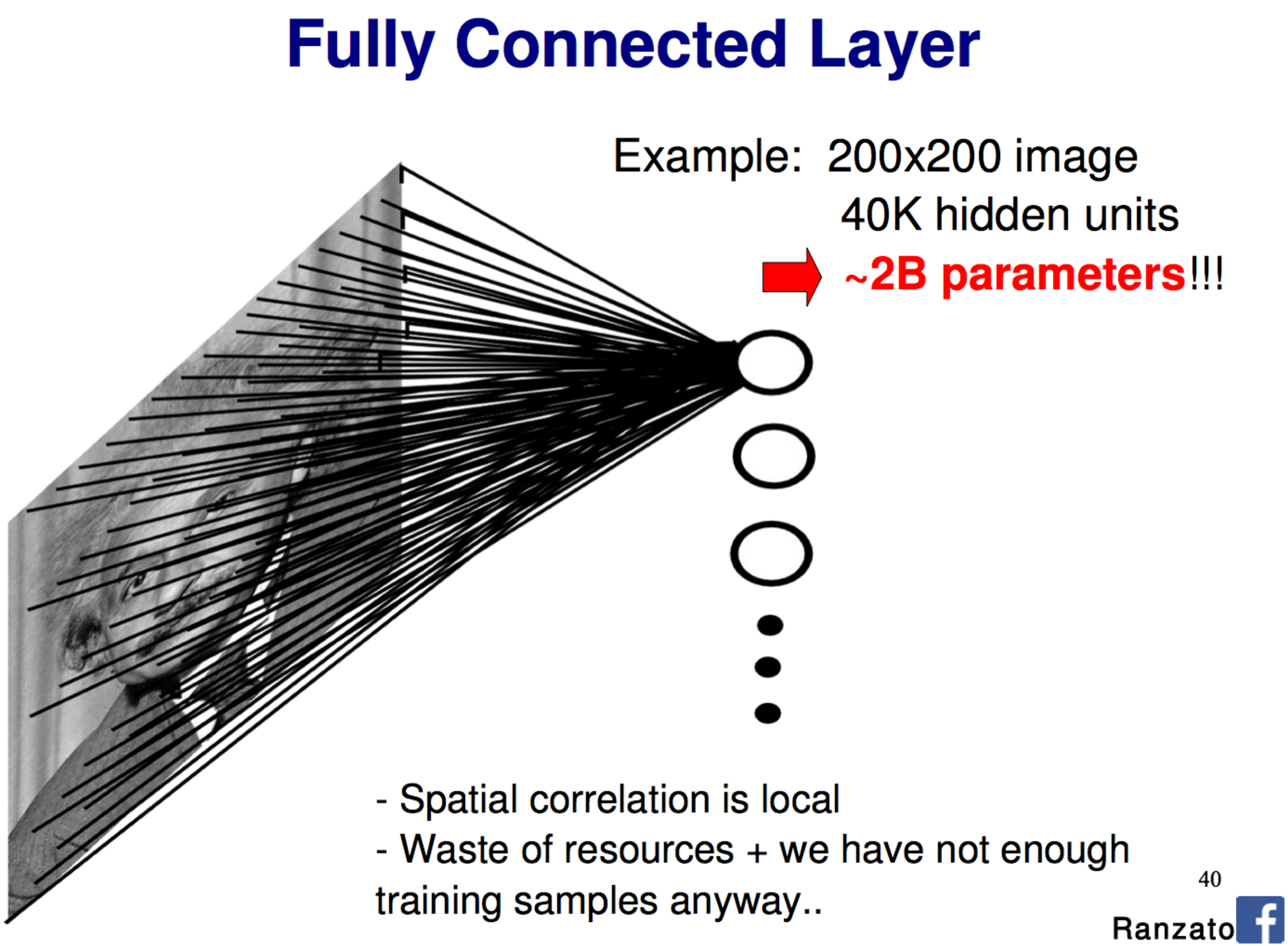
$$E_s(x \in \mathbb{R}^{n\times 3}) = \sum_{(i,j)\in E} \left(\left\| x_i - x_j \right\|^2 - 1 \right)^2$$

Convolutional models & deep networks



Honglak Lee & Andrew Ng, ICML 2010

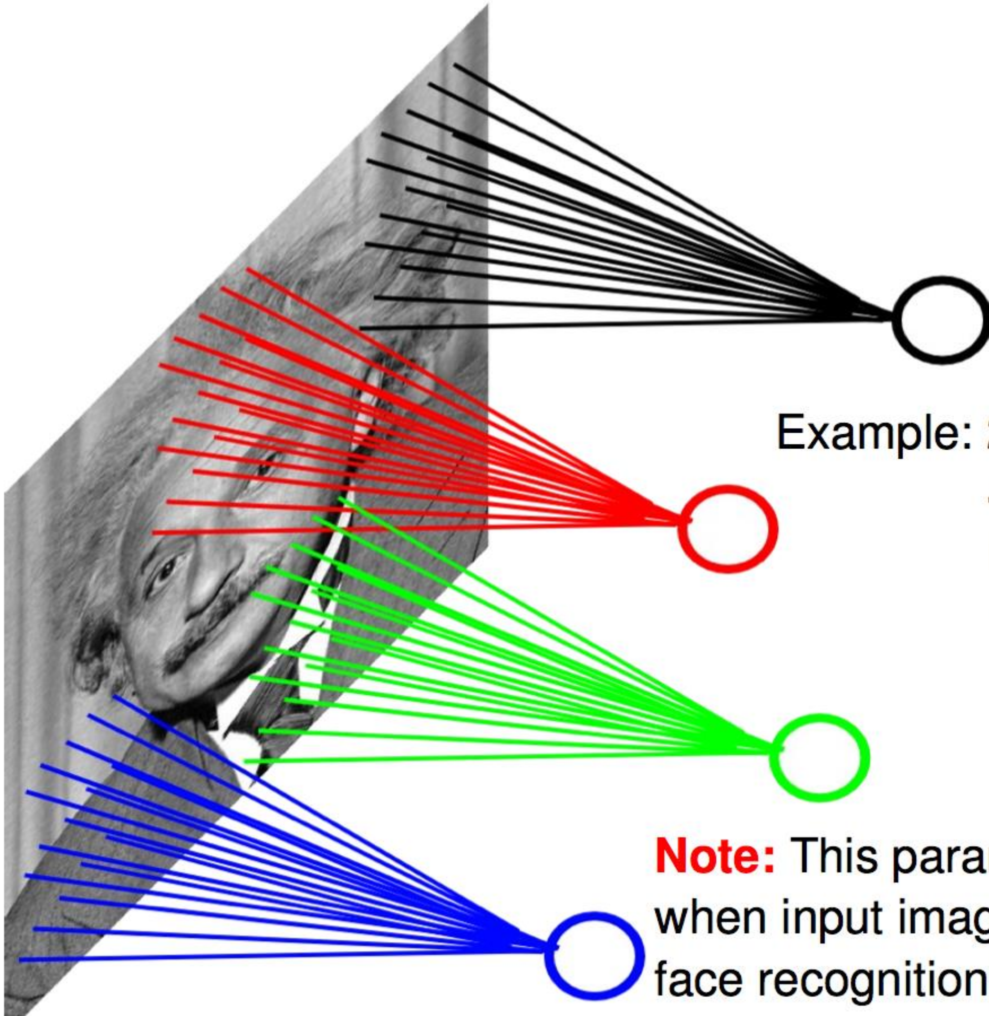
Network connectivity



Slide credits: M. A. Ranzatto

Network connectivity

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

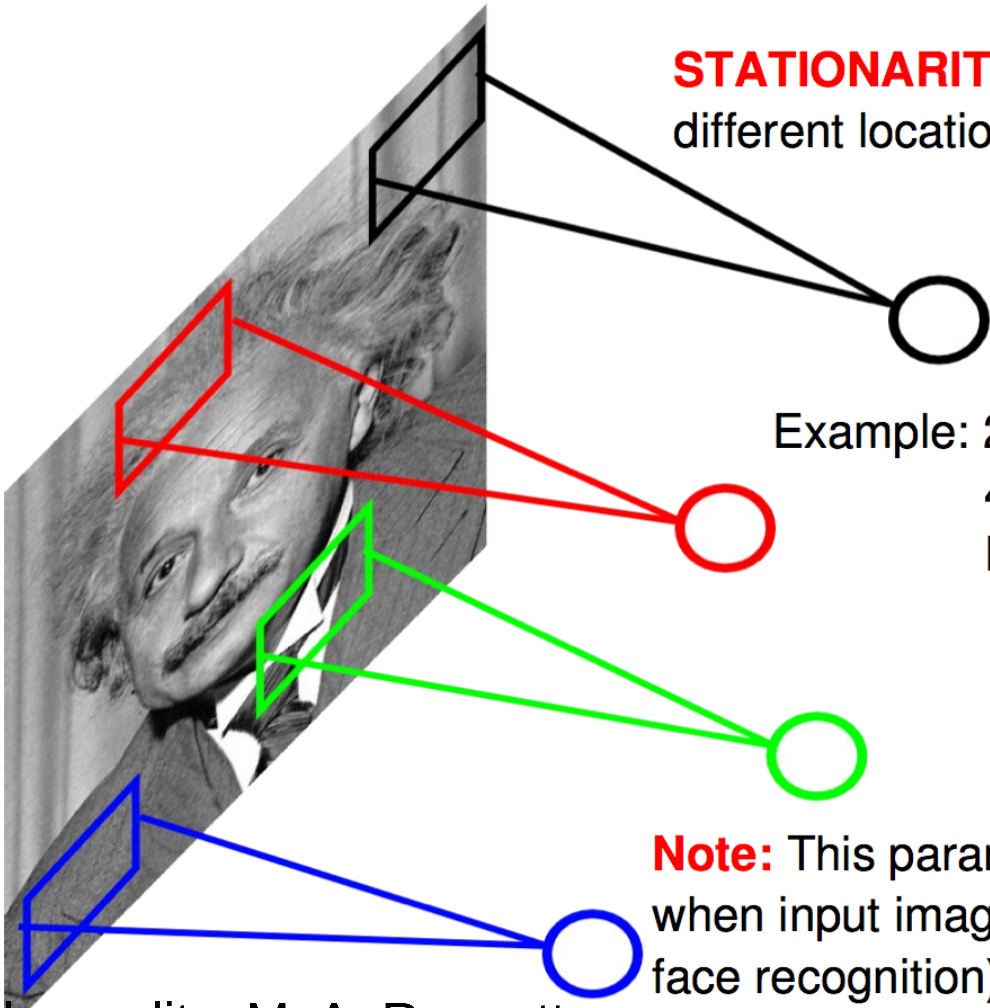
41

Ranzato

Slide credits: M. A. Ranzatto

Network connectivity

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

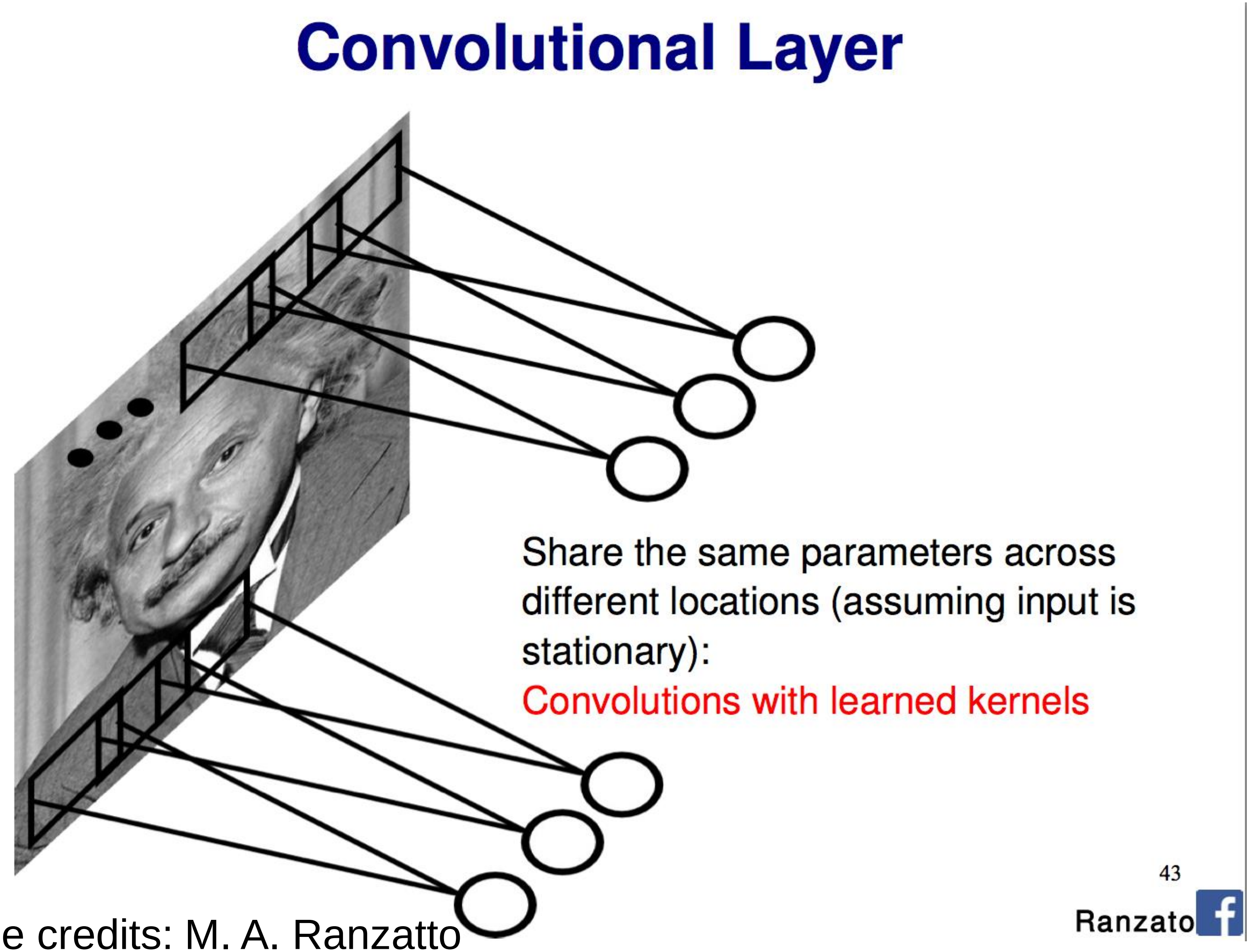
Note: This parameterization is good when input image is registered (e.g., face recognition).

42

Ranzato

Slide credits: M. A. Ranzatto

Network connectivity



Slide credits: M. A. Ranzatto

Network connectivity

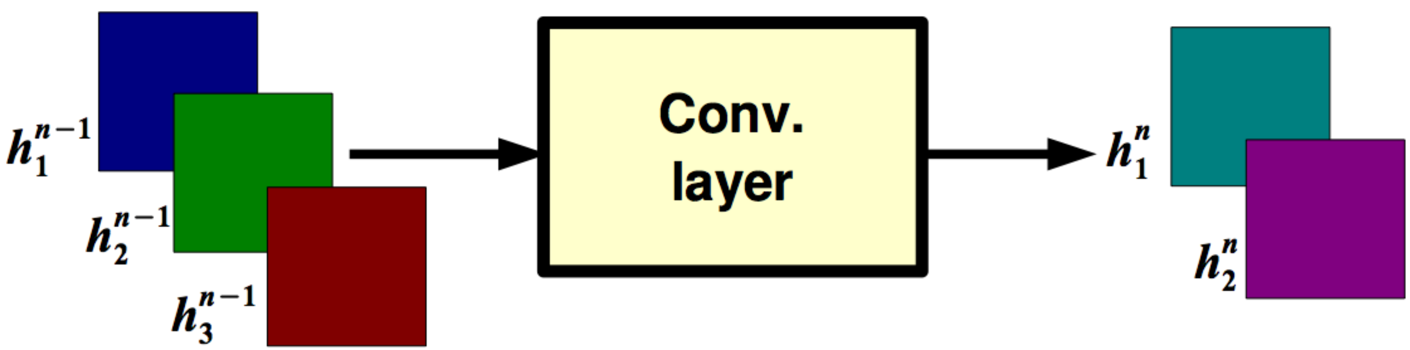
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

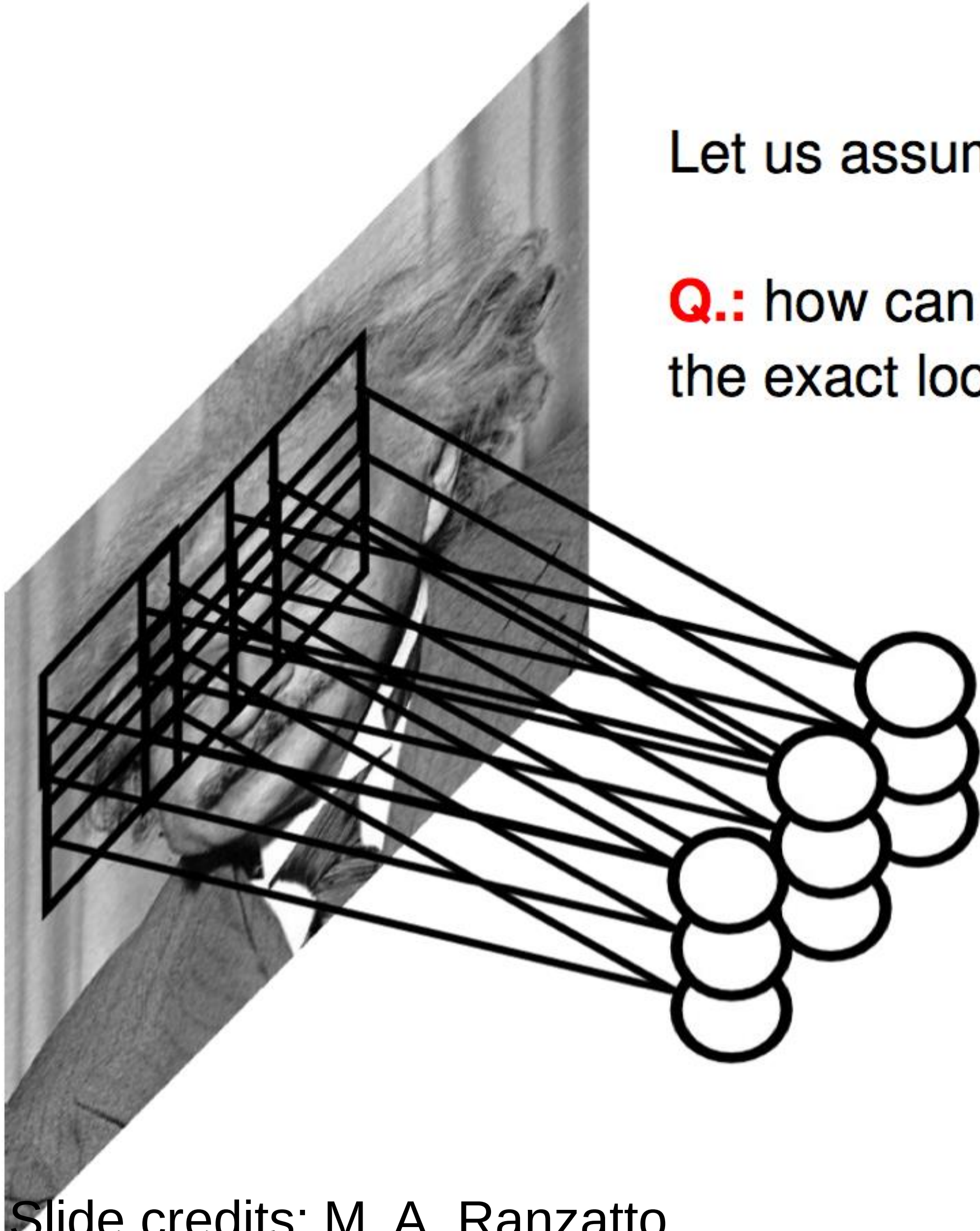
input feature map

kernel



Slide credits: M. A. Ranzatto

Pooling Layer



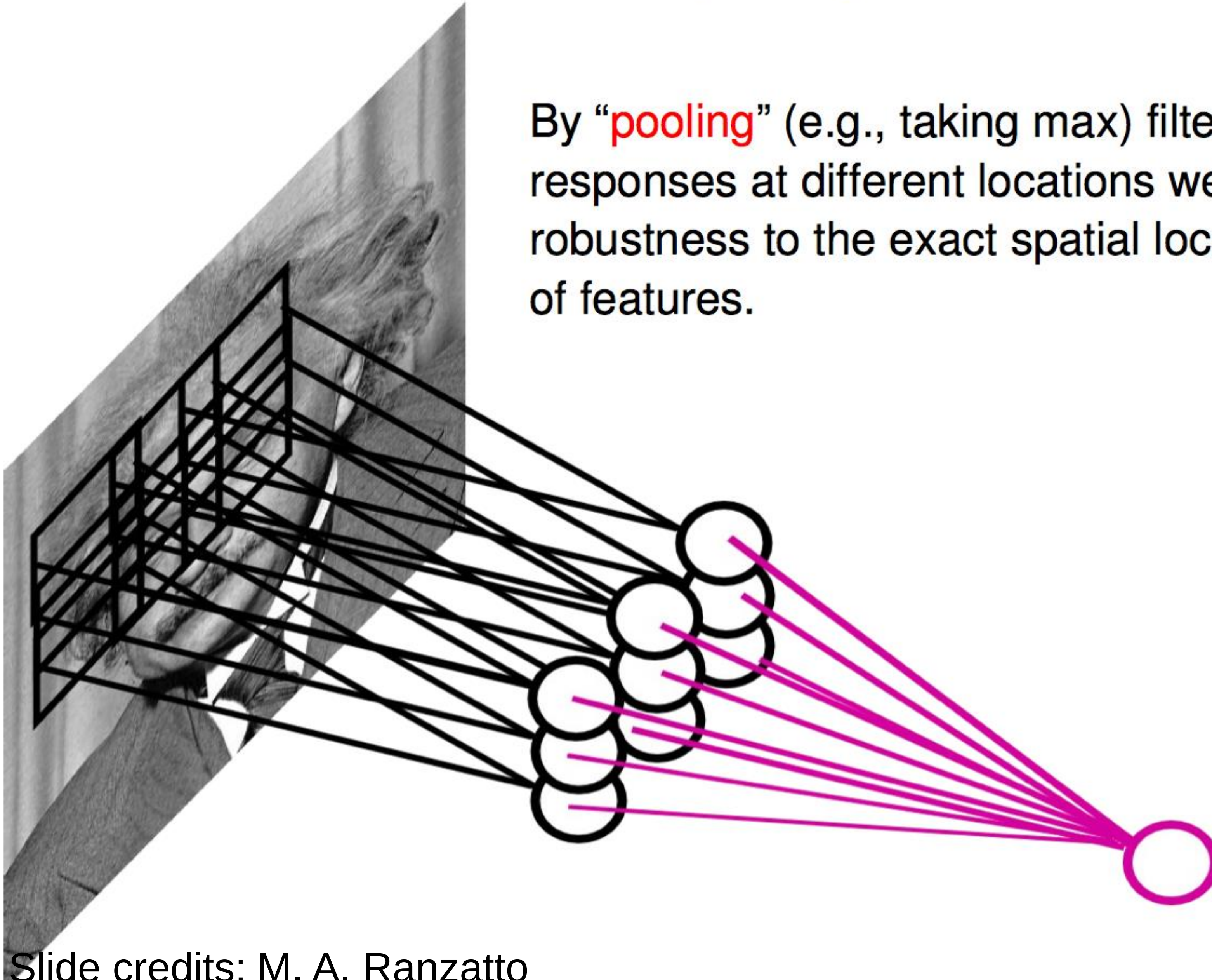
Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Ranzato

Slide credits: M. A. Ranzatto

Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slide credits: M. A. Ranzatto

Ranzato

Pooling Layer: 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:

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

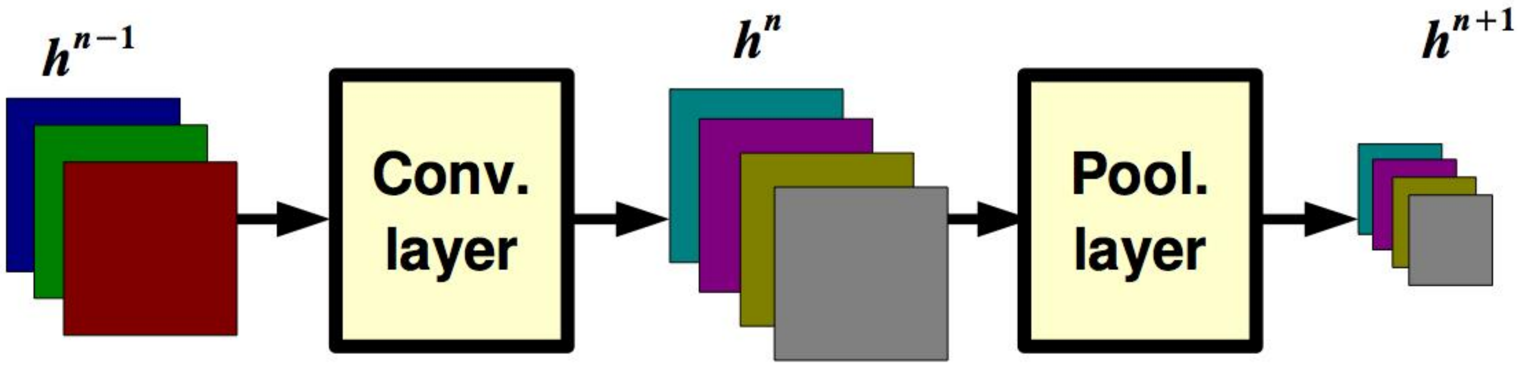
L2-pooling over features:

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

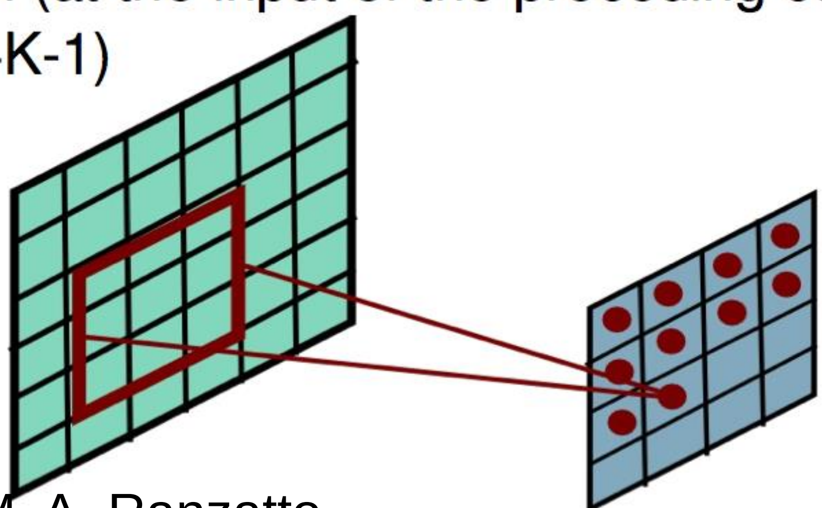
Slide credits: M. A. Ranzatto

Ranzato

Pooling Layer: Receptive Field Size



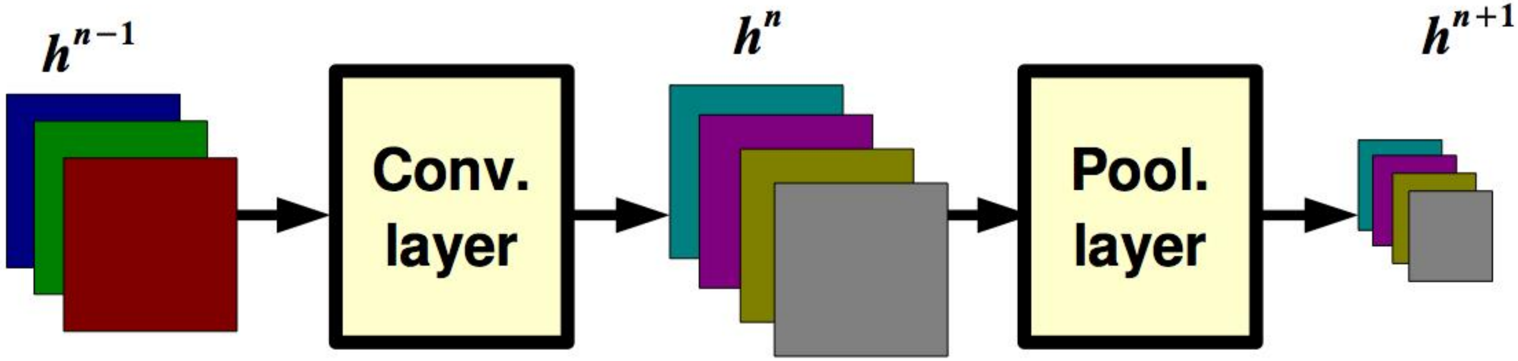
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



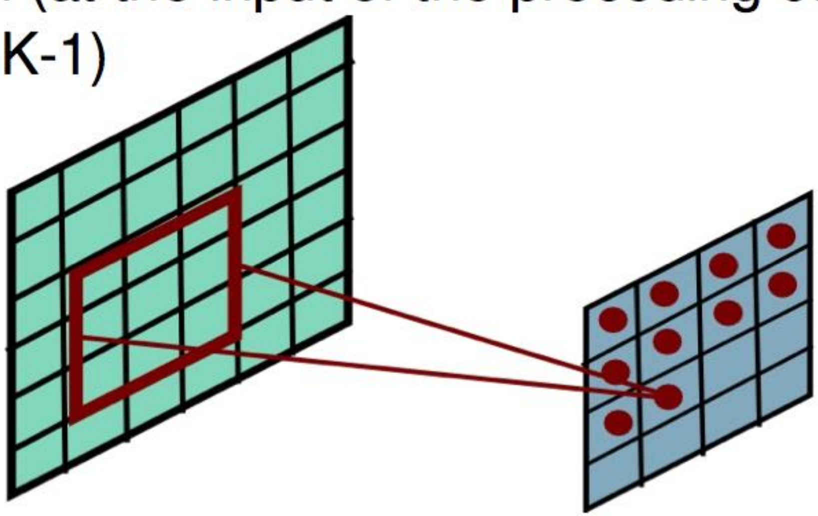
Ranzato

Slide credits: M. A. Ranzatto

Pooling Layer: Receptive Field Size

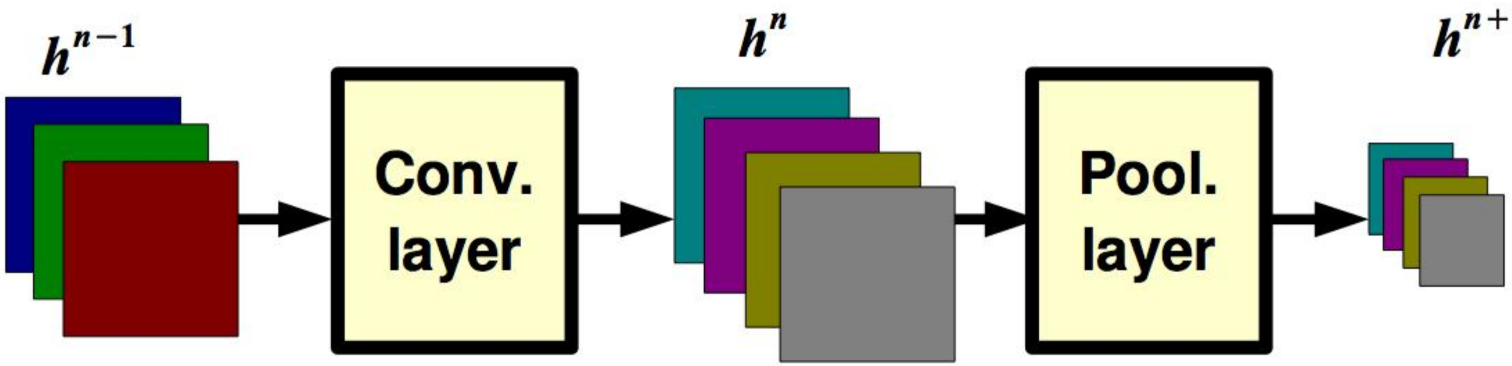


If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)

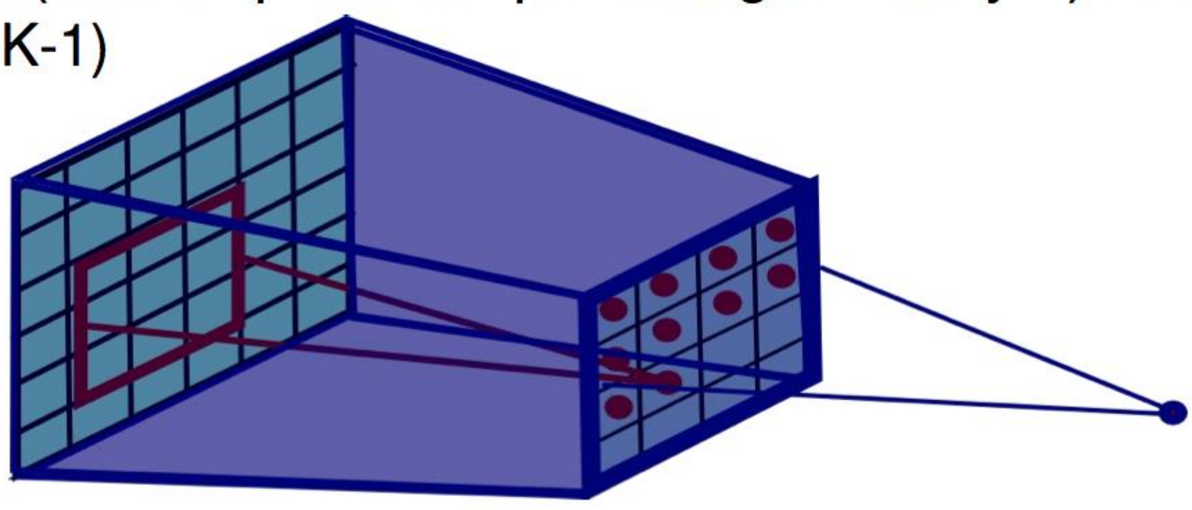


Ranzato

Pooling Layer: Receptive Field Size



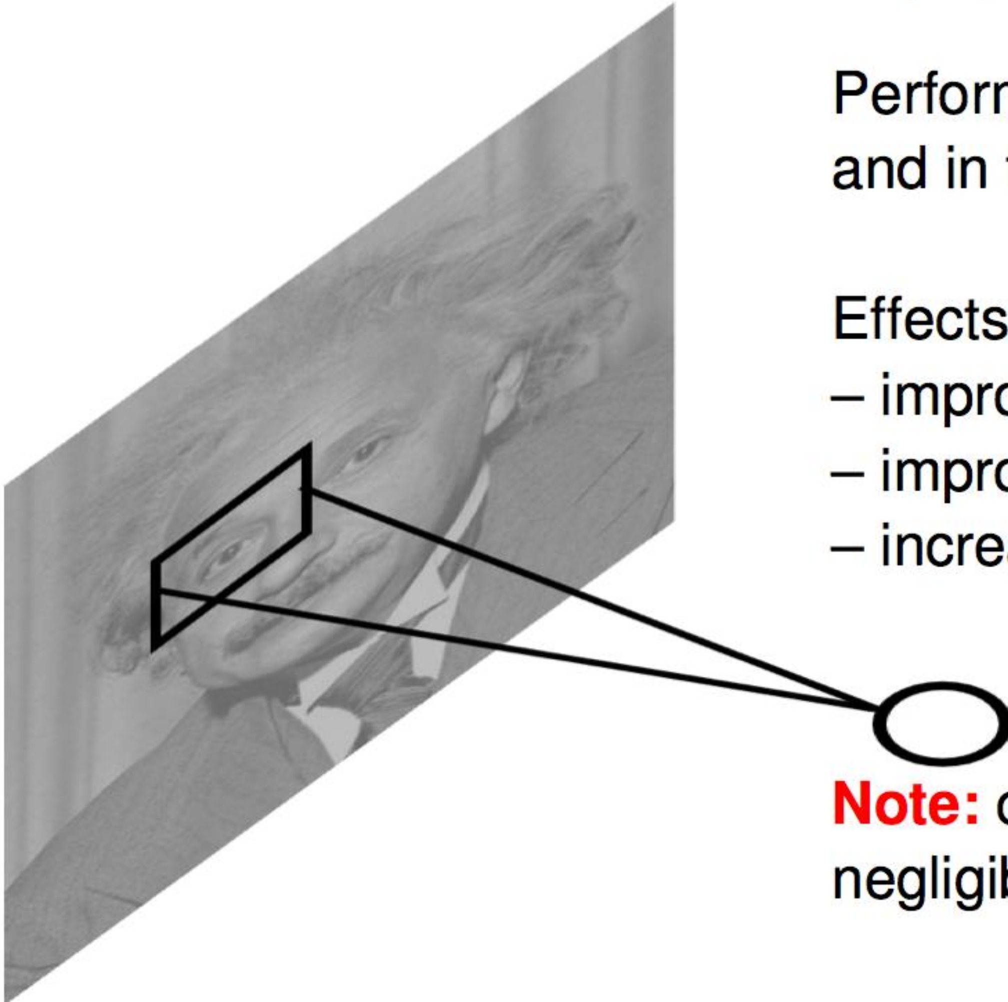
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Ranzato

Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{max(\epsilon, \sigma^i(N(x,y)))}$$



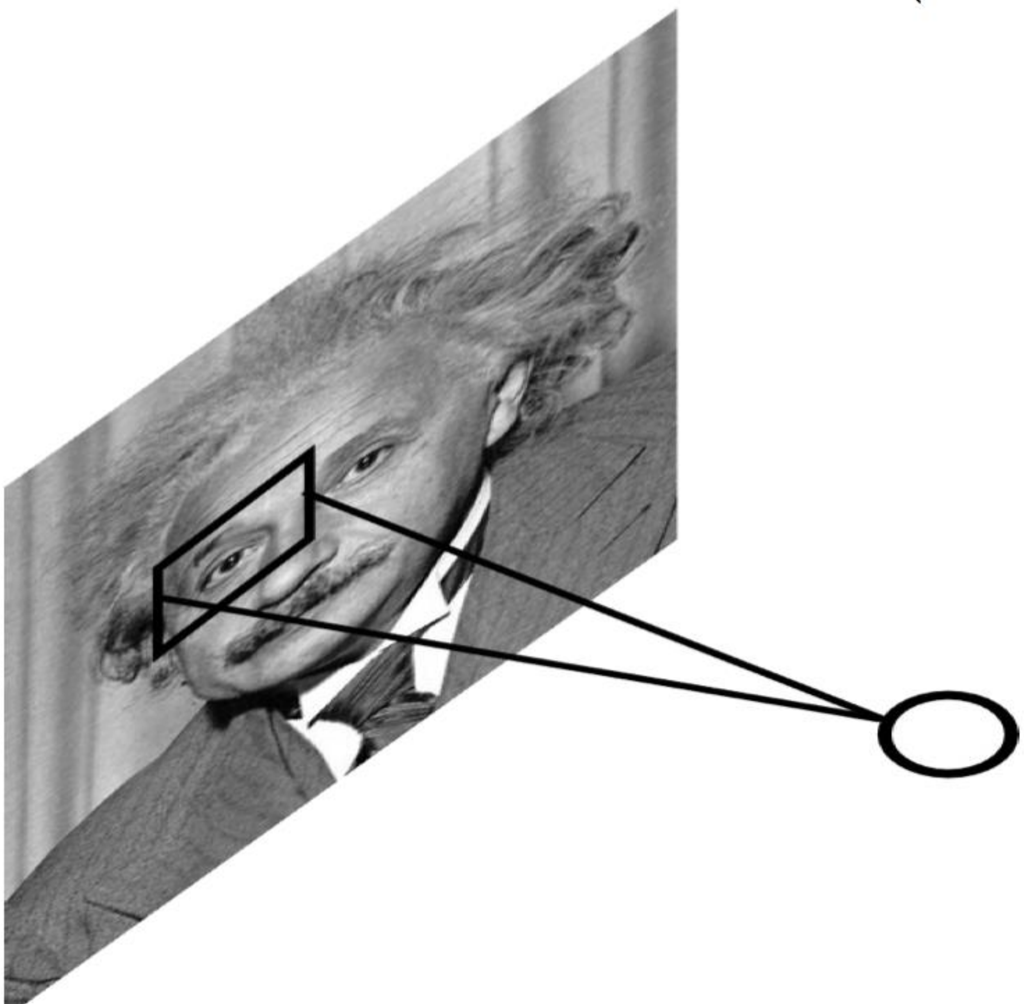
Performed also across features and in the higher layers..

Effects:
– improves invariance
– improves optimization
– increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

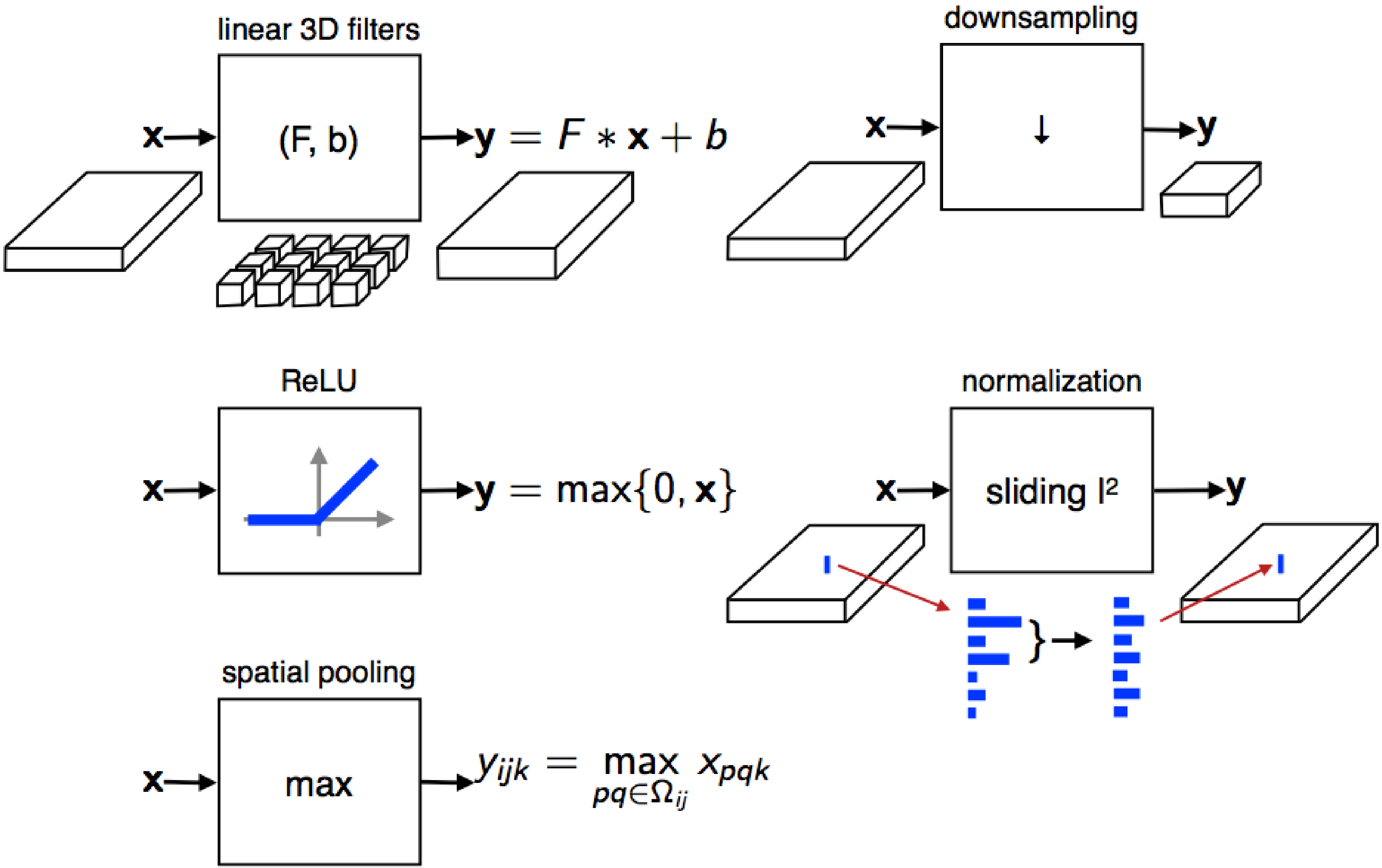
Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{max(\epsilon, \sigma^i(N(x,y)))}$$



Ranzato

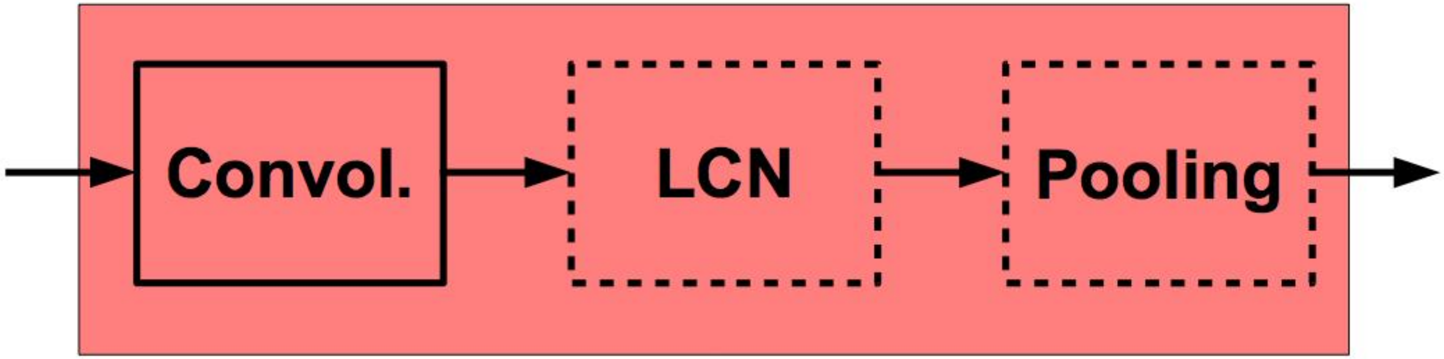
CNN components

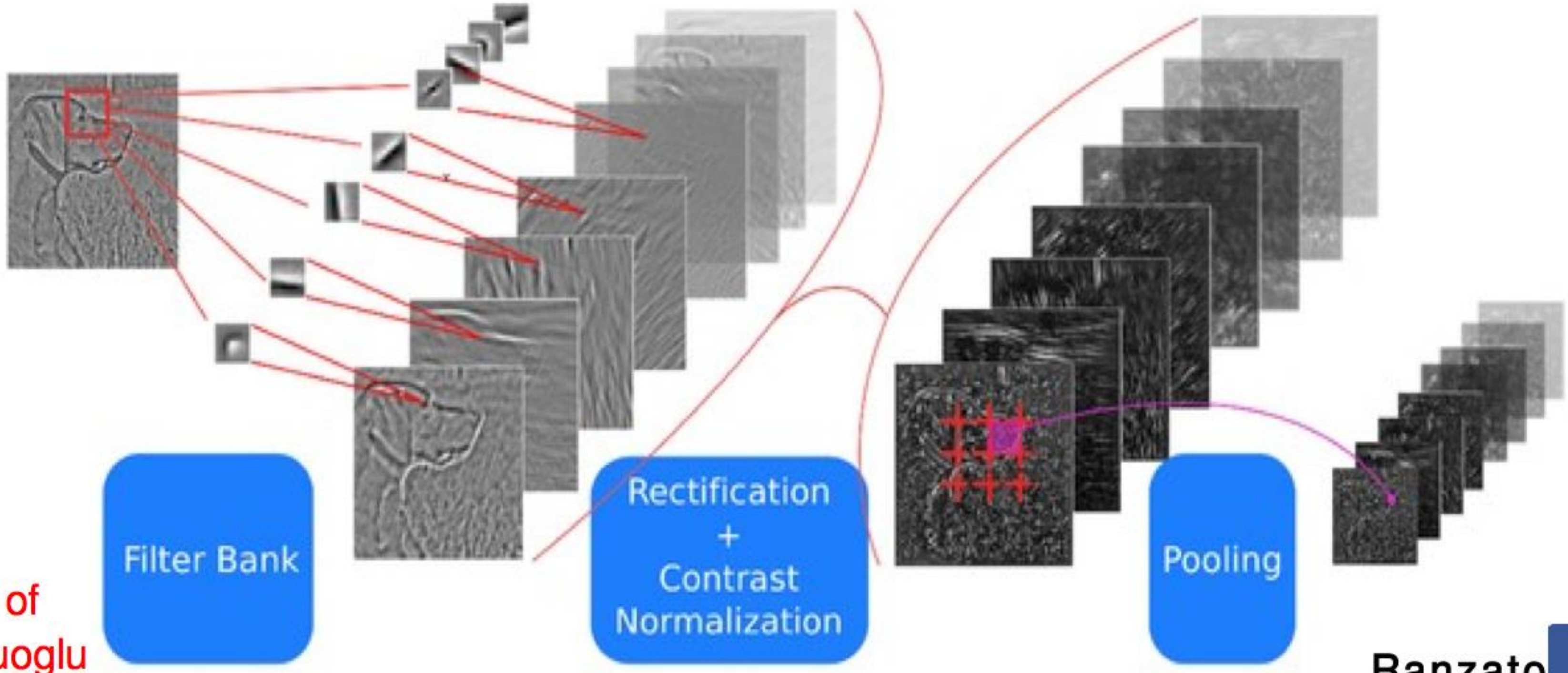


Slide credits: A. Vedaldi

ConvNets: Typical Stage

One stage (zoom)



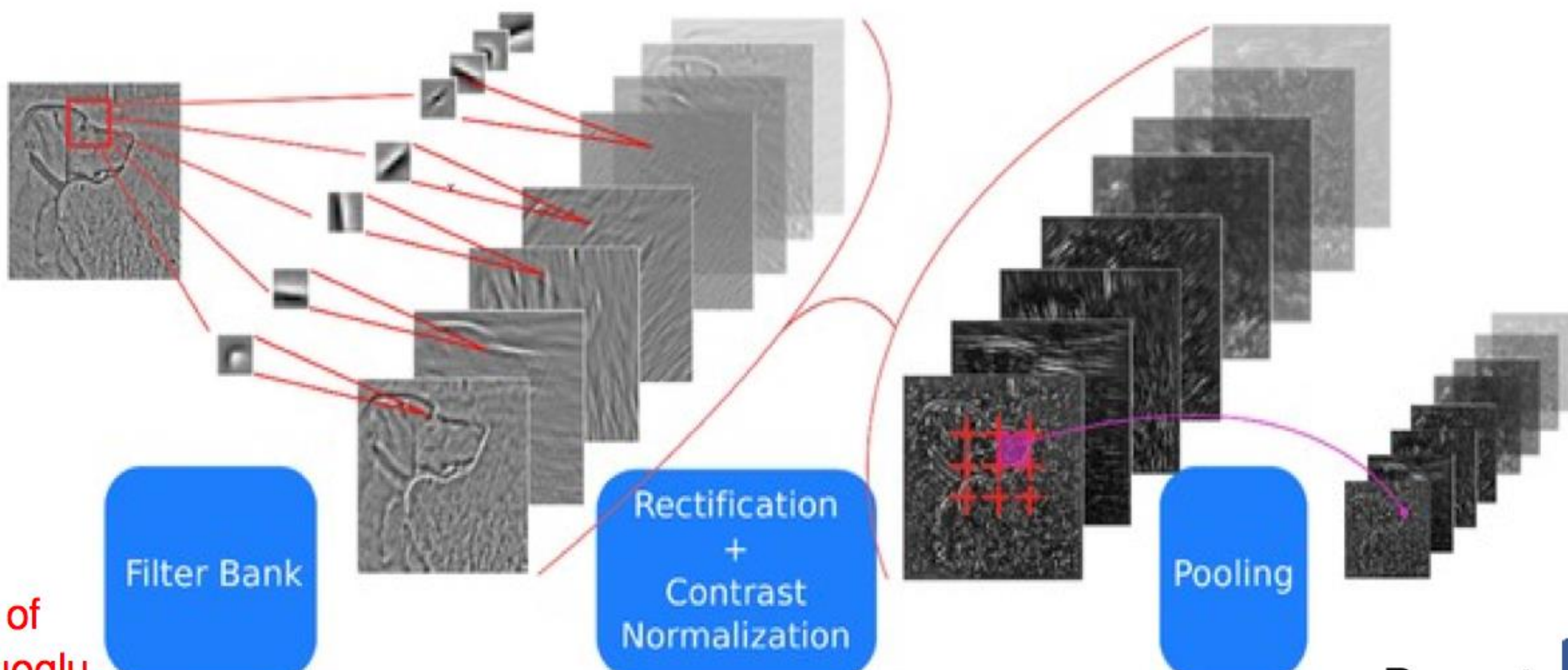


courtesy of
K. Kavukcuoglu

Ranzato

Note: after one stage the number of feature maps is usually increased (conv. layer) and the spatial resolution is usually decreased (stride in conv. and pooling layers). Receptive field gets bigger.

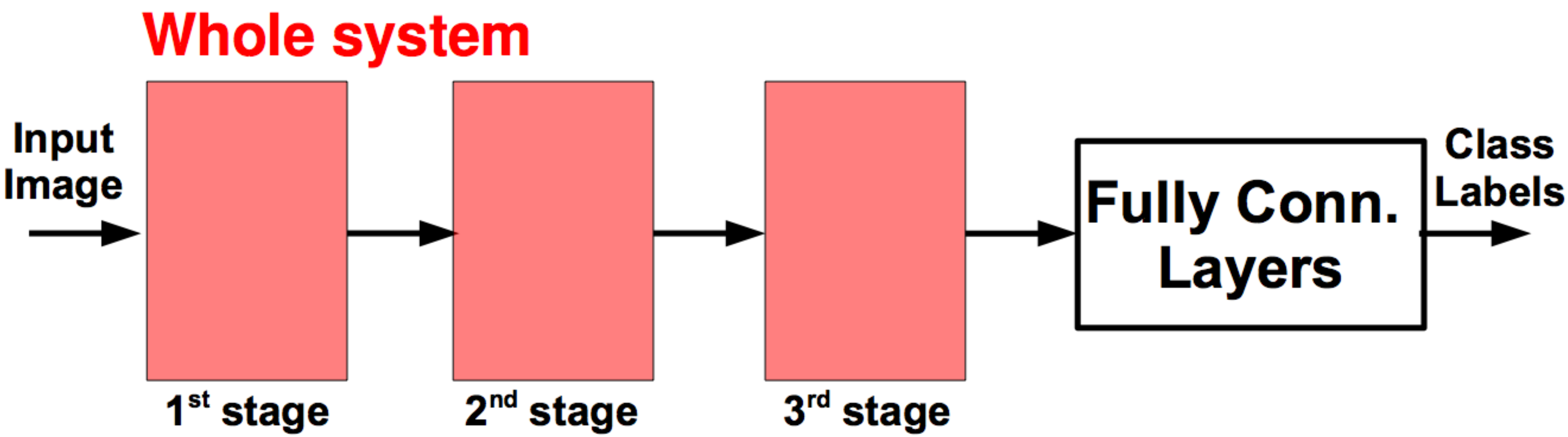
Reasons:
- gain invariance to spatial translation (pooling layer)
- increase specificity of features (approaching object specific units)



courtesy of K. Kavukcuoglu

Ranzato

ConvNets: Typical Architecture



Conceptually similar to:

SIFT → K-Means → Pyramid Pooling → SVM

Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012

Slide: M-A Ranzatto

Ranzato