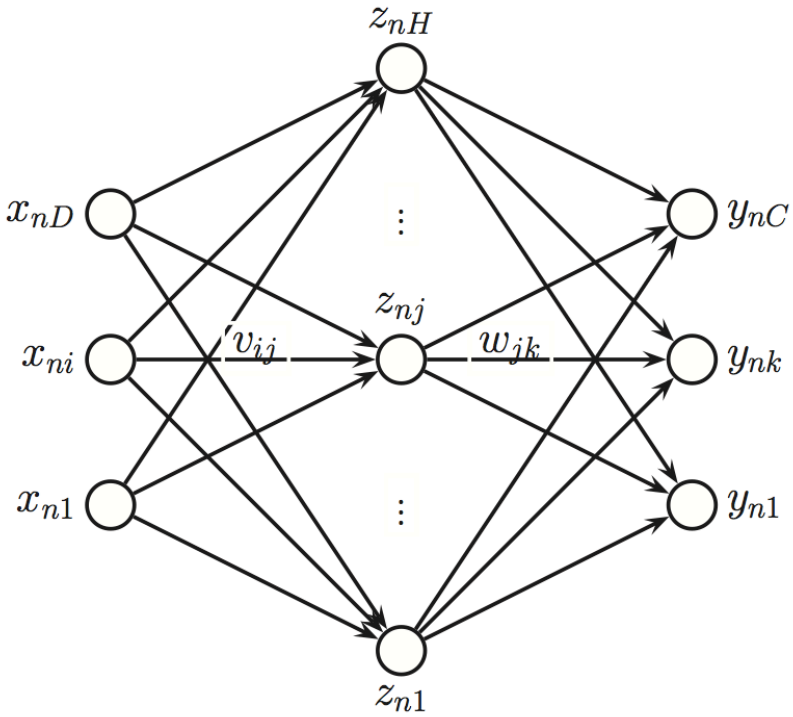
Introduction to Supervised Learning



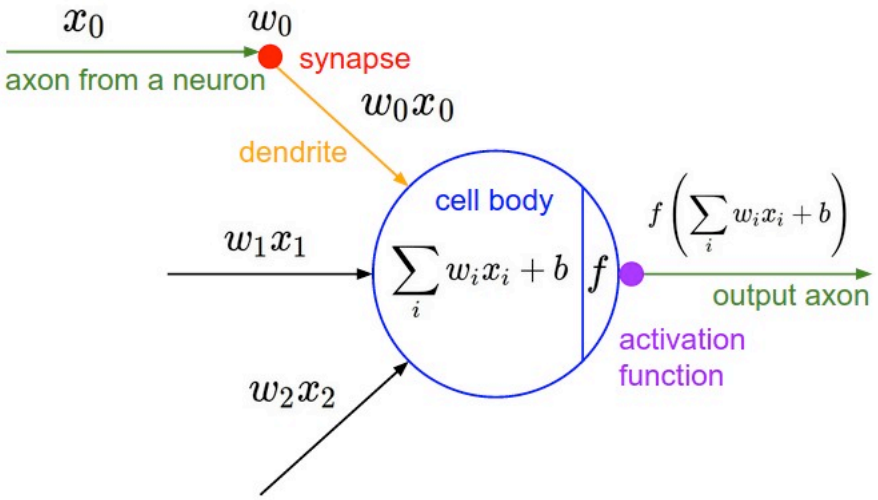
Week 8: Learning with Neural Networks

Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

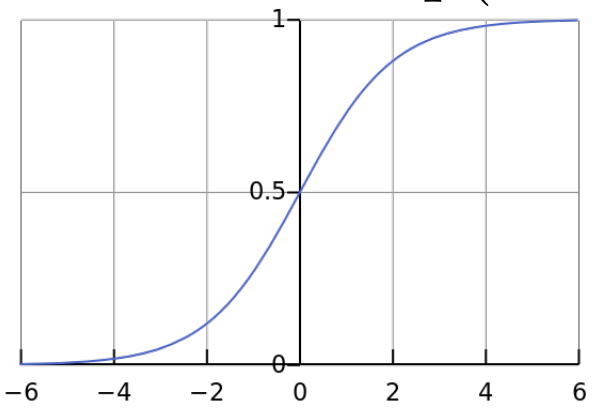
University College London

‘Neuron’: cascade of linear and nonlinear function



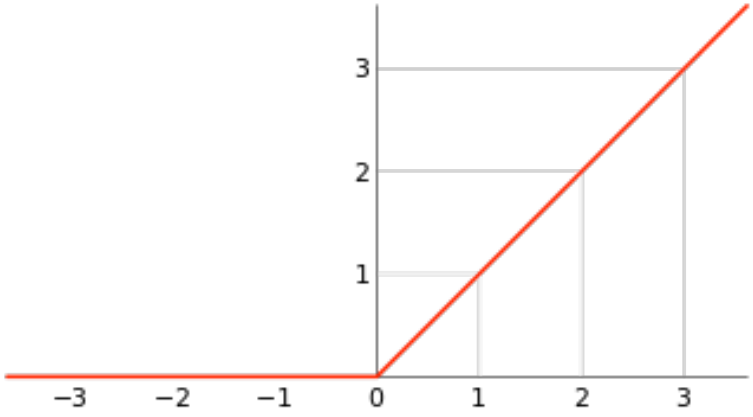
Sigmoidal (“logistic”)

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



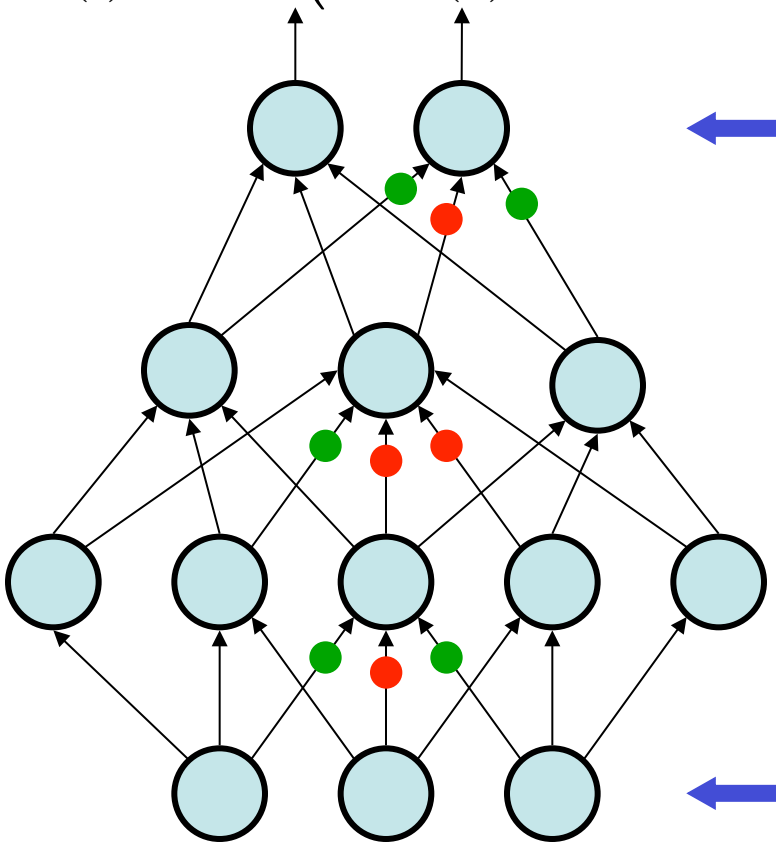
Rectified Linear Unit (RELU)

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



Multi-Layer Perceptrons (Deep Learning, 1985 A.D.)

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



outputs

hidden layers

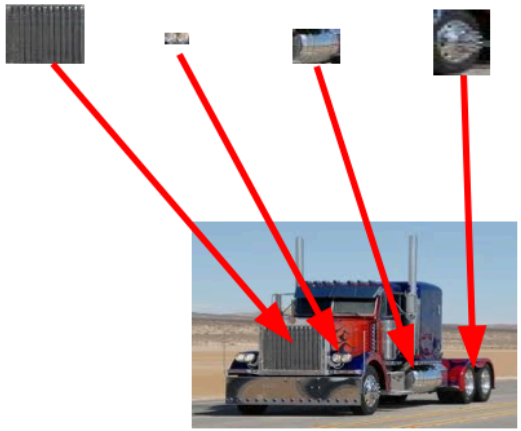
input vector

Hidden Layers: what do they do?

Intuition: learn “dictionary” for objects

“Distributed representation”:
represent (and classify) object classifier by mixing & mashing reusable parts

[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 …] truck feature



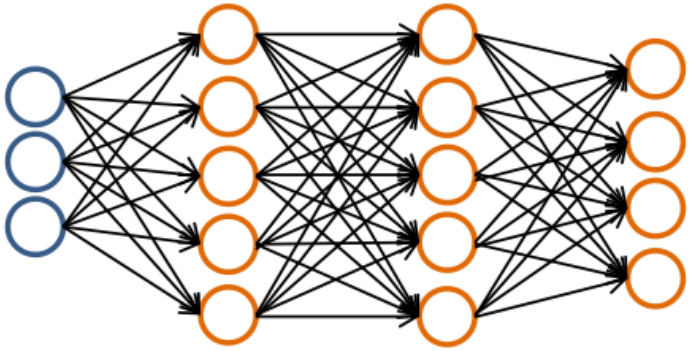
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(x) \in \mathbb{R}^4$$

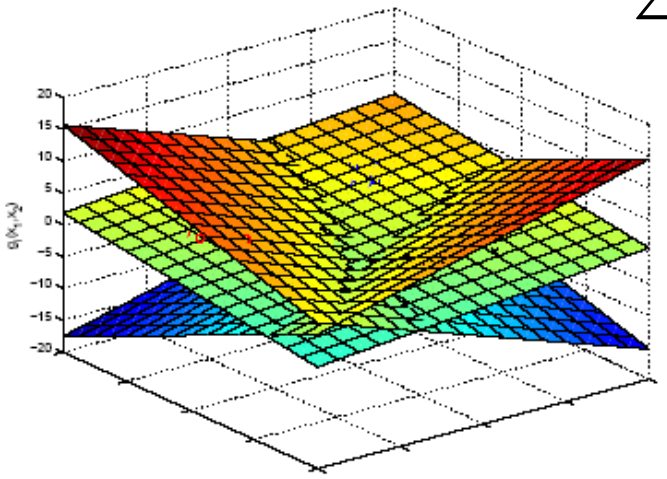
Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle

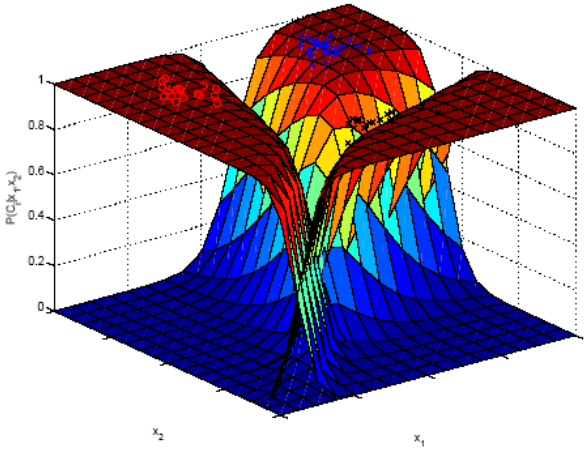
Reminder (W4): multiple classes & logistic regression

Soft maximum (softmax) of competing classes:

$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$



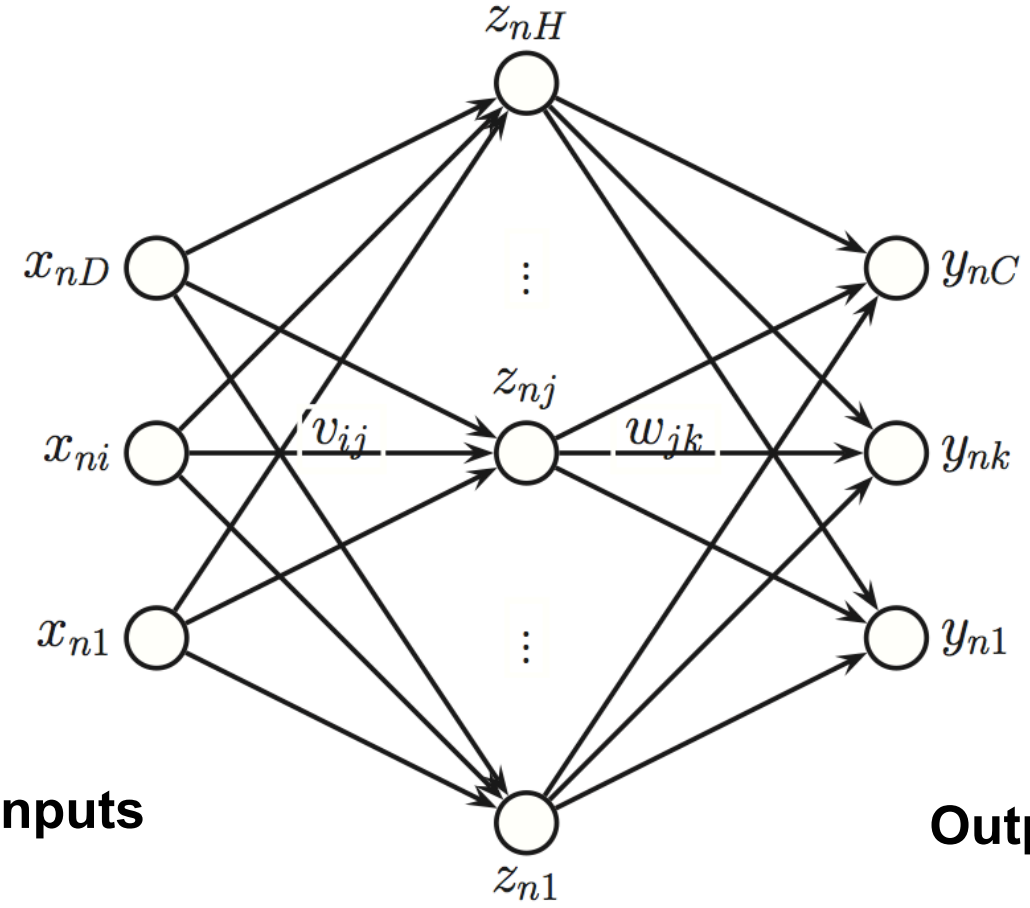
Discriminants (inputs)



Softmax (outputs)

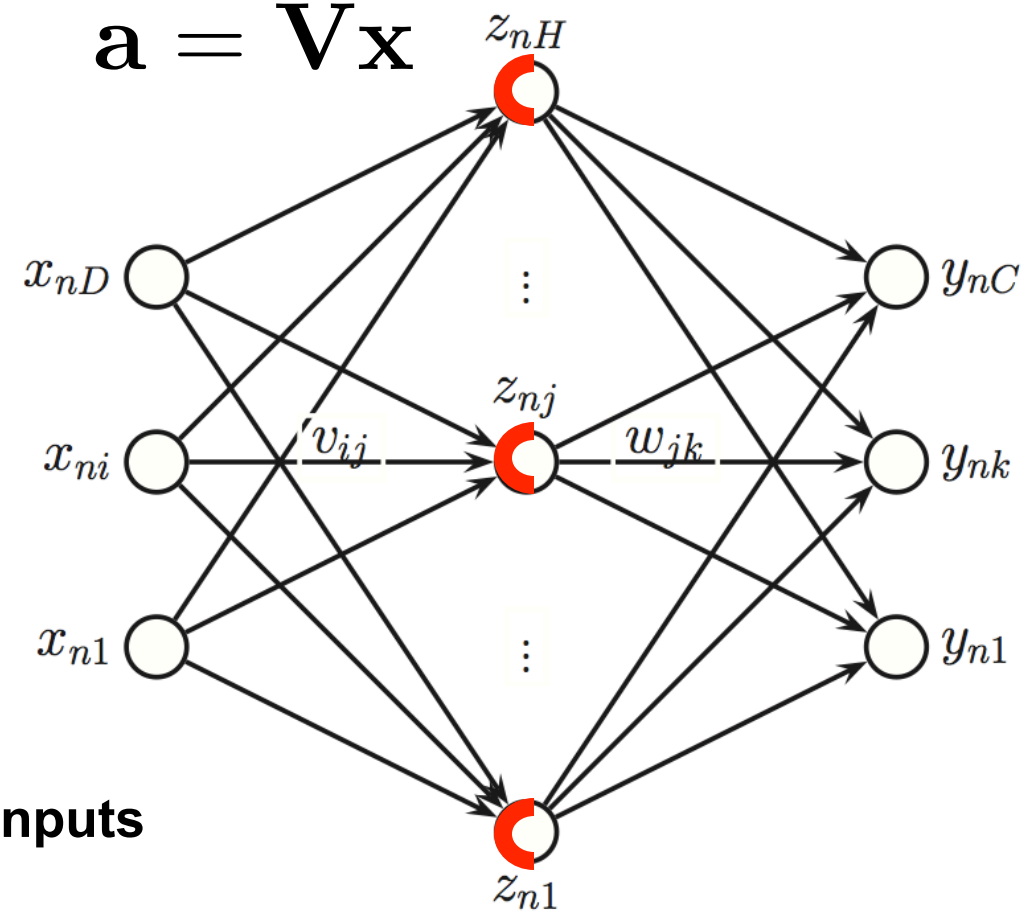
A neural network for multi-way classification

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



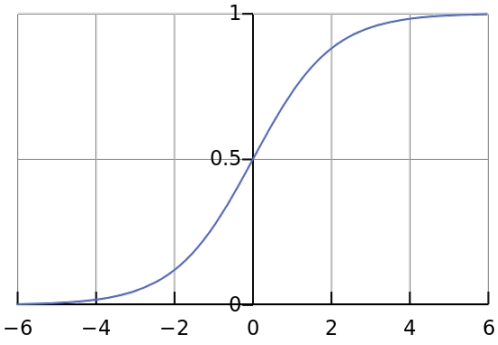
A neural network:

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

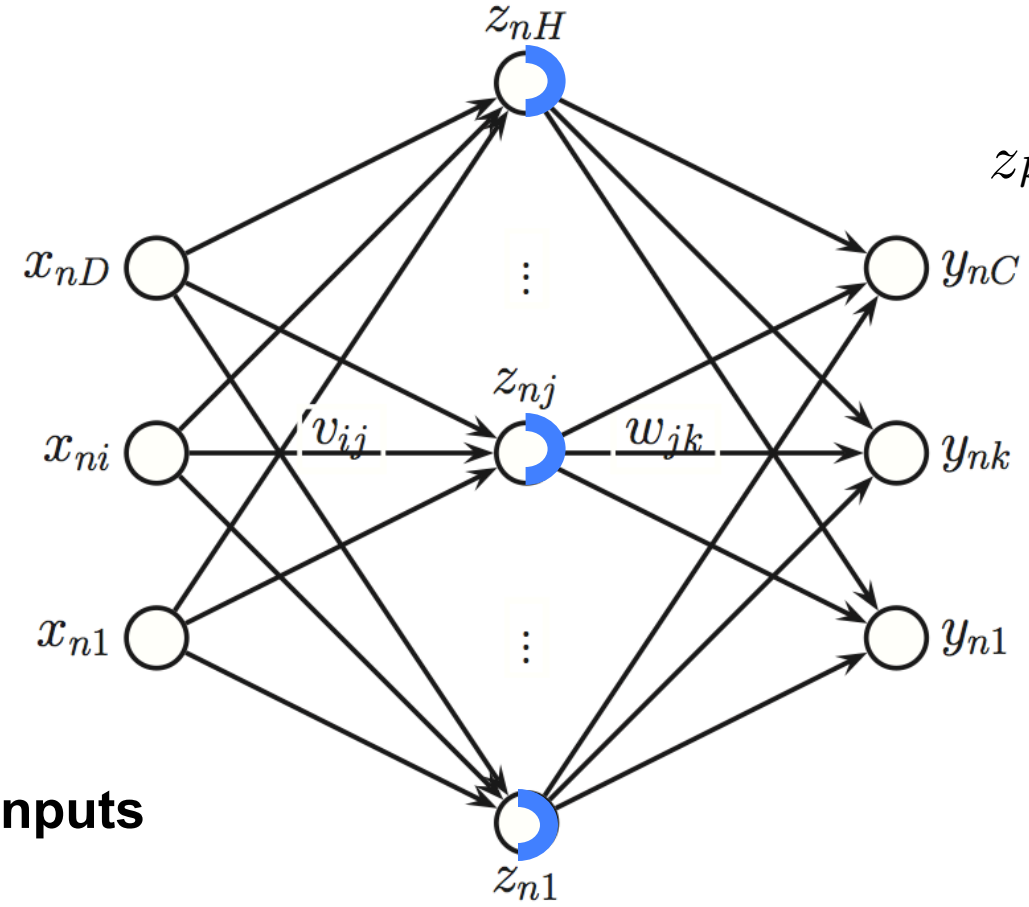


A neural network:

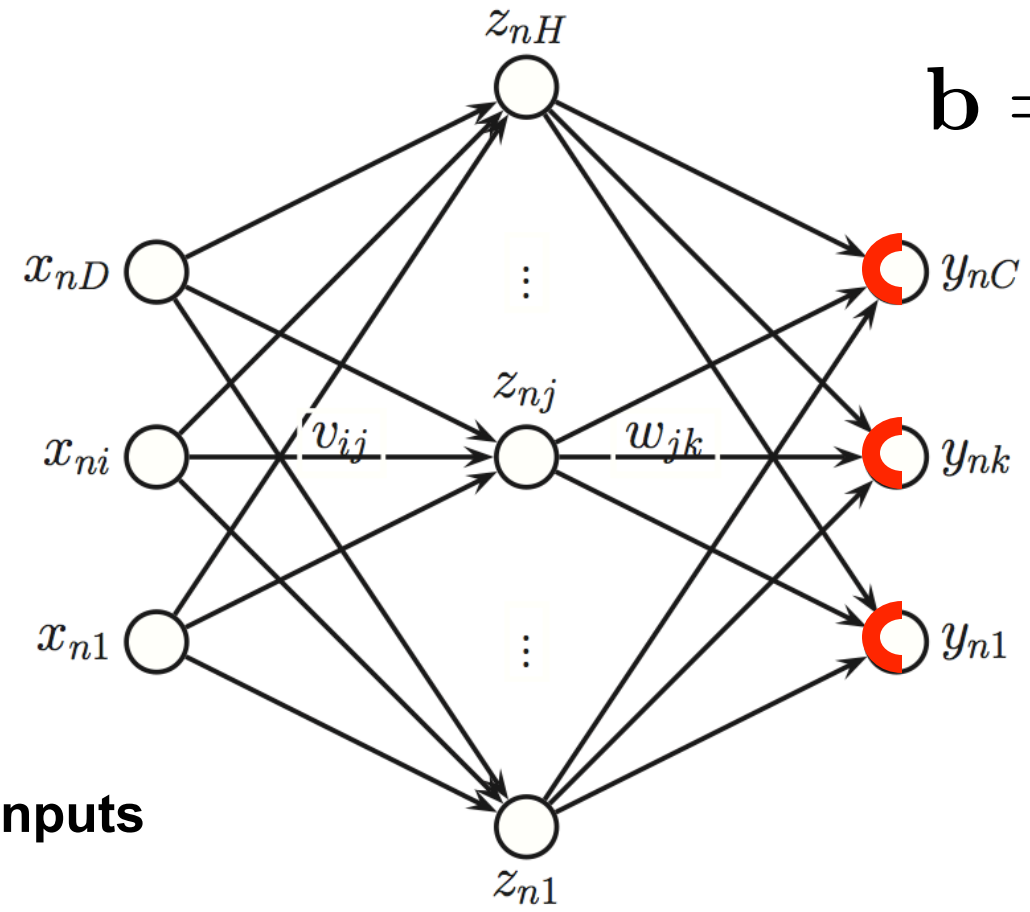
$$\mathbf{z} = g(\mathbf{a})$$



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

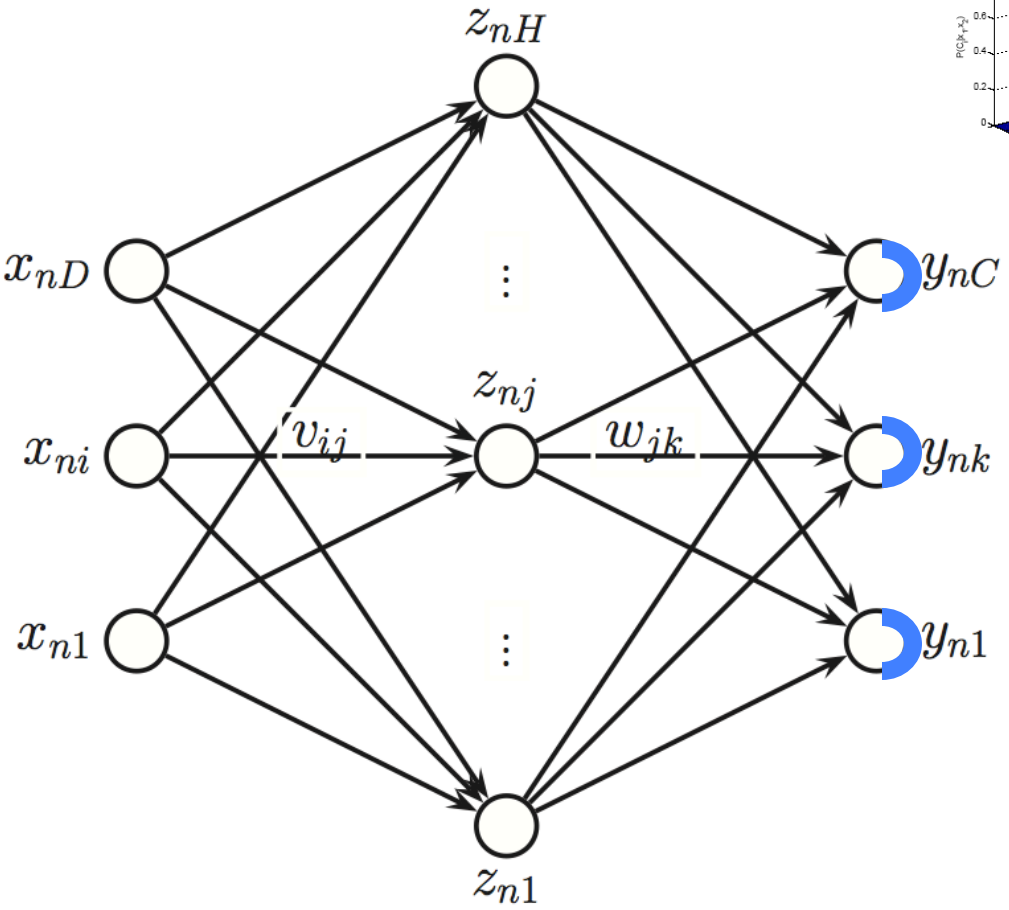


A neural network:

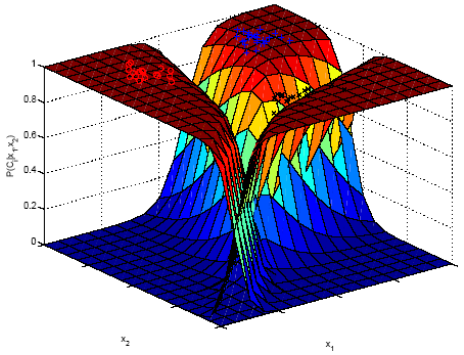


$$\mathbf{b} = \mathbf{W}\mathbf{z}$$

A neural network:



Hidden layer



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

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

Outputs

Training objective, multi-class case

One-hot label encoding: $\mathbf{y}^i = (0, 0, 1, 0)$

Likelihood of training sample: $(\mathbf{y}^i, \mathbf{x}^i)$

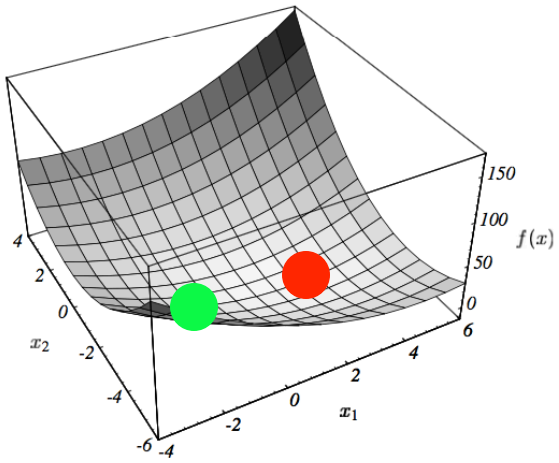
$$P(\mathbf{y}^i|\mathbf{x}^i; \mathbf{w}) = \prod_{i=1}^{N} \prod_{c=1}^{C} \left(g_c(\mathbf{x}, \mathbf{W})\right)^{\mathbf{y}_c^i}$$

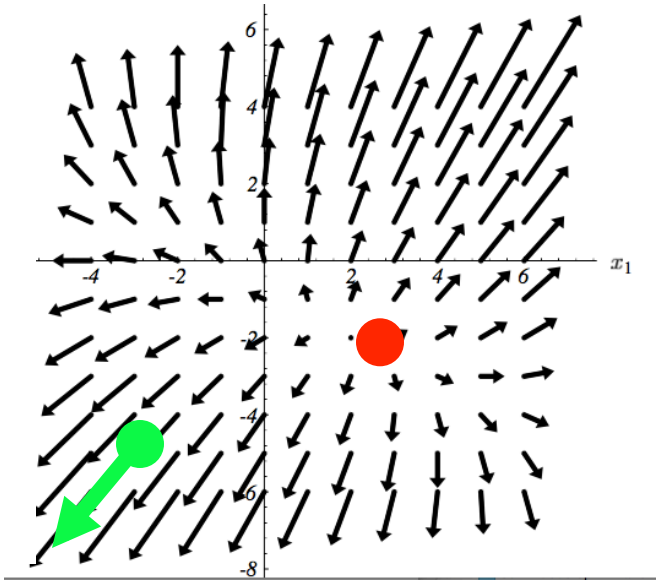
Optimization criterion:

$$L(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} \mathbf{y}_c^i \log \left(g_c(\mathbf{x}, \mathbf{W})\right)$$

Parameter estimation: Gradient of L with respect to **W**

Gradient-based minimization

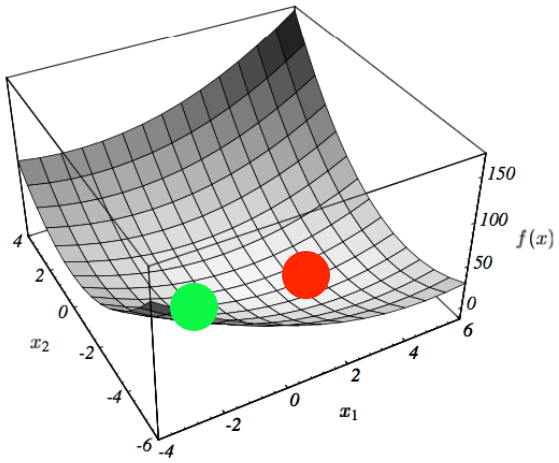


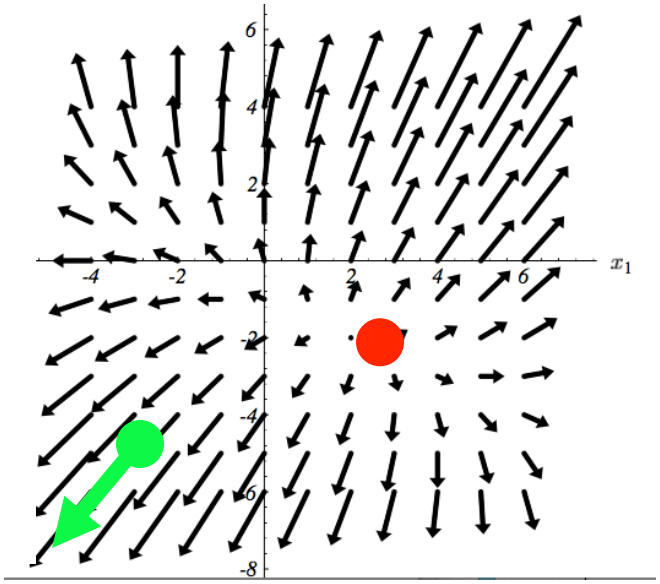


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Gradient-based minimization



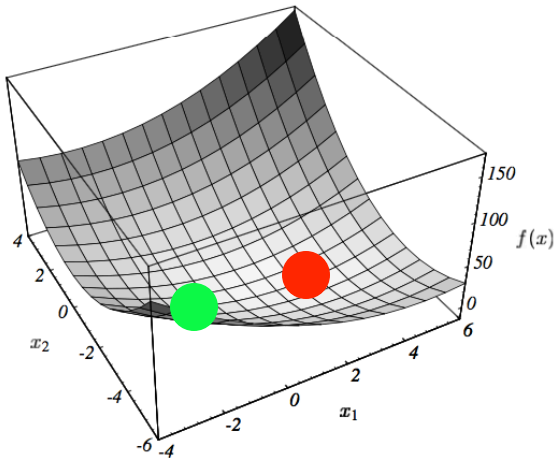


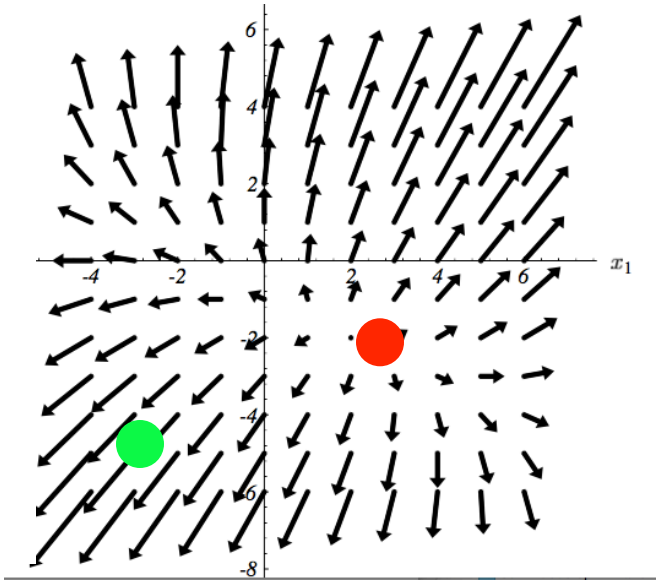
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient-based minimization



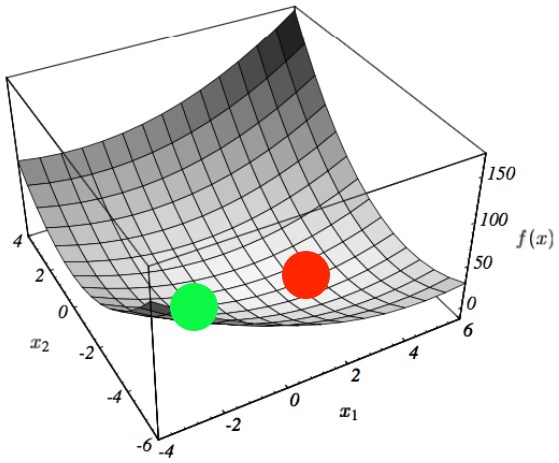


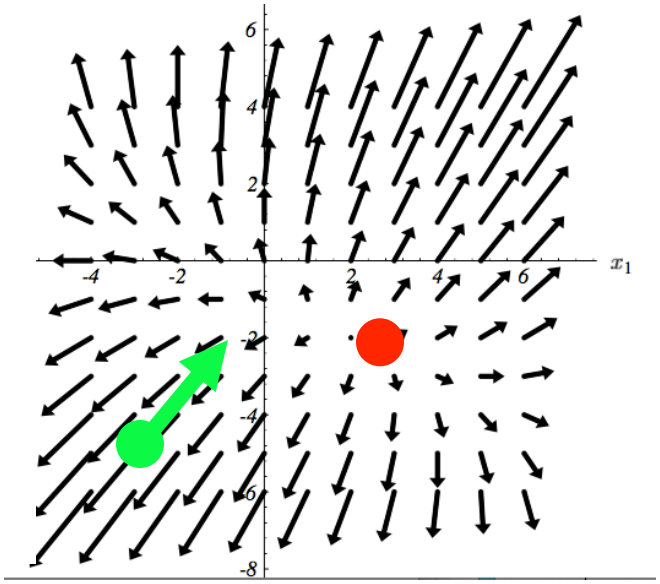
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient-based minimization



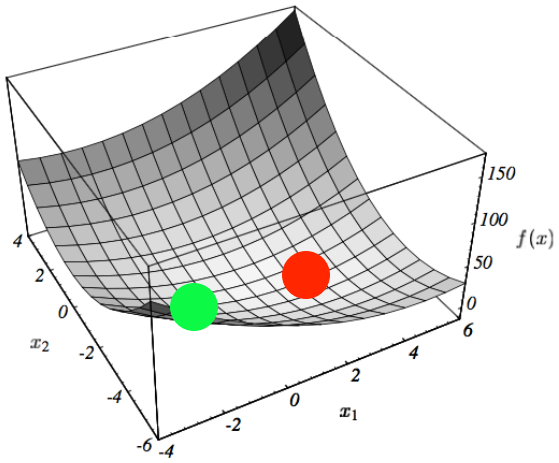


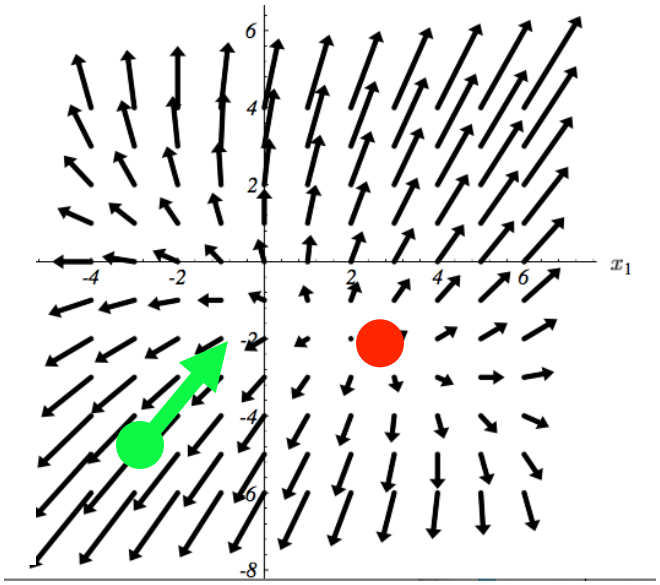
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient-based minimization





$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

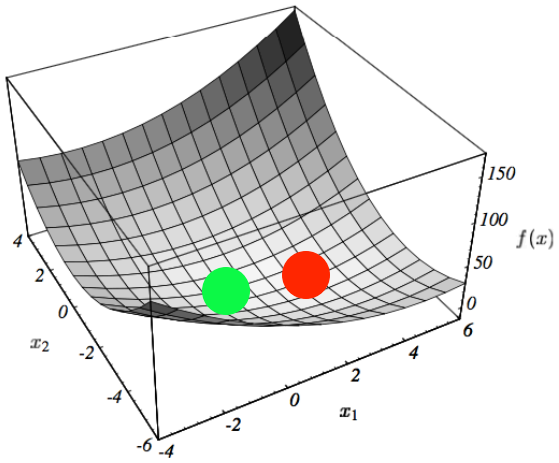
Fact: gradient at any point gives direction of fastest increase

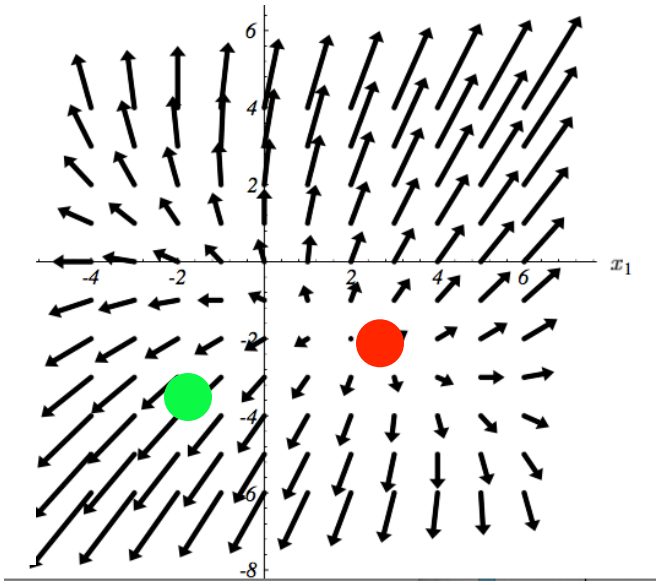
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=0**

Gradient-based minimization

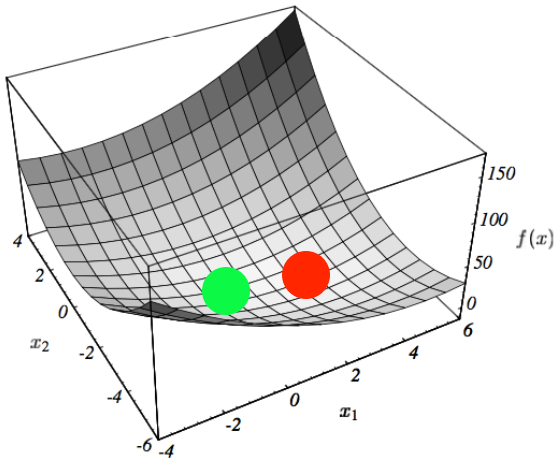


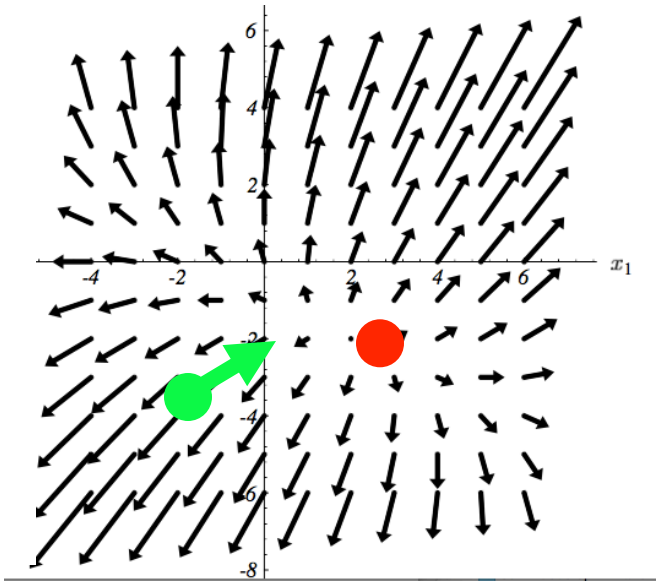


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ i=1

Gradient-based minimization

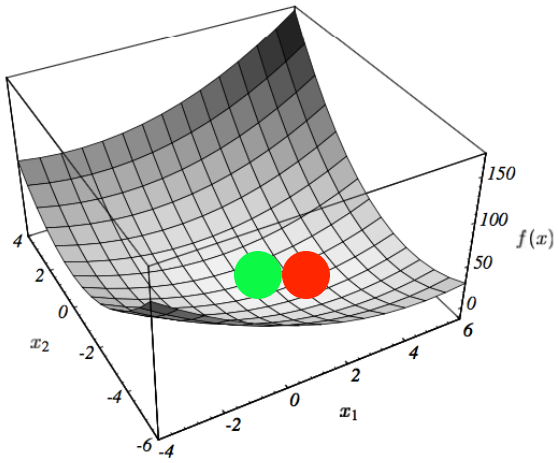


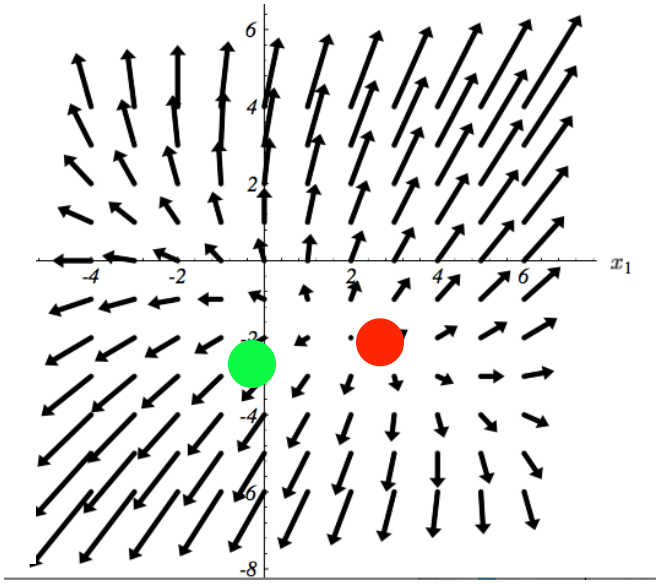


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=1**

Gradient-based minimization

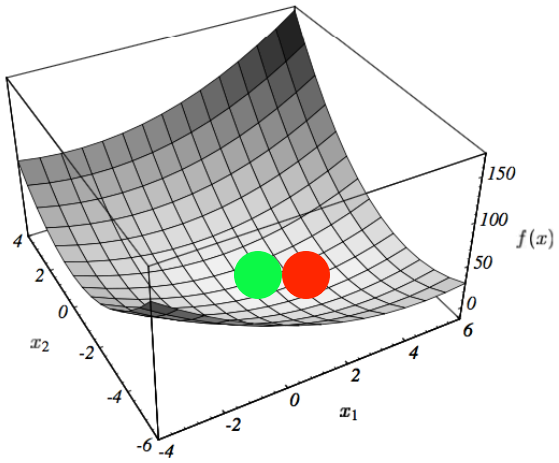


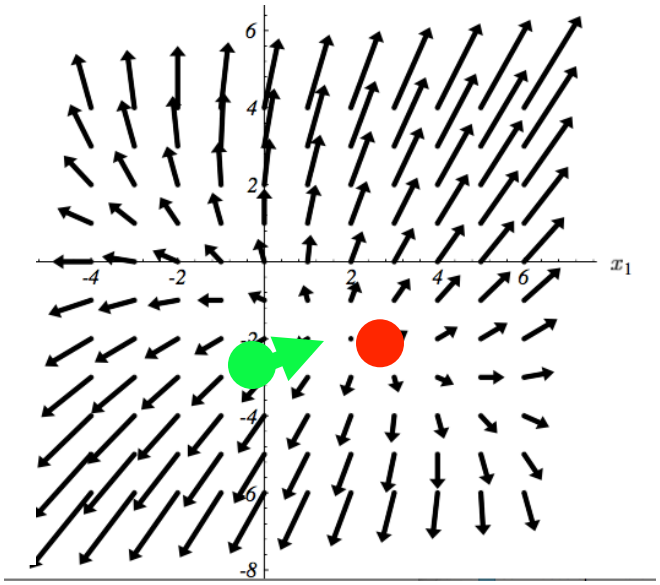


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=2**

Gradient-based minimization

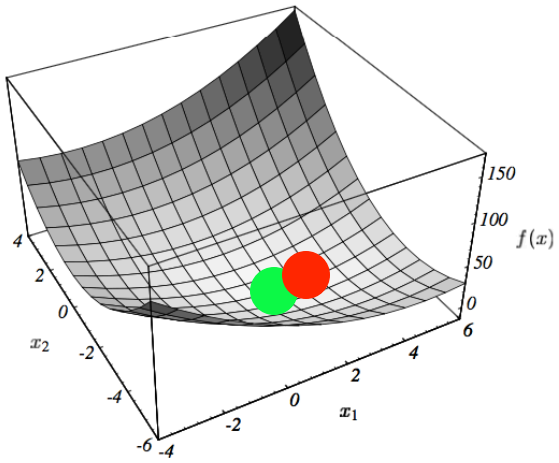


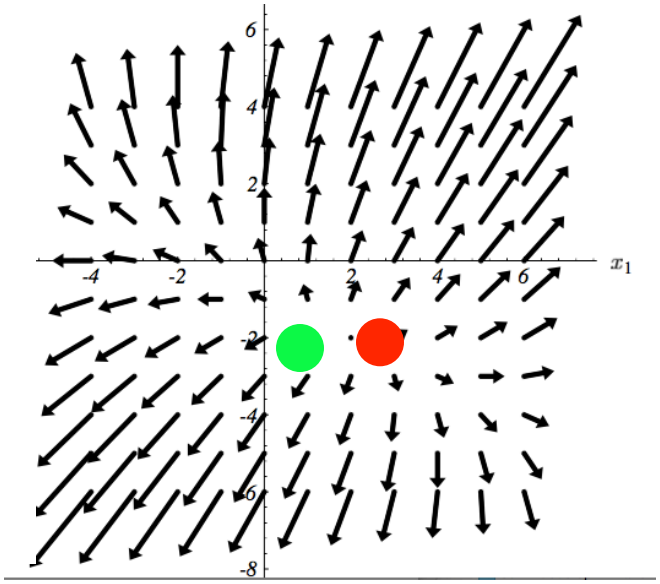


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=2**

Gradient-based minimization

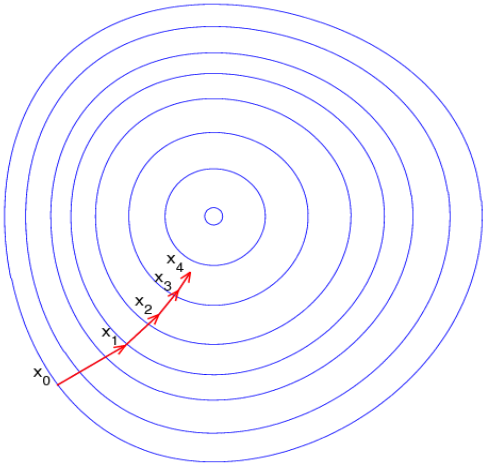




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=3**

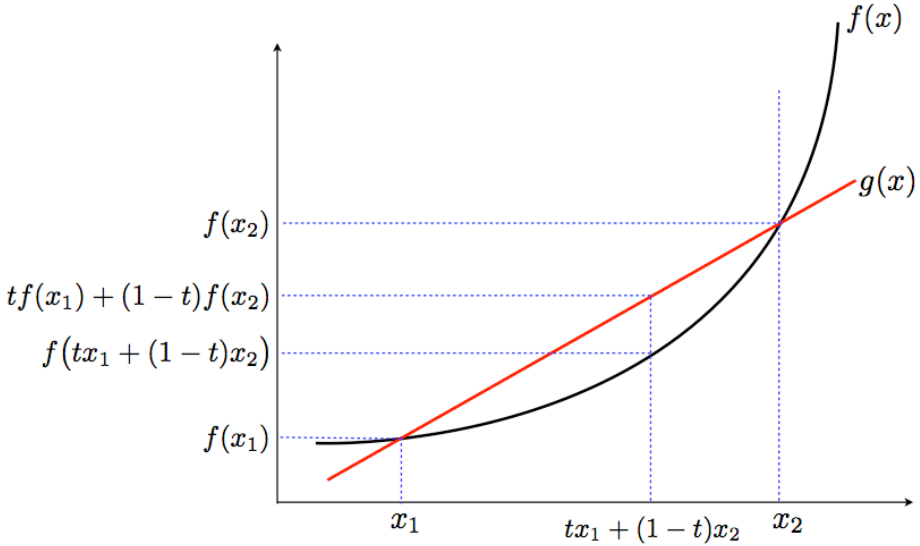
Gradient descent minimization method

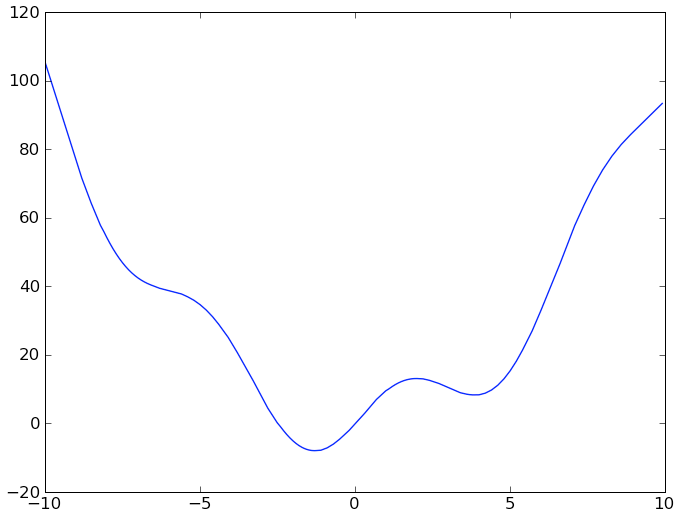


Initialize: $\mathbf{x}_0$

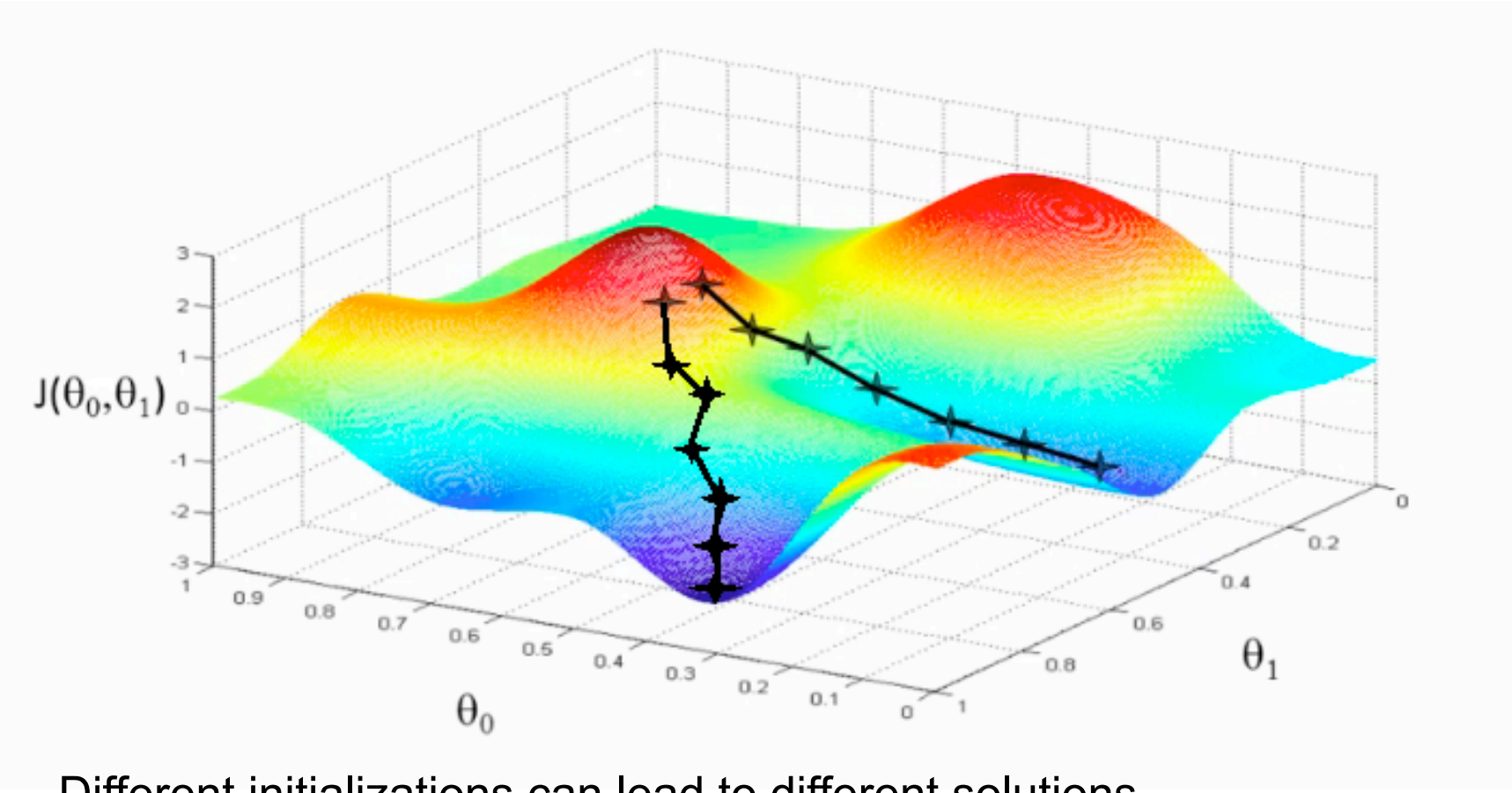
Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a **convex** function





Problems: multiple local minima



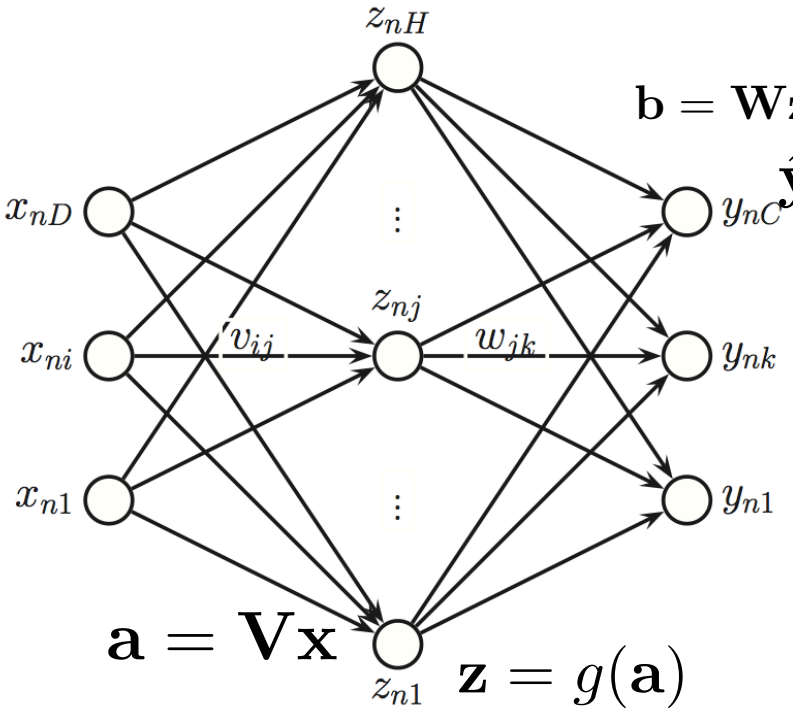
Different initializations can lead to different solutions

-Empirically all are almost equally good

-Empirically all are better than flat counterparts

On to the gradients!

Training a neural network



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

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

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

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

$$\mathcal{X} = \{(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^N, \mathbf{y}^N)\}$$

$$\hat{\mathbf{y}}_i = h(\mathbf{W}g(\mathbf{V}\mathbf{x}_i))$$

$$L(\mathcal{X}; \mathbf{W}, \mathbf{V}) = \sum_{i=1}^{N} l(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

$$\begin{aligned} l(\mathbf{y}_i, \hat{\mathbf{y}}_i) &= \log P(\mathbf{y}_i | \hat{\mathbf{y}}_i) \\ &= \log \prod_{c=1}^{C} {y_{i,c}}^{\hat{y}_{i,c}} \\ &= \sum_{c=1}^{C} y_{i,c} \log \hat{y}_{i,c} \end{aligned}$$

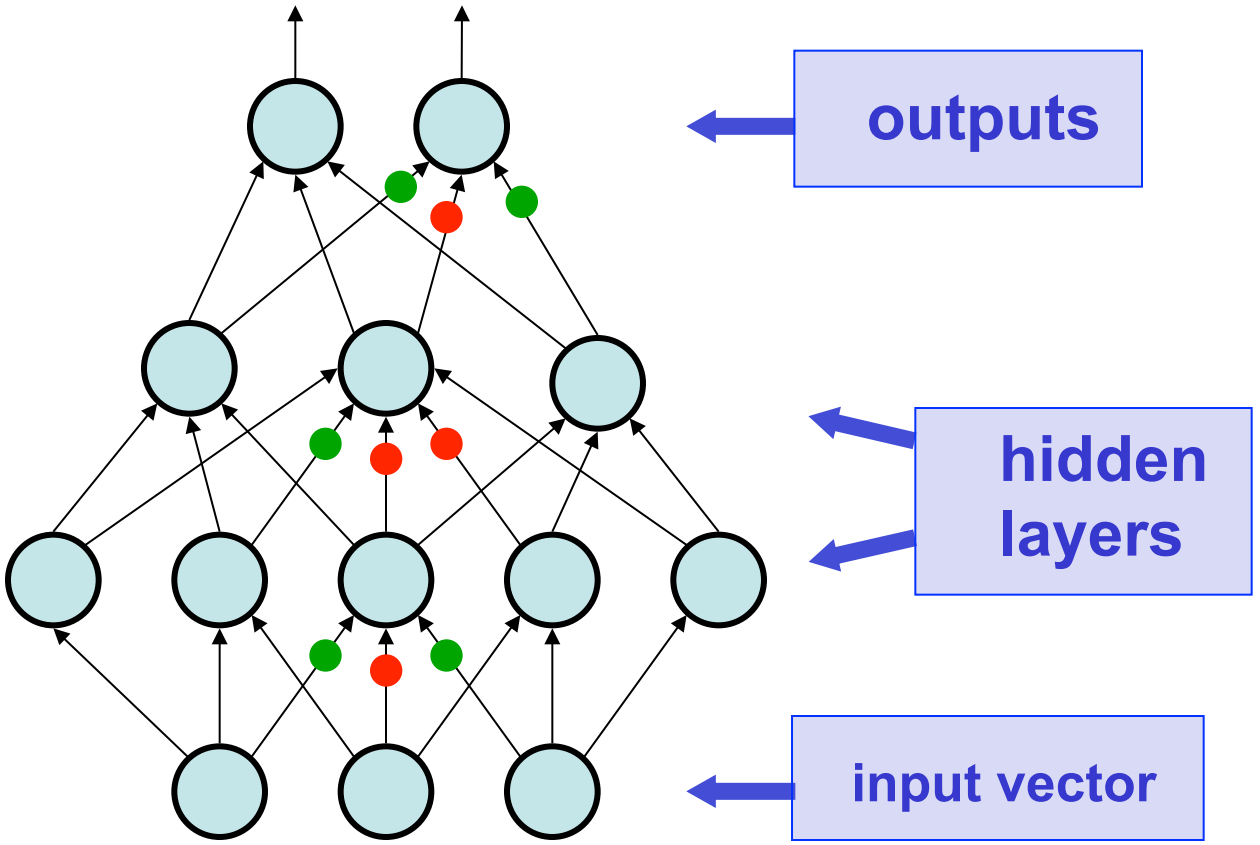
What is the gradient of the loss?

$$\frac{\partial L(\mathcal{X}; \mathbf{W}, \mathbf{V})}{\partial \mathbf{W}_{i,j}} = ? \qquad \frac{\partial L(\mathcal{X}; \mathbf{W}, \mathbf{V})}{\partial \mathbf{V}_{i,j}} = ?$$

Back-propagation algorithm

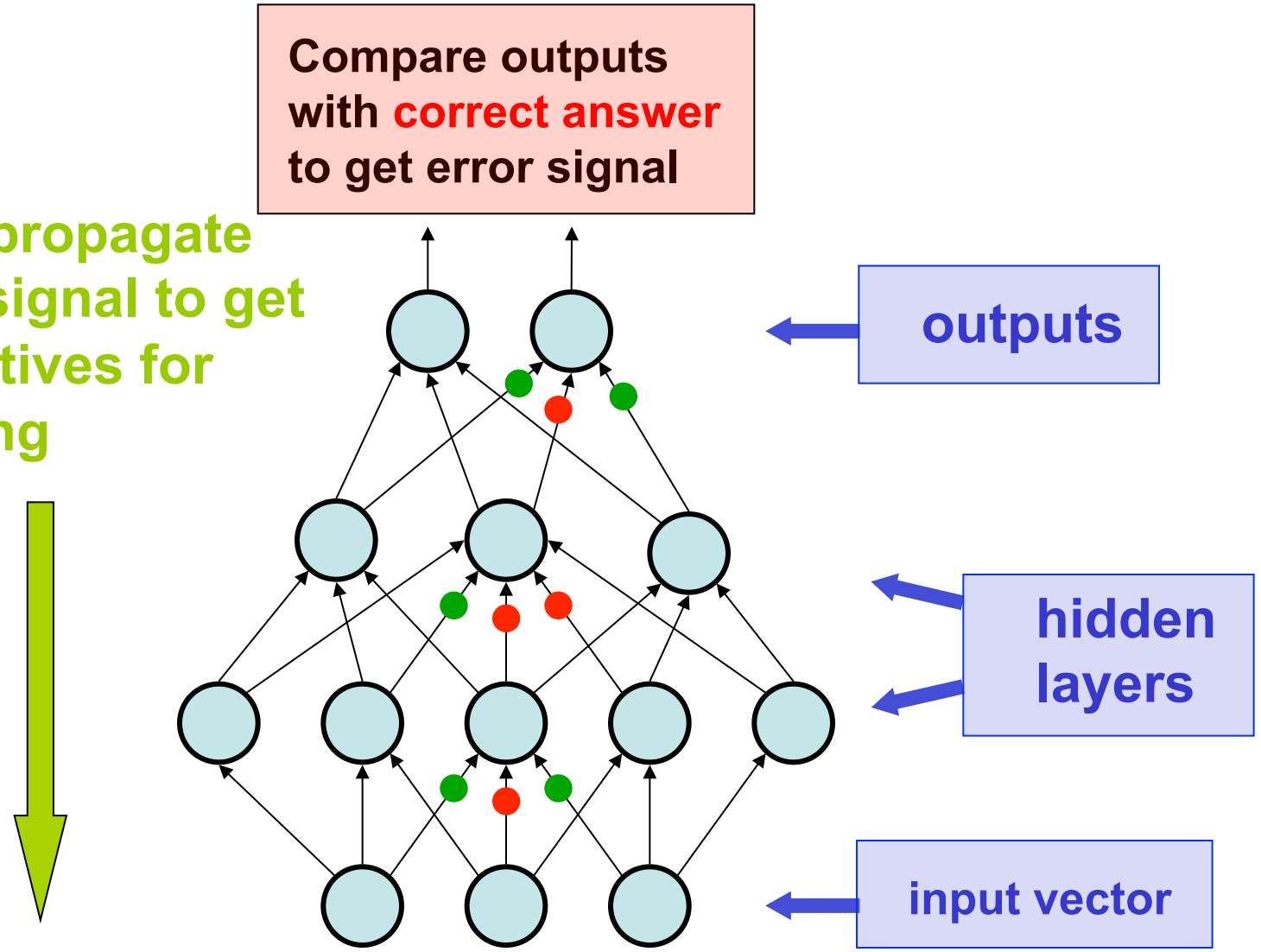
Multi-Layer Perceptrons

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



Slide credits: G. Hinton

Multi-Layer Perceptrons (~1985)



Slide credits: G. Hinton

Chain rule

$$x \xrightarrow{g} u \xrightarrow{f} y$$

y is affected by x through intermediate quantity, u:

$$u = g(x) \quad y = f(u)$$

Calculus: $(f(g(x)))' = f'(g(x))g'(x)$

Rewrite:

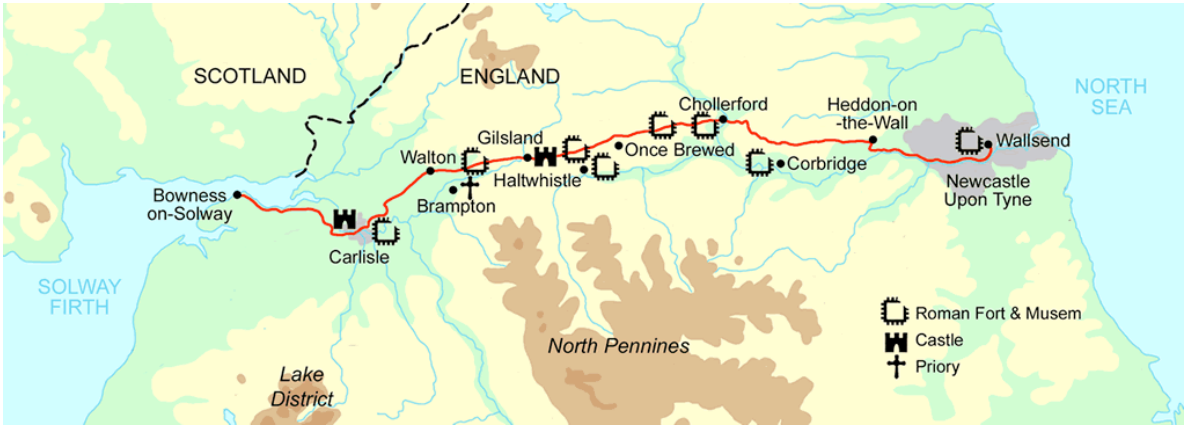
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Chain rule of differentiation– multiple variables

x(t),y(t) coordinates: given by GPS

z=f(x,y) given by map

Q: what is your speed in the vertical direction?



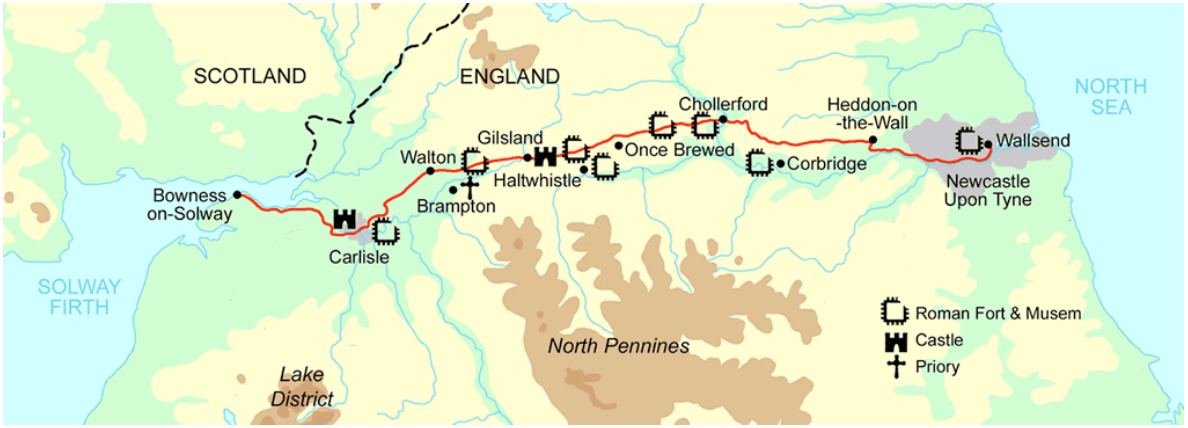
https://www.math.hmc.edu/calculus/tutorials/multichainrule/

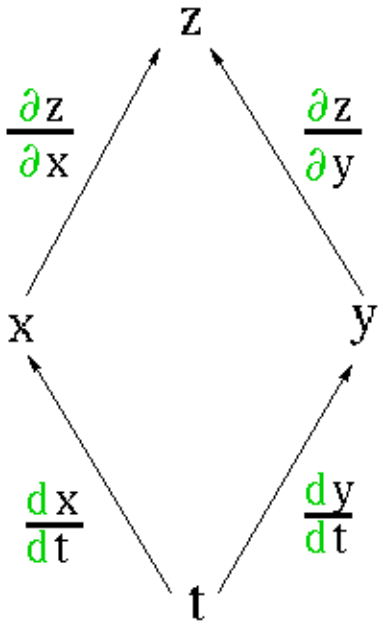
Chain rule of differentiation– multiple variables

x(t),y(t) coordinates: given by GPS

z=f(x,y) given by map

Q: what is your speed in the vertical direction?



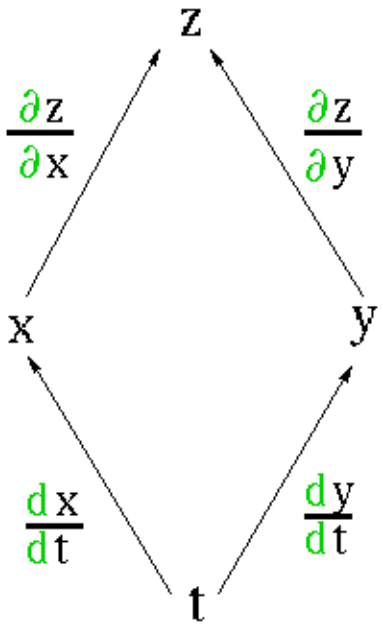


https://www.math.hmc.edu/calculus/tutorials/multichainrule/

Chain rule of differentiation– multiple variables

Let $x = x(t)$ and $y = y(t)$ be differentiable at t and suppose that $z = f(x, y)$ is differentiable at $(x(t), y(t))$. Then $z = f(x(t), y(t))$ is differentiable at t and

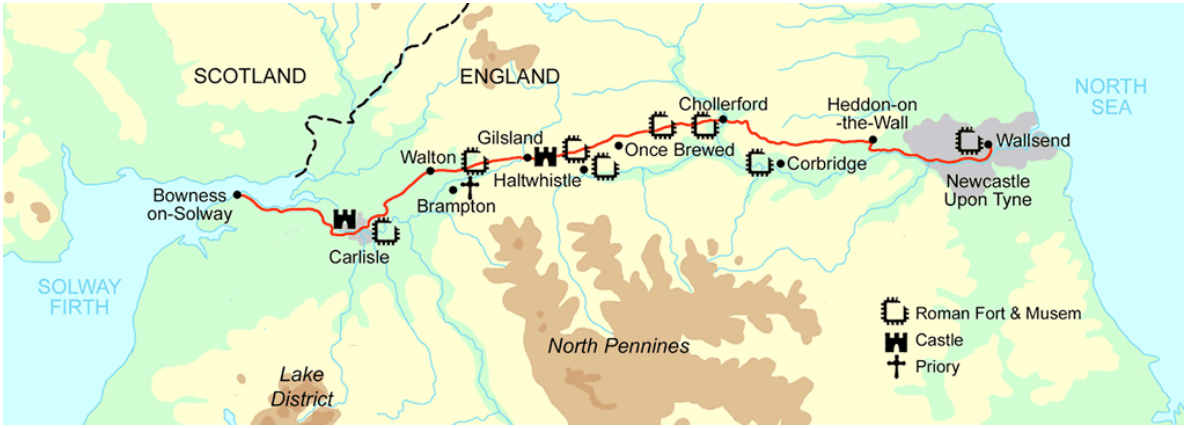
$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}.$$



x(t),y(t) coordinates: given by GPS

z=f(x,y) given by map

Q: what is your speed in the vertical direction?

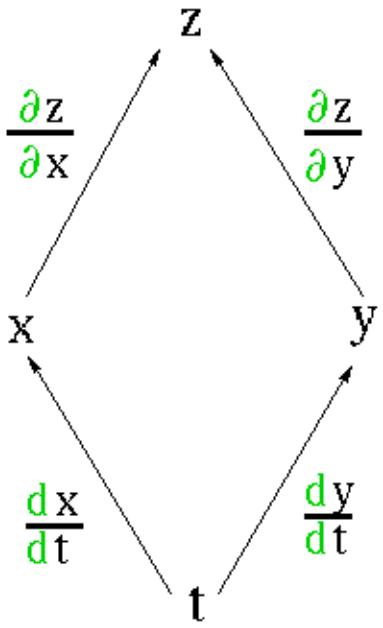


https://www.math.hmc.edu/calculus/tutorials/multichainrule/

Chain rule of differentiation– multiple variables

Let $x = x(t)$ and $y = y(t)$ be differentiable at t and suppose that $z = f(x, y)$ is differentiable at $(x(t), y(t))$. Then $z = f(x(t), y(t))$ is differentiable at t and

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}.$$



Let $z = x^2y - y^2$ where $x = t^2$ and $y = 2t$. Then

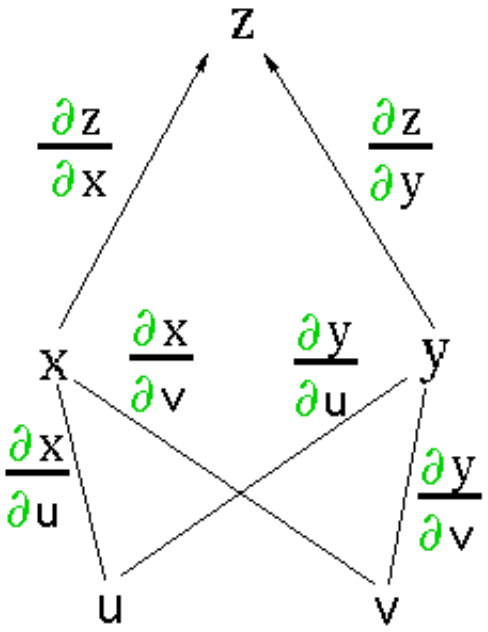
$$\begin{aligned}
\frac{dz}{dt} &= \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \\
&= (2xy)(2t) + (x^2 - 2y)(2) \\
&= (2t^2 \cdot 2t)(2t) + \left((t^2)^2 - 2(2t)\right)(2) \\
&= 8t^4 + 2t^4 - 8t \\
&= 10t^4 - 8t.
\end{aligned}$$

https://www.math.hmc.edu/calculus/tutorials/multichainrule/

Chain rule of derivative – multiple variables

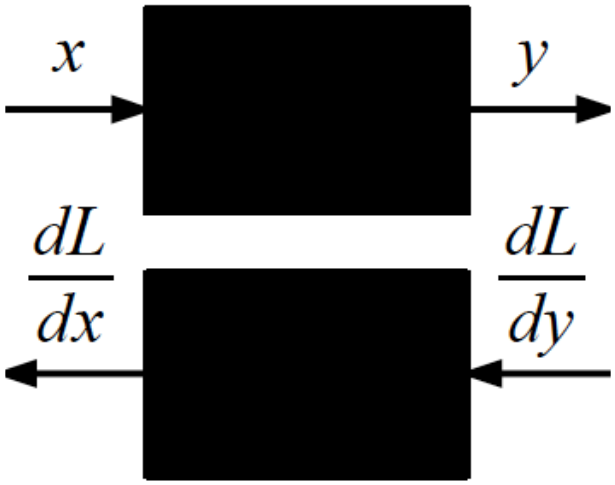
Let $x = x(u,v)$ and $y = y(u,v)$ have first-order partial derivatives at the point (u,v) and suppose that $z = f(x,y)$ is differentiable at the point $(x(u,v), y(u,v))$. Then $f(x(u,v), y(u,v))$ has first-order partial derivatives at (u,v) given by

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$
$$\frac{\partial z}{\partial v} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}.$$



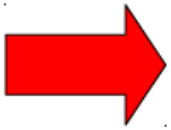
https://www.math.hmc.edu/calculus/tutorials/multichainrule/

Chain Rule

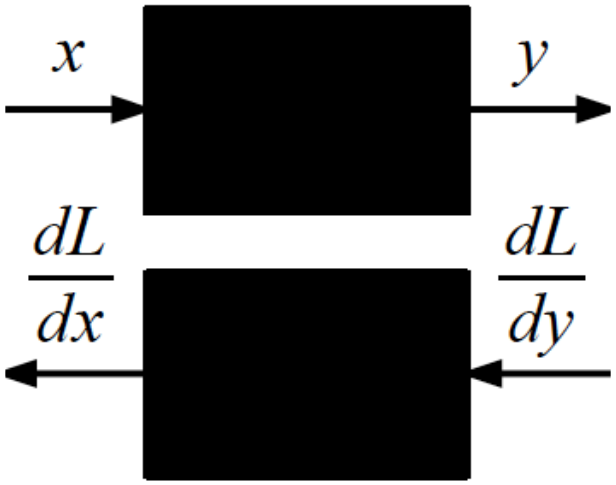


Given $y(x)$ and dL/dy,

What is dL/dx ?



Chain Rule

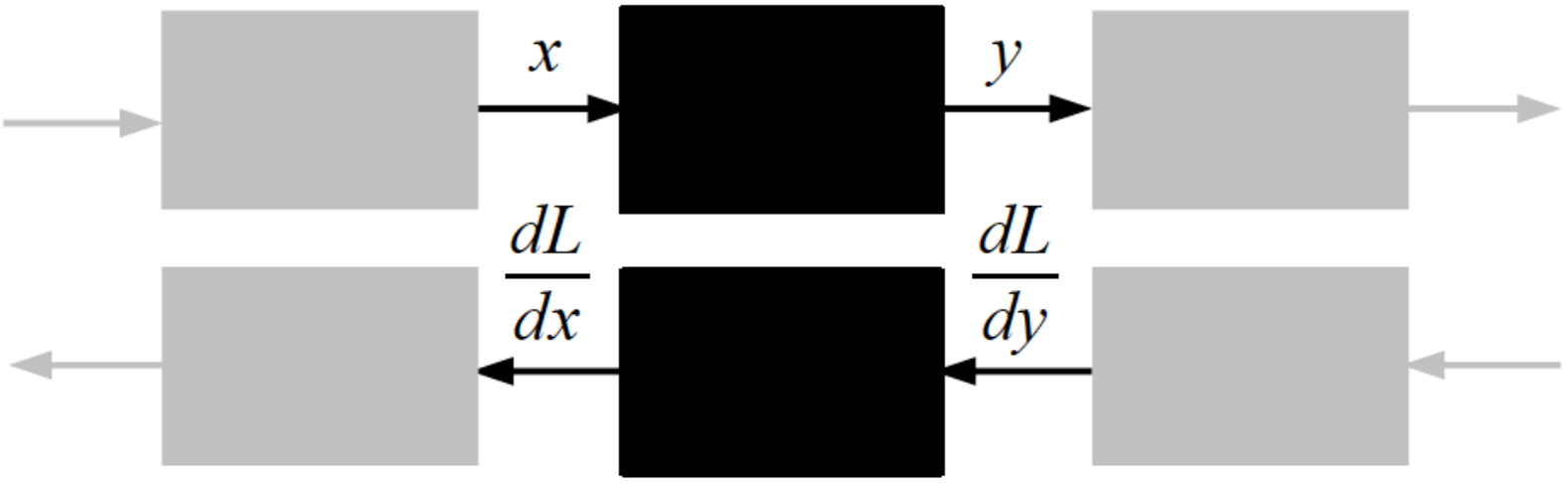


Given $y(x)$ and dL/dy,

What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

‘another brick in the wall’



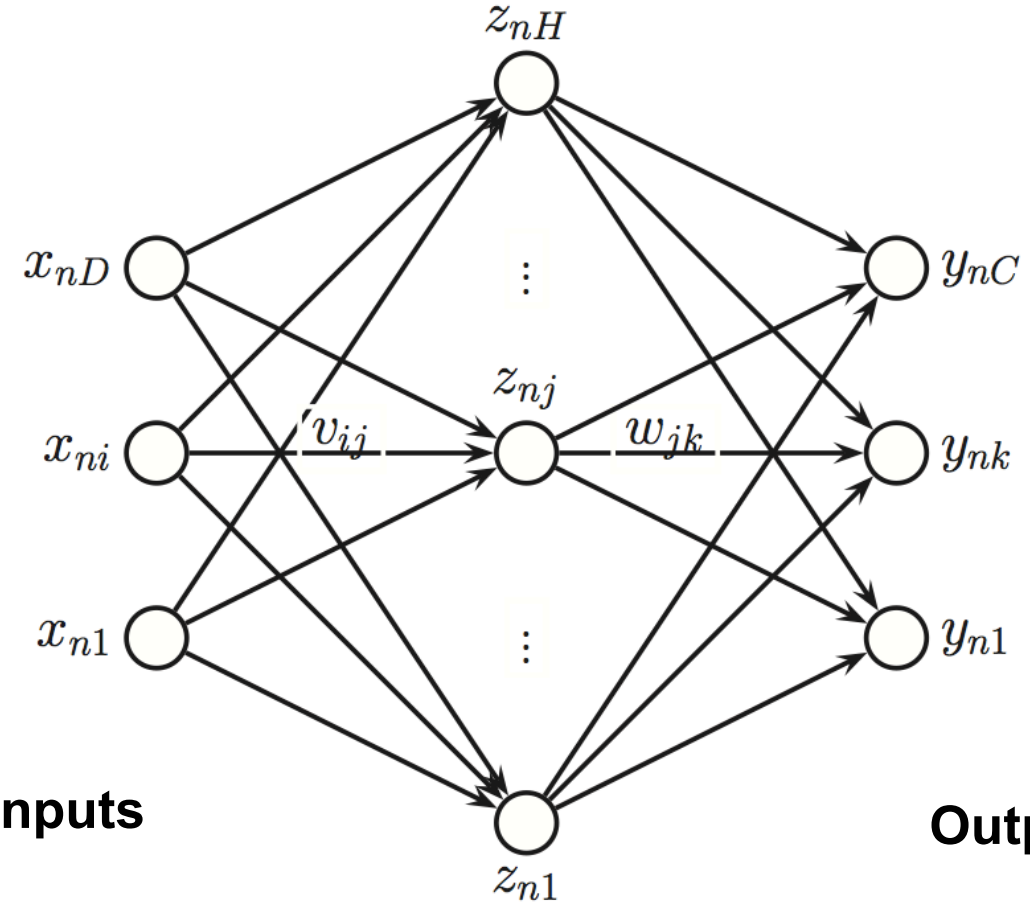
Given $y(x)$ and dL/dy,

What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

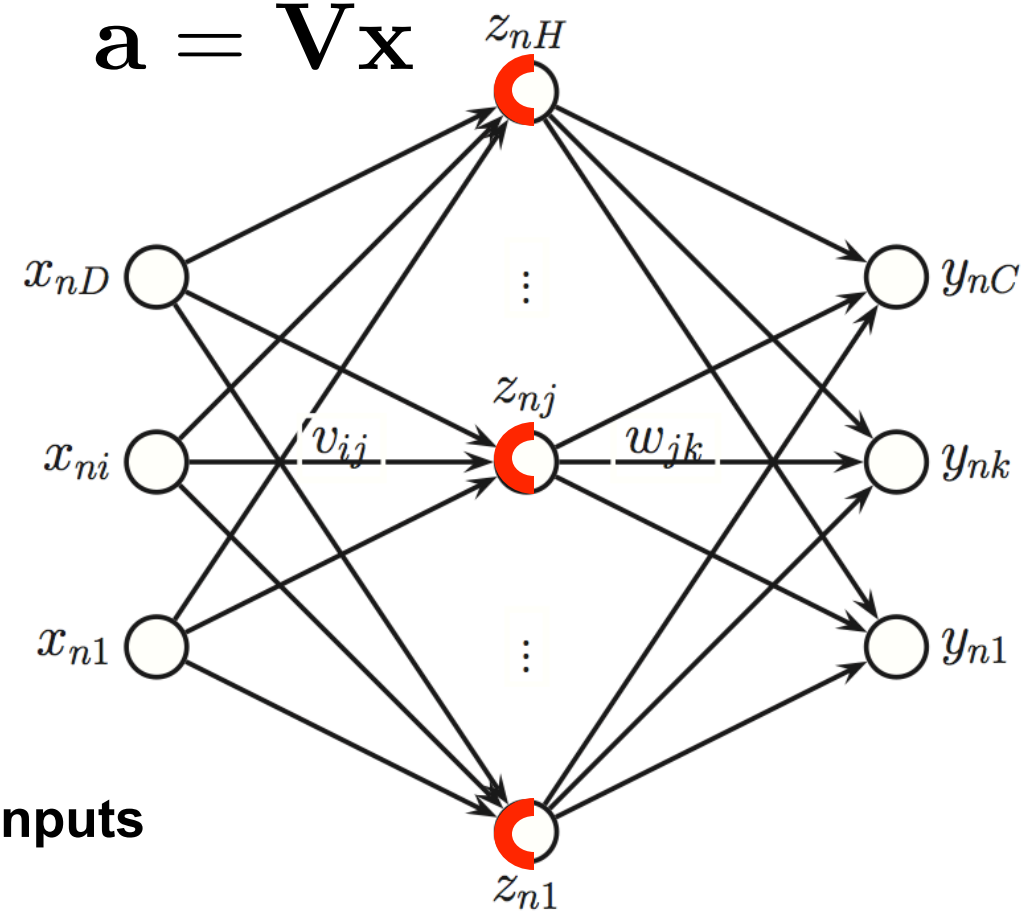
A neural network for multi-way classification

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



A neural network in forward mode:

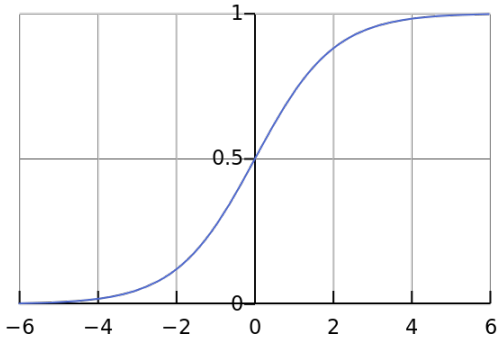
$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

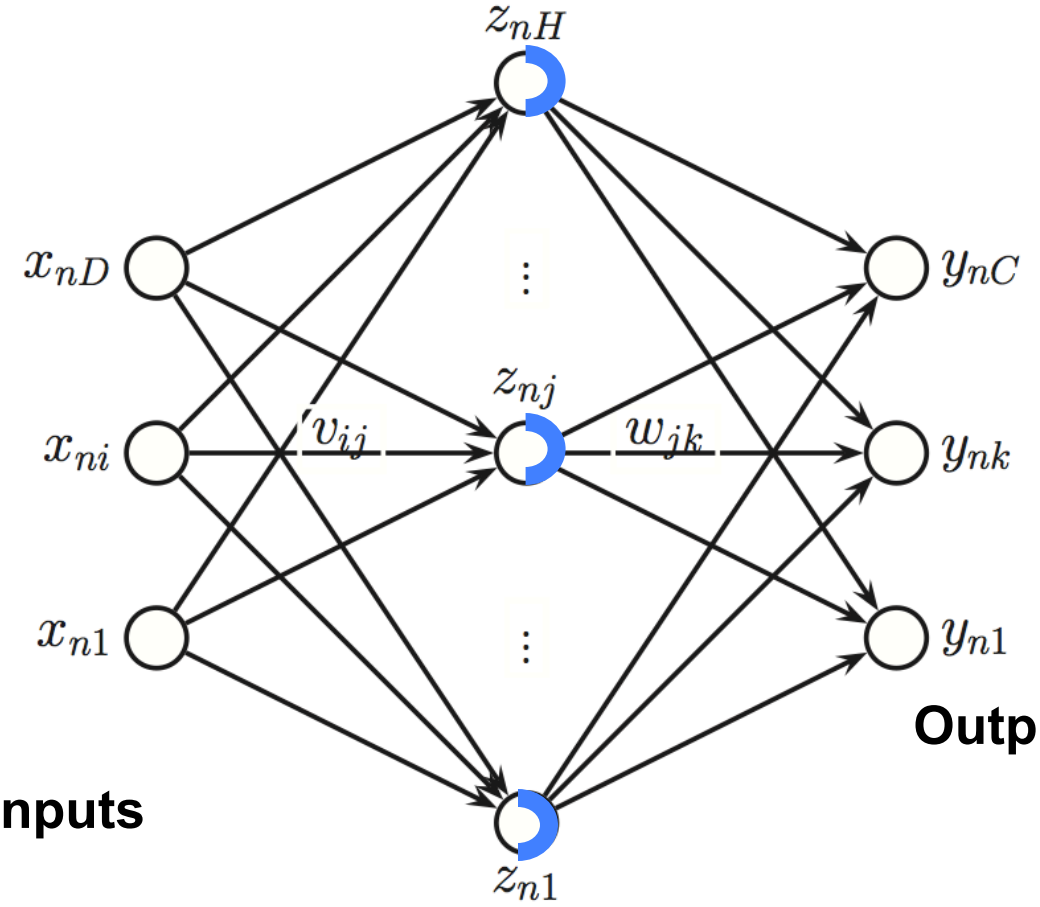


A neural network in forward mode:

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

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



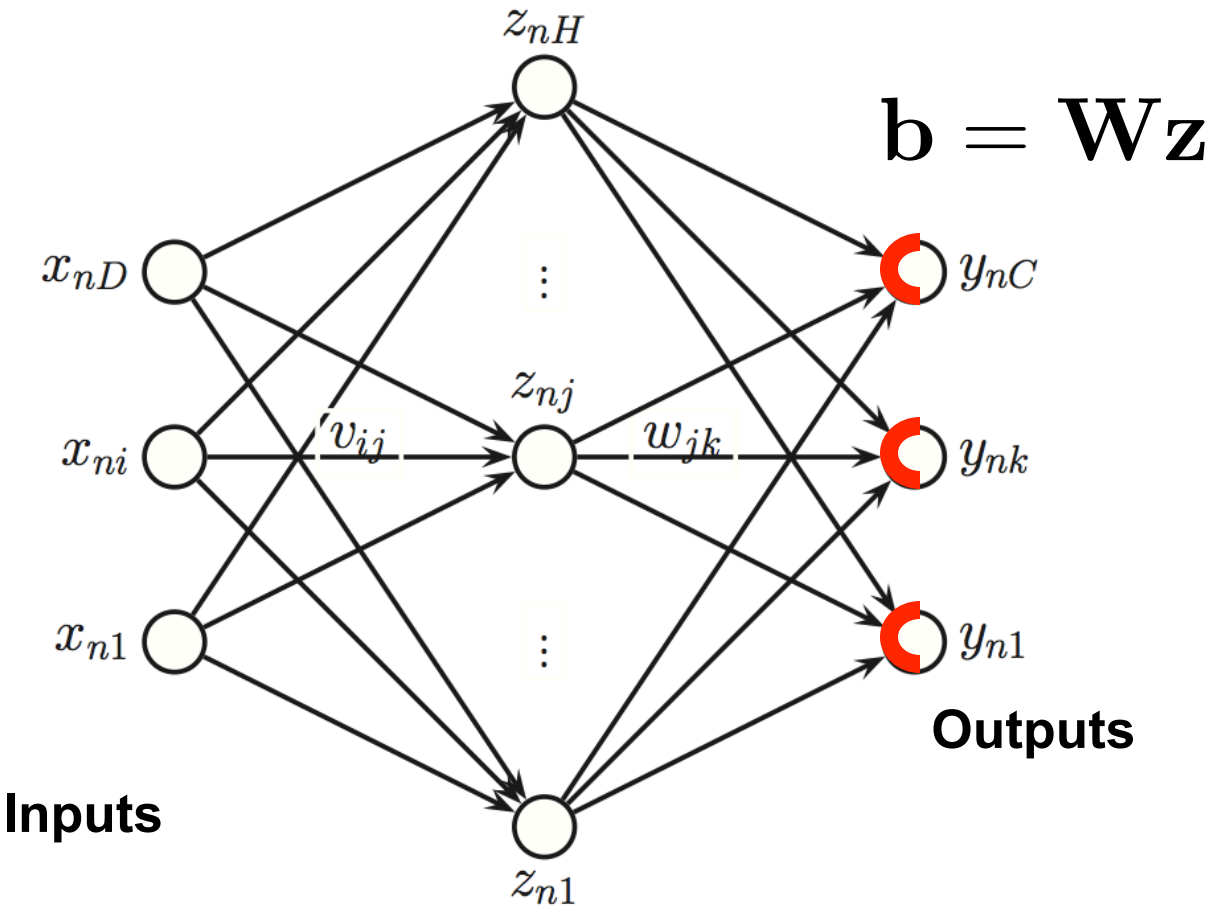


Outputs

Inputs

Hidden layer

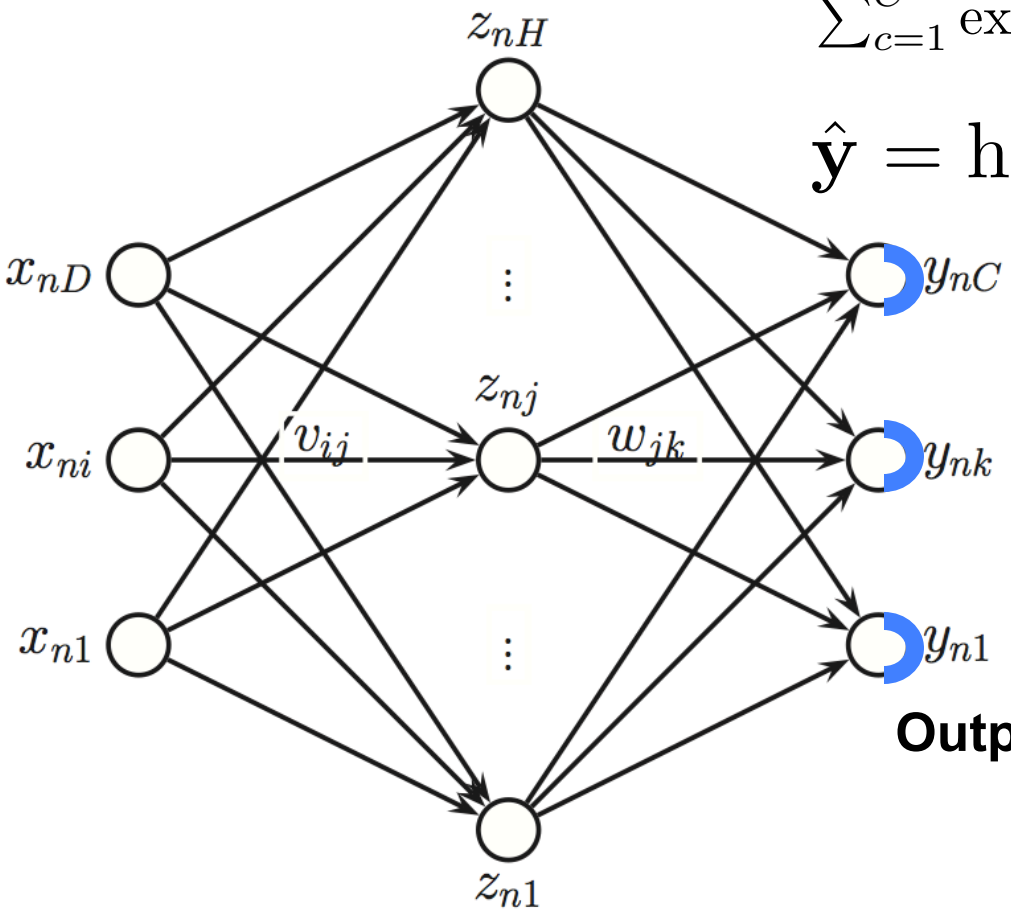
A neural network in forward mode:



A neural network in forward mode:

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

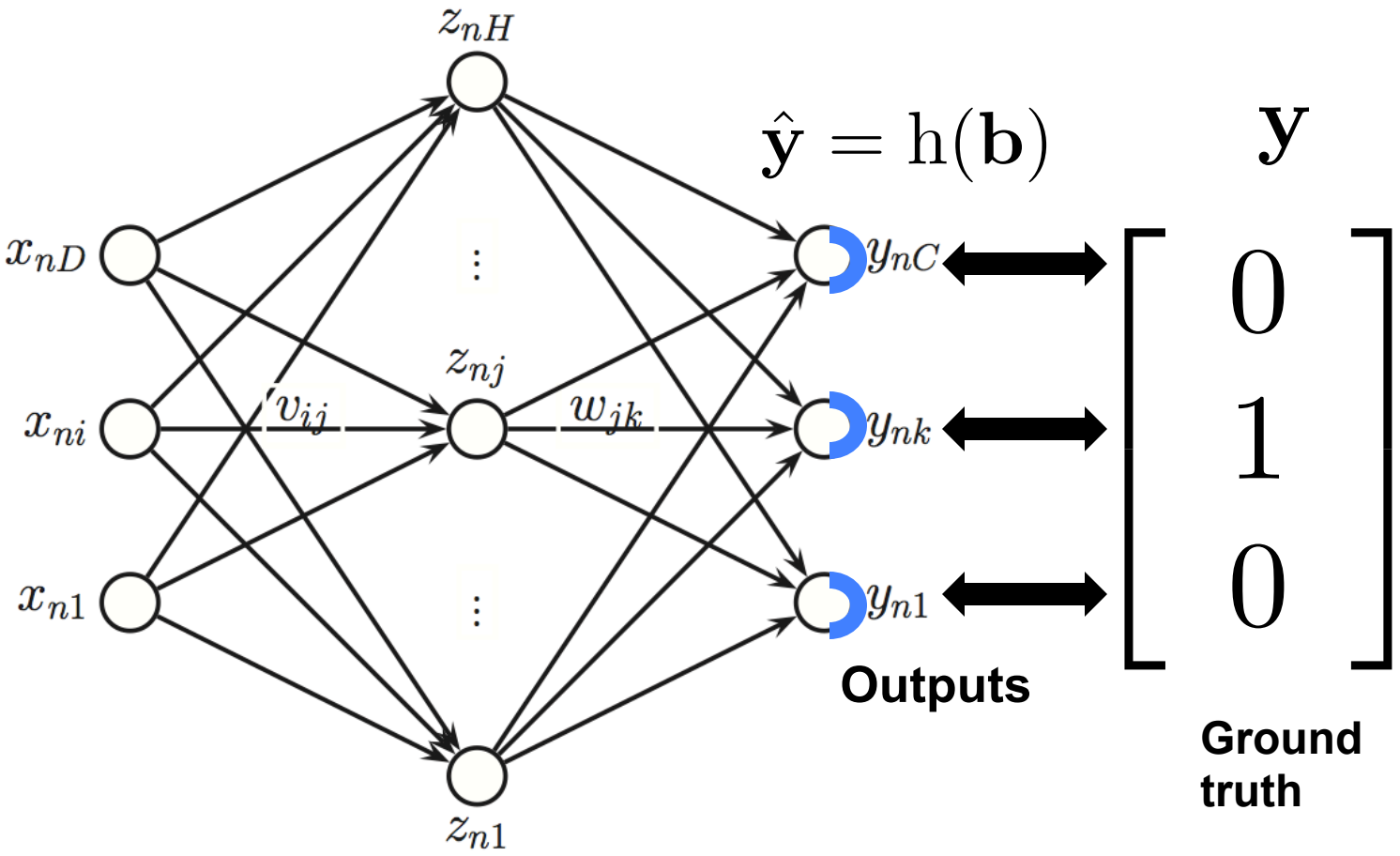
$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$



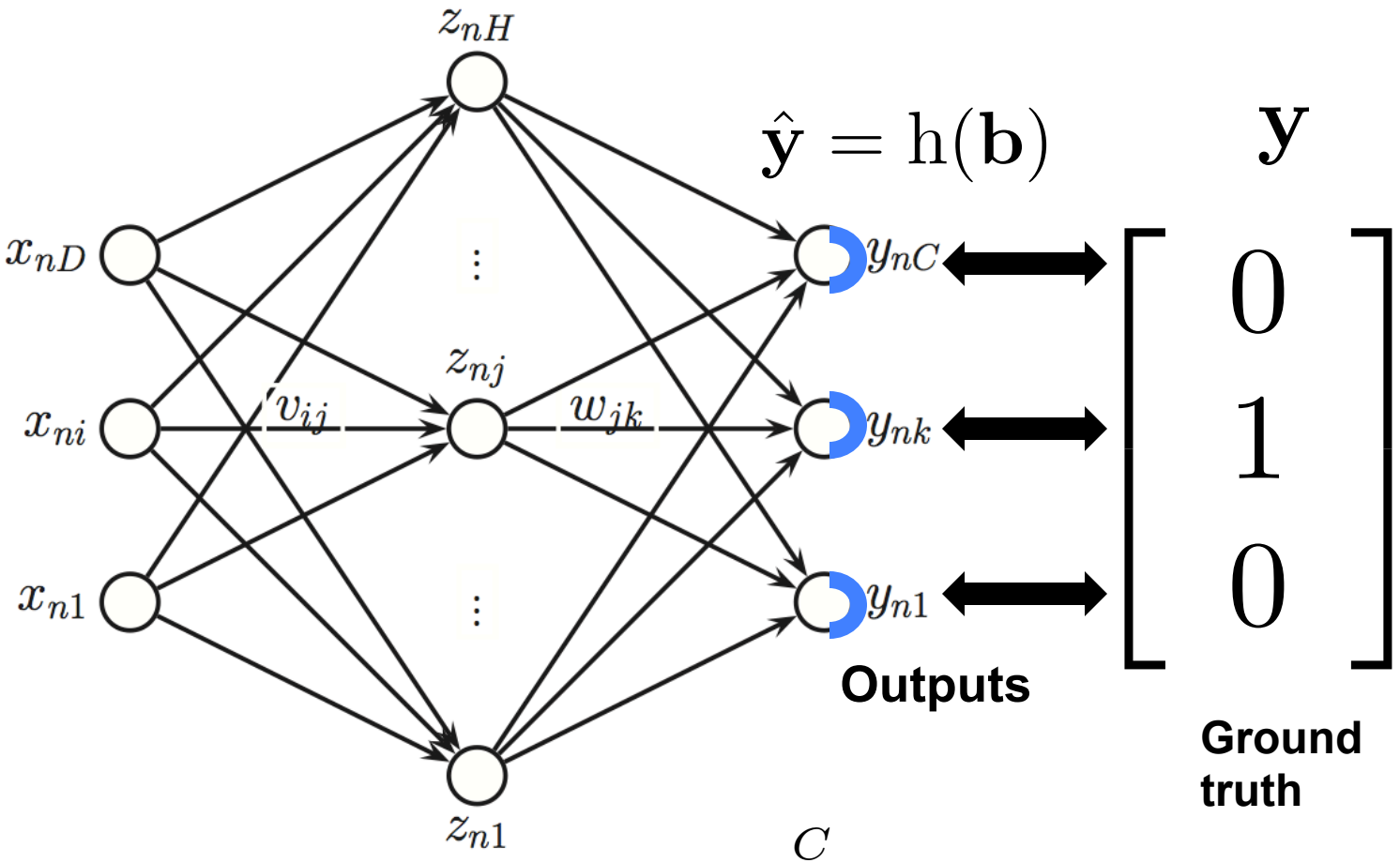
Outputs

Hidden layer

Objective for multi-class classification

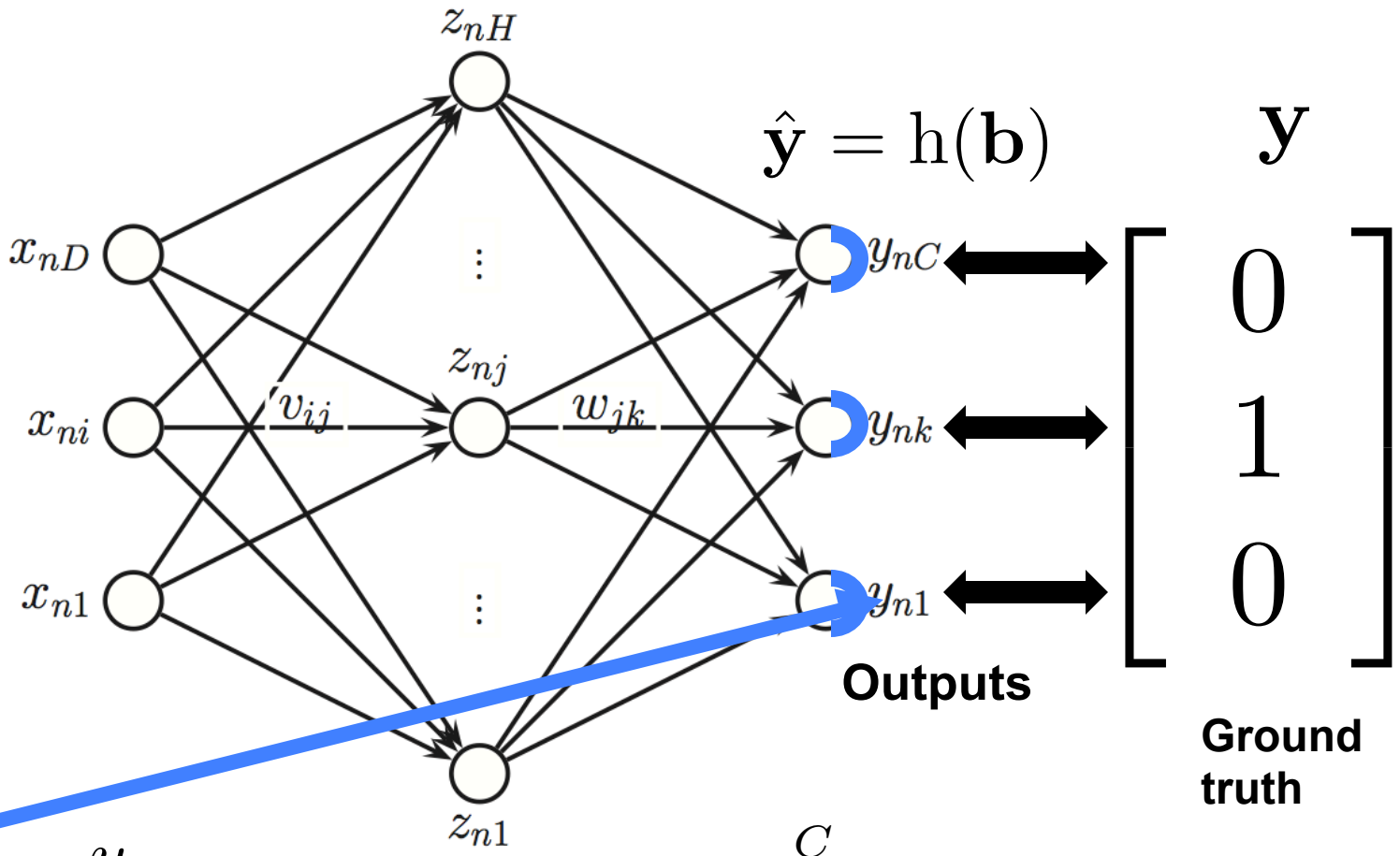


Objective for multi-class classification



$$L(\mathbf{W}) = -\sum_{c=1}^{C} y_c \log\left(\hat{y}_c(\mathbf{x}; \mathbf{W})\right)$$

Derivative of loss w.r.t. top-layer neurons

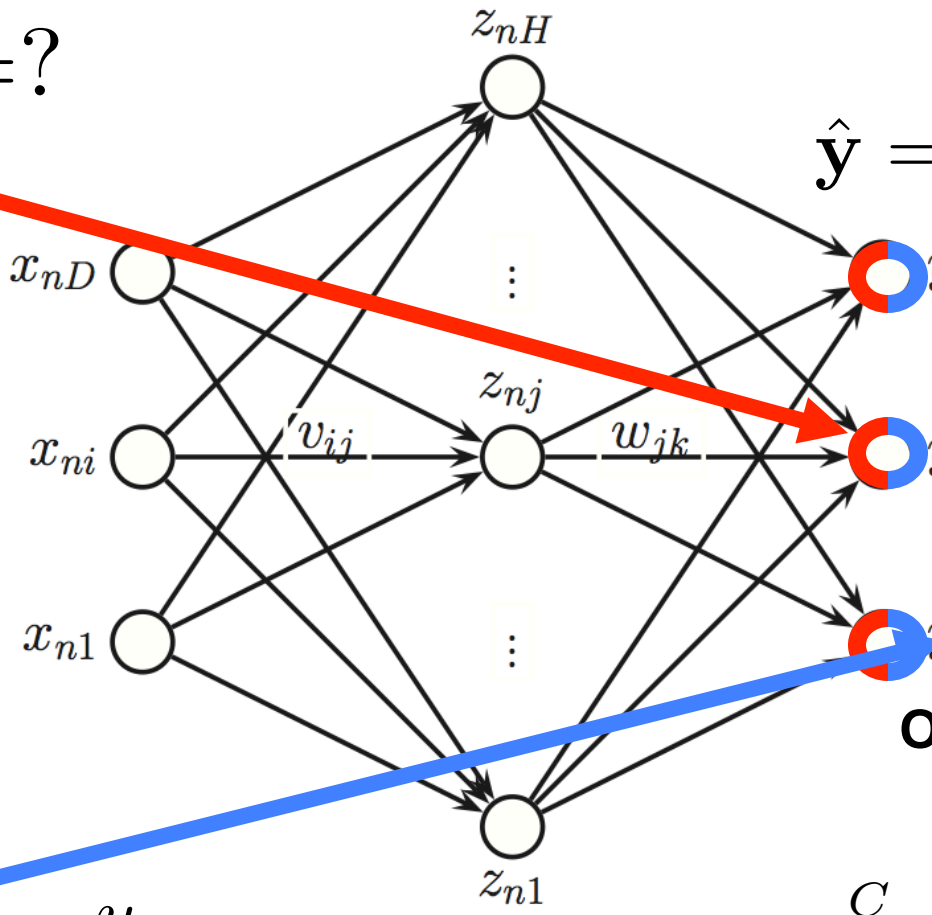


$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

$$L(\mathbf{W}) = -\sum_{c=1}^{C} y_c \log\left(\hat{y}_c(\mathbf{x}; \mathbf{W})\right)$$

A neural network in backward mode:

$$\frac{\partial L}{\partial b_k} = ?$$



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

Outputs

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

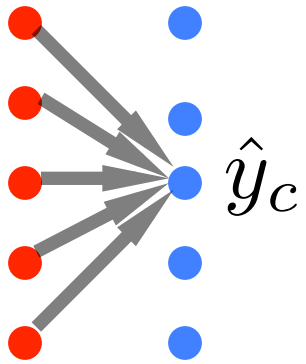
Ground truth

$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

$$L(\mathbf{W}) = -\sum_{c=1}^{C} y_c \log\left(\hat{y}_c(\mathbf{x}; \mathbf{W})\right)$$

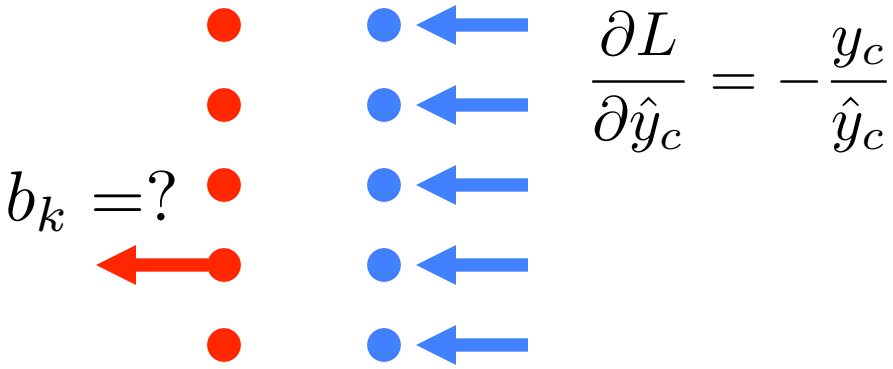
Softmax in forward mode: all for one

$$\hat{y}_c = \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b'_c)}$$



Softmax in backward mode?

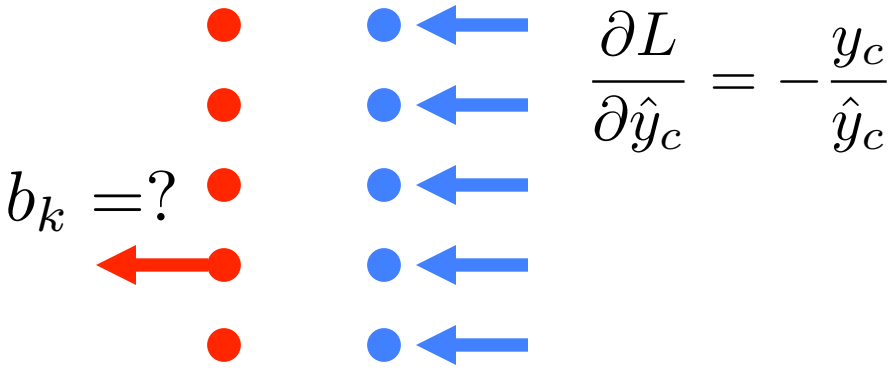
$$\hat{y}_c = \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b'_c)}$$



$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

Softmax in backward mode?

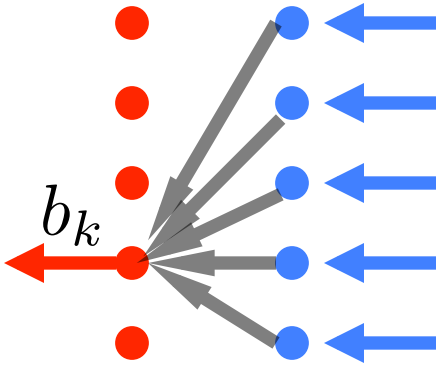
$$\hat{y}_c = \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b'_c)}$$



$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

Softmax in backward mode: one from all

$$\hat{y}_c = \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b'_c)}$$



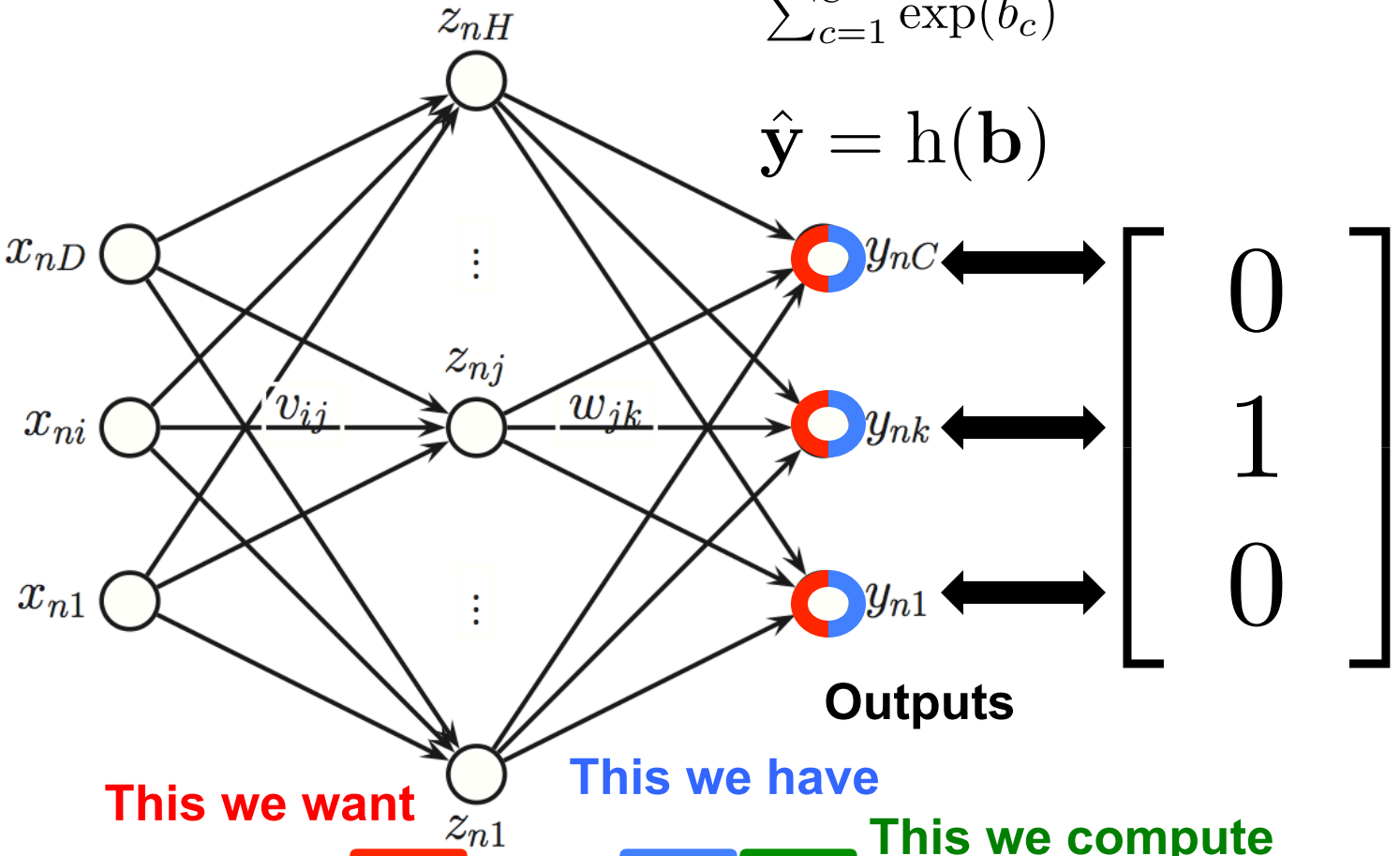
$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

$$\frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k}$$

A neural network in backward mode:

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

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



This we want

$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

This we have

$$\frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k}$$

This we compute

$$= \hat{y}_k - y_k$$

In backward mode?

$$\frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k} \qquad \hat{y}_c = \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b'_c)} \qquad \frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

$$\frac{\partial y_c}{\partial b_k} = \frac{\frac{\partial \exp(b_c)}{\partial b_k}}{\sum_{c'=1}^{C} \exp(b_{c'})} - \frac{\exp(b_c) \frac{\partial \sum_{c'=1}^{C} \exp(b_{c'})}{\partial b_k}}{(\sum_{c'=1}^{C} \exp(b_{c'}))^2}$$

$$= \frac{[c=k]\exp(b_k)}{\sum_{c'} \exp(b'_c)} - \frac{\exp(b_c)}{\sum_{c'=1}^{C} \exp(b_{c'})} \frac{\exp(b_k)}{\sum_{c'=1}^{C} \exp(b_{c'})}$$

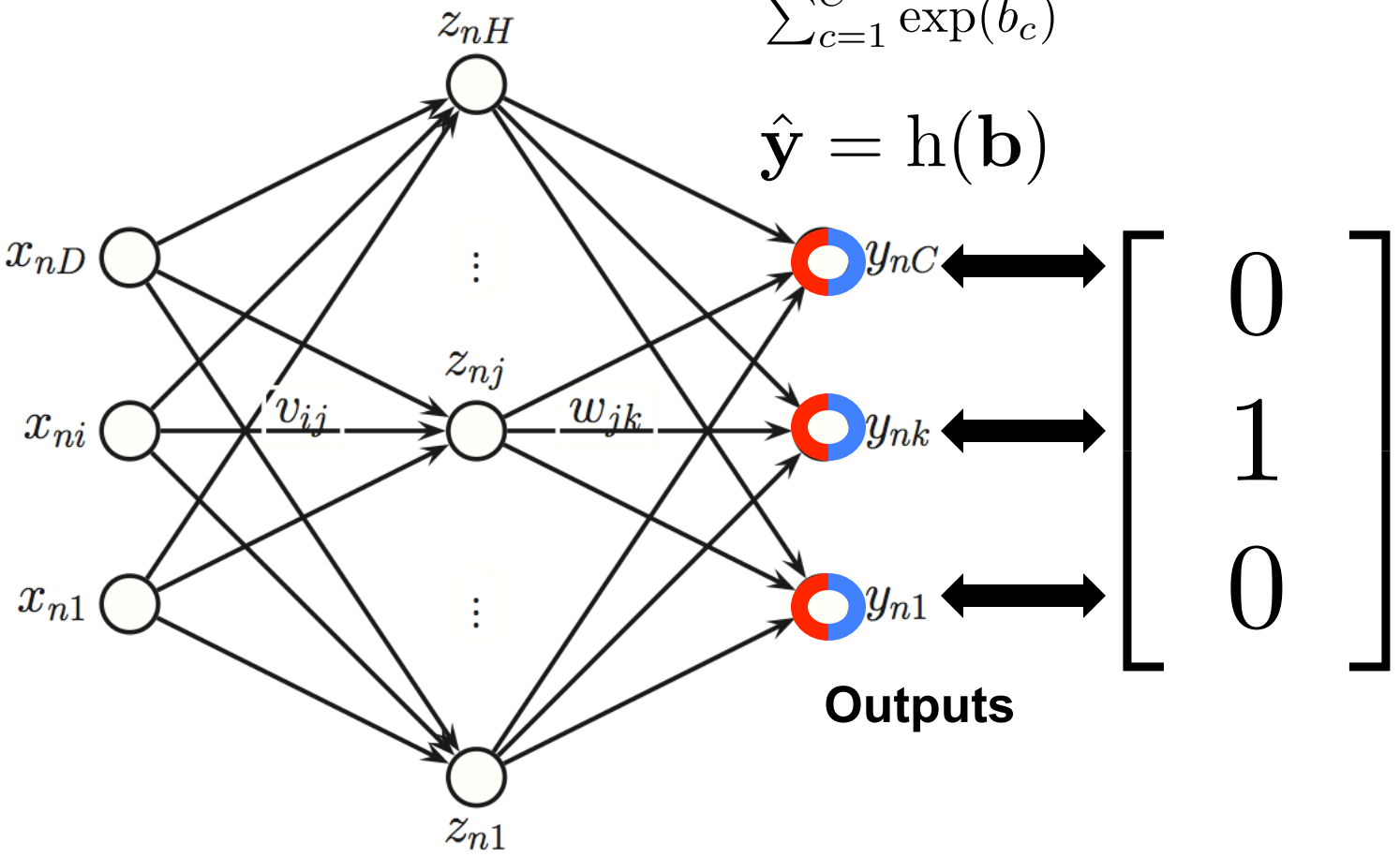
$$= [c=k]\hat{y}_k - \hat{y}_c\hat{y}_k \quad = ([c=k] - \hat{y}_c)\hat{y}_k$$

$$\frac{\partial L}{\partial b_k} = \sum_{c=1}^{C} -\frac{y_c}{\hat{y}_c} ([c=k]\hat{y}_k - \hat{y}_c\hat{y}_k) = -y_k - \sum_{c=1}^{C} (-y_c)\hat{y}_k \; = \hat{y}_k - y_k$$

A neural network in backward mode:

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

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

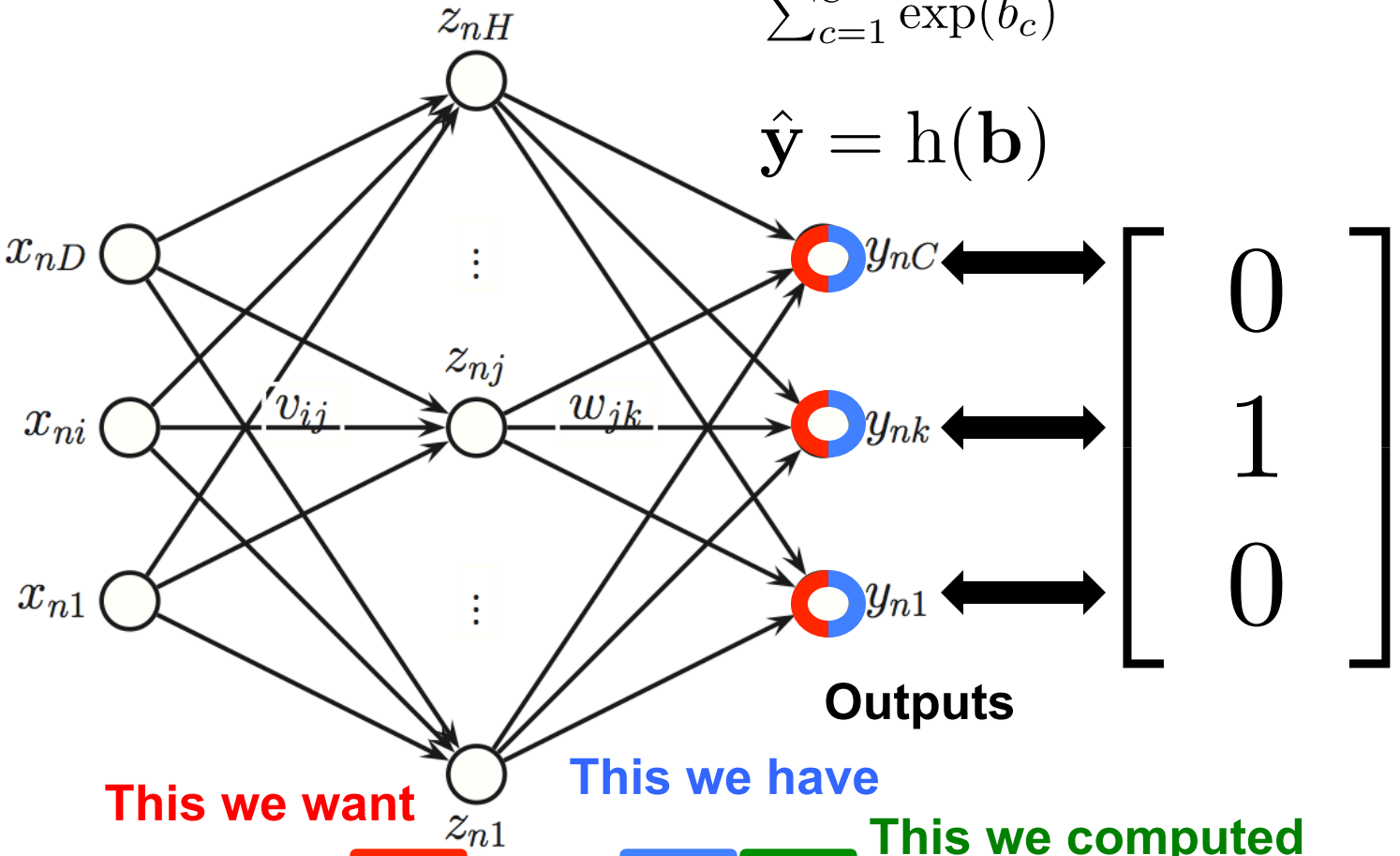


$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c} \qquad \frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k}$$

A neural network in backward mode:

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

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



This we want

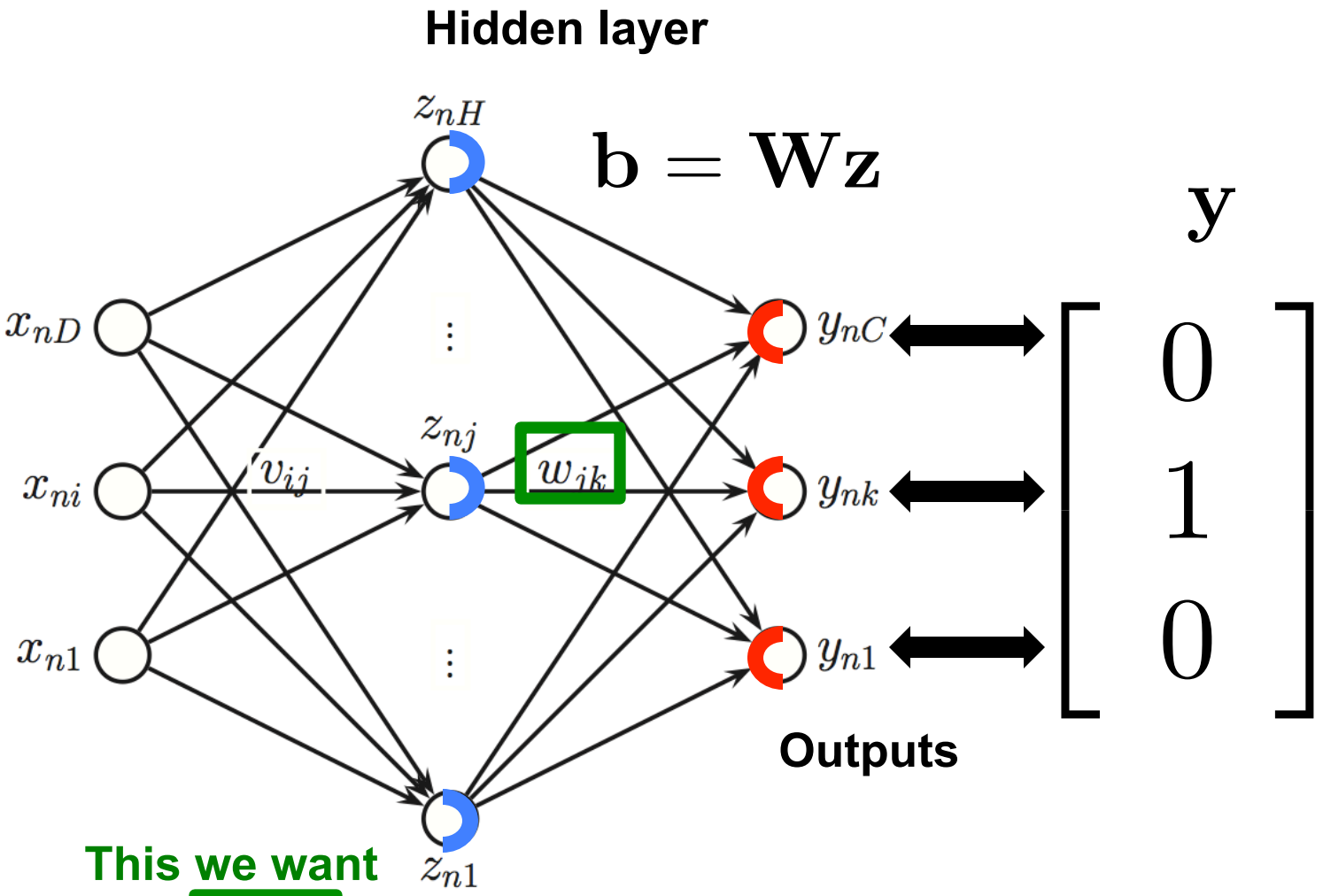
$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

This we have

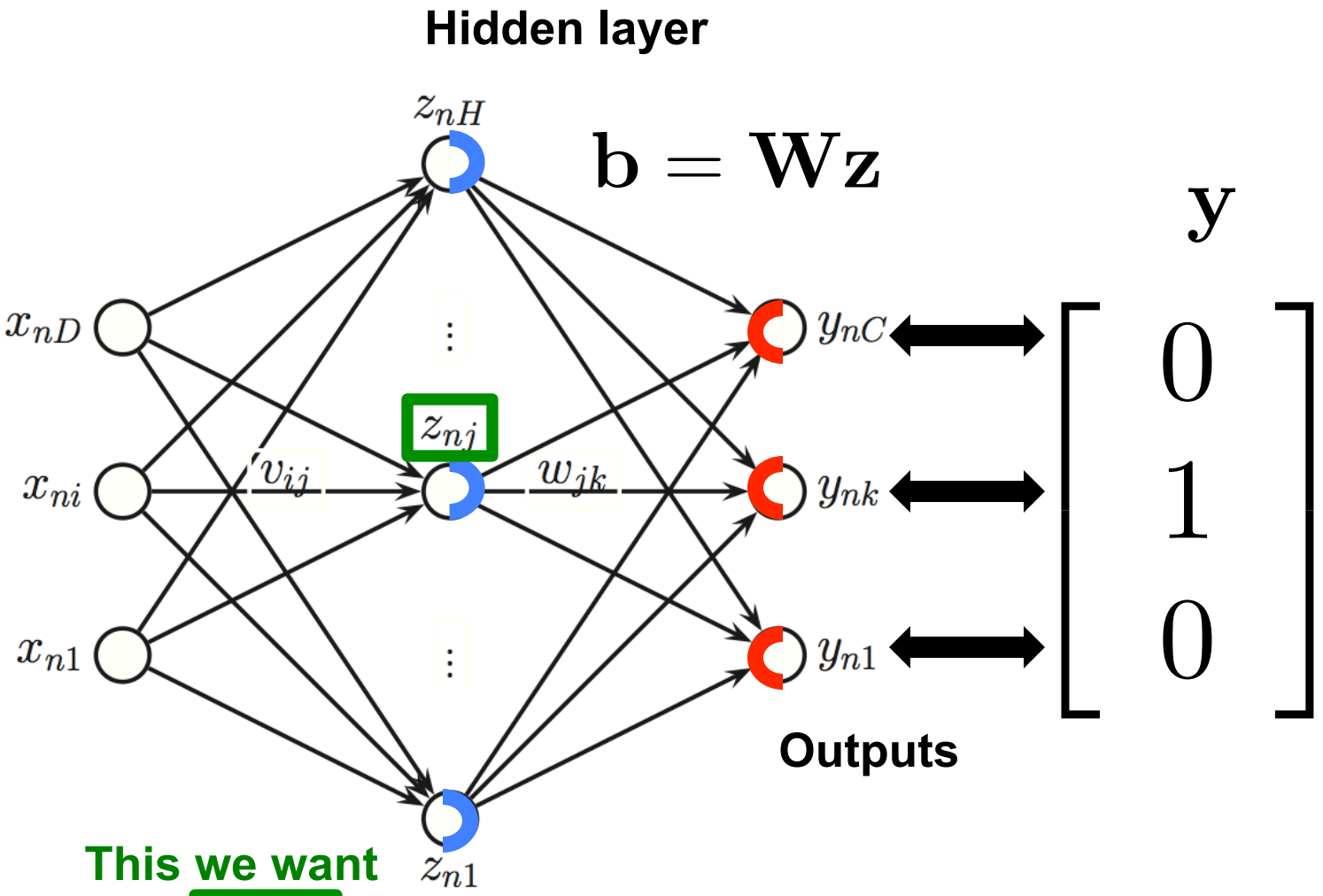
This we computed

$$\frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k} = \hat{y}_k - y_k$$

A neural network in backward mode:

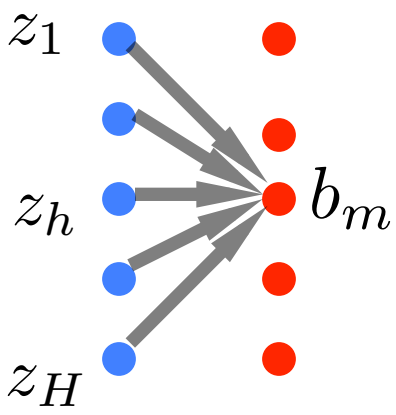


A neural network in backward mode:



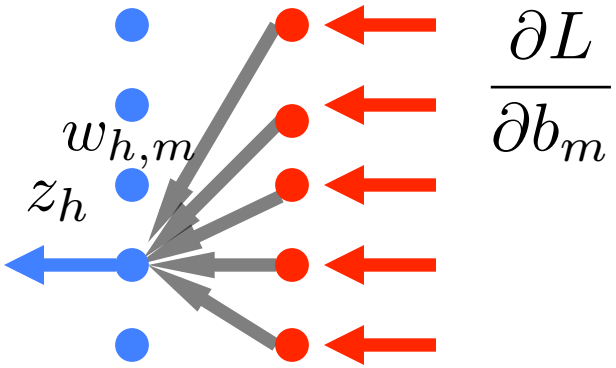
Linear layer in forward mode: all for one

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



Linear layer in backward mode: one from all

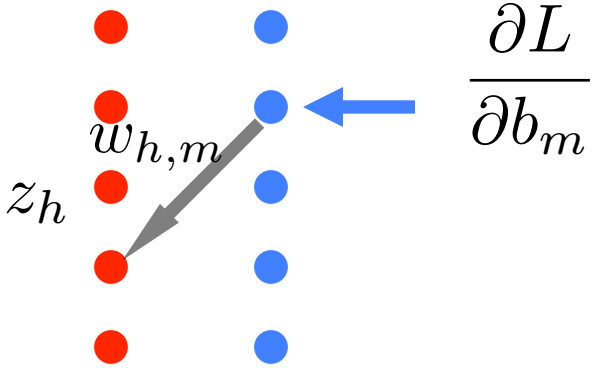
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} w_{h,c}$$

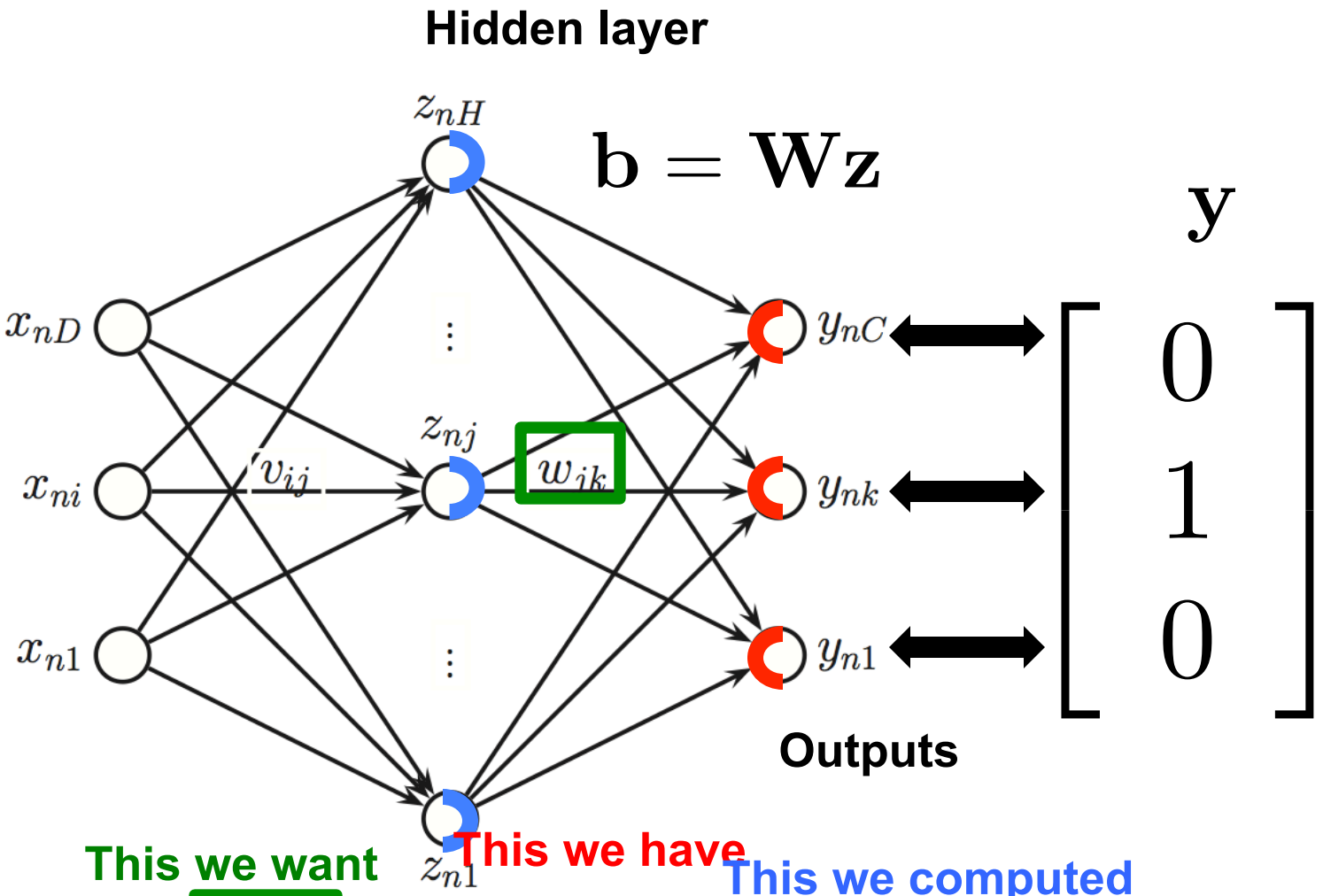
Linear layer parameters in backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$

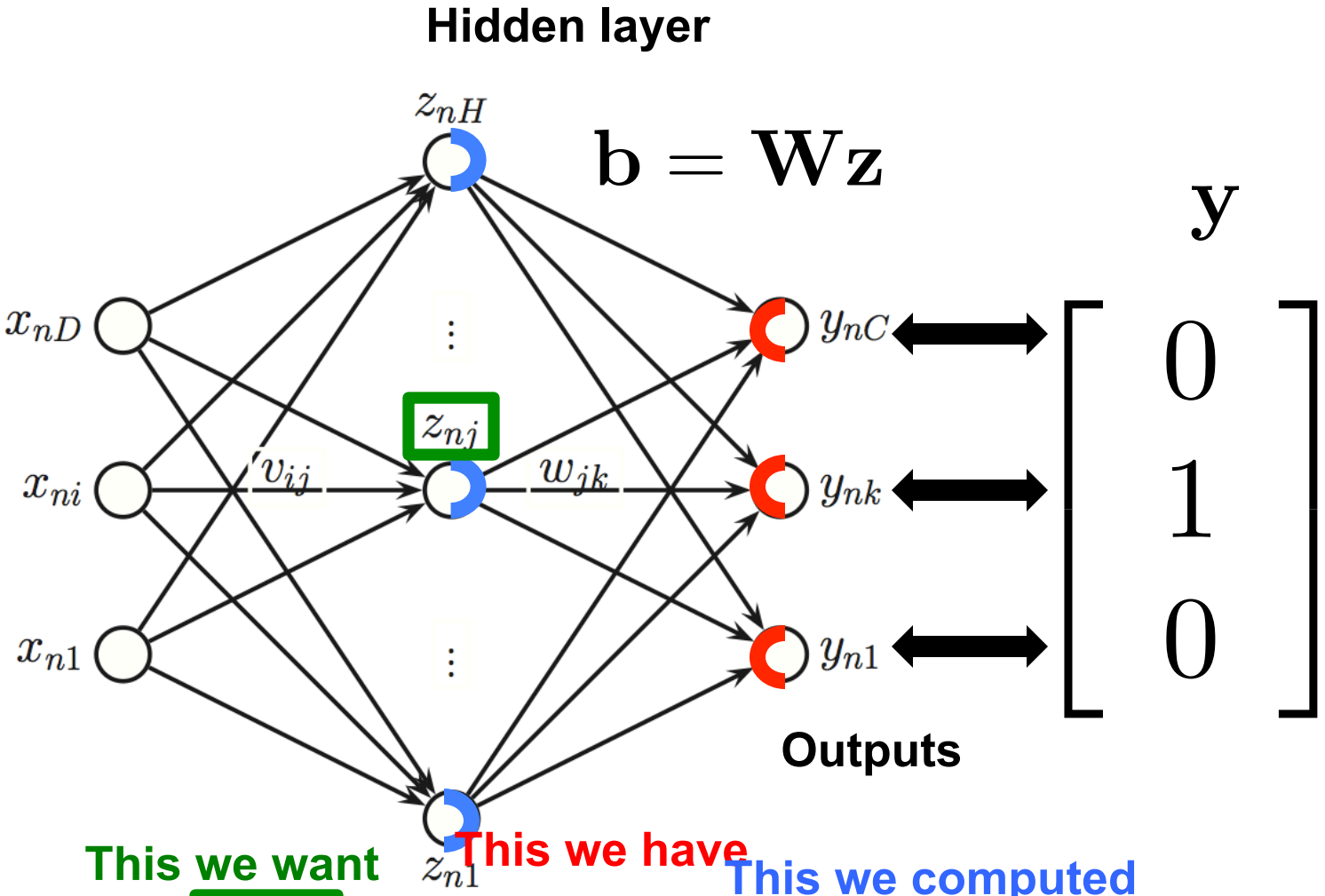


$$\frac{\partial L}{\partial w_{h,m}} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial w_{h,m}} = \frac{\partial L}{\partial b_m} z_h$$

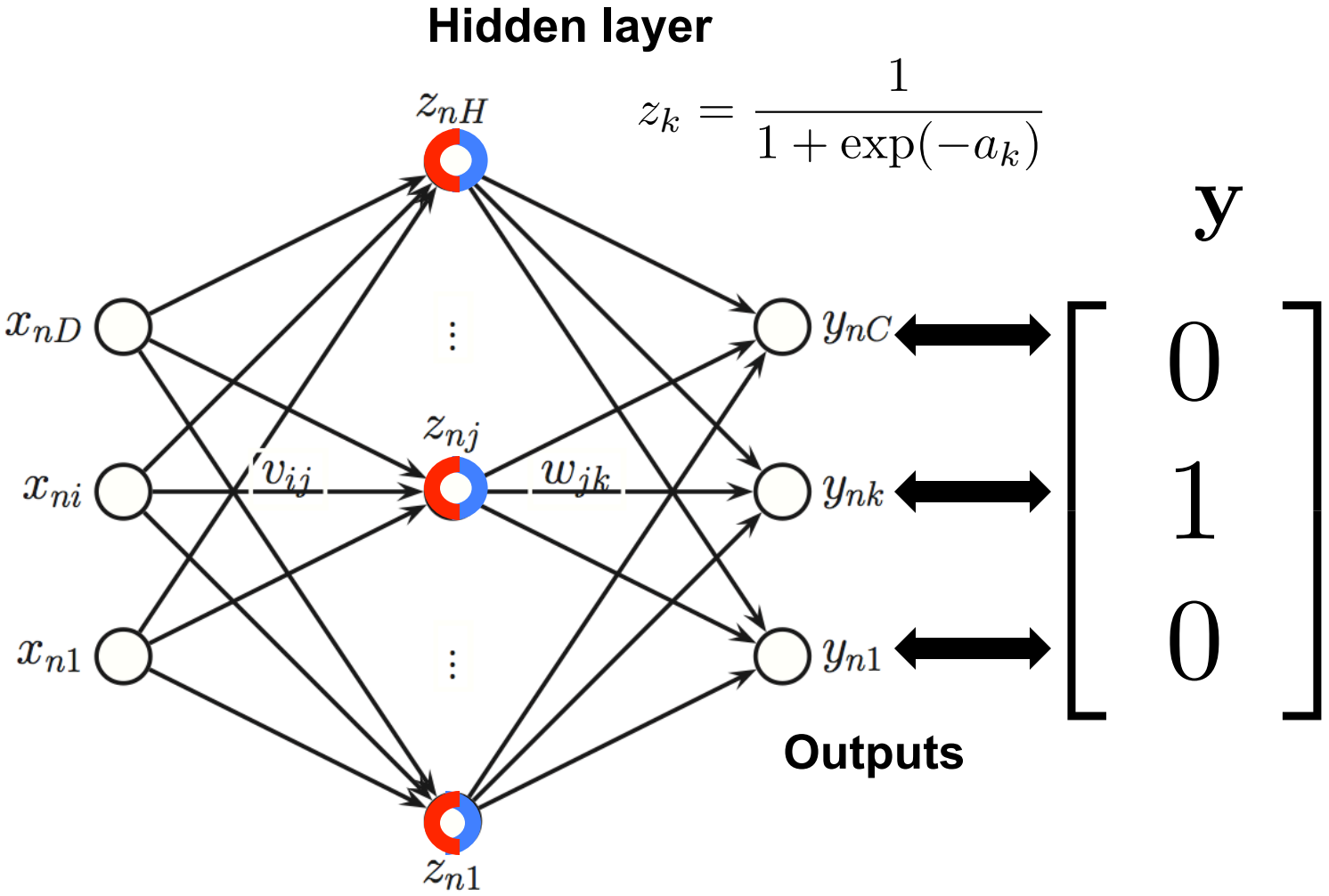
A neural network in backward mode:



A neural network in backward mode: ⏪

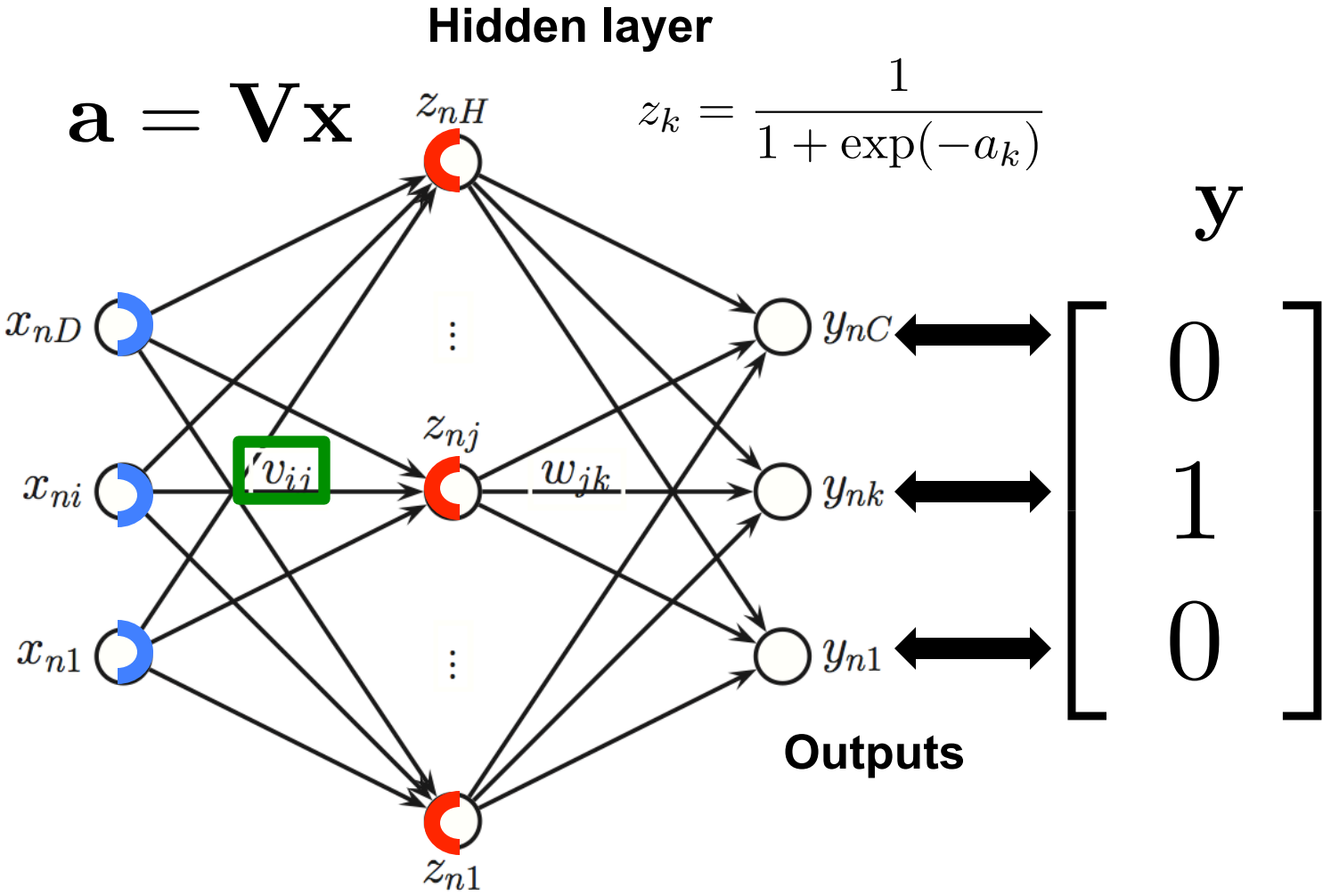


A neural network in backward mode:



$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1-g(a_k))$$

A neural network in backward mode:



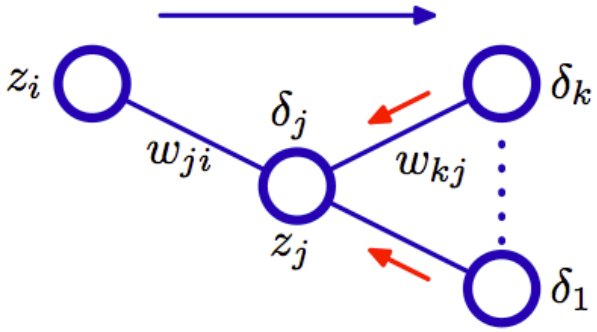
$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k}\frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

Back-propagation in a nutshell

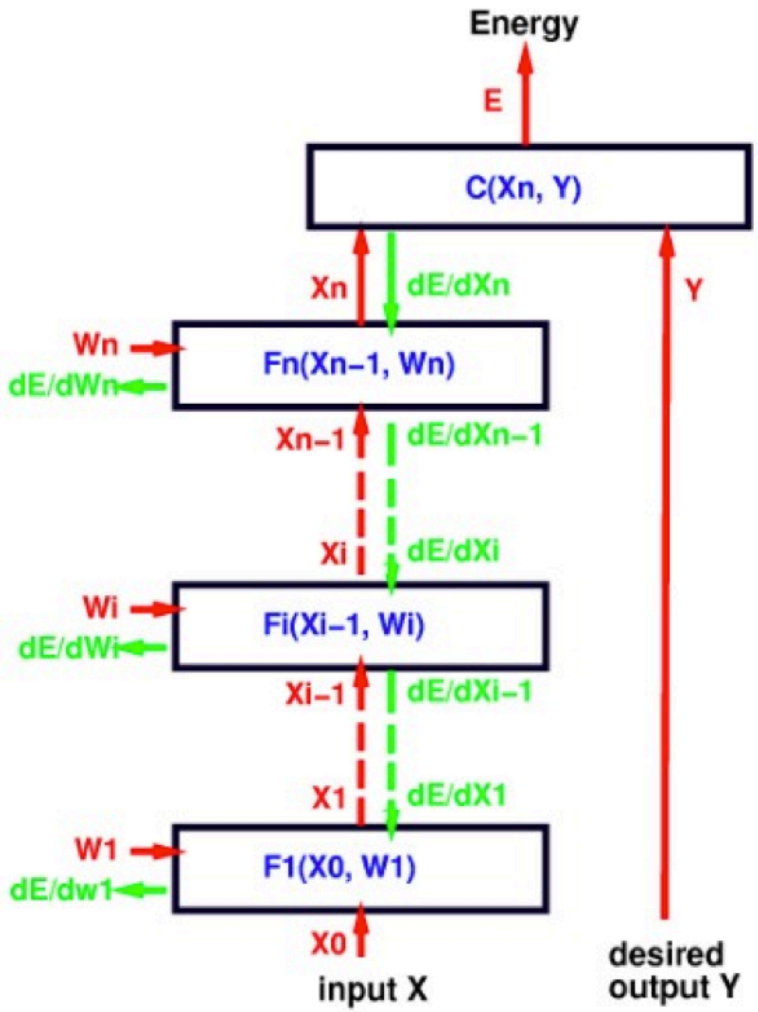
Error messages, ‘δ’, at current layer:
sensitivity of loss to activations of current layer

Every node estimates its error message by forming a weighted sum of the error messages of its recipients

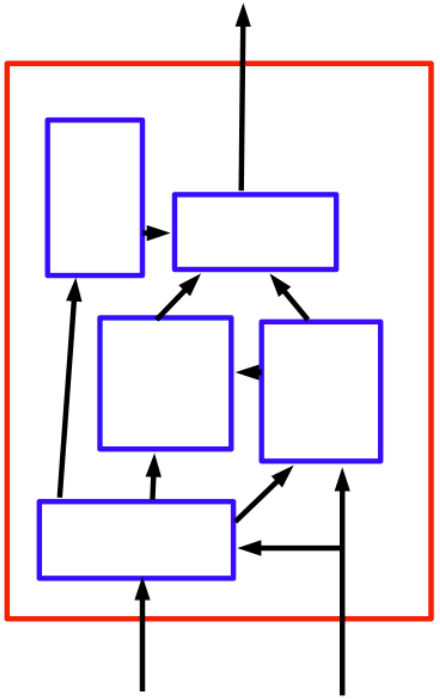
Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



Forward/backward information flow



We can use the same algorithm to learn in any Directed Acyclic Graph!

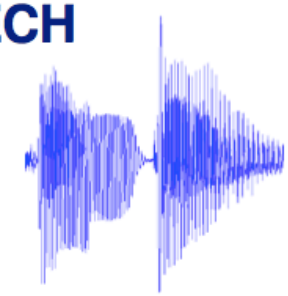


Deep Learning: breakthrough for all of AI

VISION

$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

SPEECH



$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

NLP

This burrito place is yummy and fun!

$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

Next 10 slides: Week 1 reminder

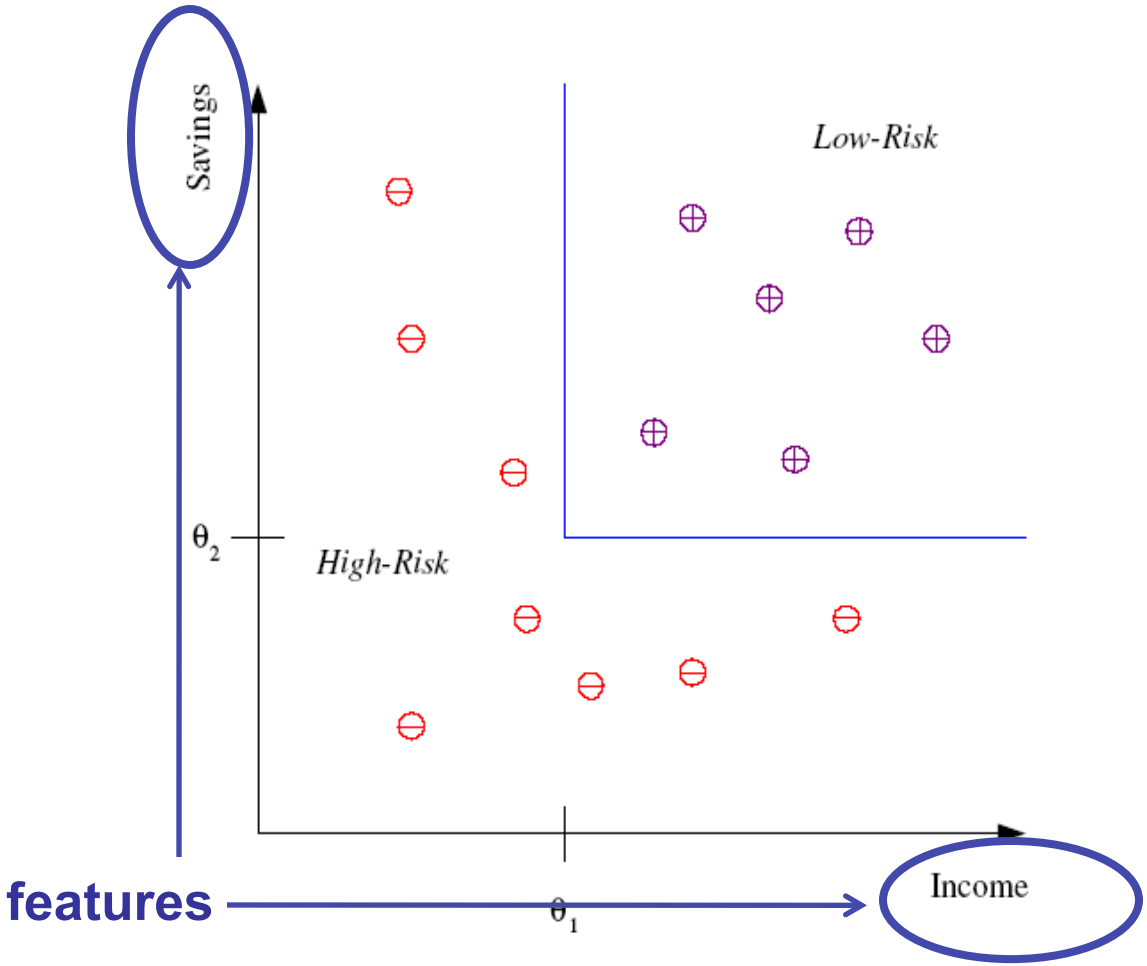
Our goal in this course: learn an input-output mapping

$$y = f_w(x) \quad (= f(x, w))$$

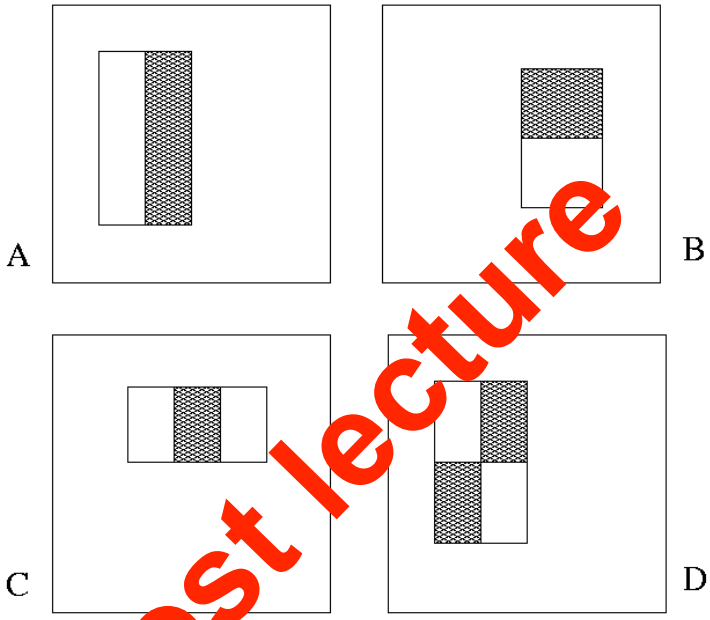
- Output: y
- Input: x
- Method: f
- Parameters: w

How to construct this function? $y = f_w(x)$

- Step 1: Determine its inputs, x



Feature example: Haar wavelets (NOT part of our course)



Value = ∑ (pixels in white area) – ∑ (pixels in black area)

Why these features?
Extremely fast to compute
(4 pixel operations per box)

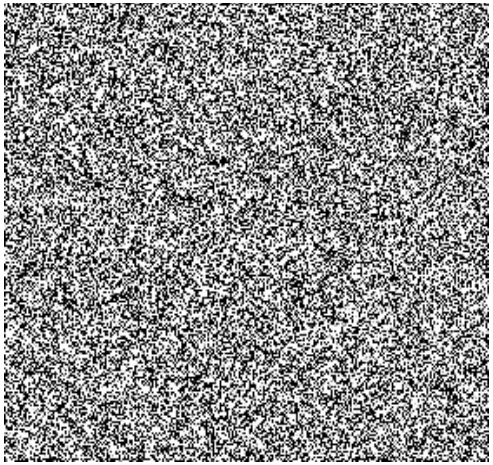
1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

input image

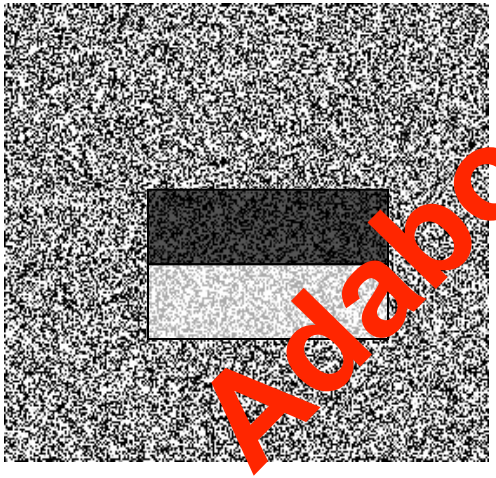
0	0	0	0	0	0
0	1	3	5	9	10
0	4	10	13	22	25
0	6	15	21	32	39
0	10	20	31	46	59
0	16	29	42	58	74

integral image

One Haar wavelet feature

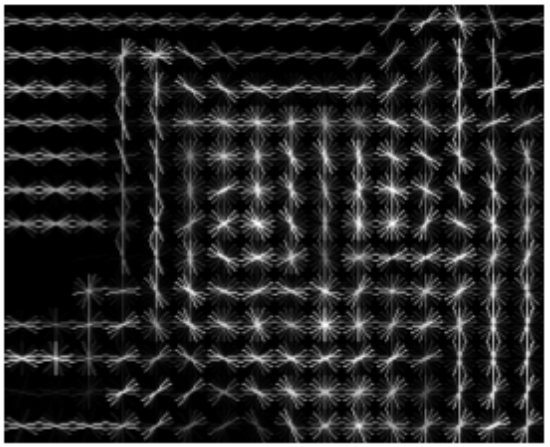


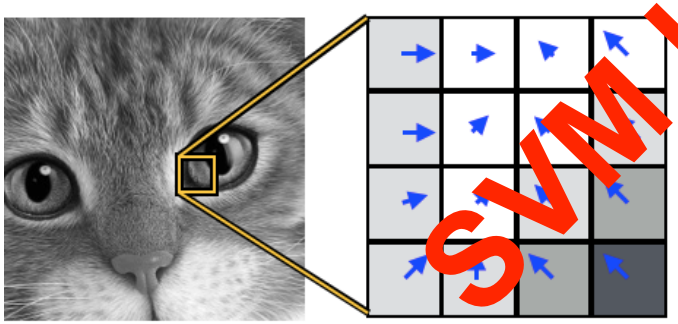
Source

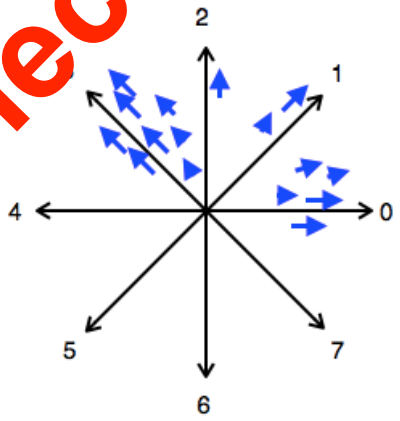


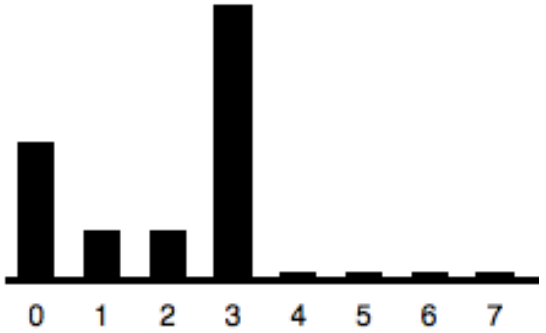
Result

Feature example: Histogram-of-gradient features (NOT part of our course)



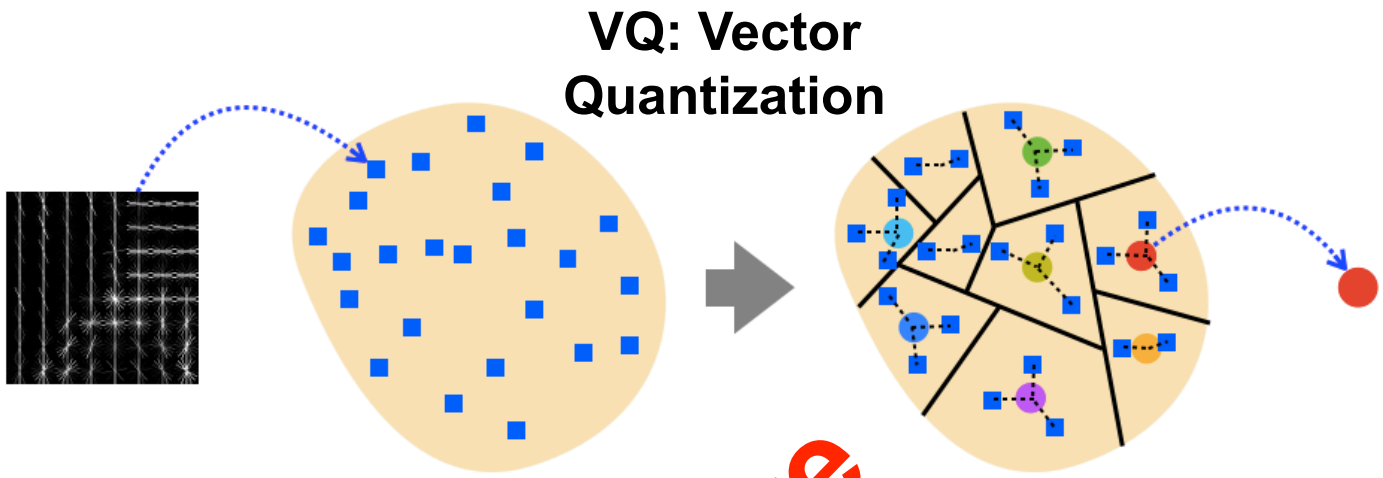






SVM lecture

Feature example: Bag-of-word features (NOT part of our course)



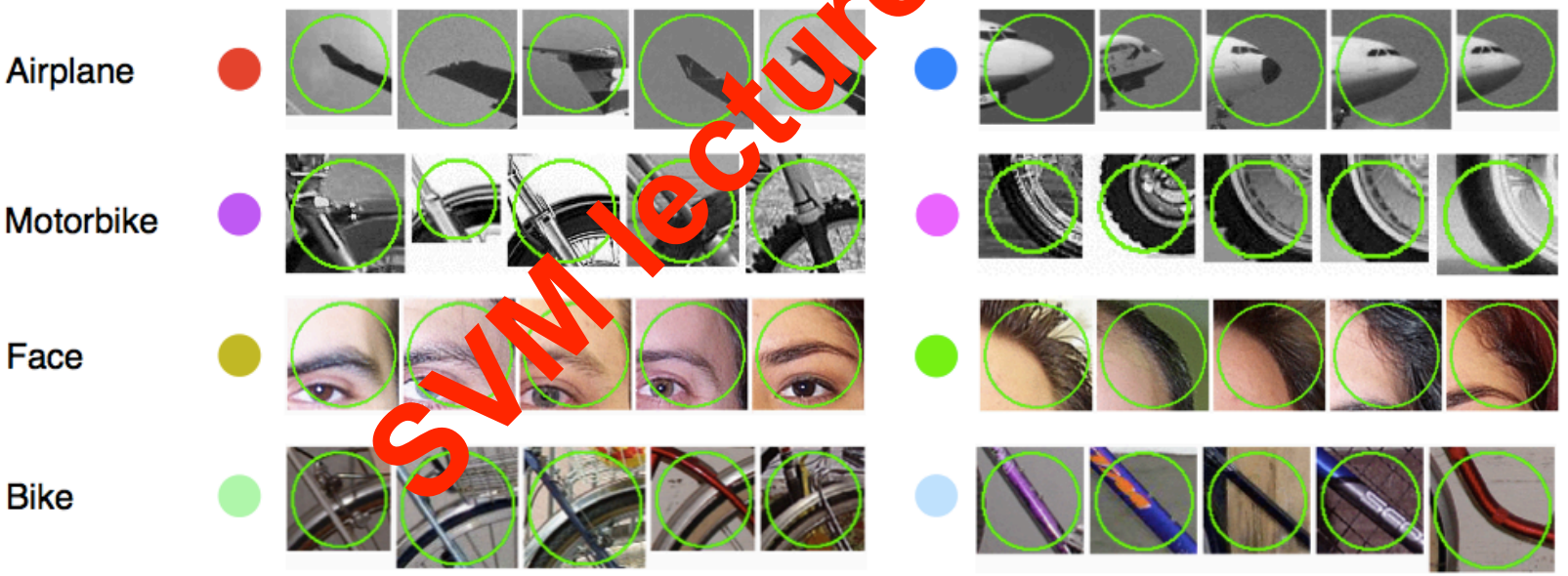
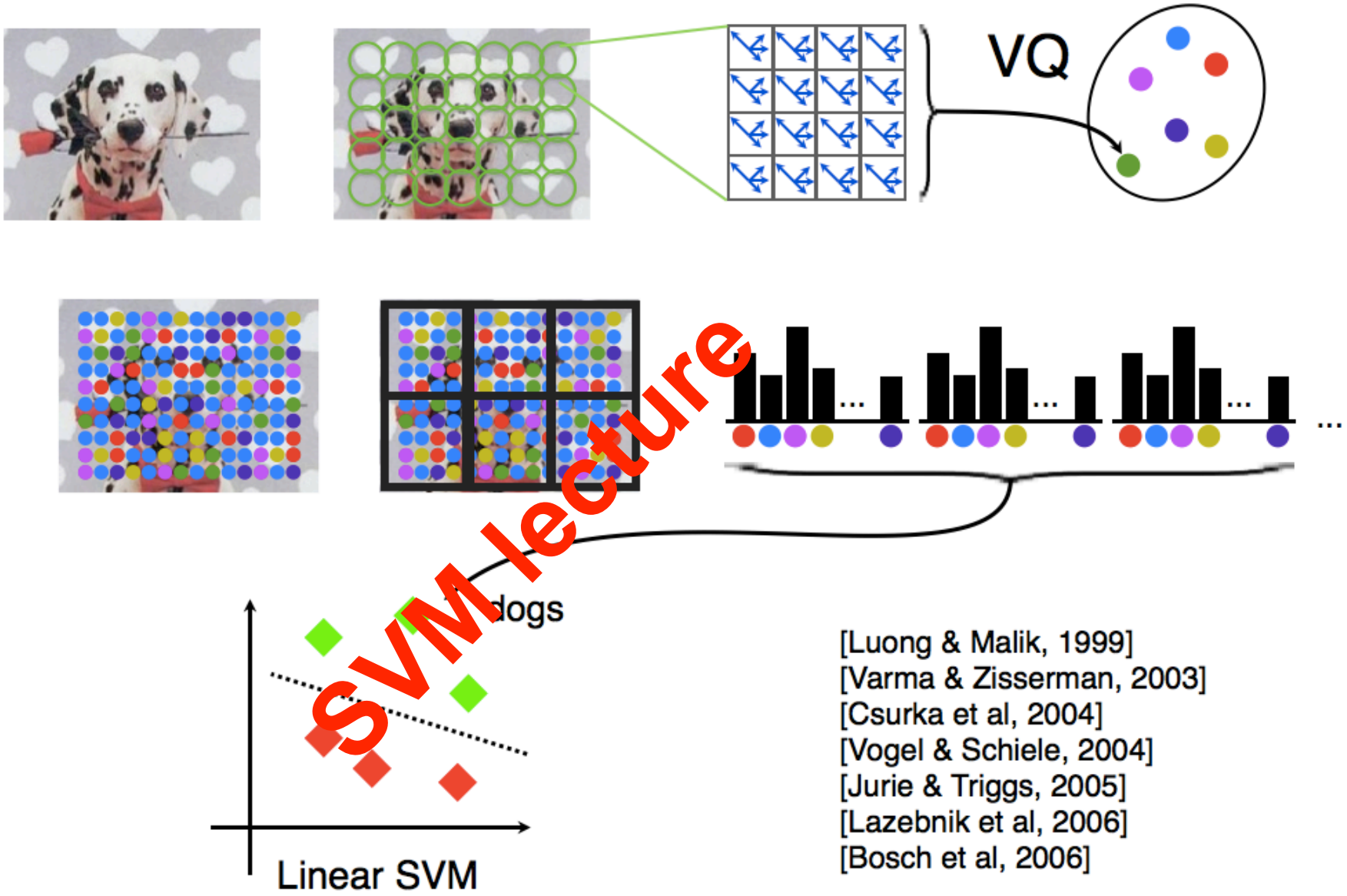
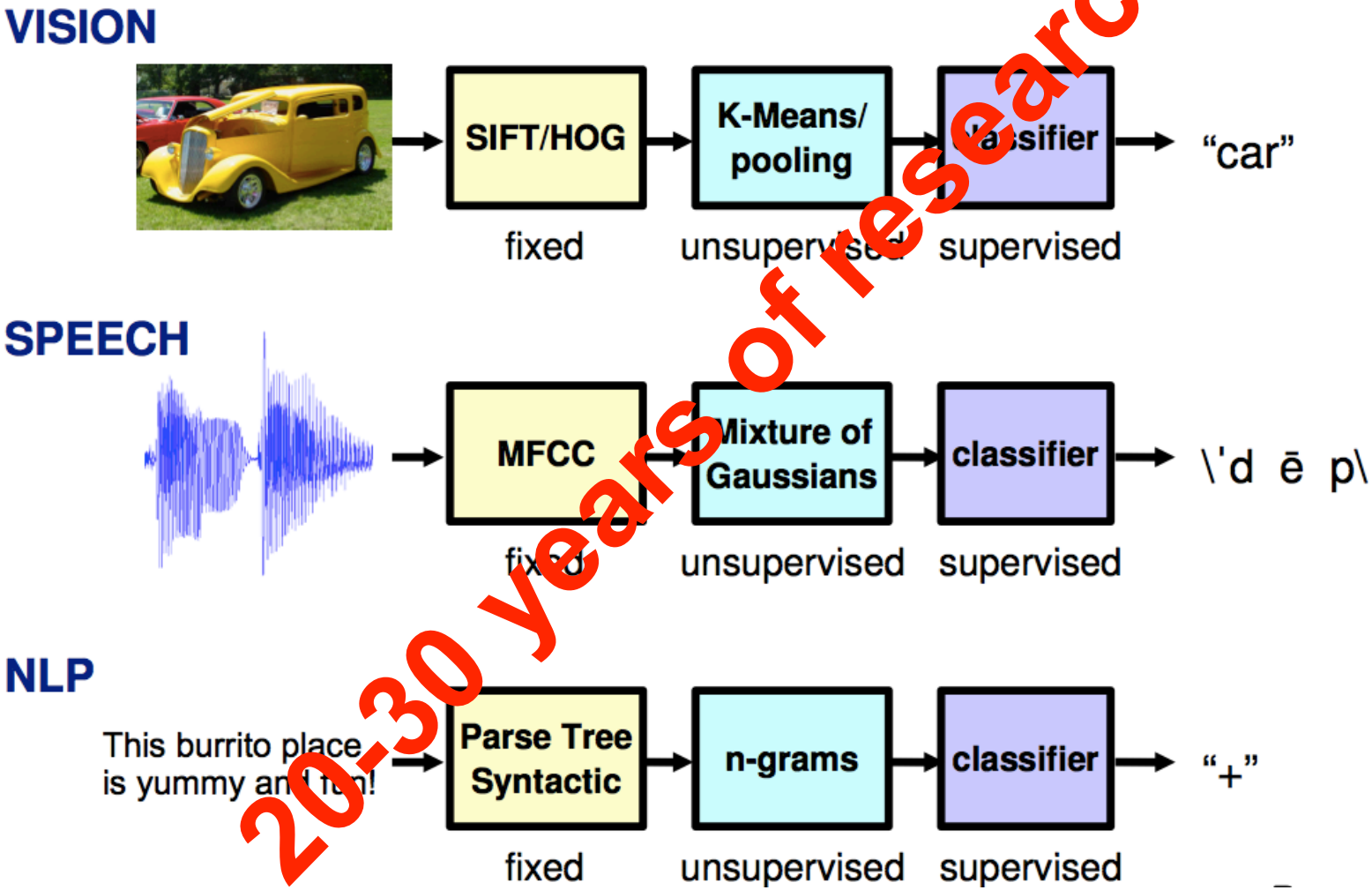


Image classification in a nutshell (NOT part of our course)

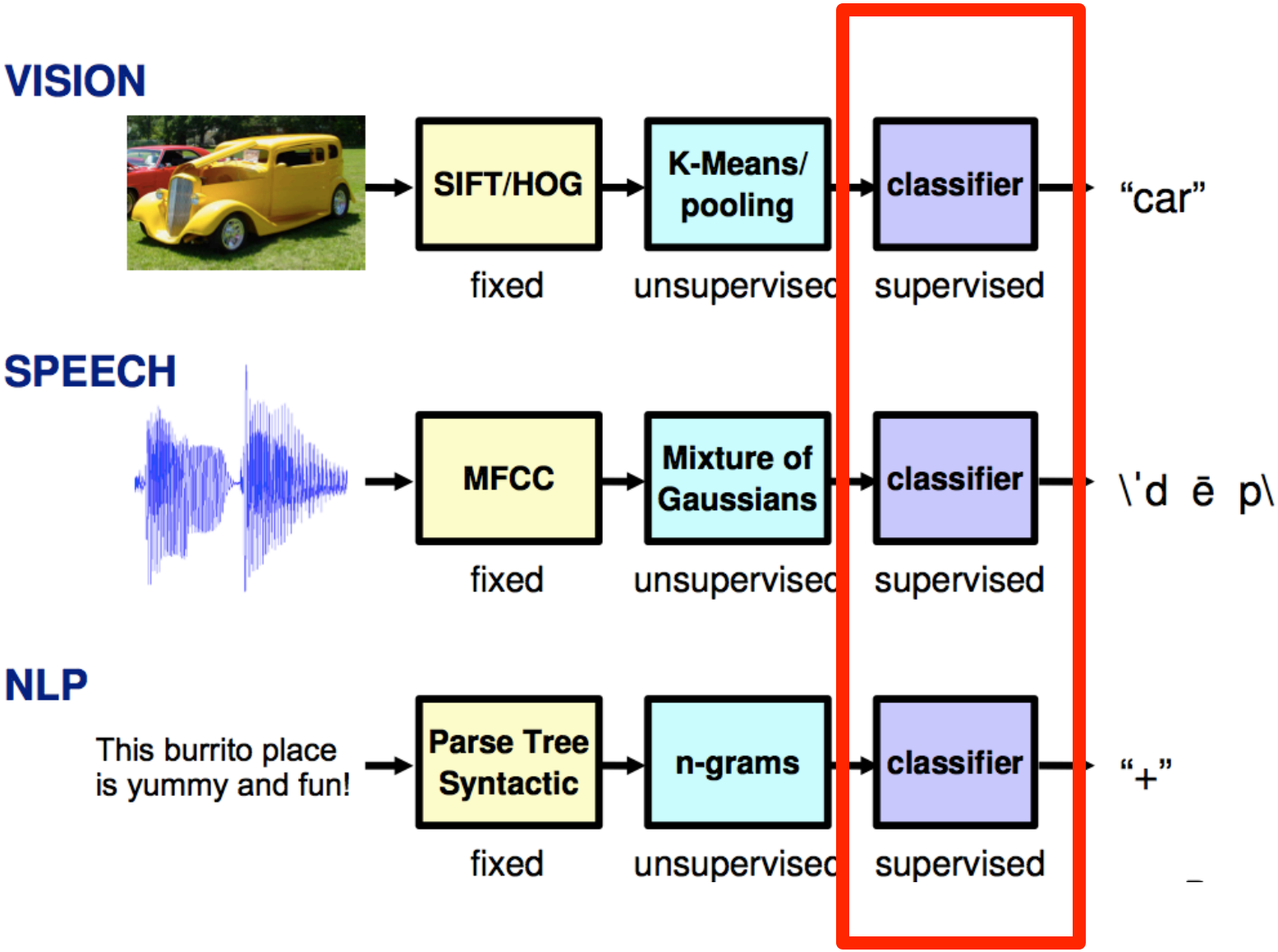


[Luong & Malik, 1999]
[Varma & Zisserman, 2003]
[Csurka et al, 2004]
[Vogel & Schiele, 2004]
[Jurie & Triggs, 2005]
[Lazebnik et al, 2006]
[Bosch et al, 2006]

Machine Learning for X: features for X + ML



Our course

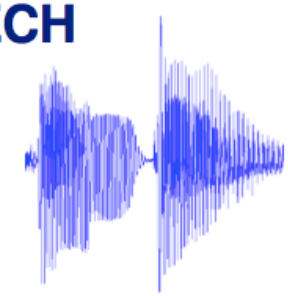

VISION
SIFT/HOG
K-Means/ pooling
classifier
"car"
fixed
unsupervised
supervised
SPEECH
MFCC
Mixture of Gaussians
classifier
\'d ē p\
fixed
unsupervised
supervised
NLP
This burrito place is yummy and fun!
Parse Tree Syntactic
n-grams
classifier
"+"
fixed
unsupervised
supervised

Deep Learning: learn the features!

VISION

$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

SPEECH



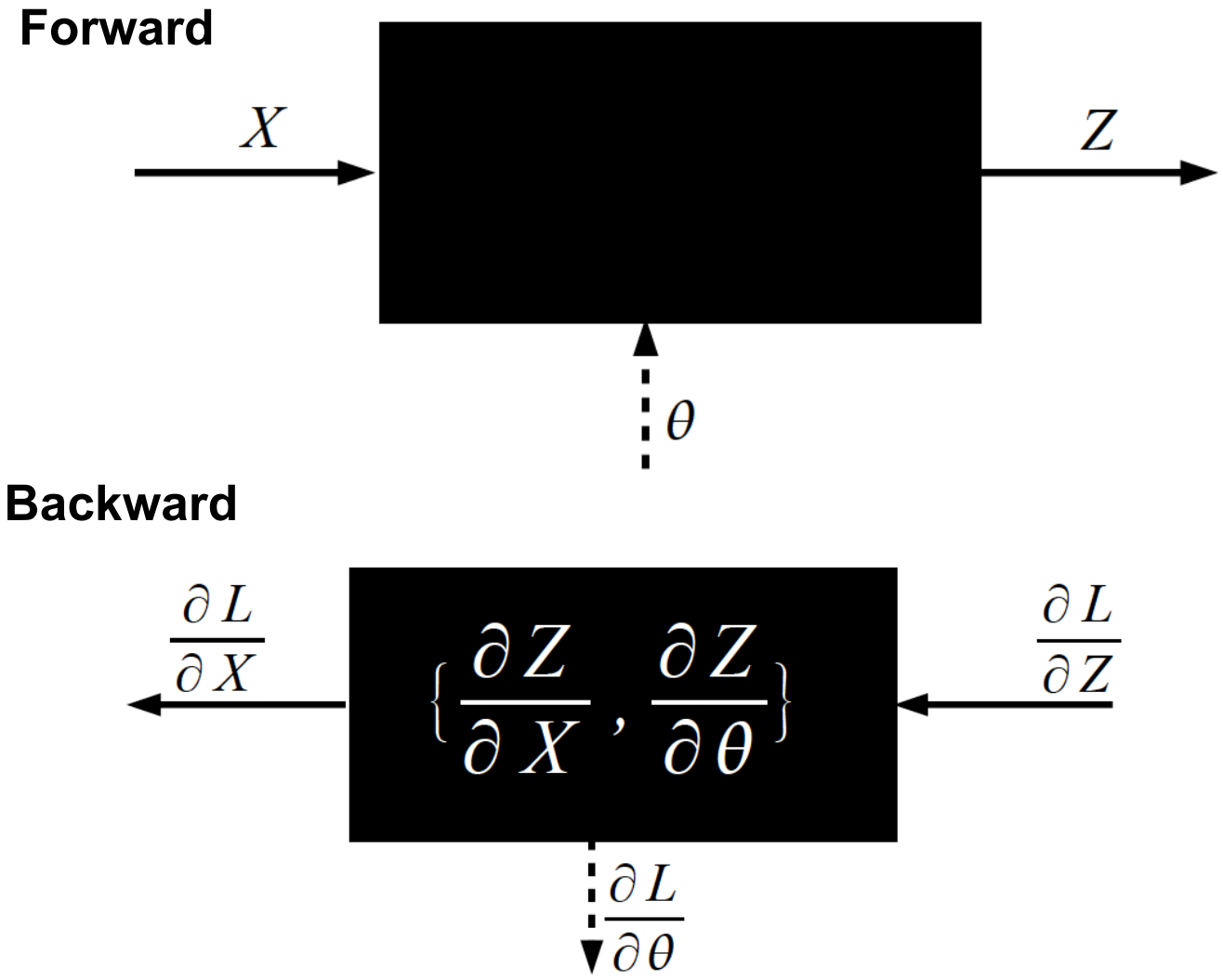
$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

NLP

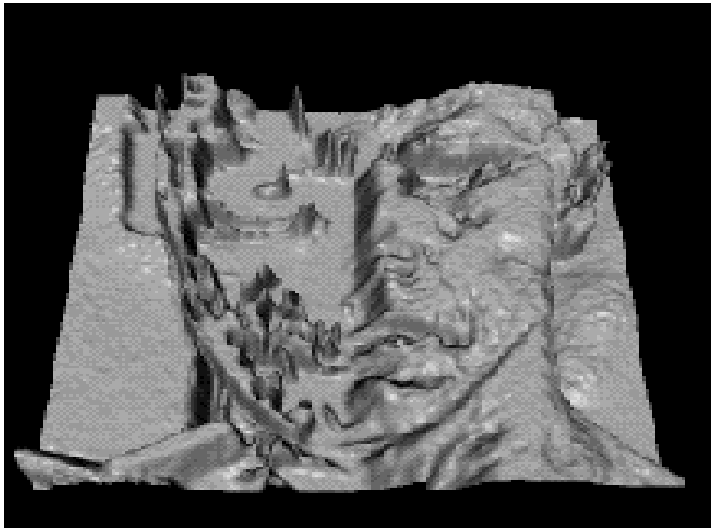
This burrito place is yummy and fun!

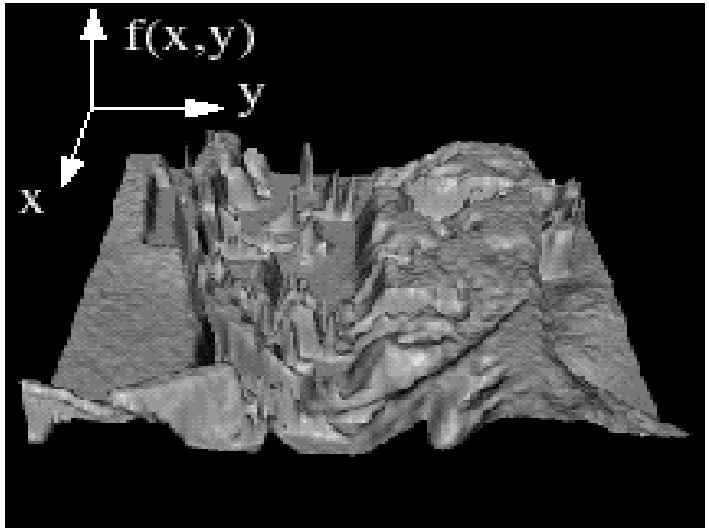
$x \rightarrow \max(0, W^1 x) \rightarrow h^1 \rightarrow \max(0, W^2 h^1) \rightarrow h^2 \rightarrow W^3 h^2 \rightarrow o$

All you need is gradients

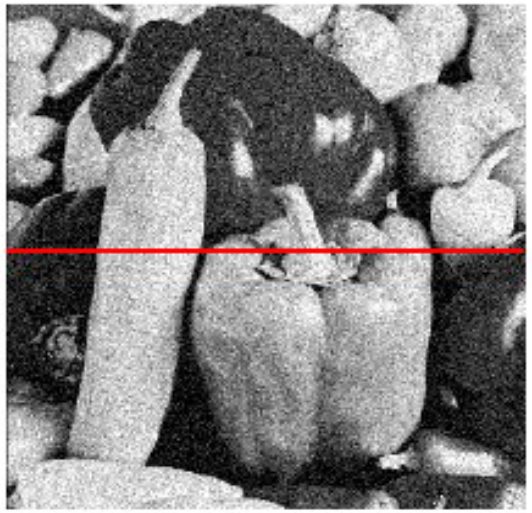


Images as functions





Gaussian Noise

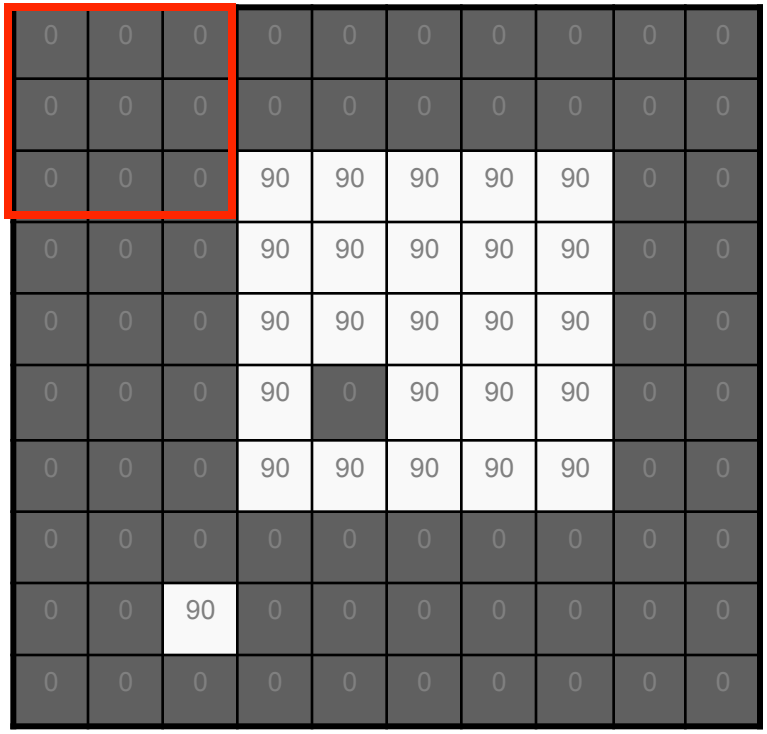


$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

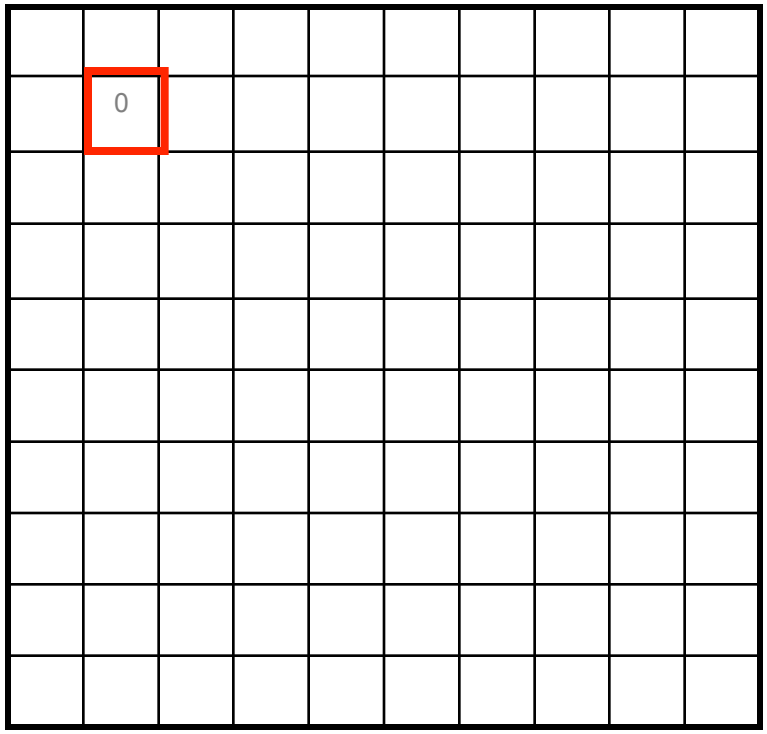
Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu, \sigma)$

Moving Average in 2D

$F[x, y]$



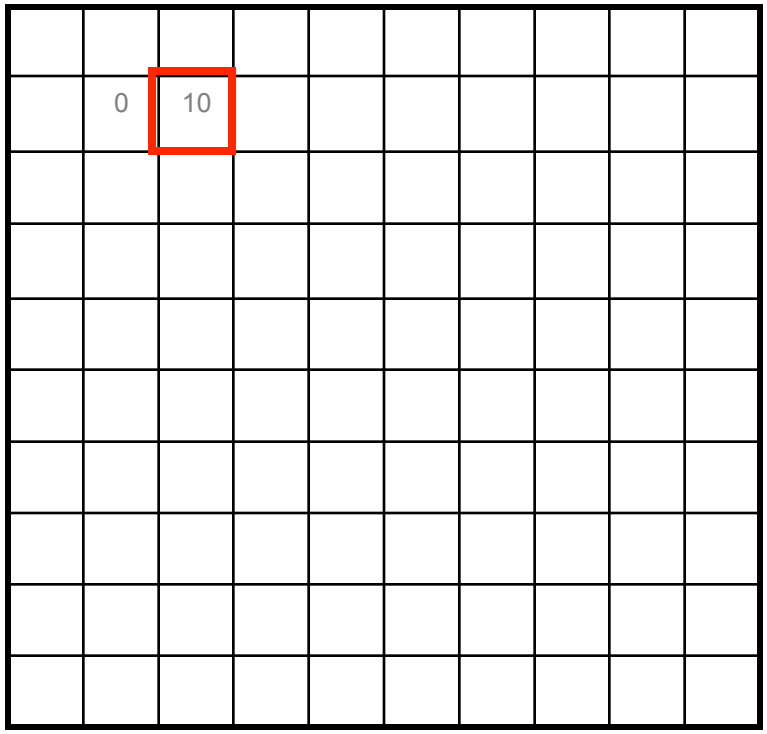
$G[x, y]$



Moving Average in 2D

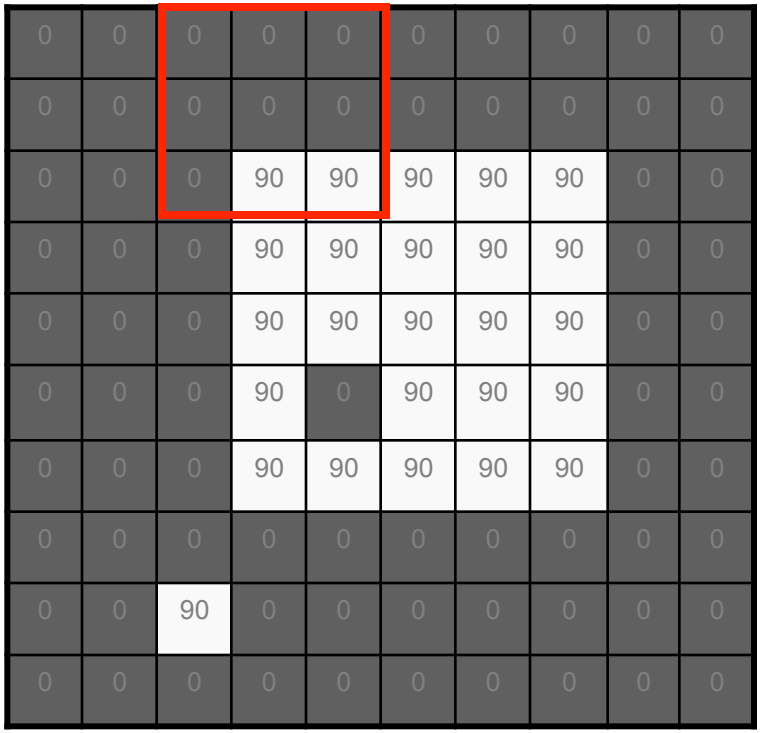
$F[x, y]$

$G[x, y]$

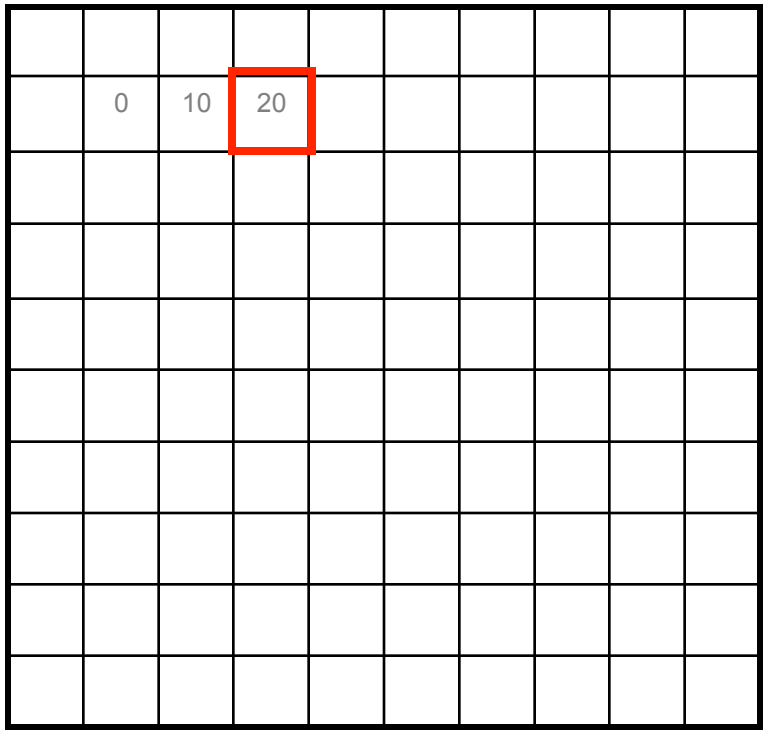


Moving Average in 2D

$F[x, y]$

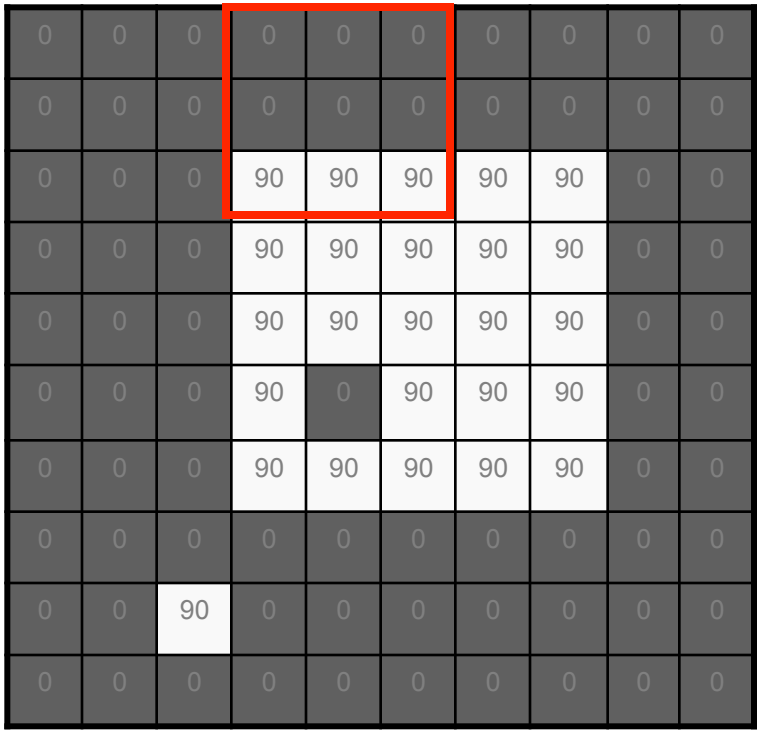


$G[x, y]$

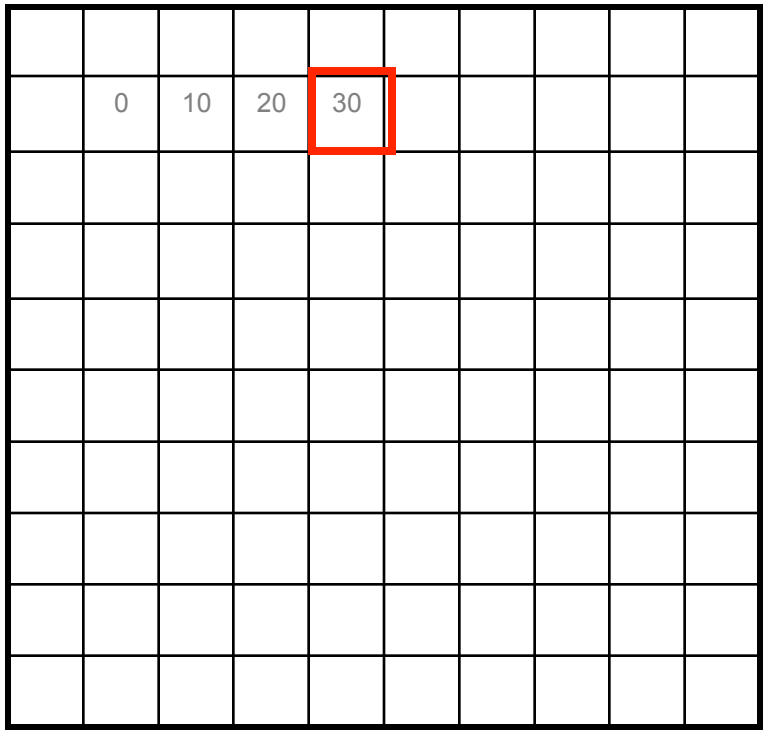


Moving Average in 2D

$F[x, y]$

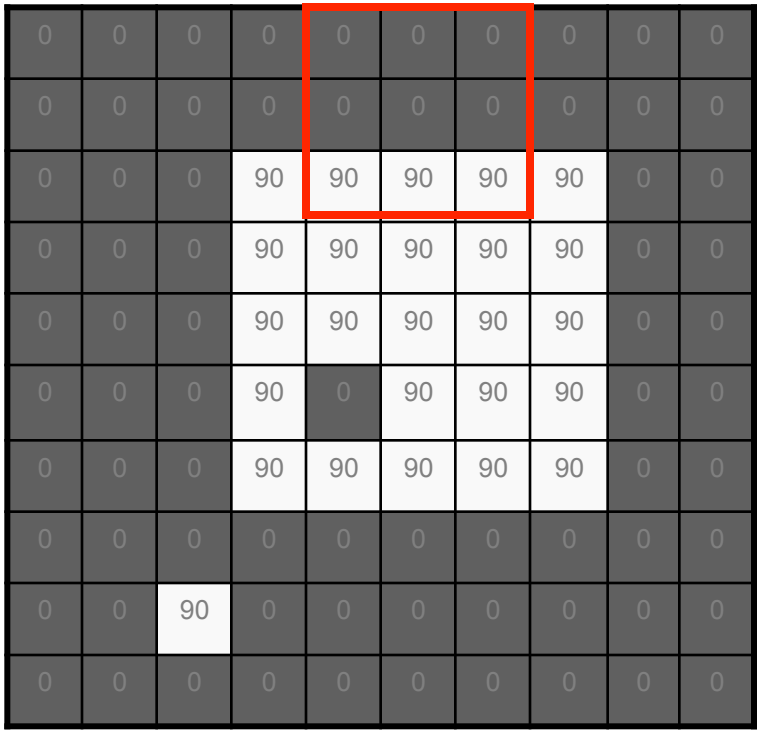


$G[x, y]$

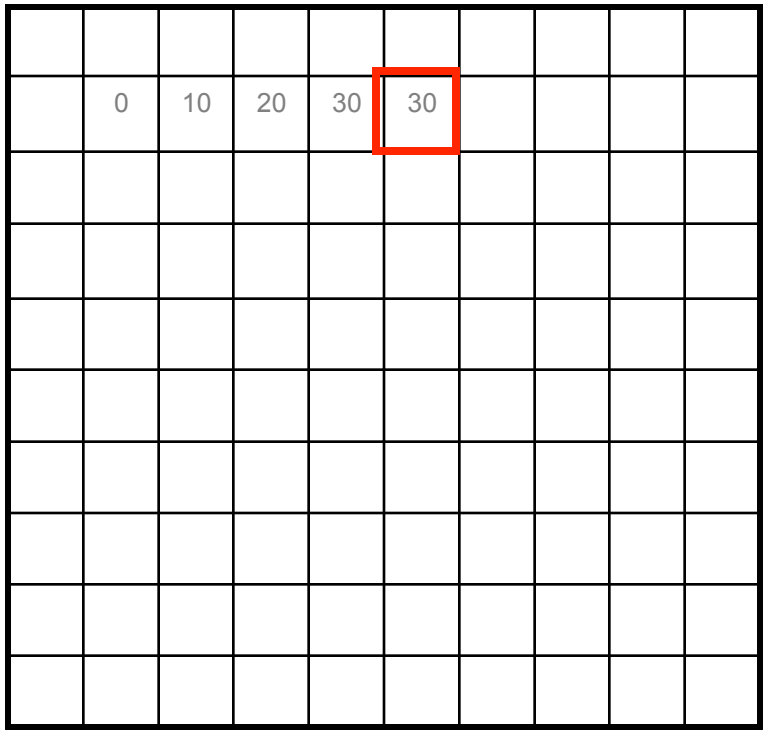


Moving Average in 2D

$F[x, y]$



$G[x, y]$



Moving Average in 2D

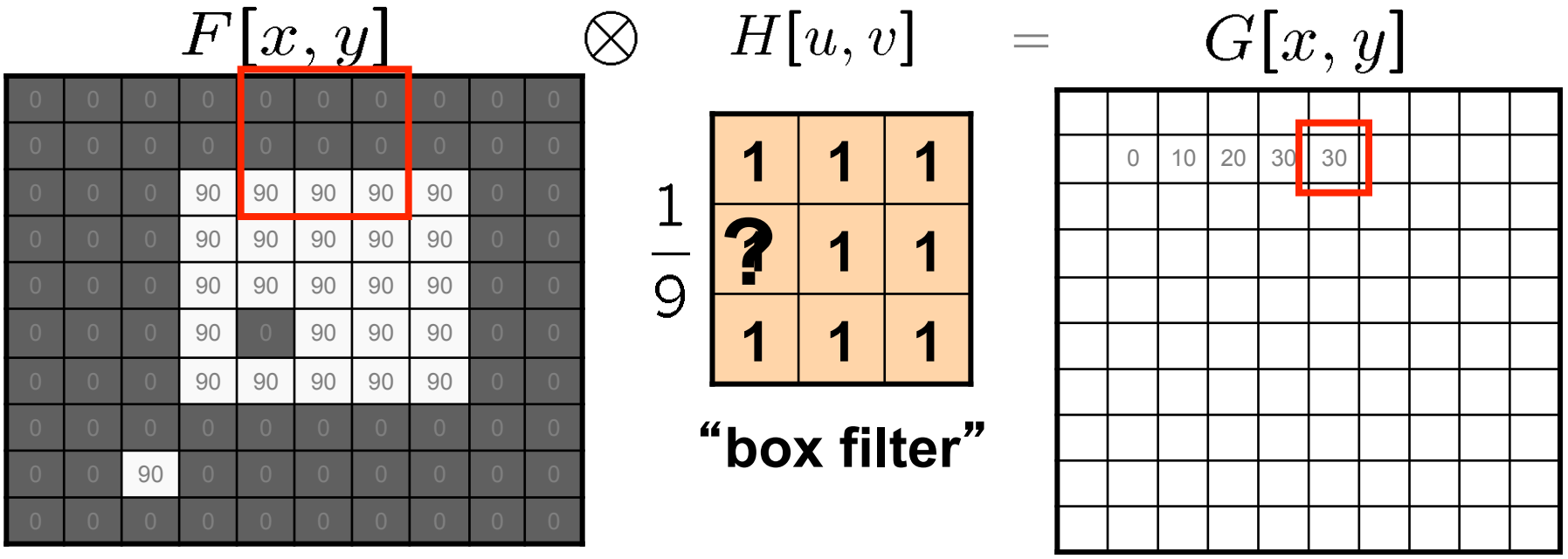
$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Averaging = Filtering



"box filter"

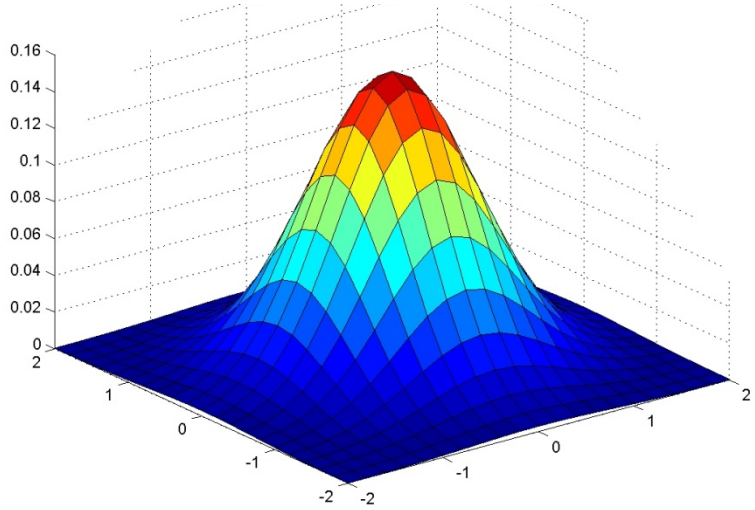
$$G = H \otimes F$$

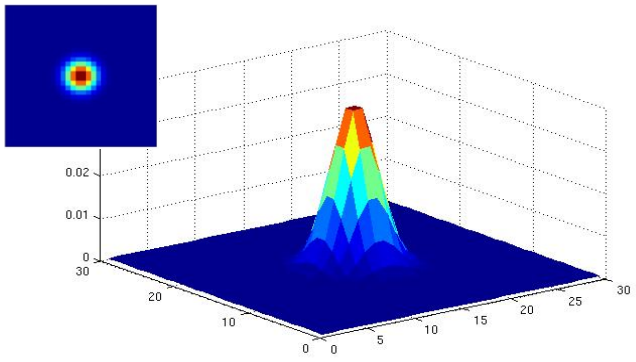
Gaussian Smoothing

- Gaussian kernel

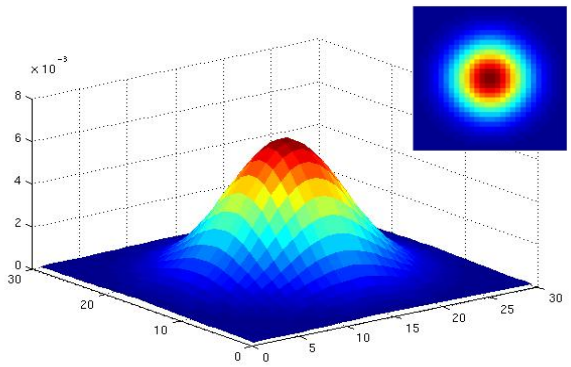
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Weighs nearby pixels more than distant ones



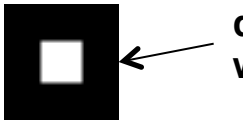


$\sigma = 2$ **with** 30×30 **kernel**



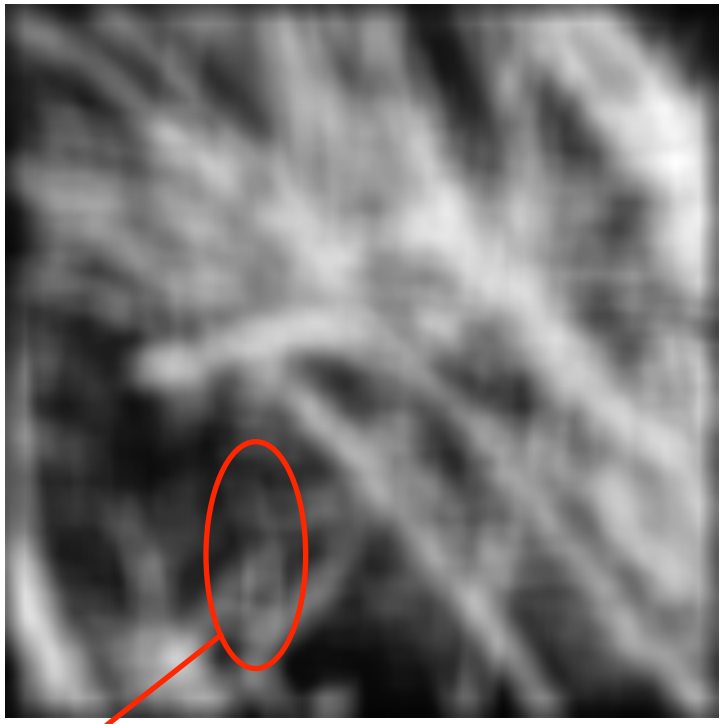
$\sigma = 5$ **with** 30×30 **kernel**

Smoothing by Averaging



depicts box filter:
white = high value, black = low value

Original

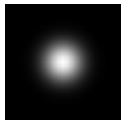


Filtered

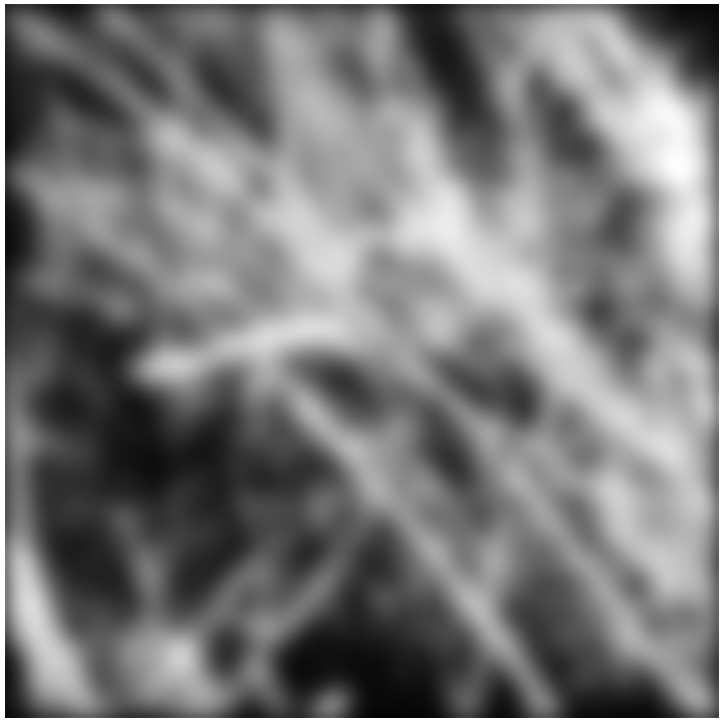
"Ringing" artifacts!

Image Source: Forsyth & Ponce

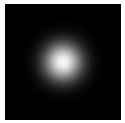
Smoothing by Gaussian filtering



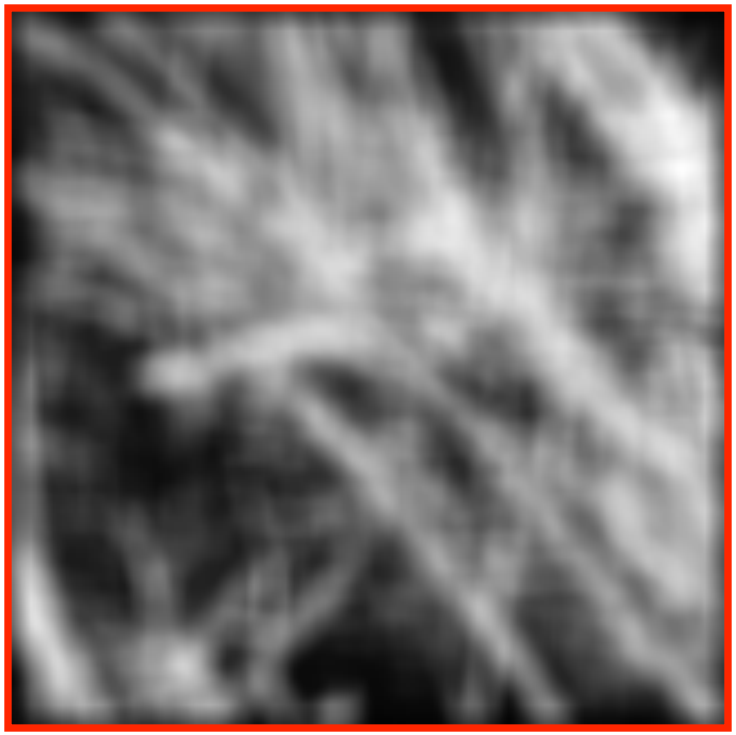
Original



Filtered

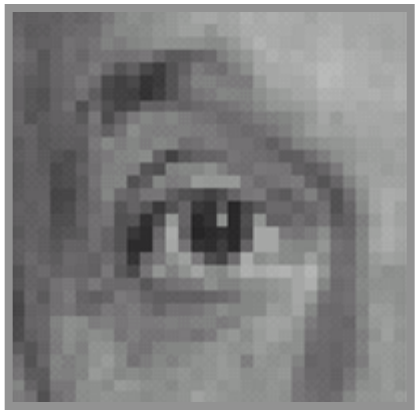


Original

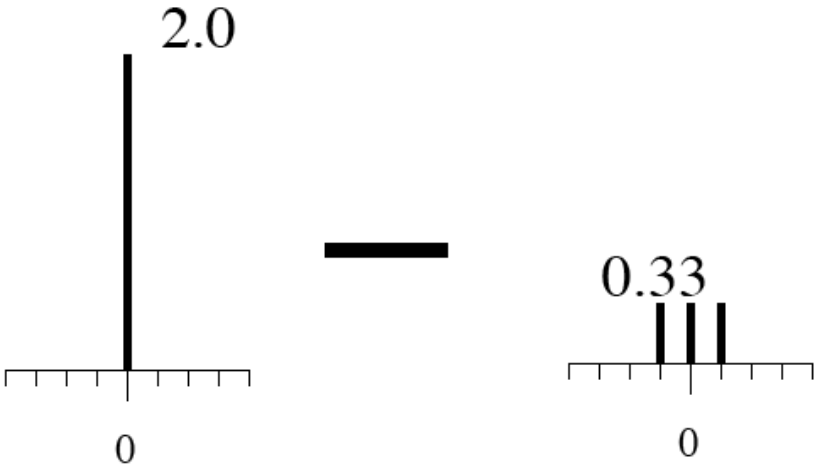


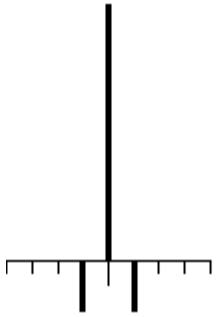
Filtered

Sharpening Filter



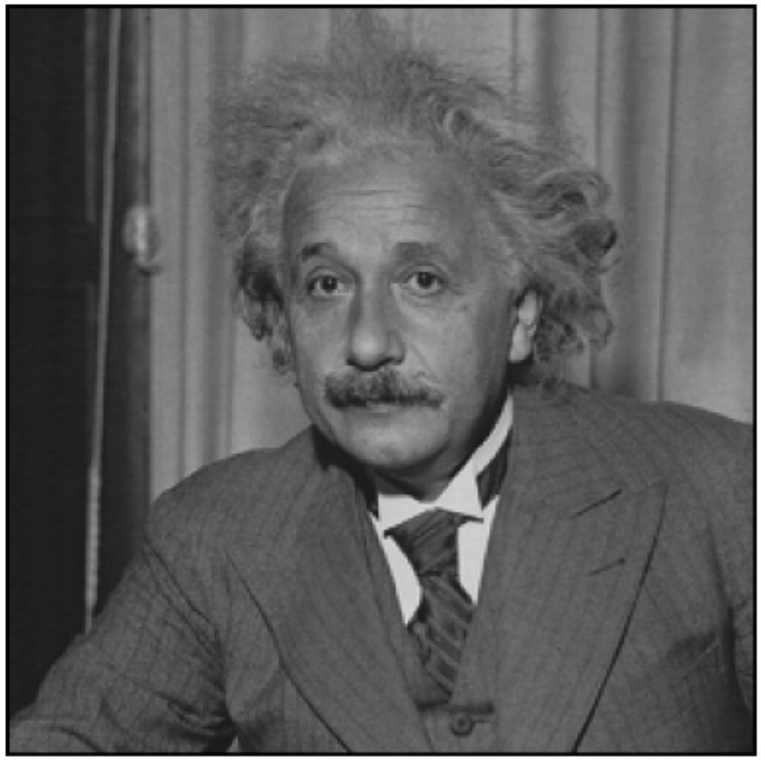
Original



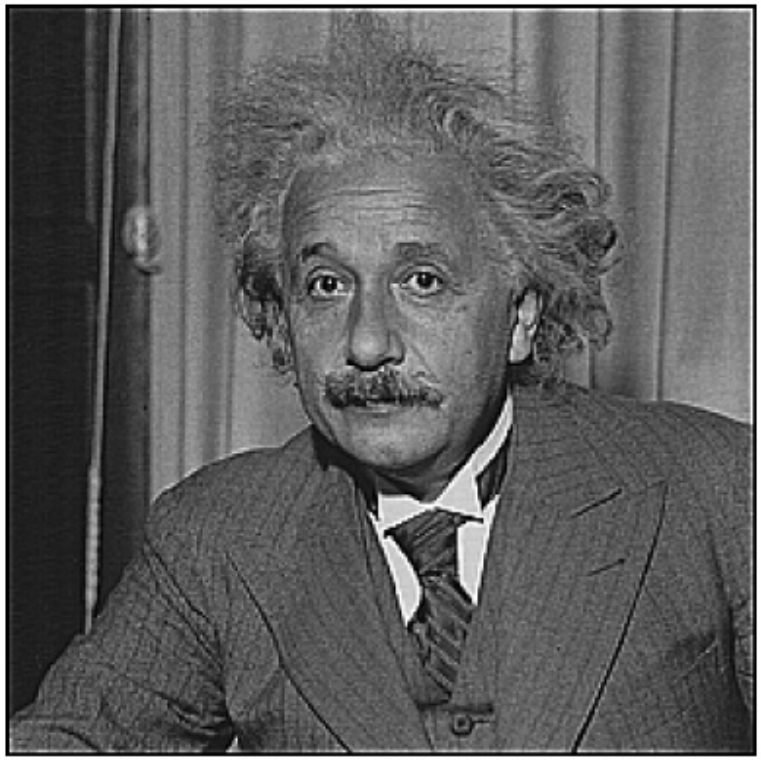


Sharpening filter

Sharpening Filter



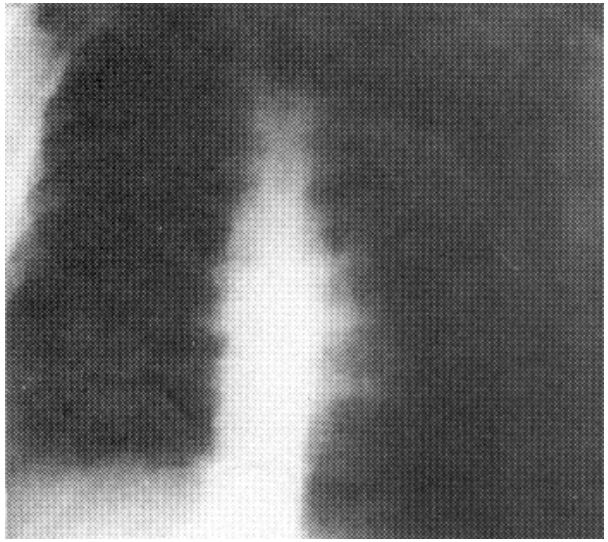
before

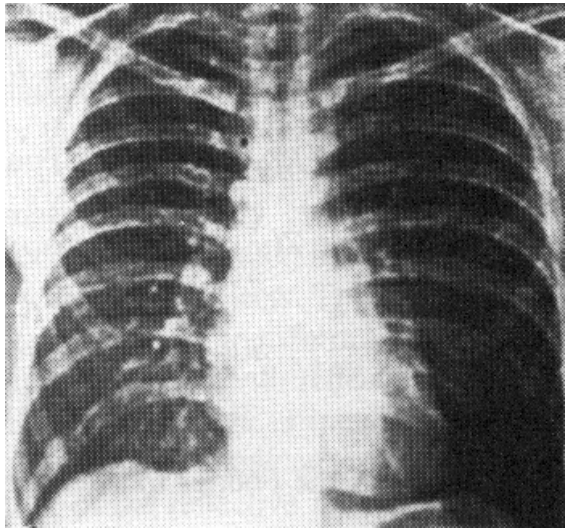


after

Image processing application

Original

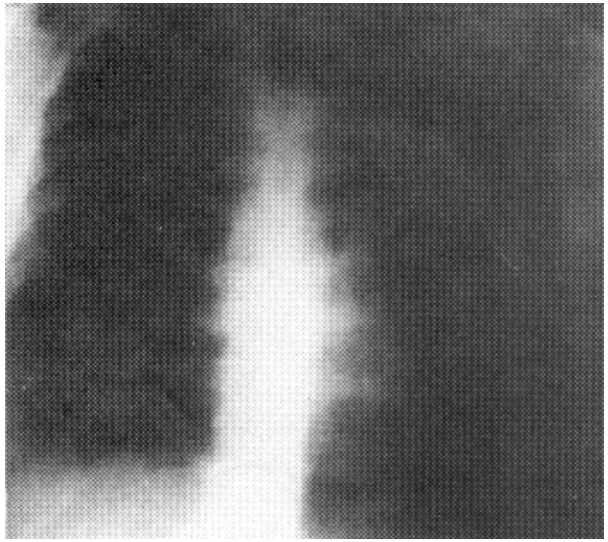


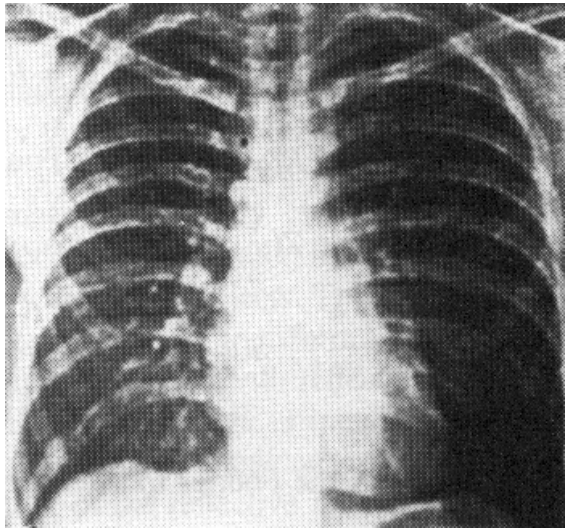


High Frequency Emphasis
+
Histogram Equalization

Image processing application

Original





High Frequency Emphasis
+
Histogram Equalization

Filters, filters, filters

Decades of research in image processing

Demos:

http://setosa.io/ev/image-kernels/

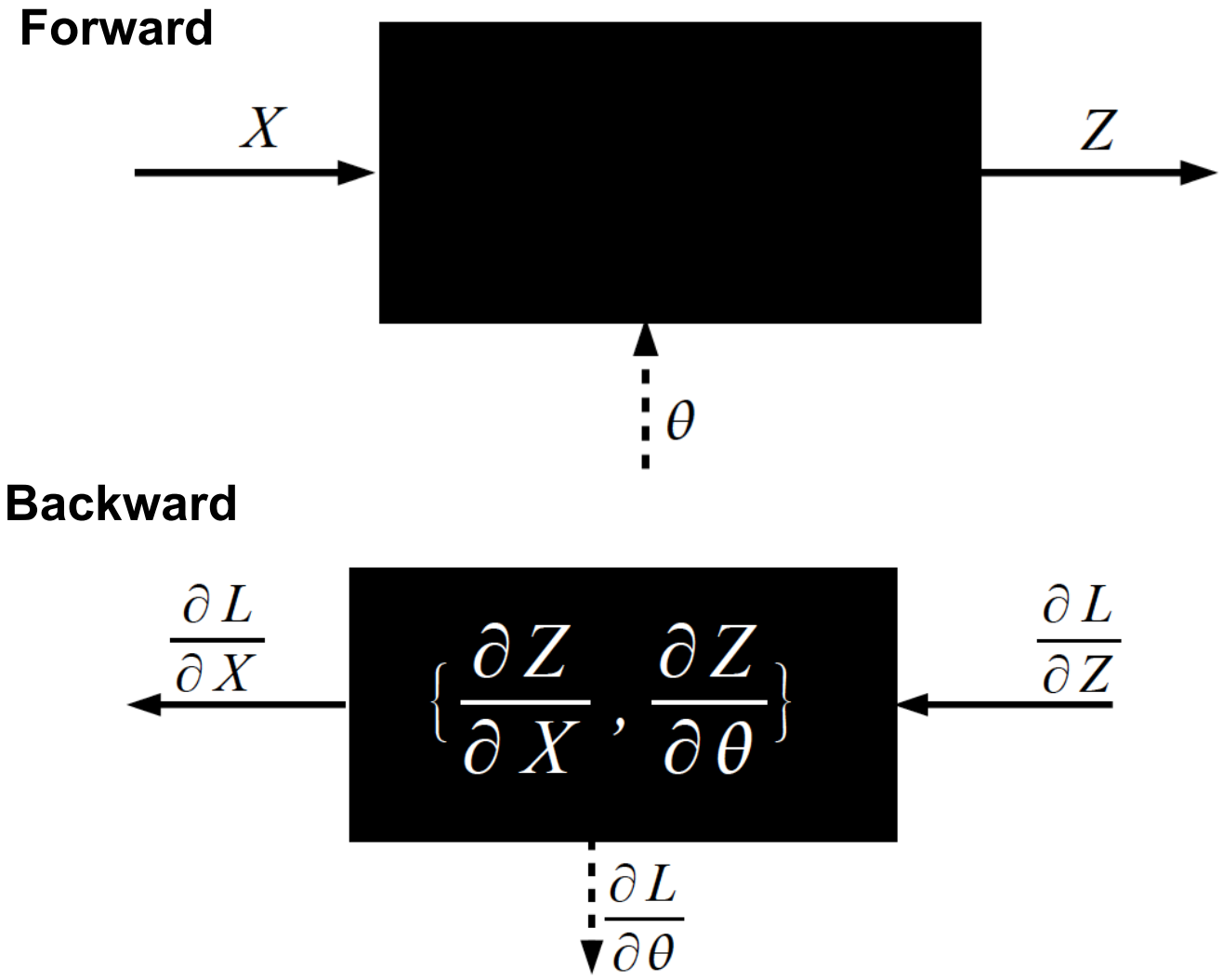
Methods:

http://www.ipol.im/

Open-ended: how does it connect with a given application?

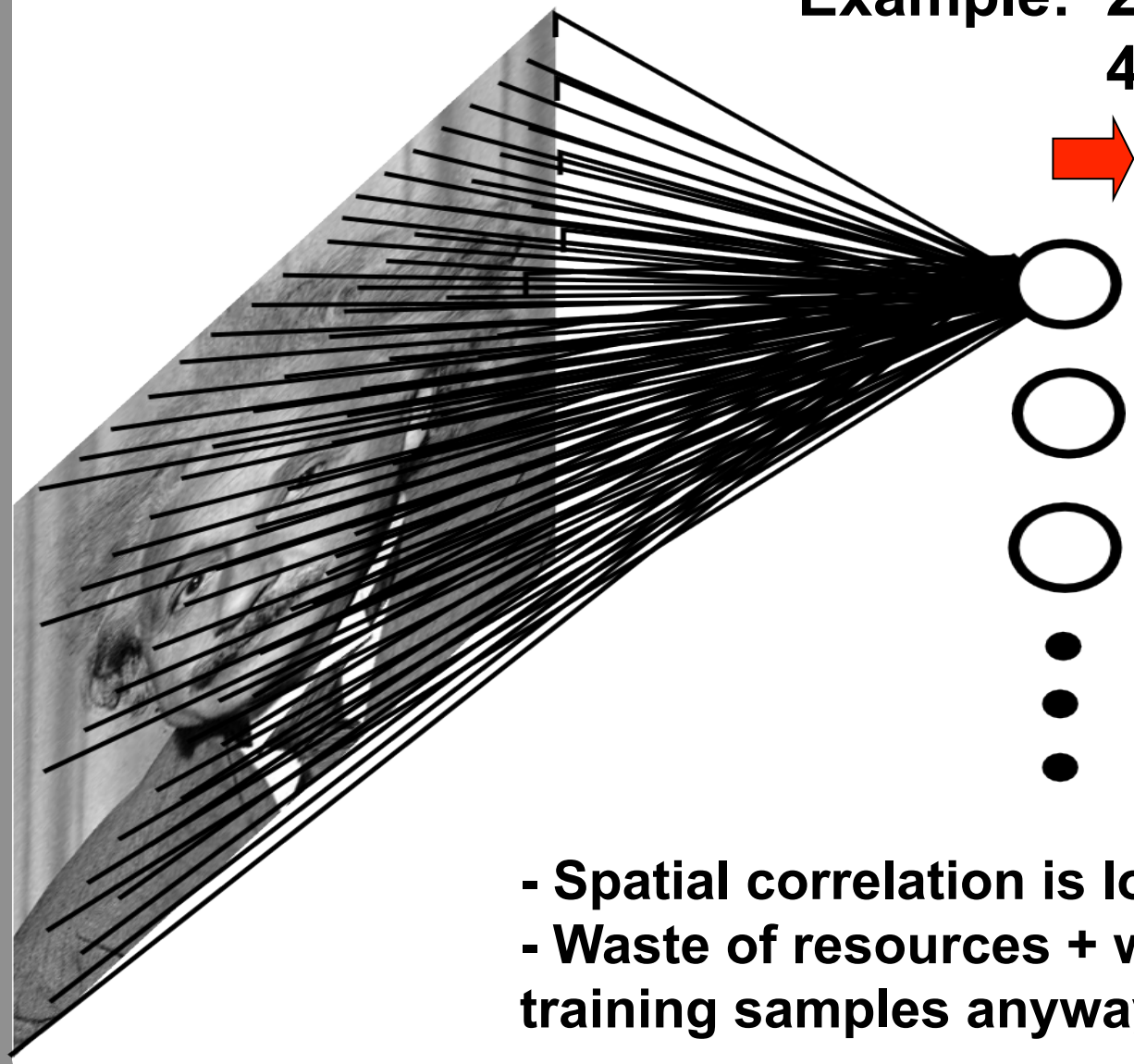
Question: can we learn how to do this?
(or any other type of image processing)

Answer: all you need is gradients



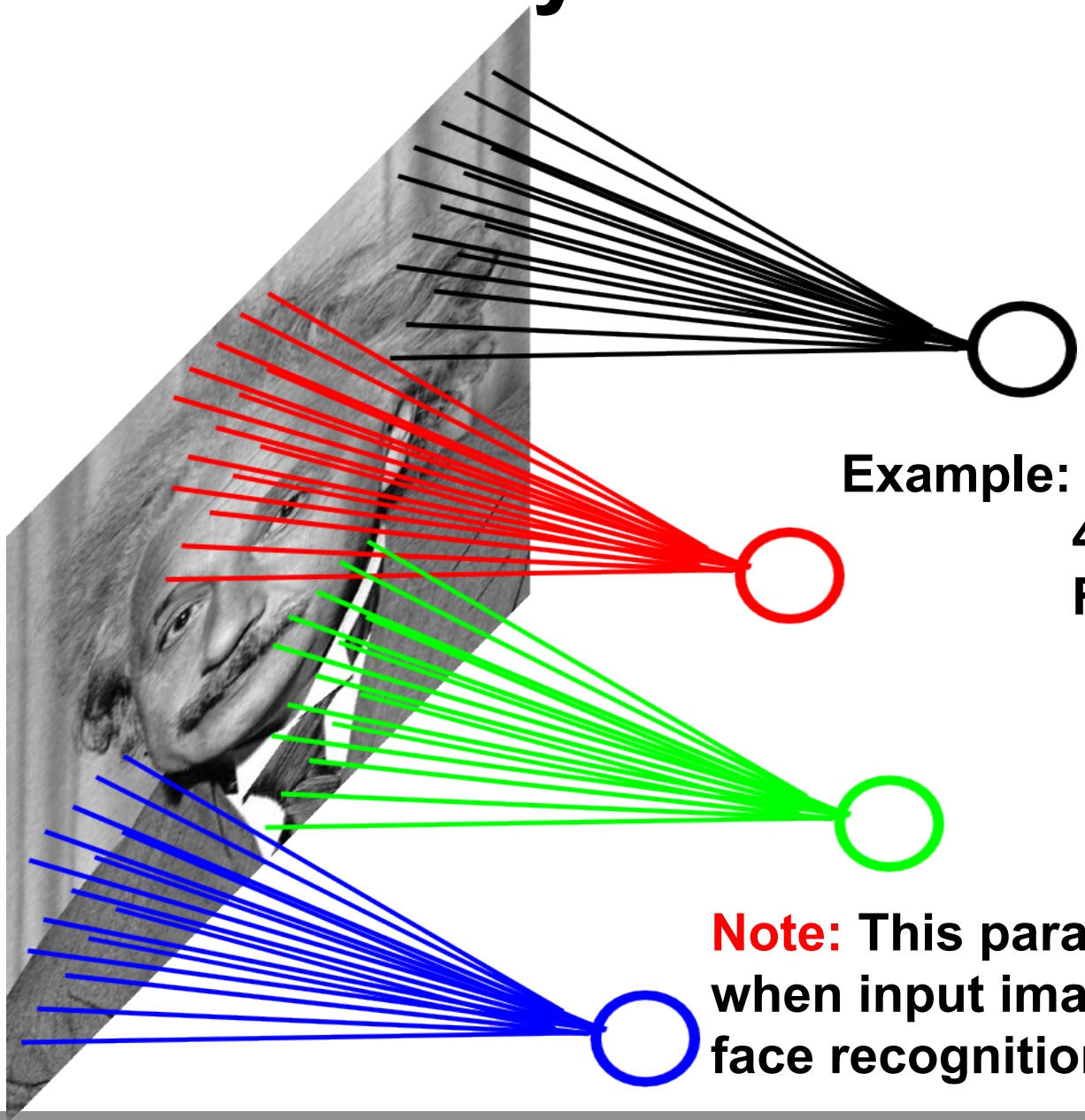
Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!



- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

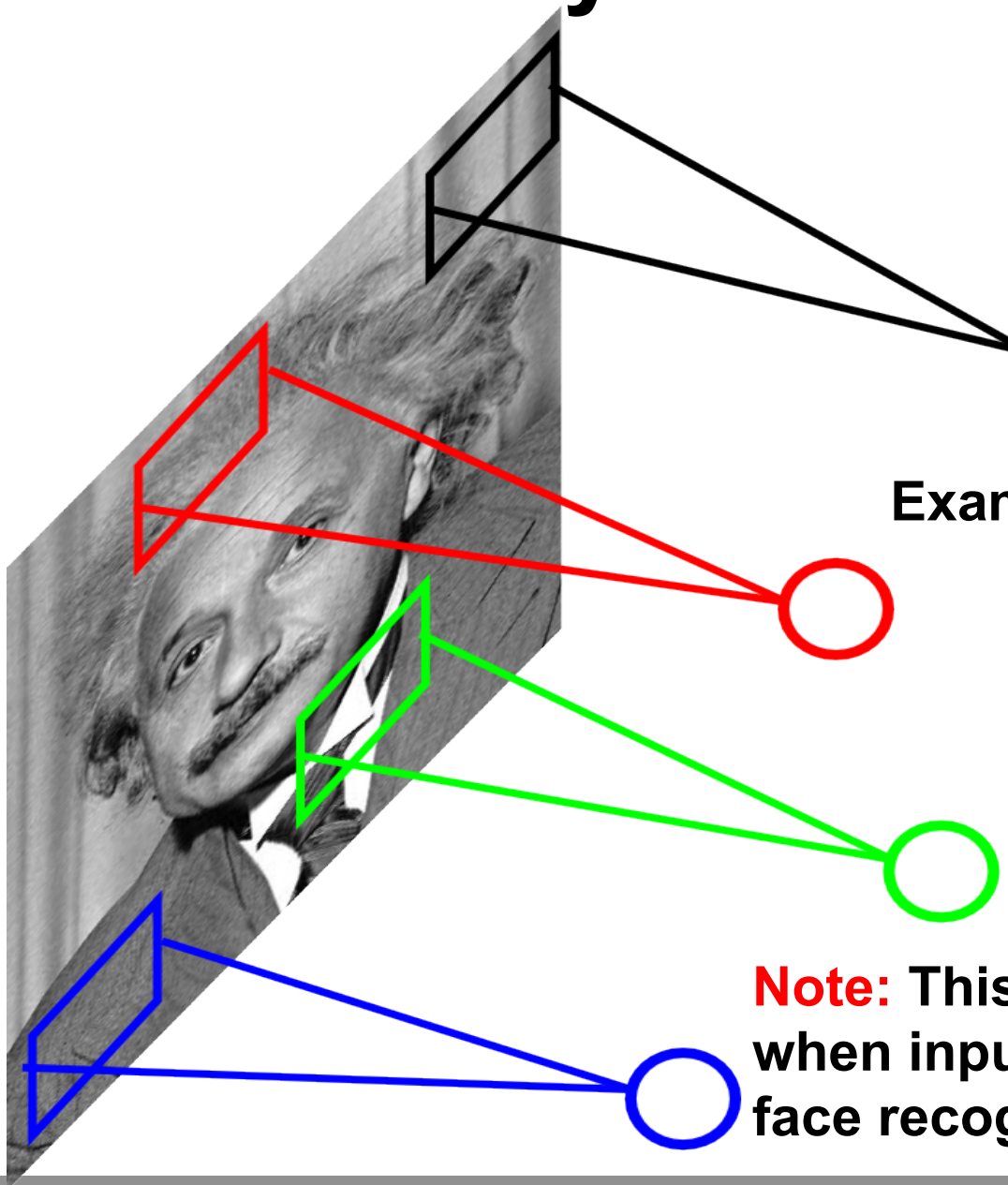
Locally Connected Layer



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).

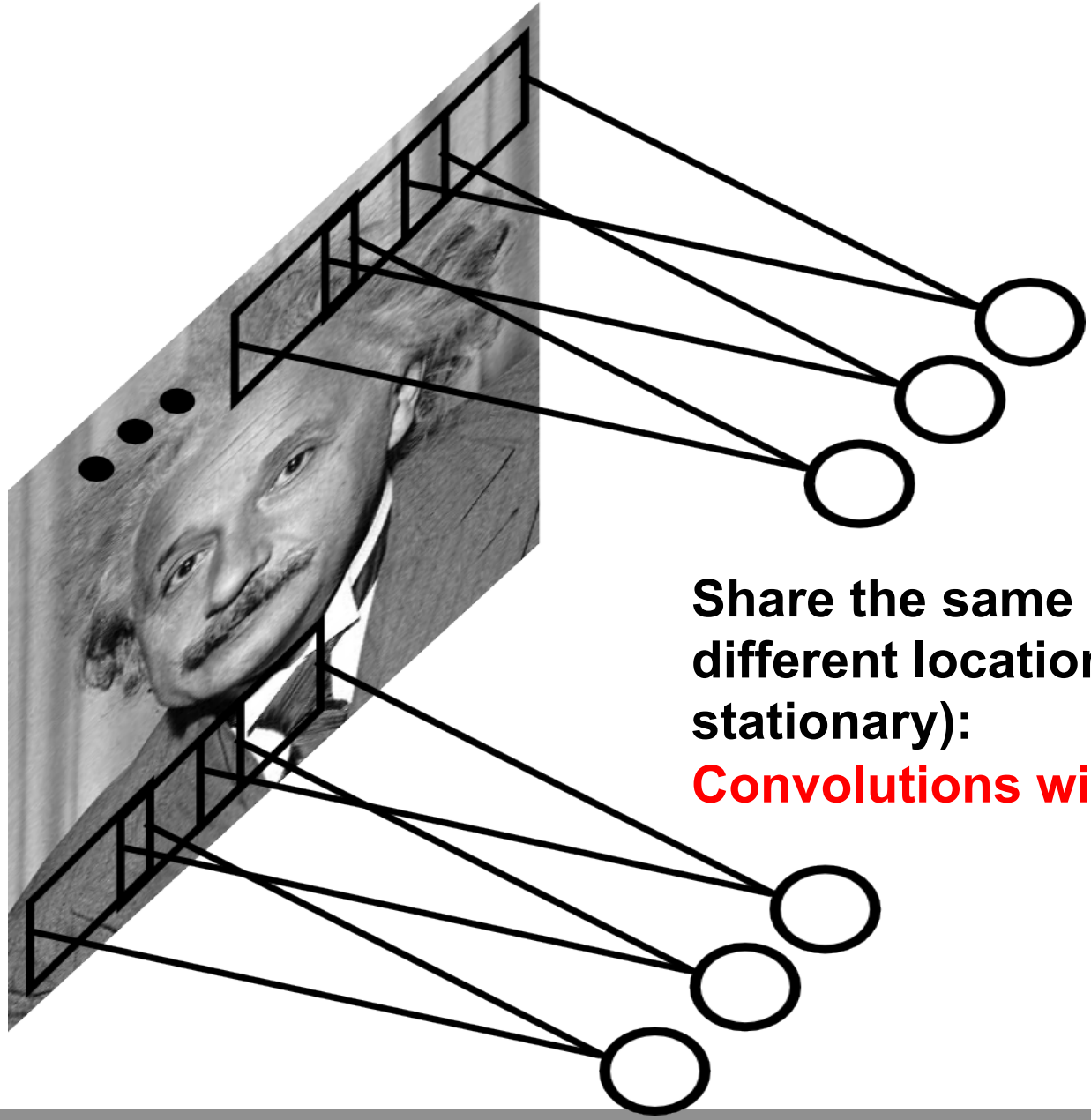
Locally Connected Layer



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).

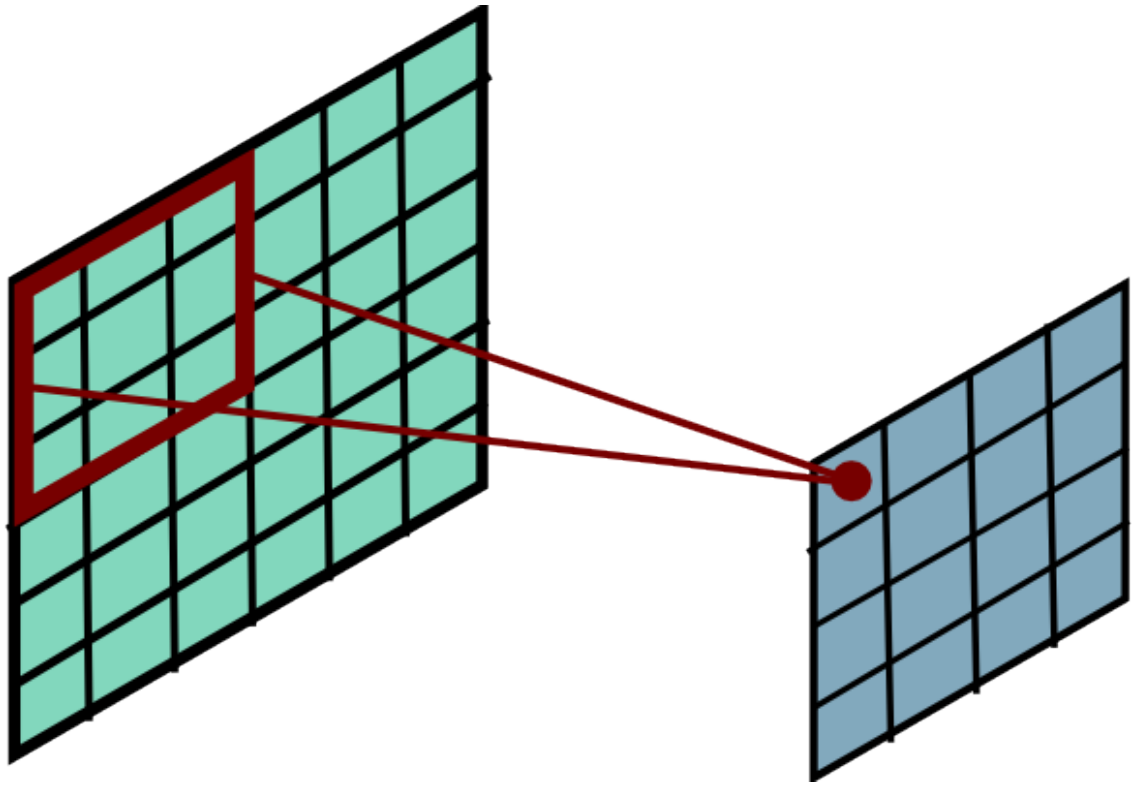
Convolutional Layer



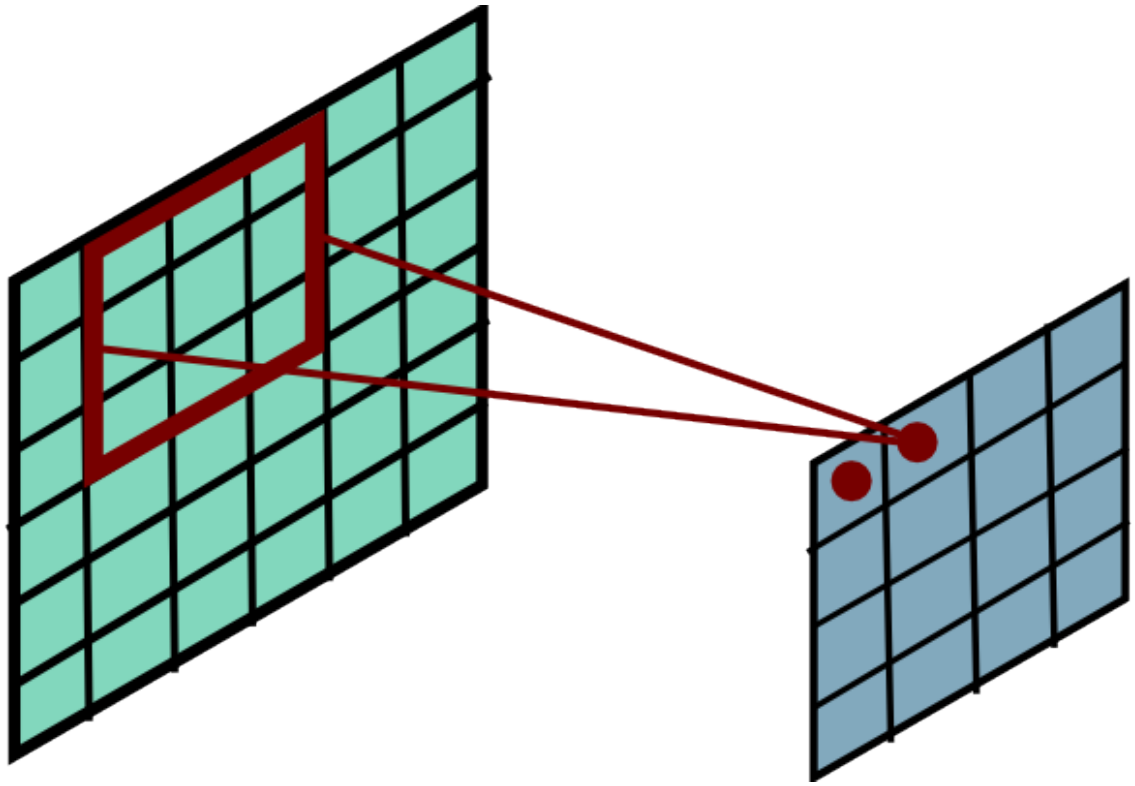
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

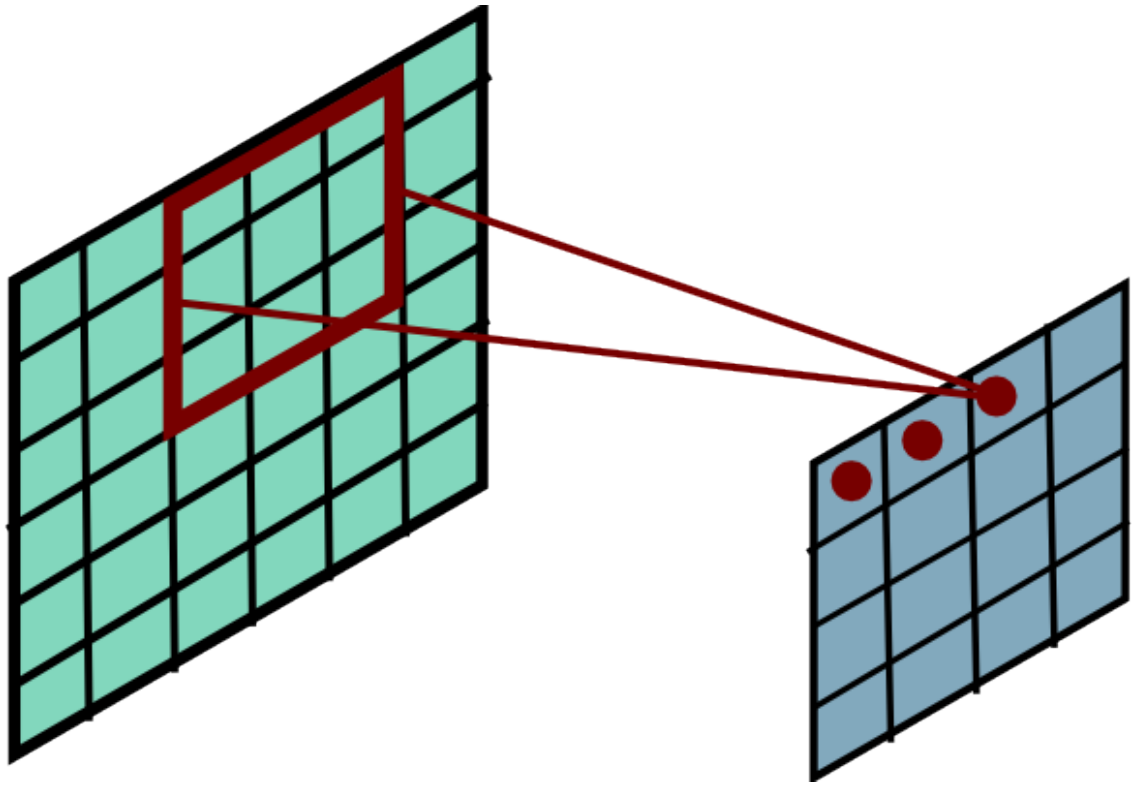
Convolutional Layer



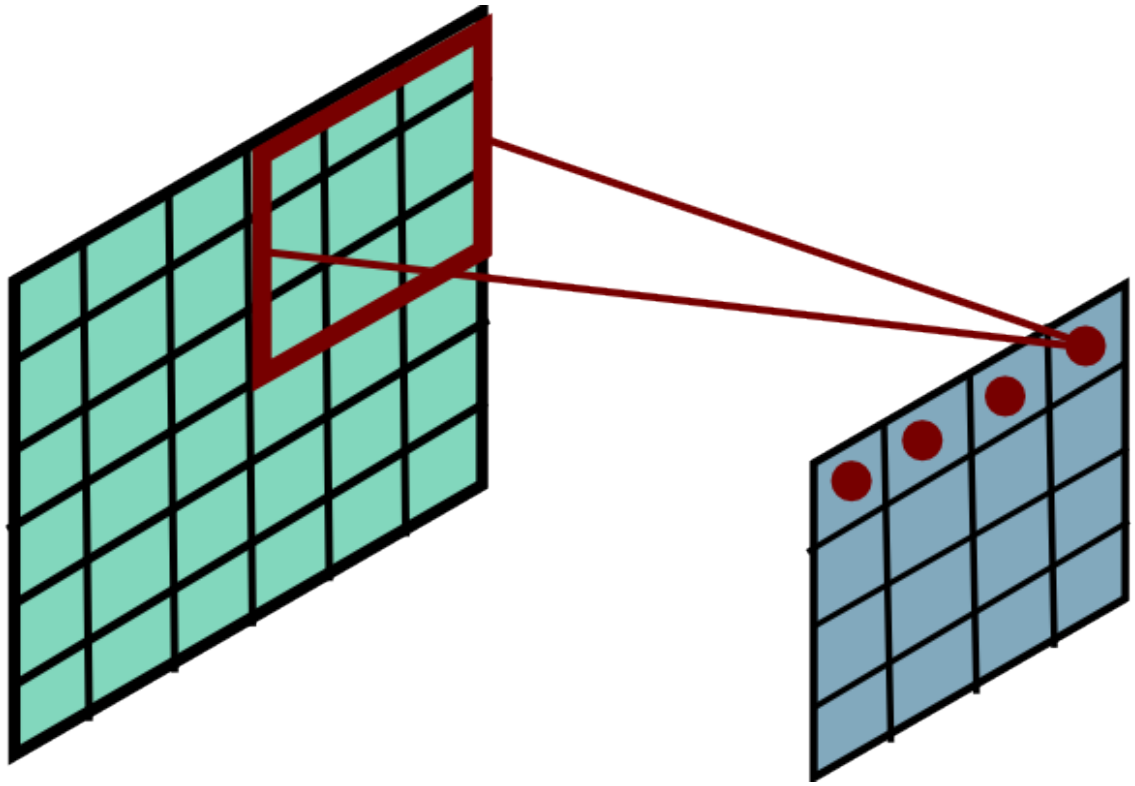
Convolutional Layer



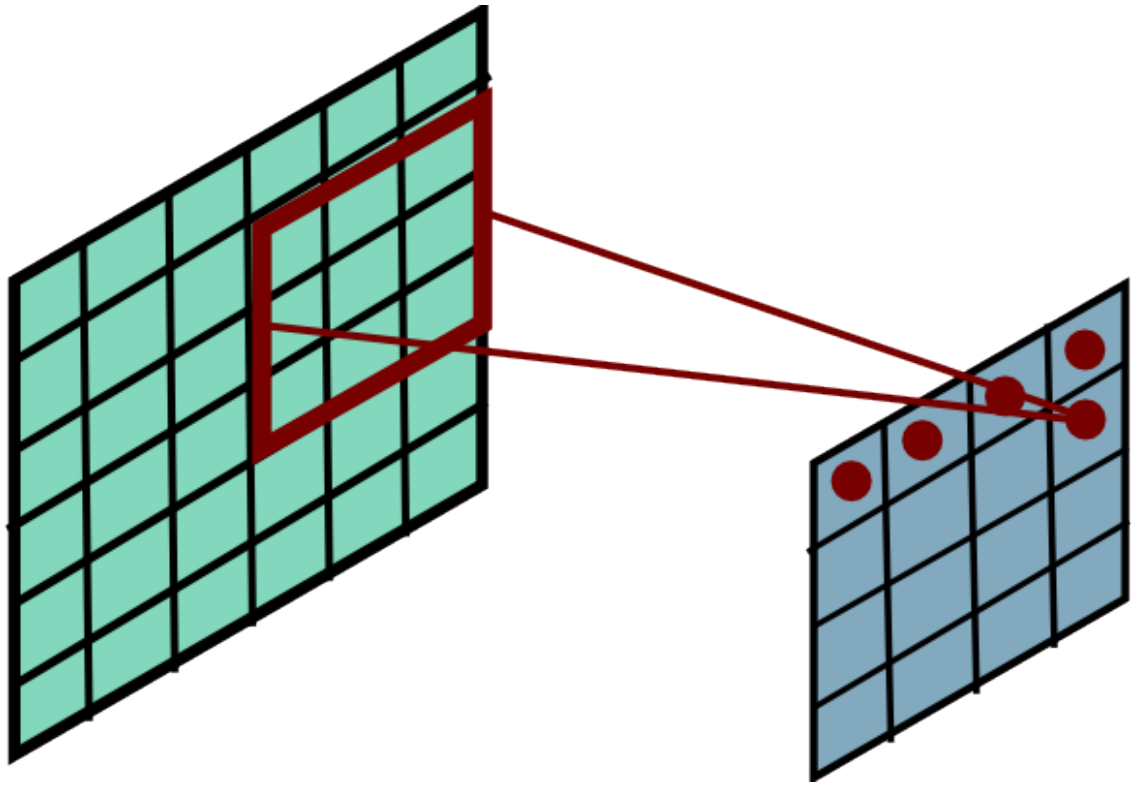
Convolutional Layer



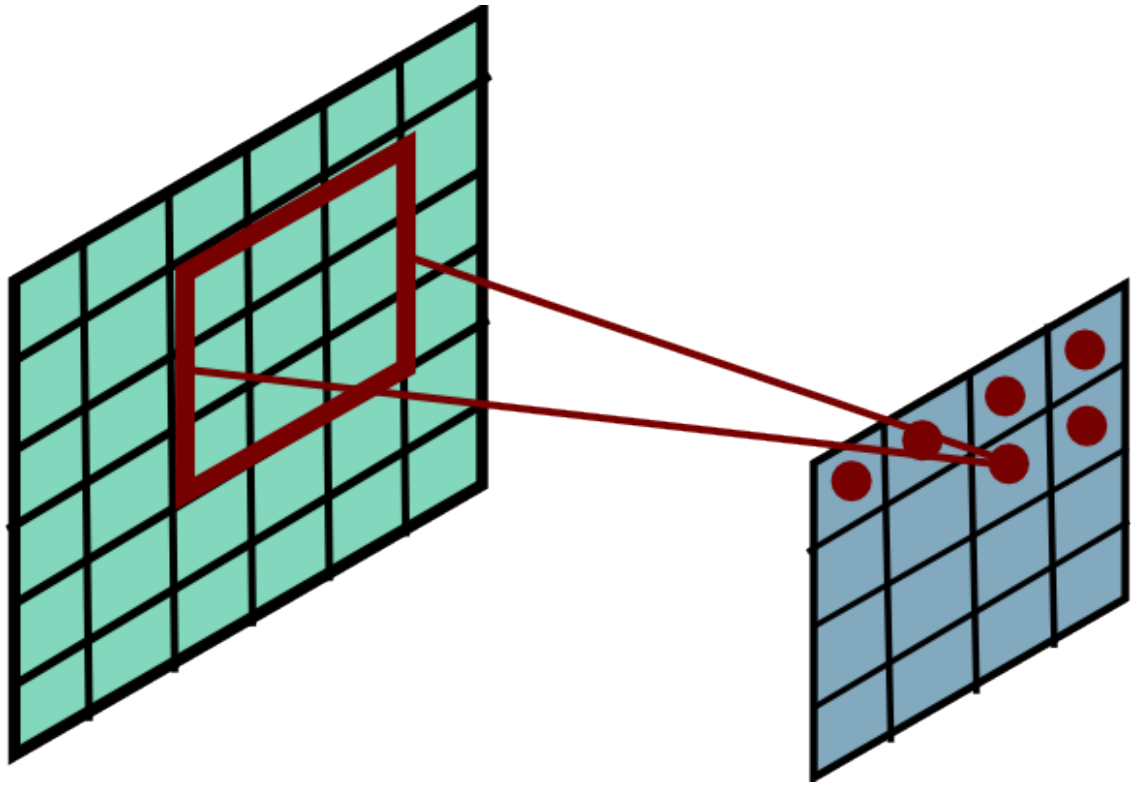
Convolutional Layer



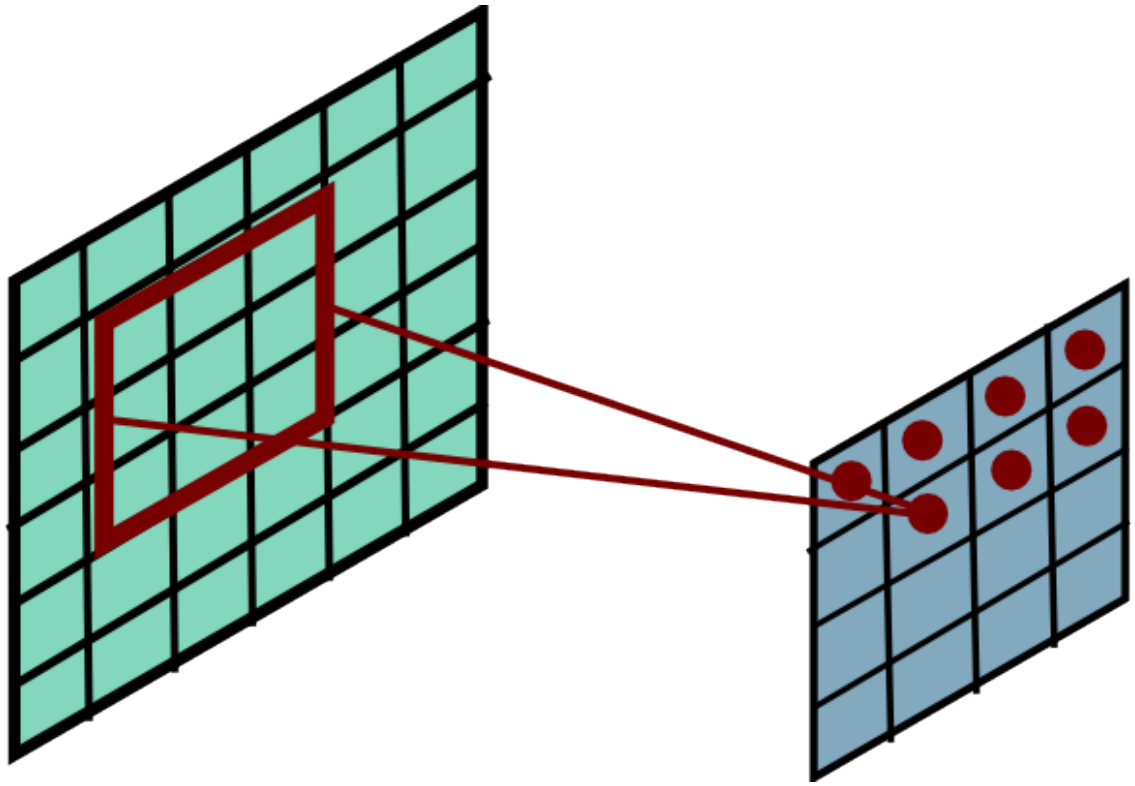
Convolutional Layer



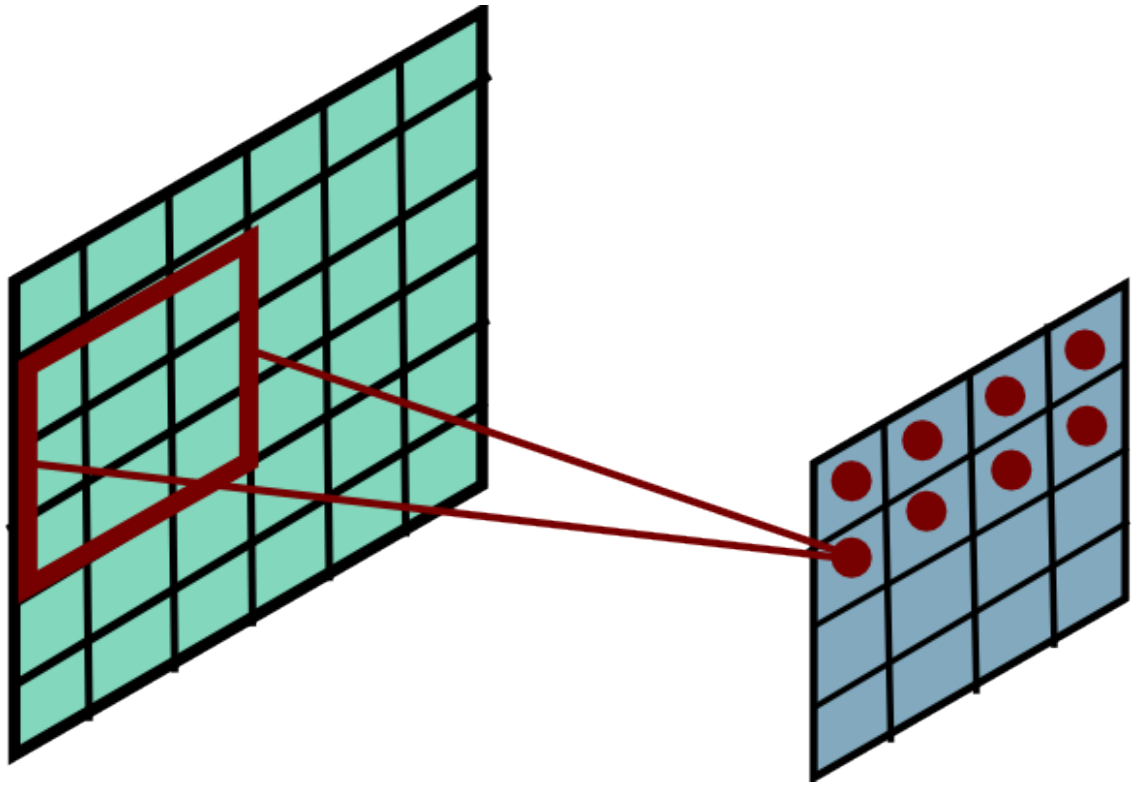
Convolutional Layer



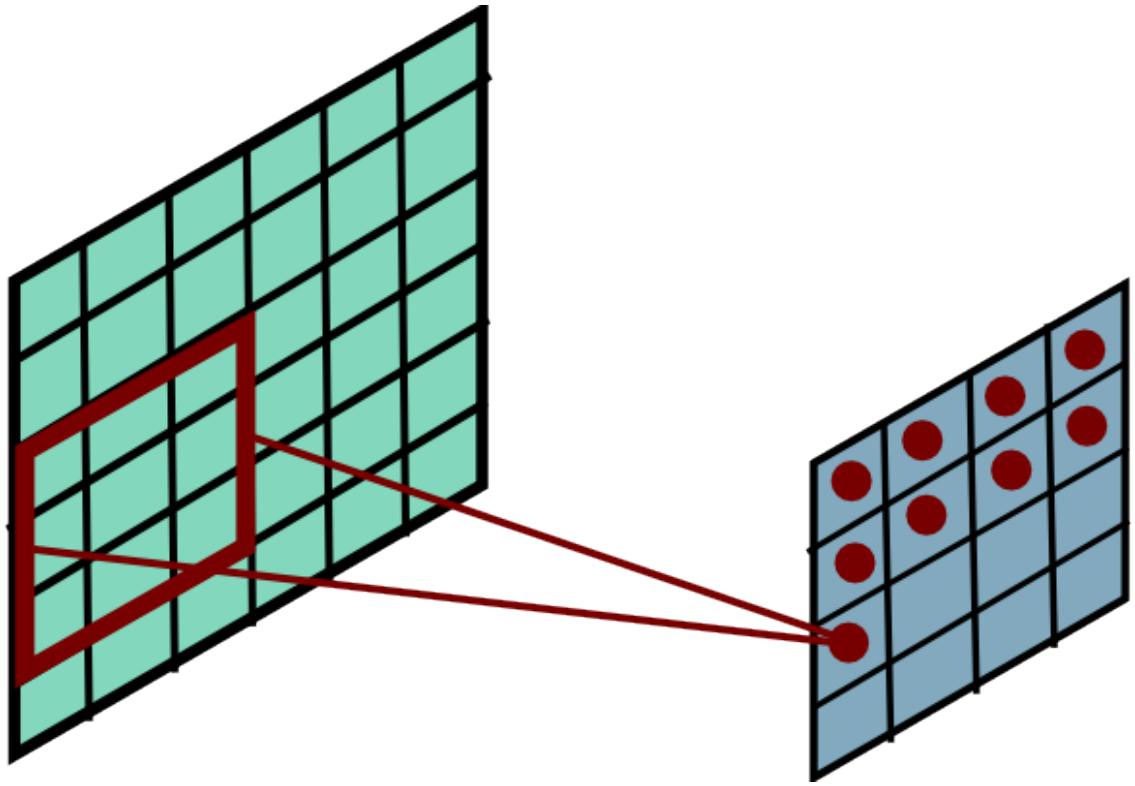
Convolutional Layer



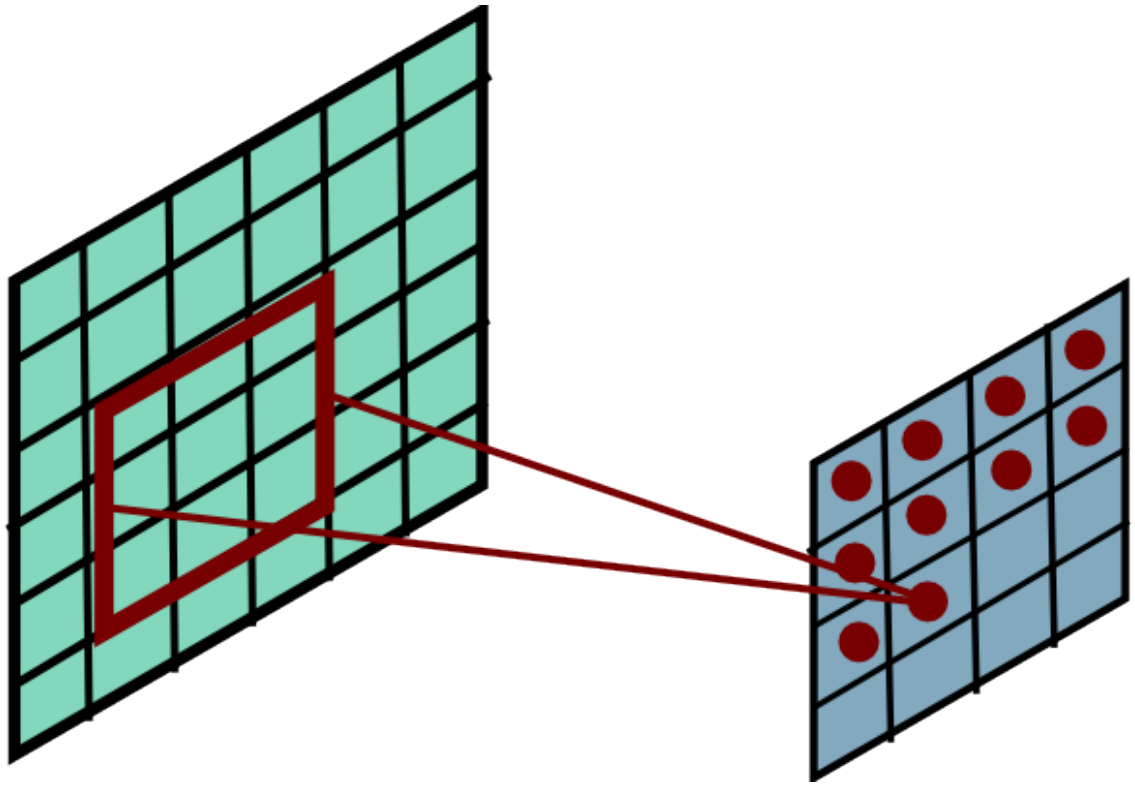
Convolutional Layer



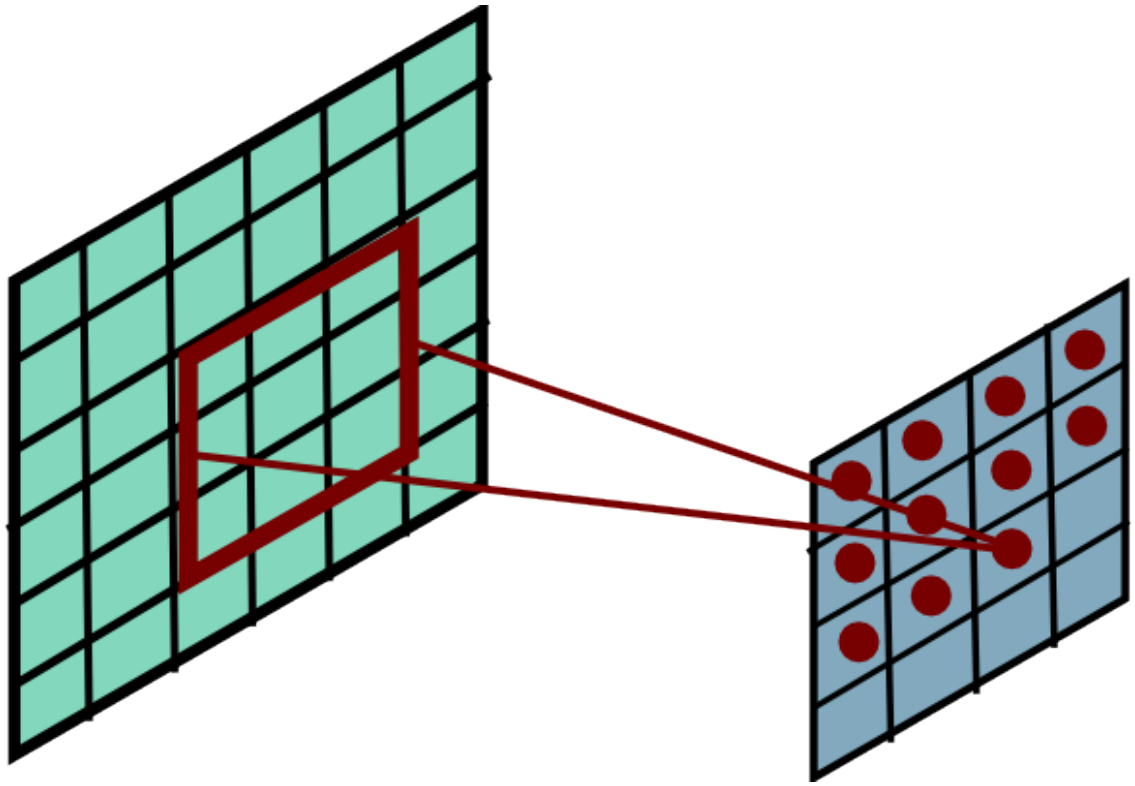
Convolutional Layer



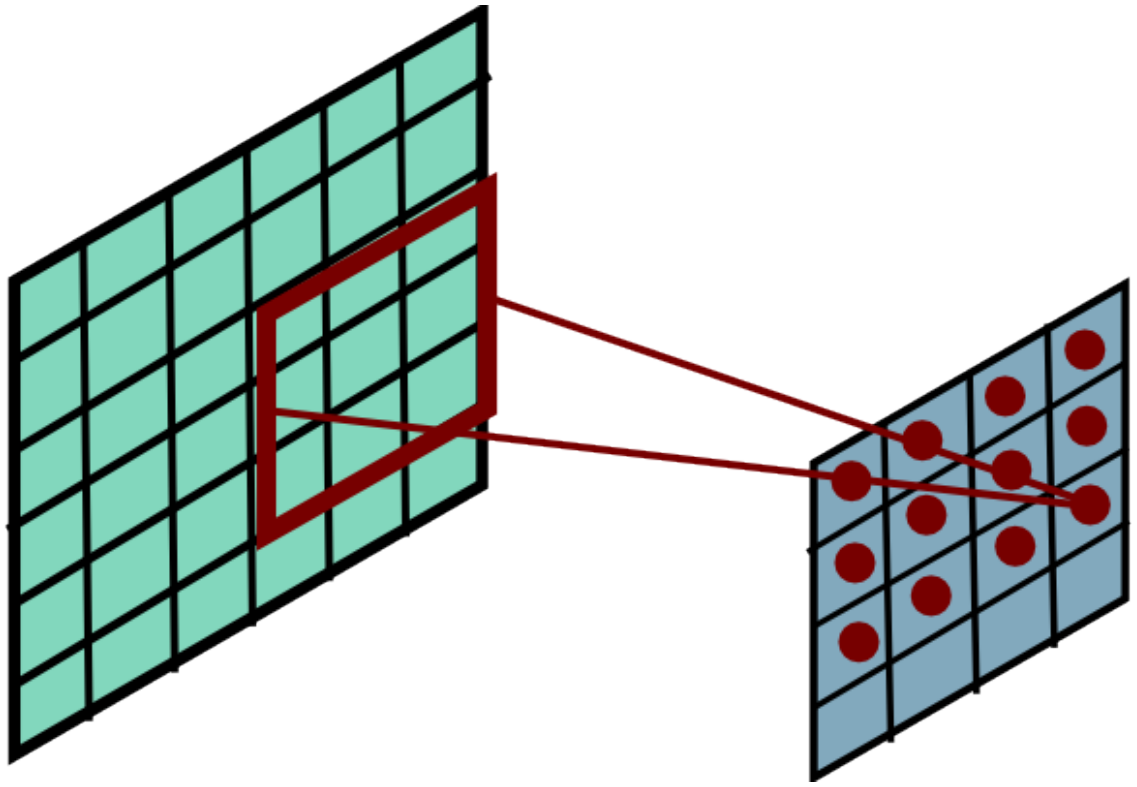
Convolutional Layer



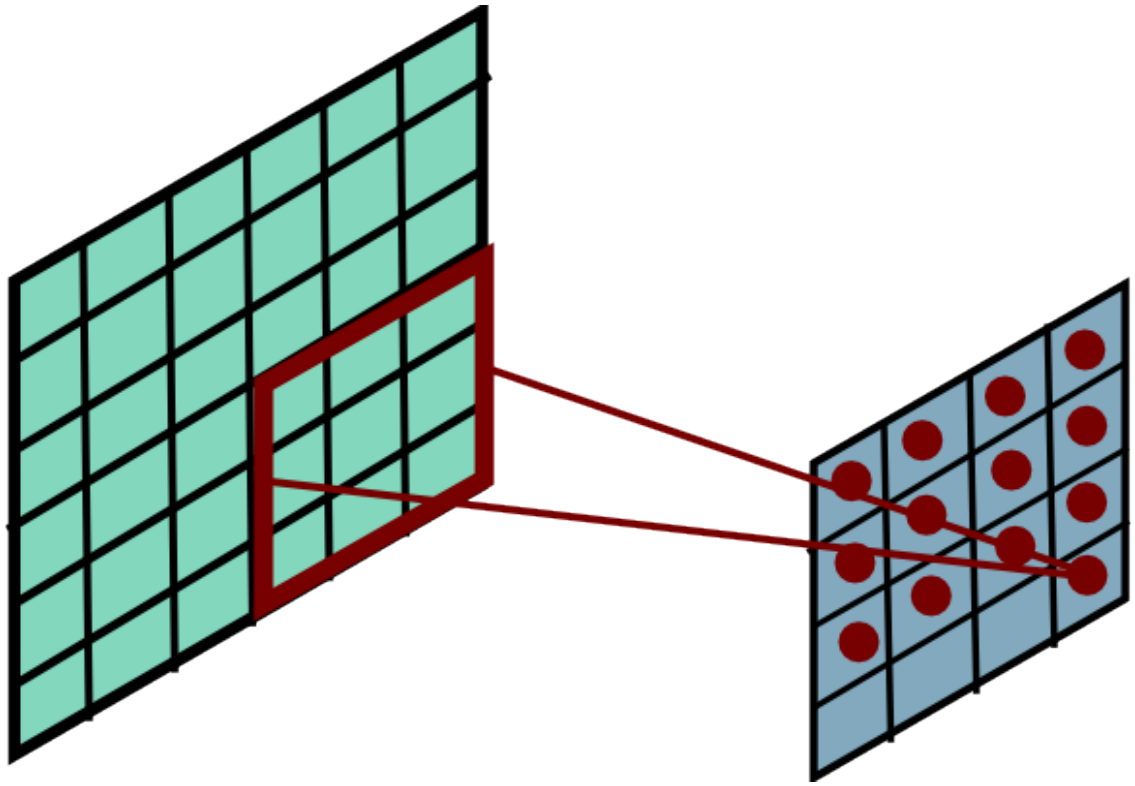
Convolutional Layer



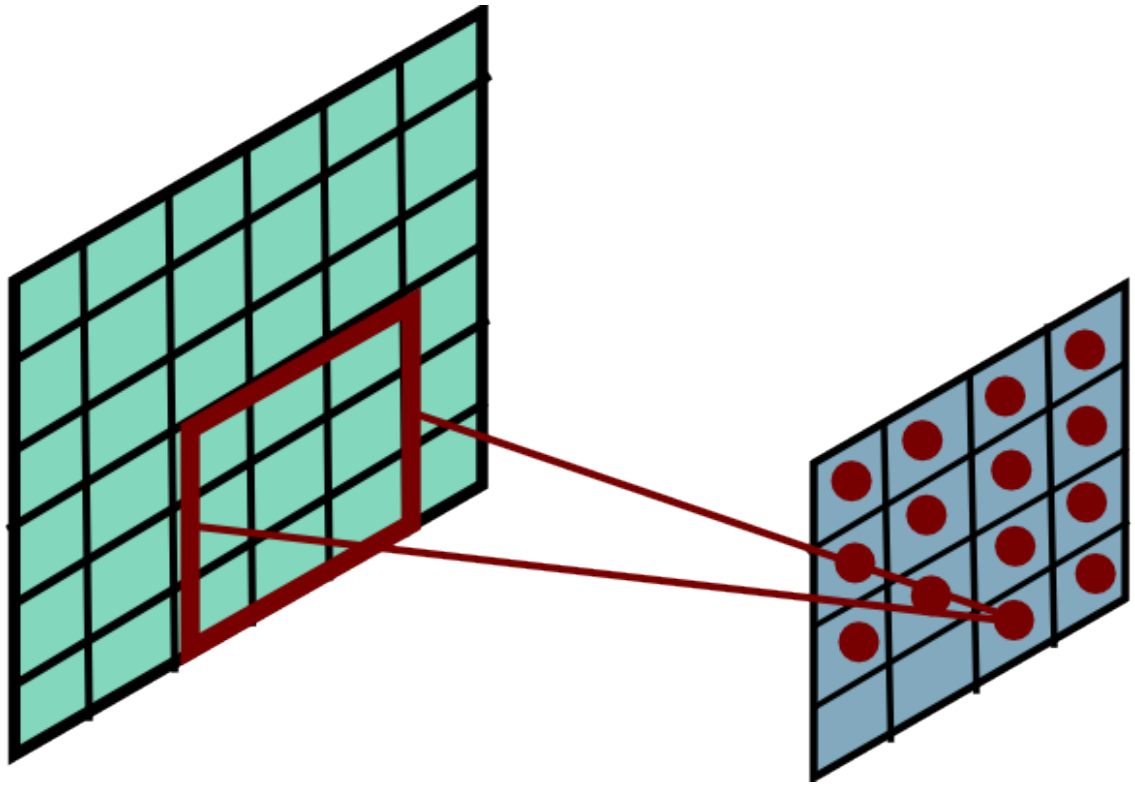
Convolutional Layer



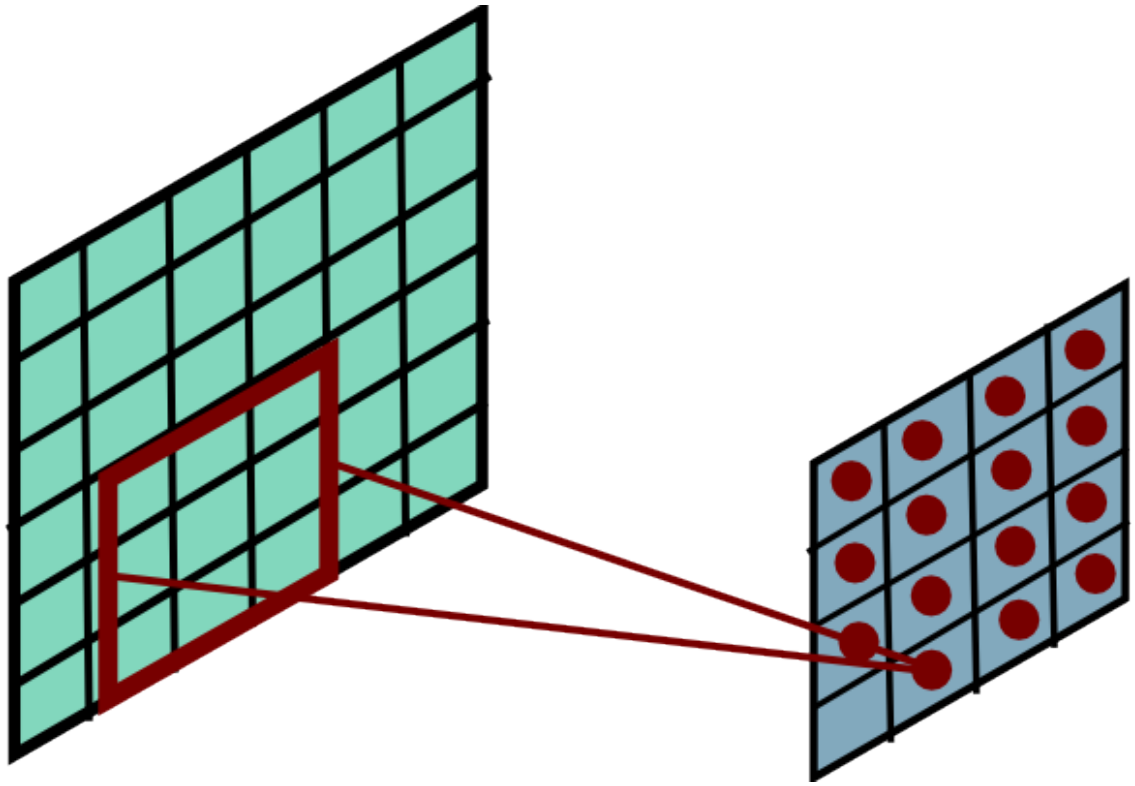
Convolutional Layer



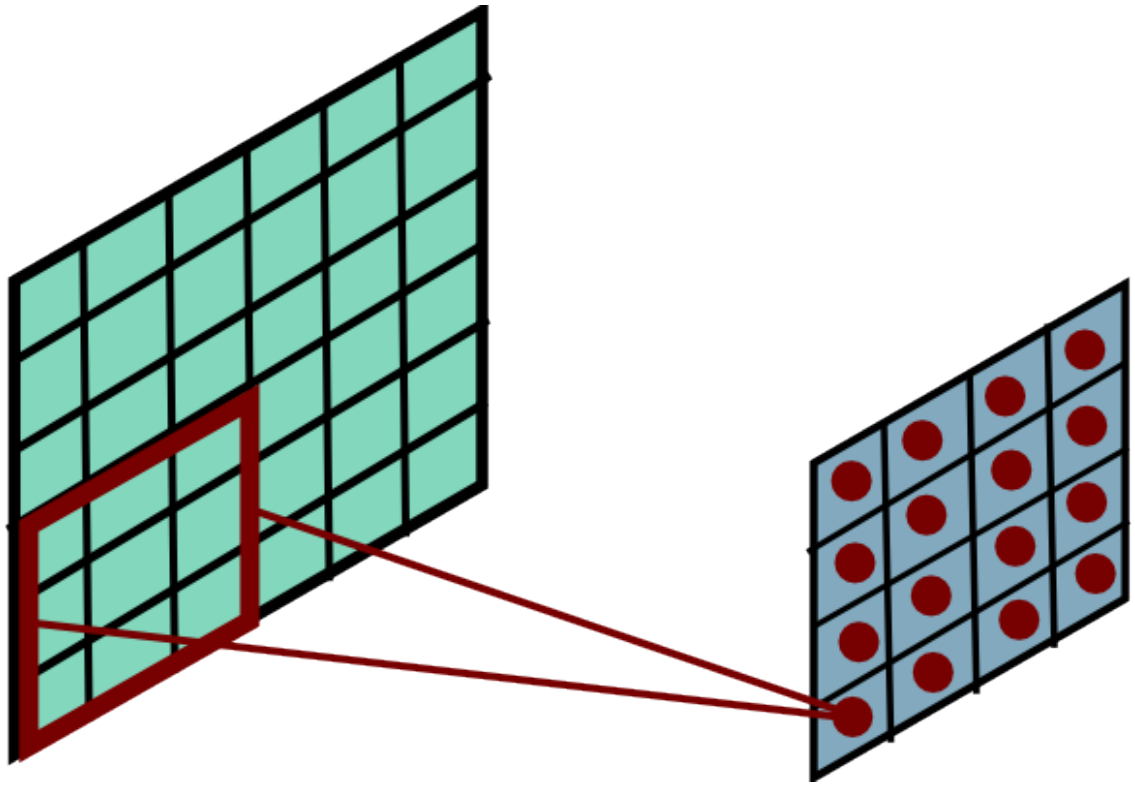
Convolutional Layer



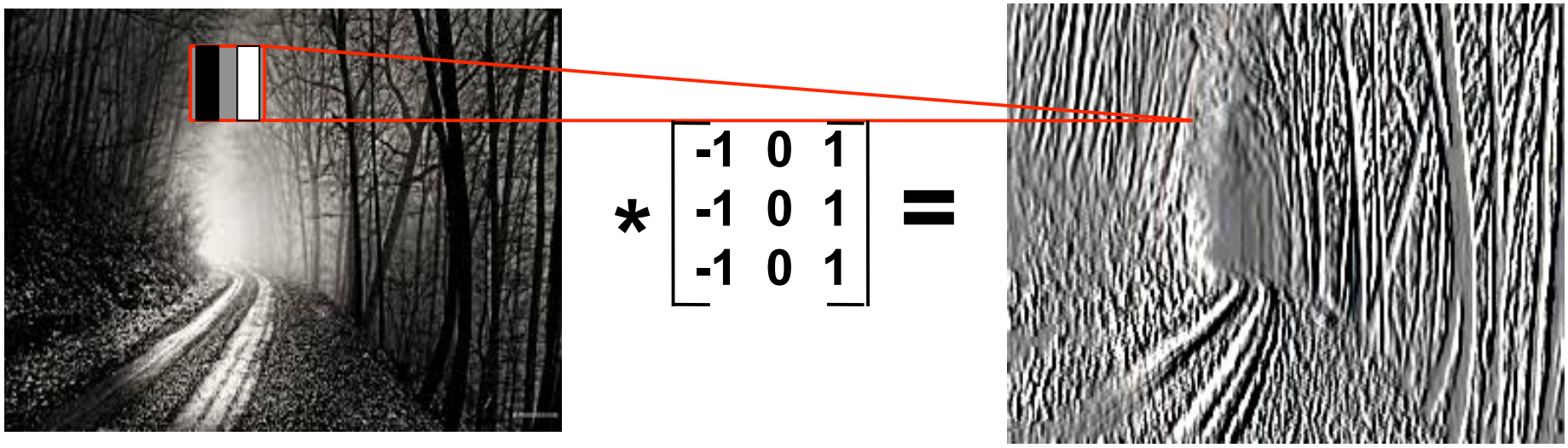
Convolutional Layer



Convolutional Layer

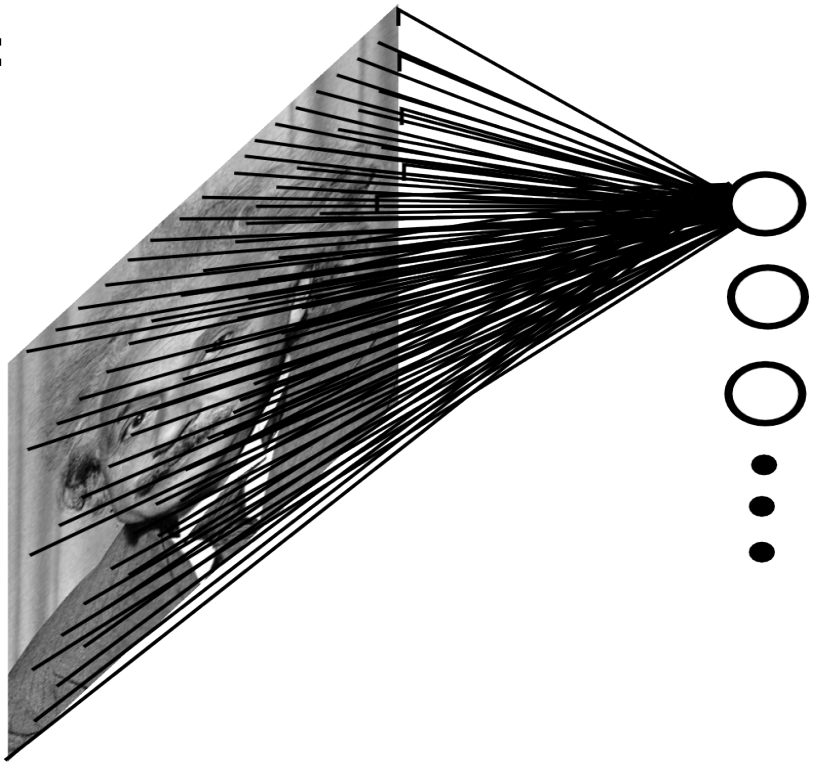


Convolutional Layer



“Fully-connected” layer

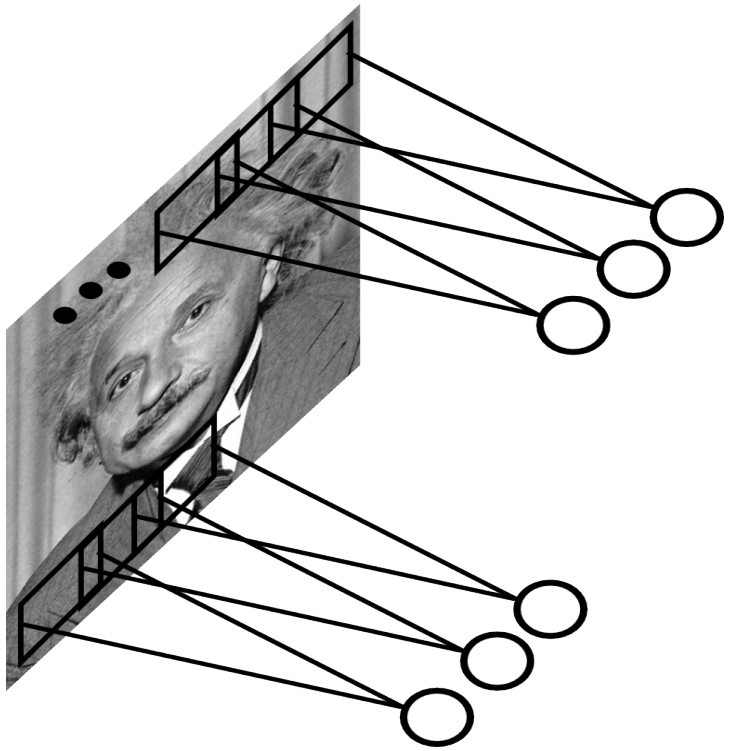
#of parameters:
K^2



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & \dots & w_{1,K} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & \dots & w_{2,K} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & \dots & w_{3,K} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & \dots & w_{4,K} \\ & & \vdots & & & \\ w_{K,1} & w_{K,2} & w_{K,3} & w_{K,4} & \dots & w_{K,K} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

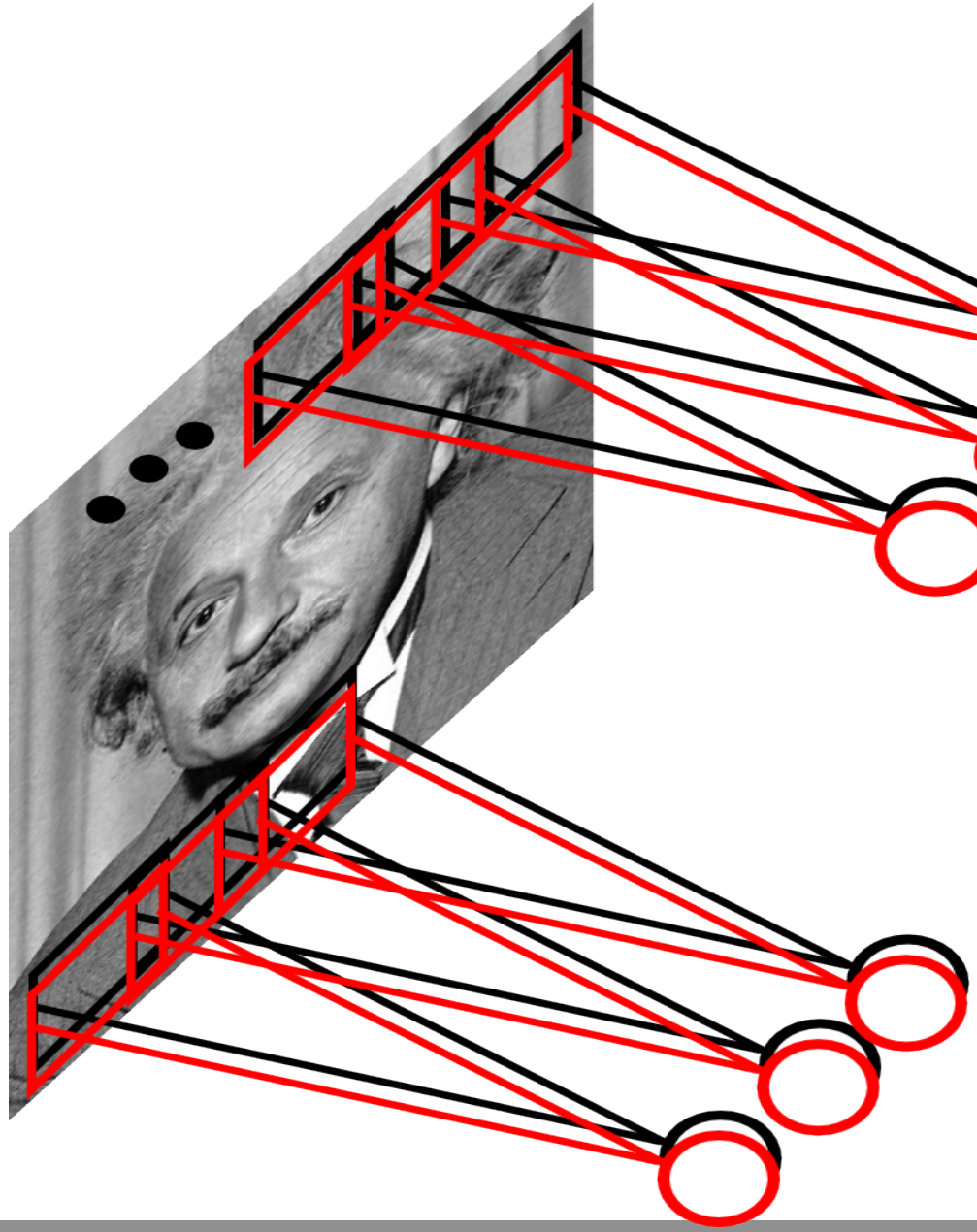
“Convolutional” layer

#of parameters:
size of window



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_0 & w_1 & w_2 & 0 & \dots & 0 \\ 0 & w_0 & w_1 & w_2 & \dots & 0 \\ 0 & 0 & w_0 & w_1 & \dots & 0 \\ 0 & 0 & 0 & w_0 & \dots & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & w_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

Convolutional Layer



Learn **multiple filters.**

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

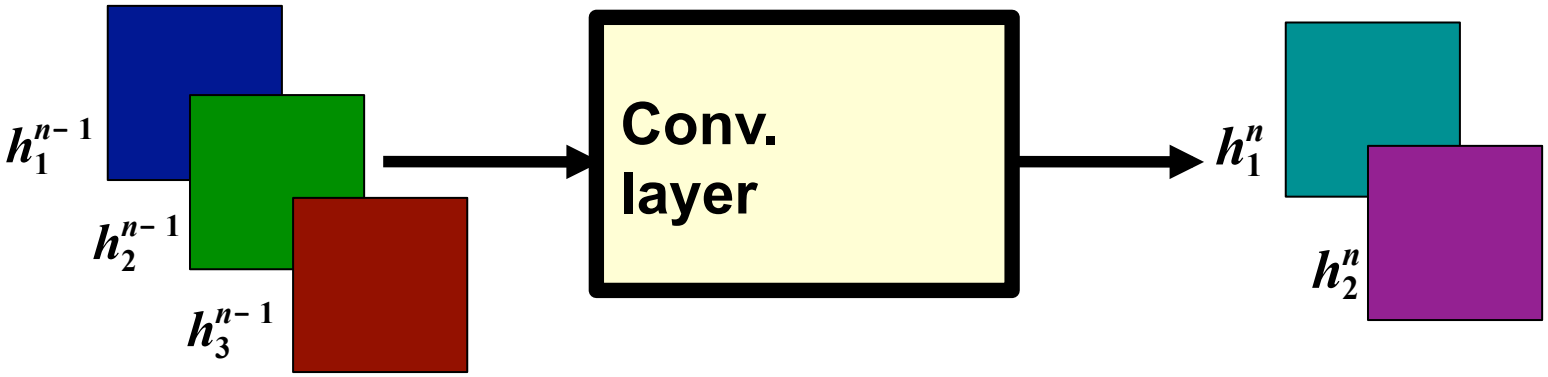
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

kernel



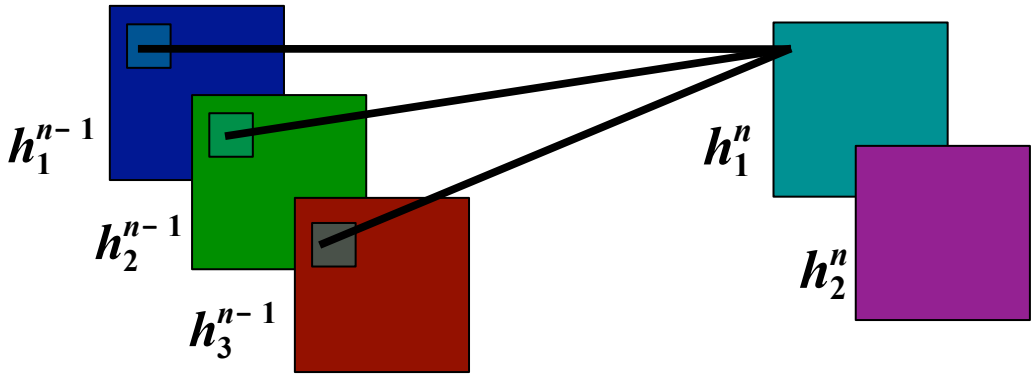
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

kernel



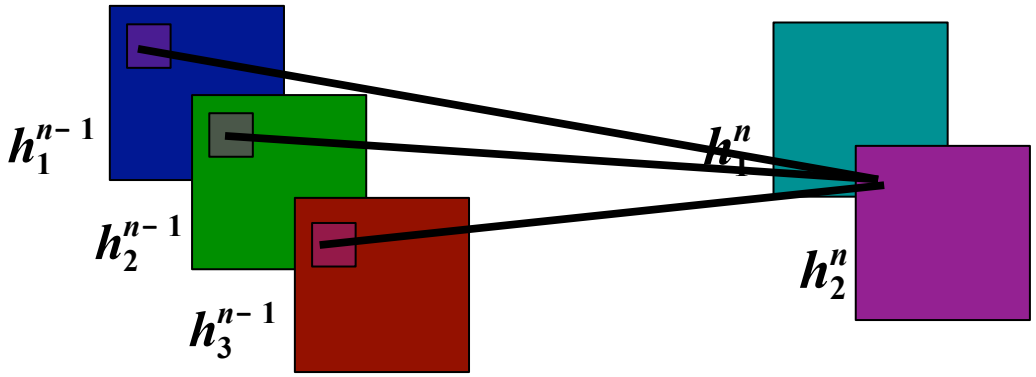
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

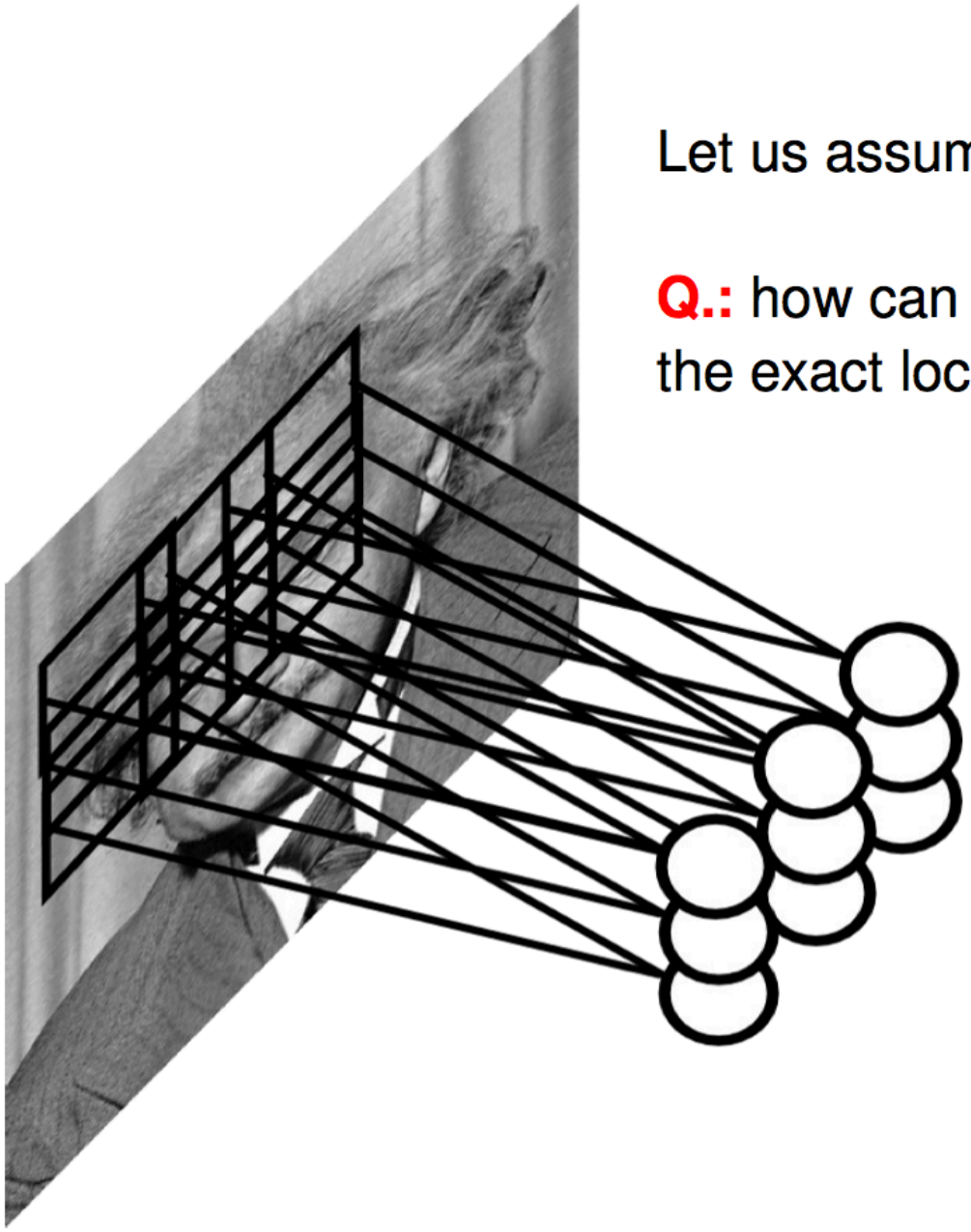
output feature map

input feature map

kernel



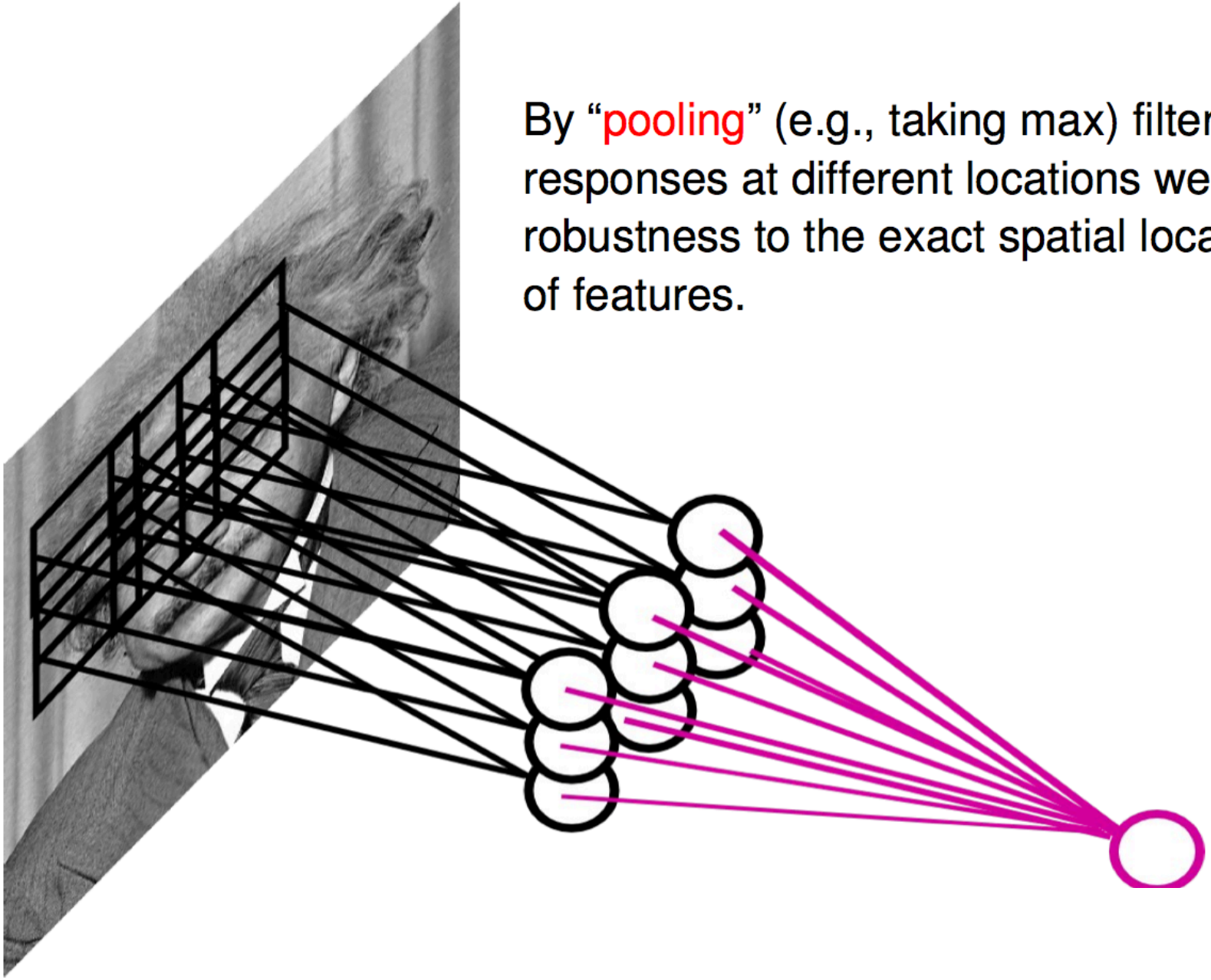
Pooling Layer



Let us assume filter is an “eye” detector.

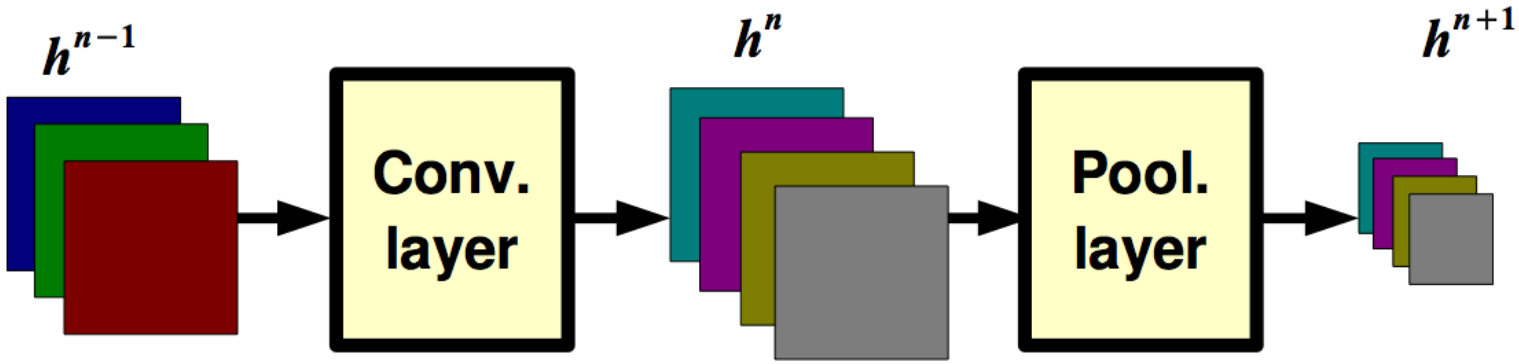
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer

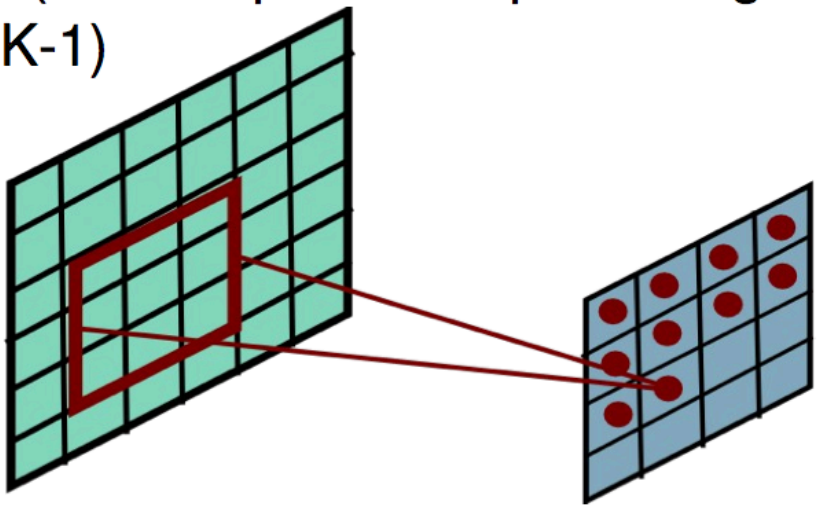


By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

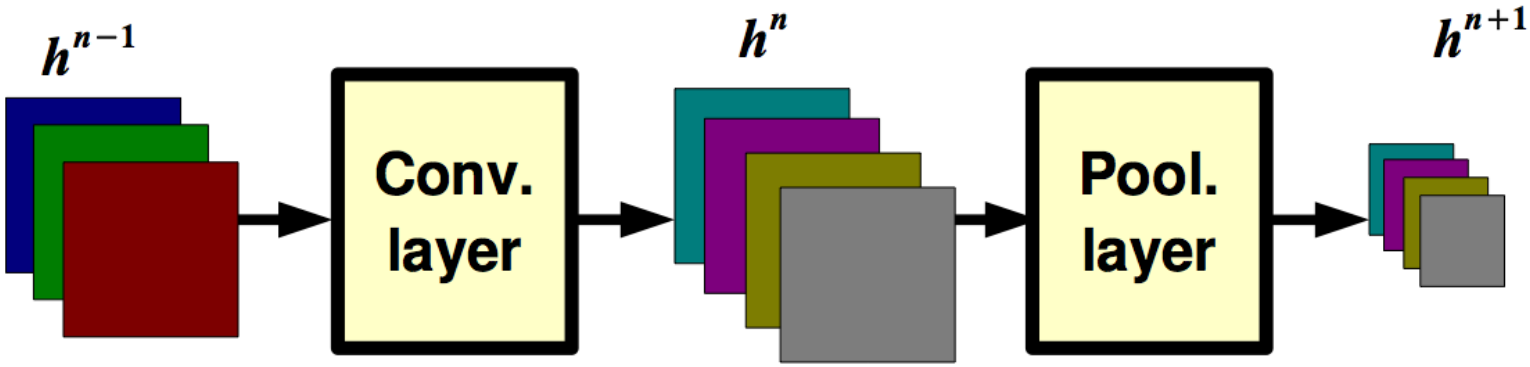
Pooling Layer: Receptive Field Size



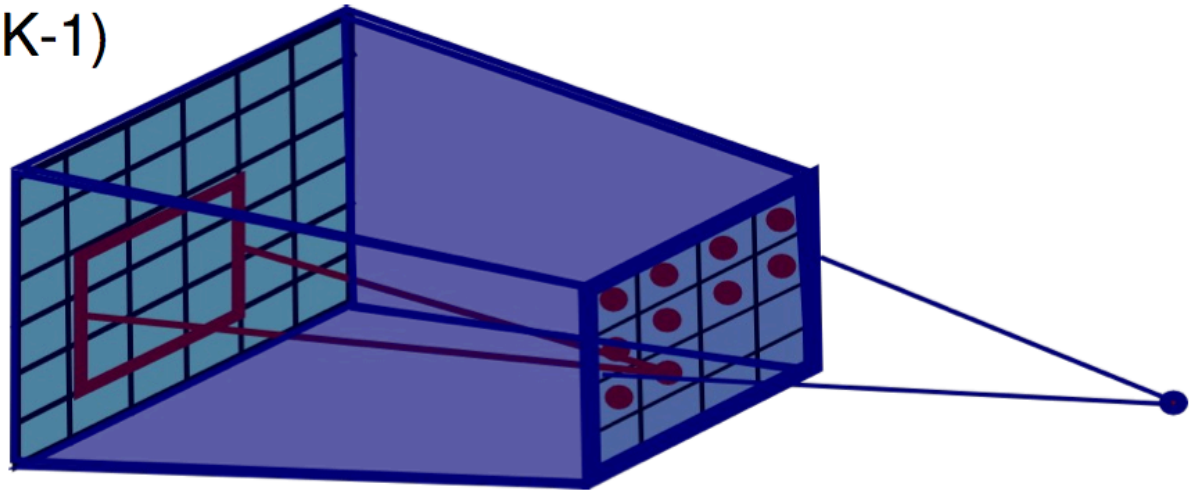
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



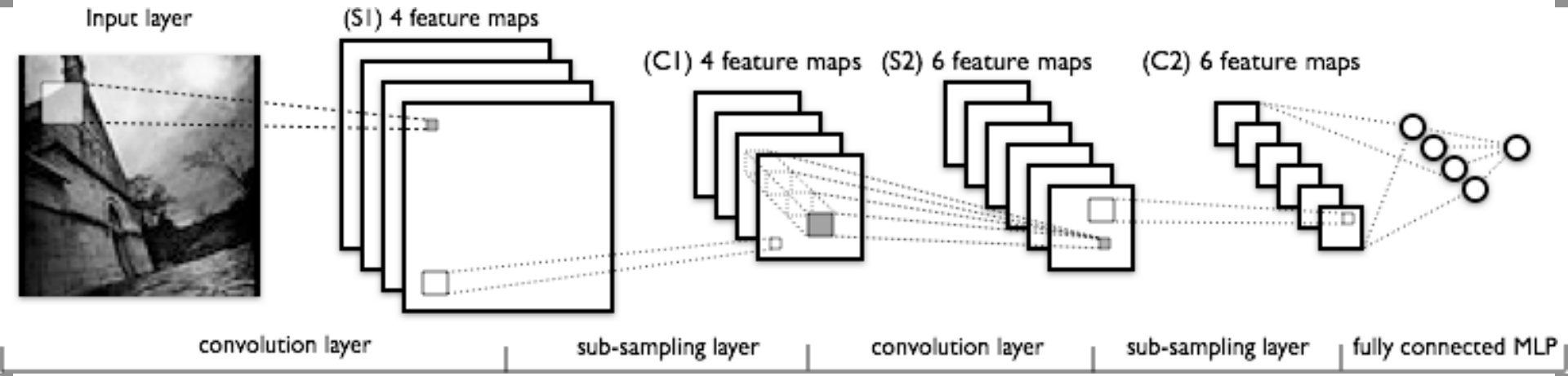
Pooling Layer: Receptive Field Size



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)



Convolutional Nets



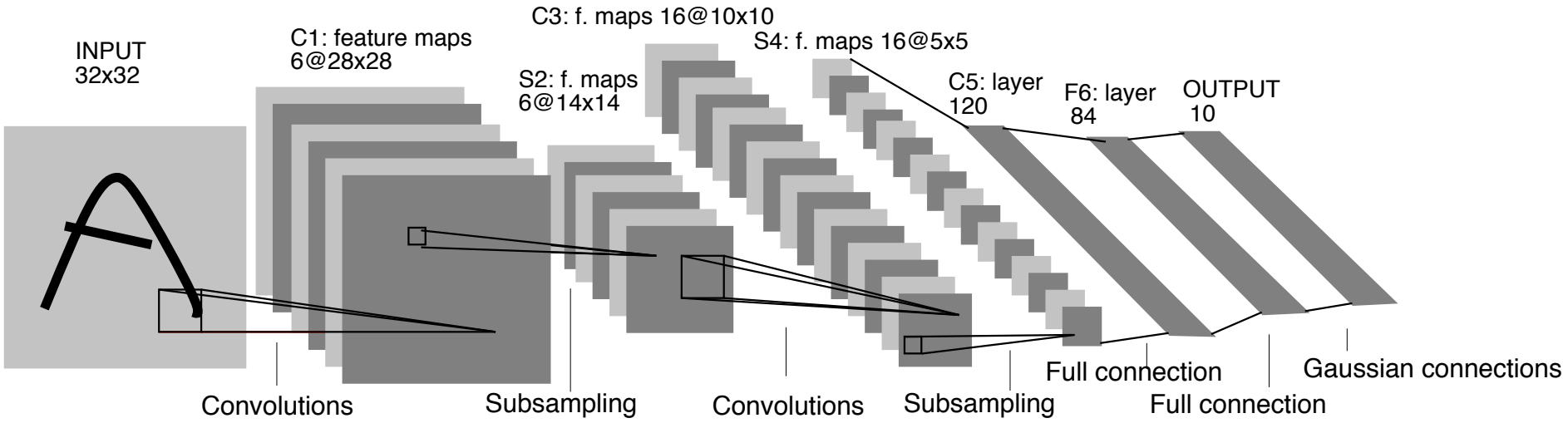


Image Credit: Yann LeCun, Kevin Murphy

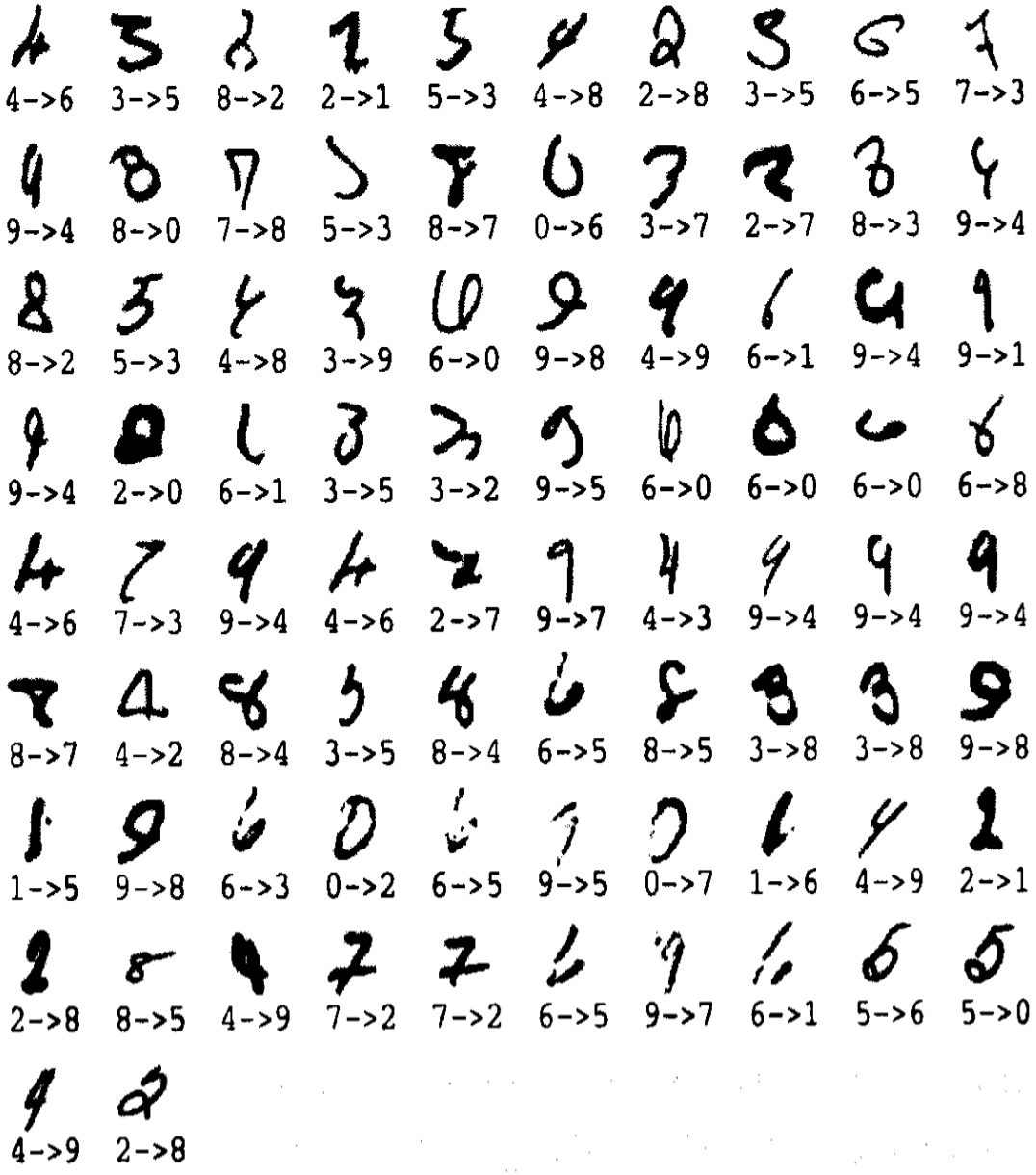
Deep learning, AD 1993

http://yann.lecun.com/exdb/lenet/

(LeNet 5, 1998)

https://www.youtube.com/watch?v=FwFduRA_L6Q

(LeNet 1, 1993)

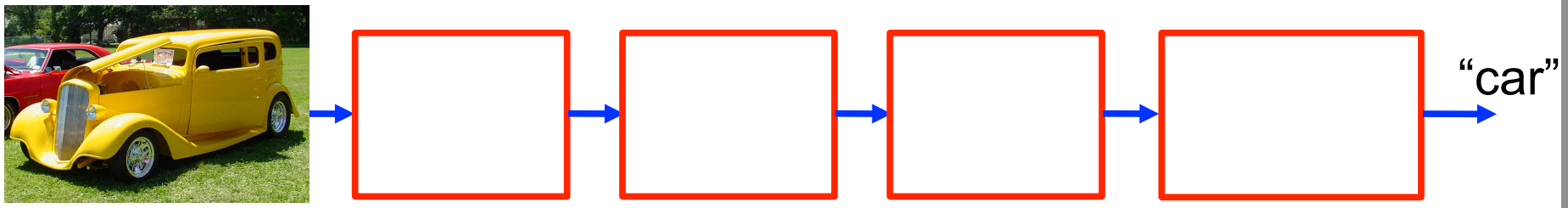


The 82 errors made by LeNet5

Notice that most of the errors are cases that people find quite easy.

The human error rate is probably 20 to 30 errors but nobody has had the patience to measure it.

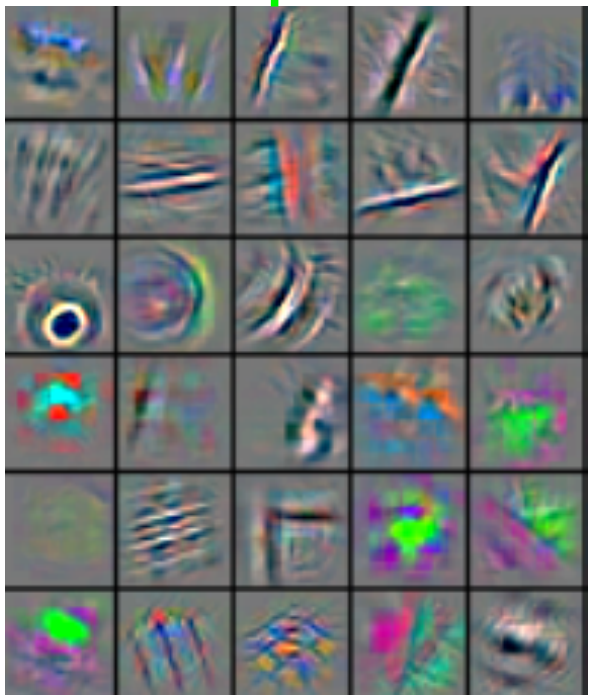
Deep Learning = Hierarchical Compositionality



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Deep Learning = Hierarchical Compositionality

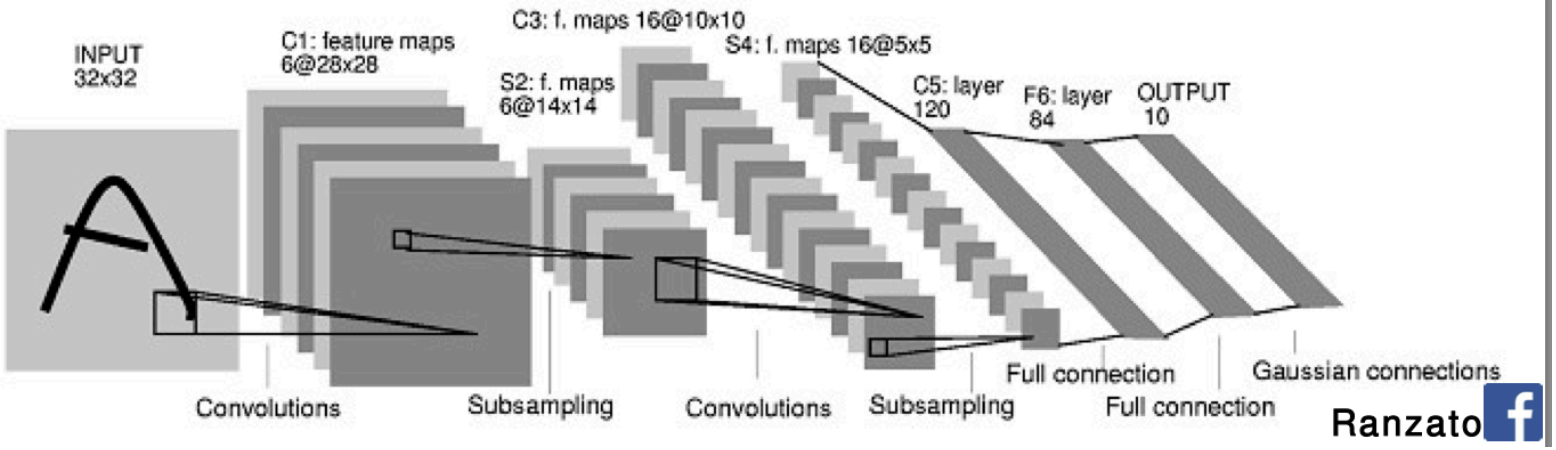
Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier → "car"



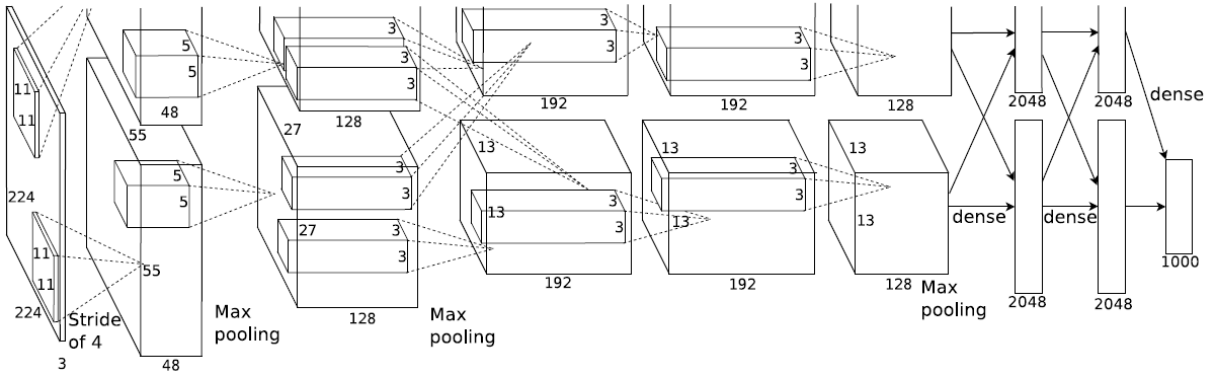
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Famous networks

http://yann.lecun.com/exdb/lenet/



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998



A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*13

Latest and greatest

Visual Geometry Group (VGG) network, Simonyan & Zisserman (2014)

- Small 3x3 kernels, much deeper the predecessors (16 vs 8 layers)

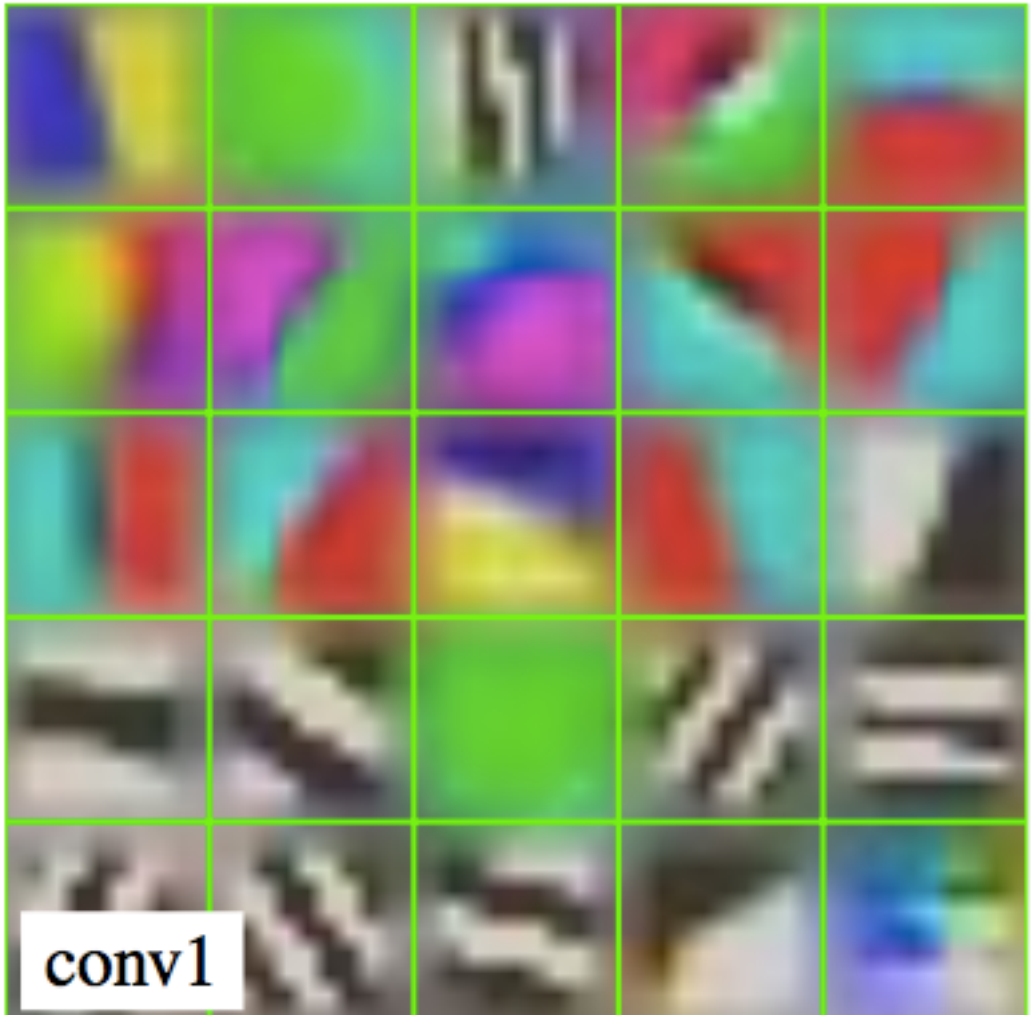
GoogLeNet, C. Szegedy et al, 2014

- Deep supervision (early losses), compression (bottleneck layers)

Deep Residual Learning (DRL) networks, He et al (CVPR 2016)

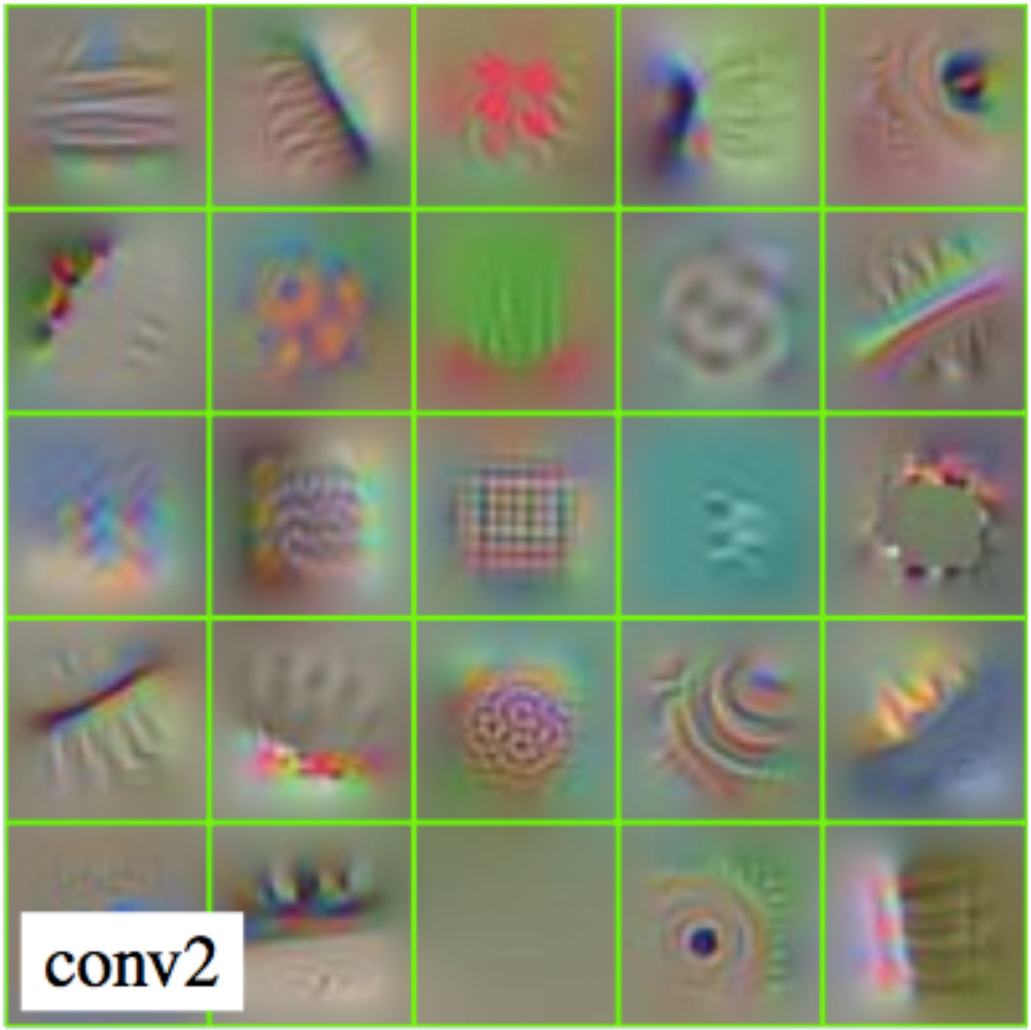
- Ultra-deep: 100+ layers, 1000 layers for smaller problems

What is in the network?



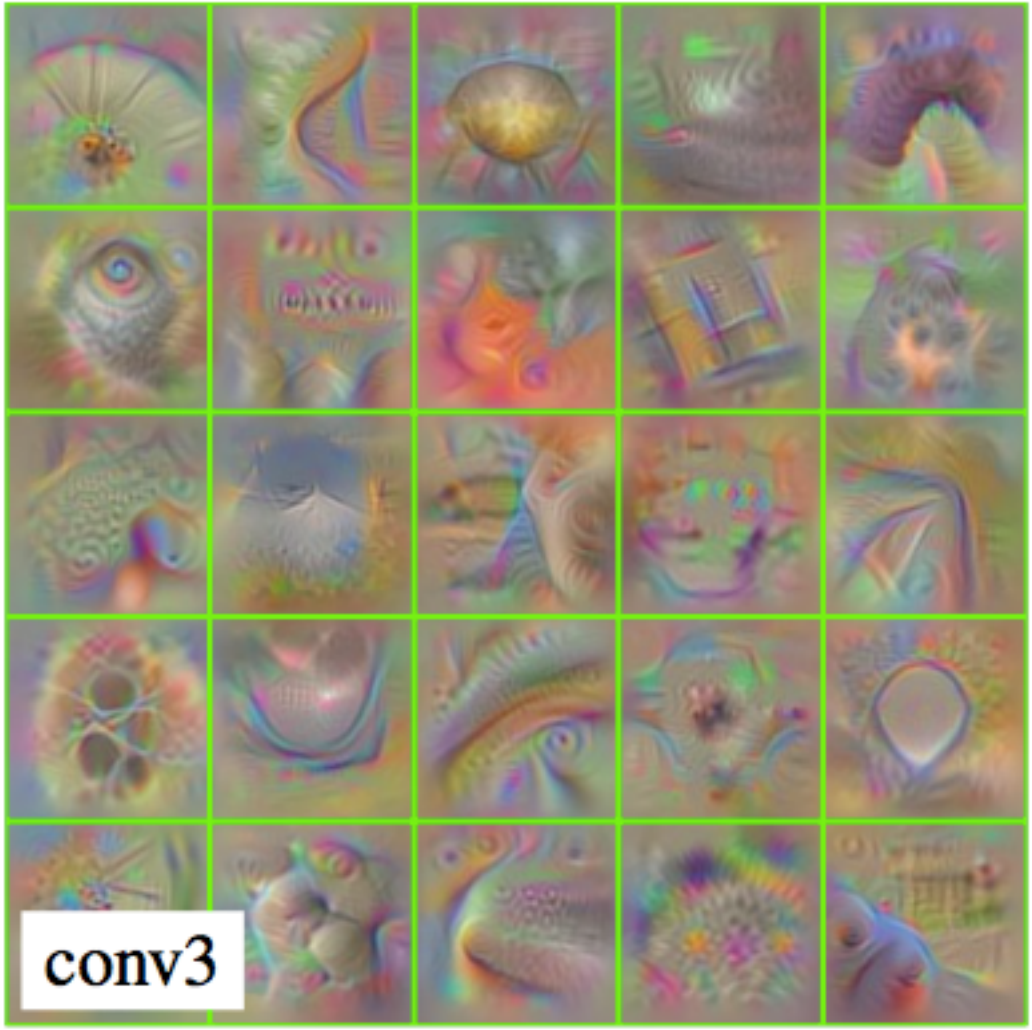
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



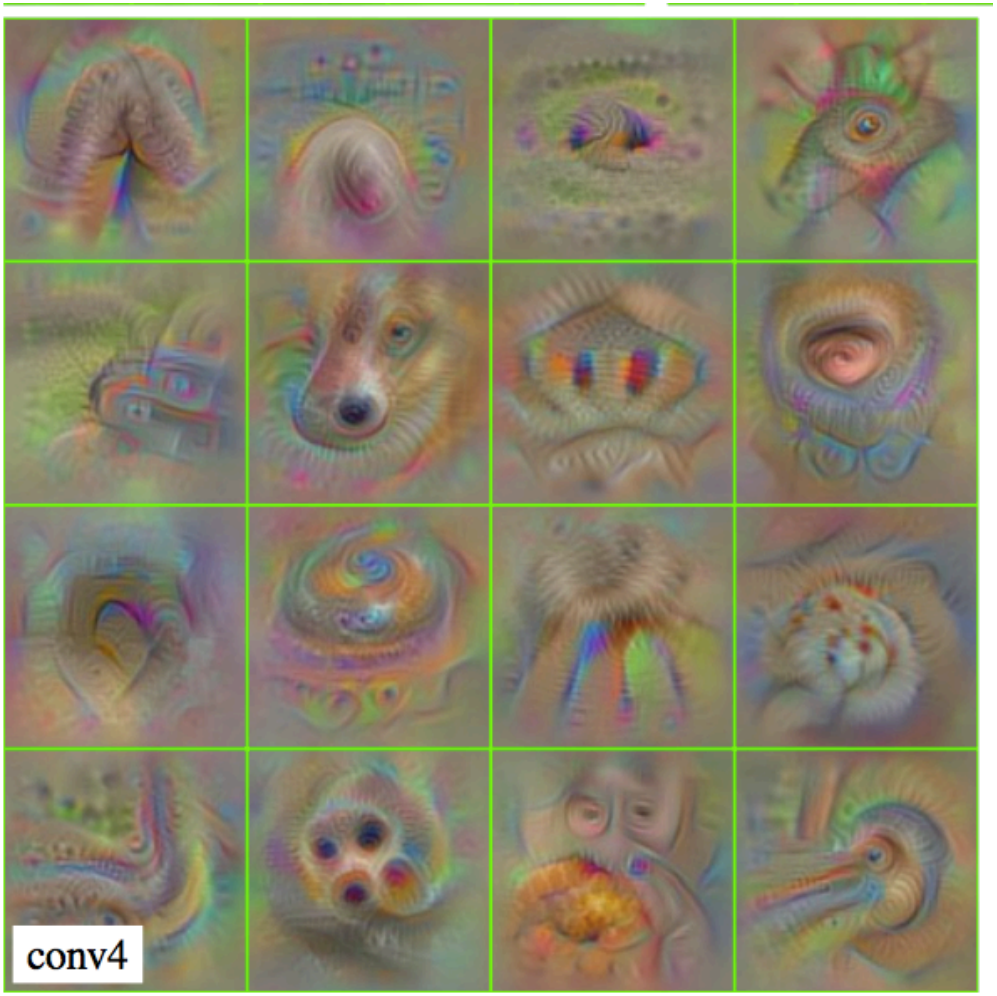
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



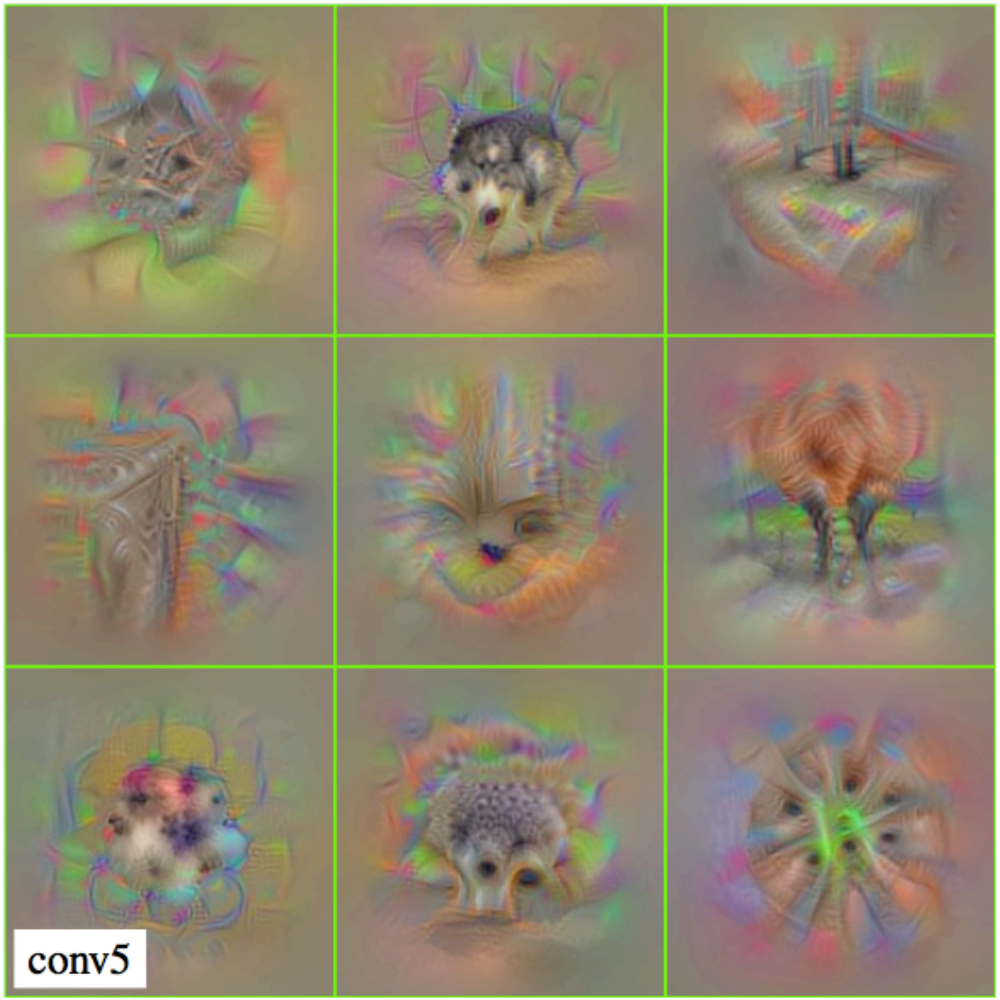
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



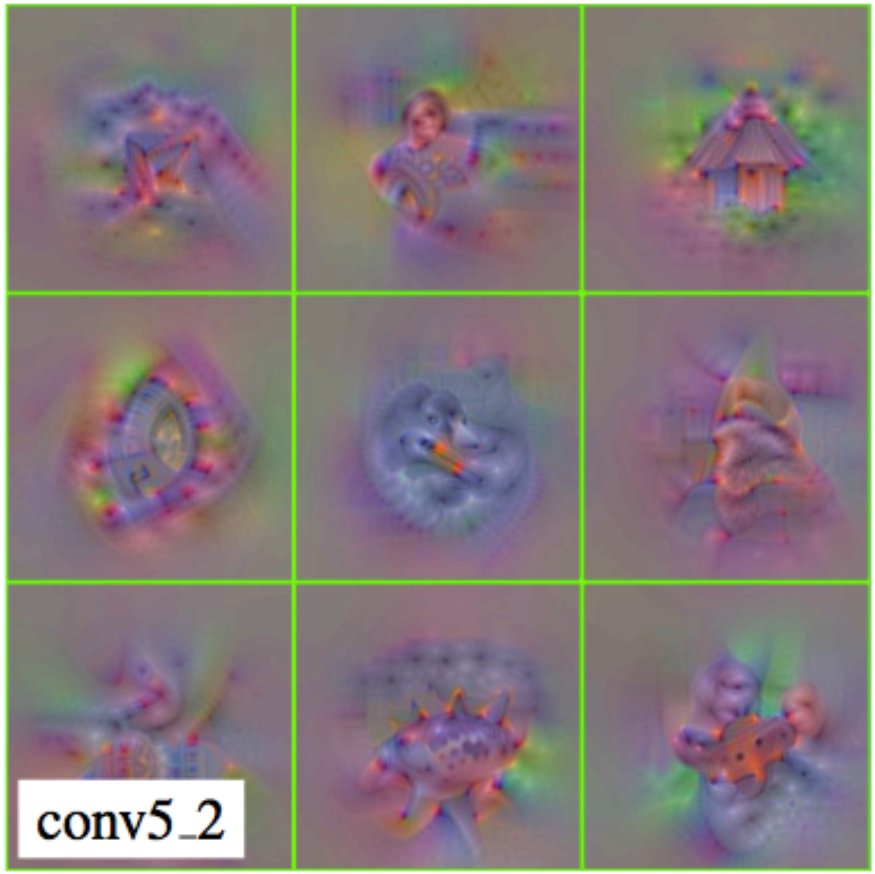
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

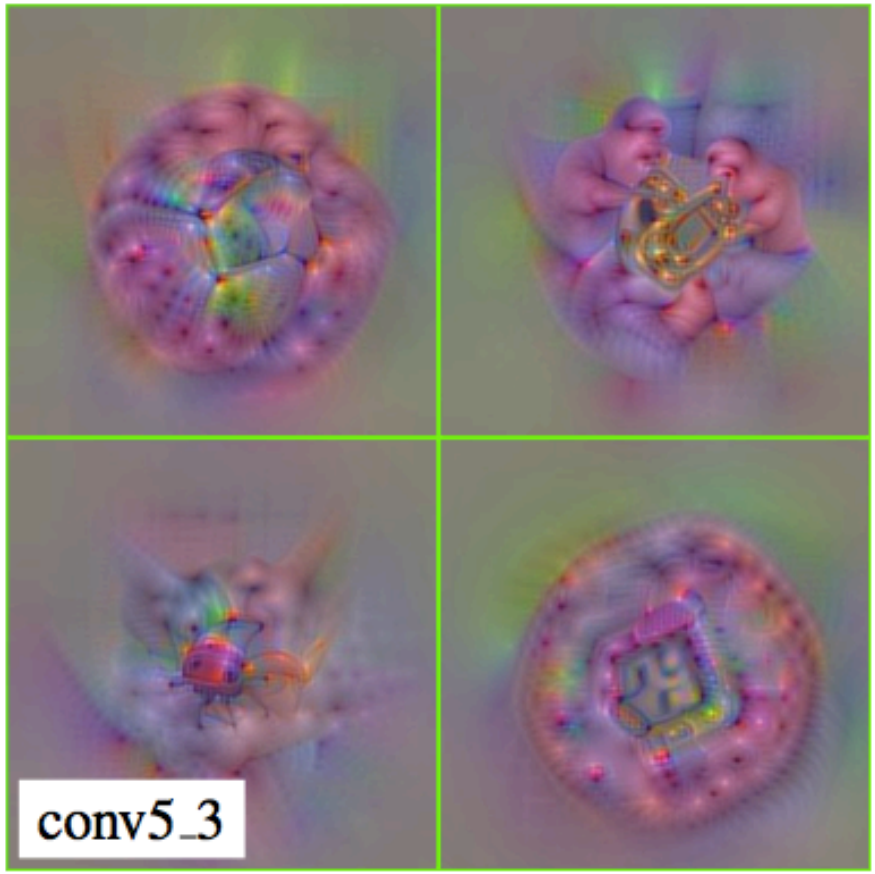
What is in the network?



Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

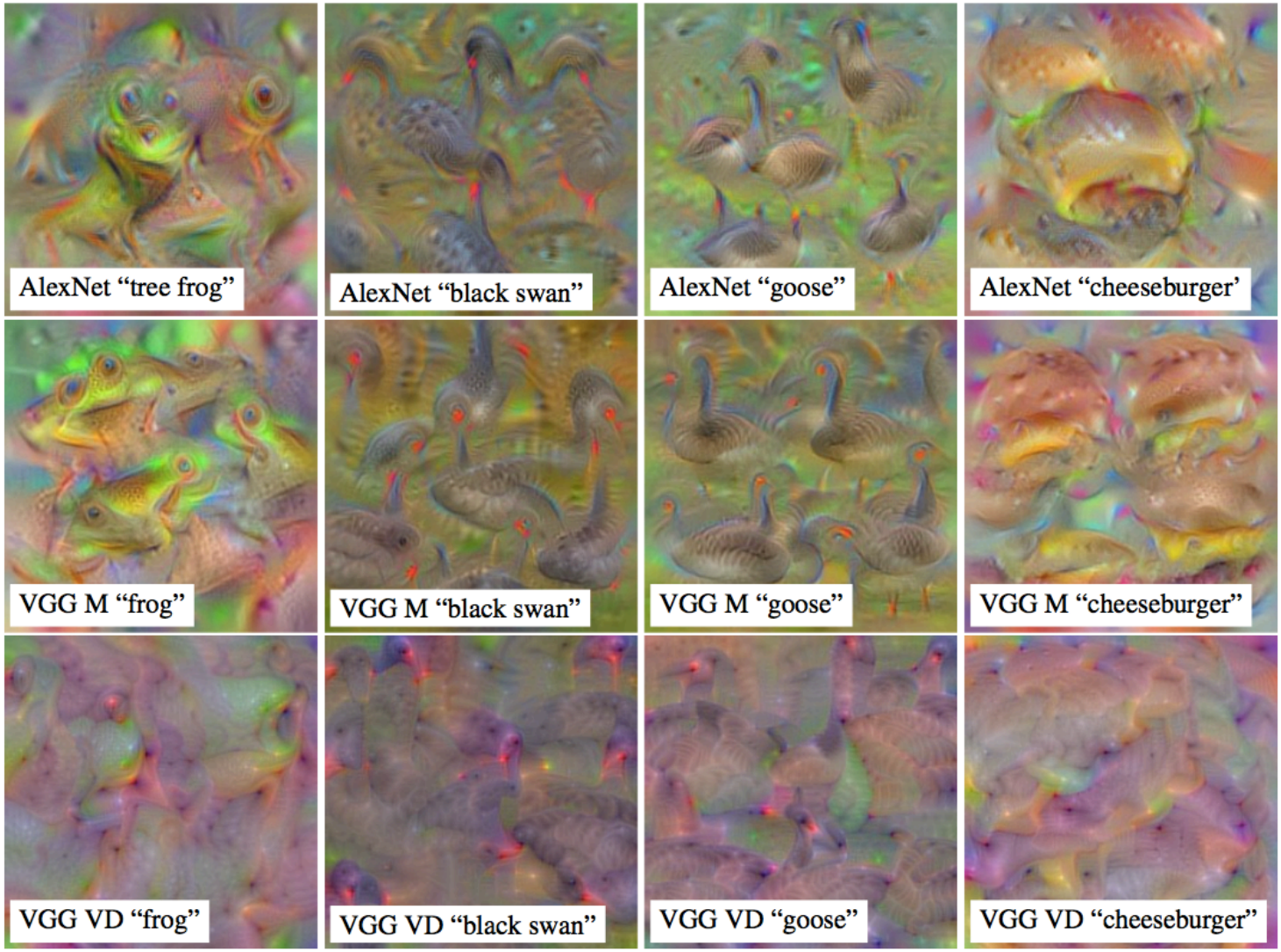
What is in the network?





Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



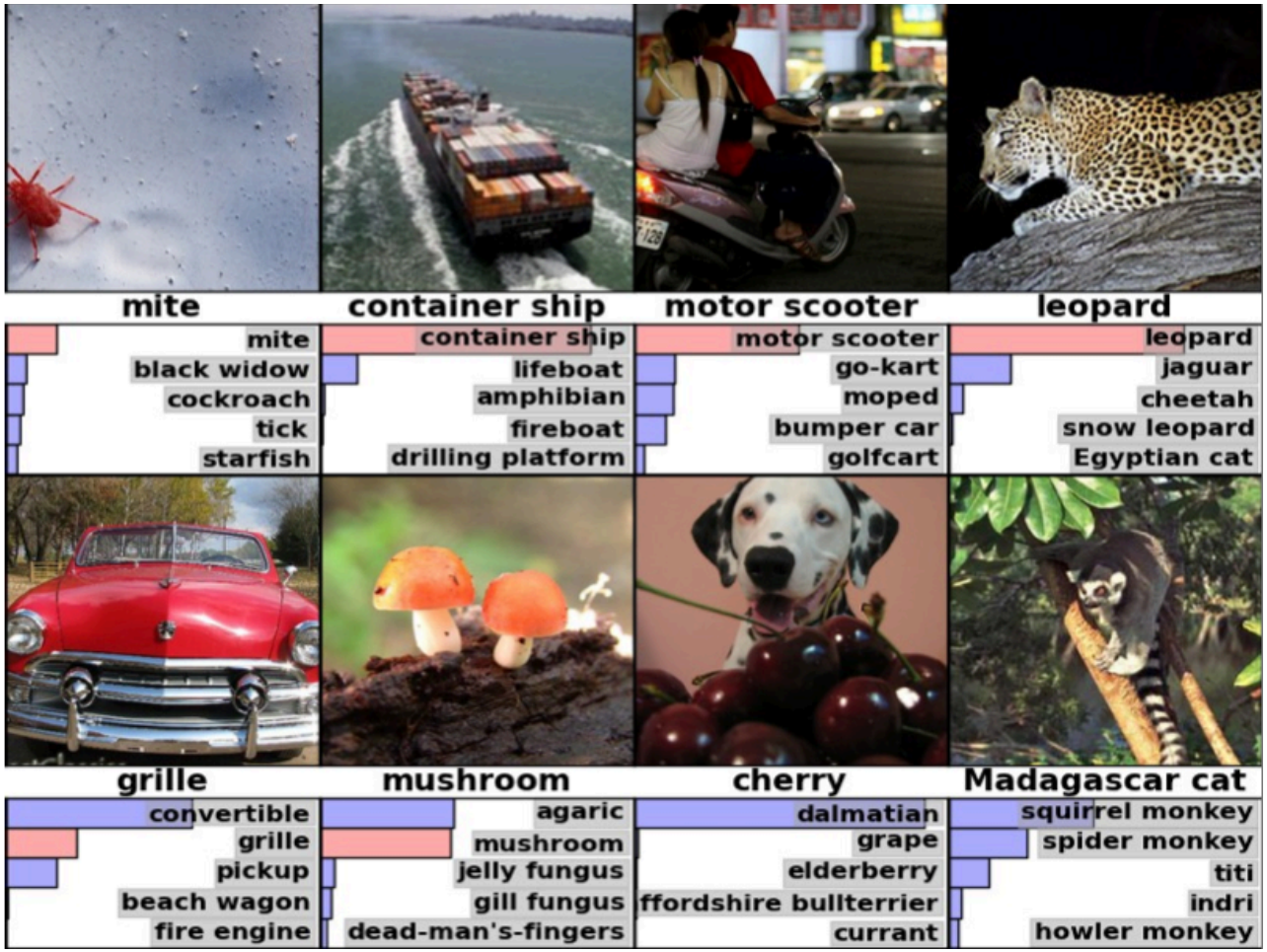
Visualizing deep convolutional neural networks using natural pre-images, A. Mahindra and A. Vedaldi

Examples from the test set (with the network's guesses)

Error rates on the ILSVRC-2012 competition

	classification	classification &localization
• University of Toronto (Alex Krizhevsky)	• 16.4%	34.1%
• University of Tokyo	• 26.1%	53.6%
• Oxford University Computer Vision Group	• 26.9%	50.0%
• INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)	• 27.0%	
• University of Amsterdam	• 29.5%	

Imagenet top-5 error rates



Humans:5.4%

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*13 [18%] (best shallow competitor: 36%)

K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, ICCV 2015. [4.5%]

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015. [4.5%]

K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, CVPR, 2016 [3.6%]