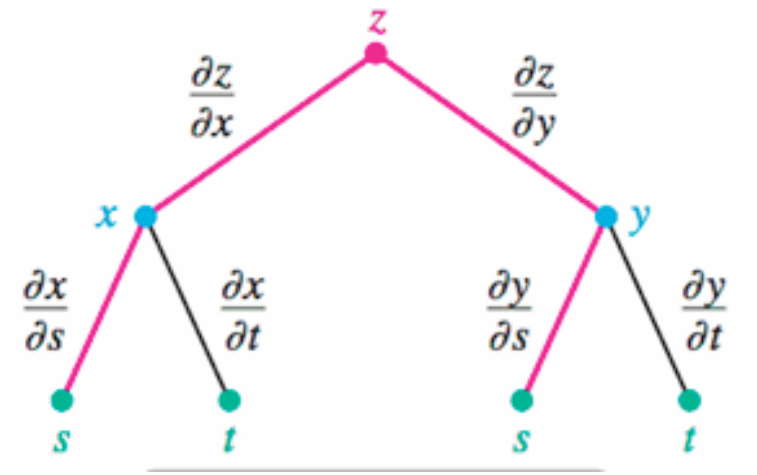
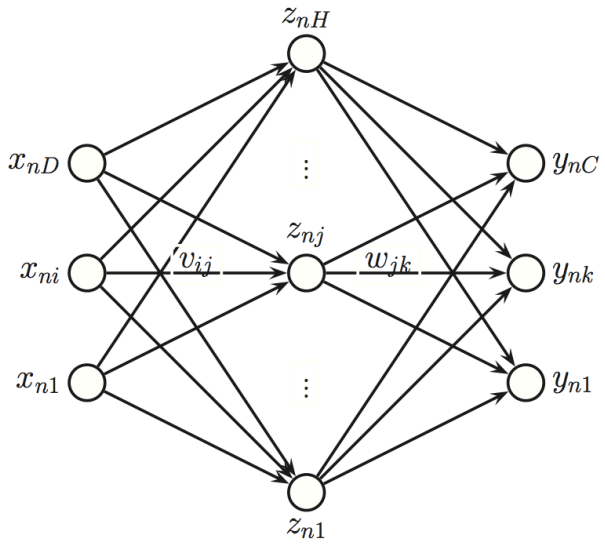
Introduction to Supervised Learning





Week 7:
Introduction to Neural Networks

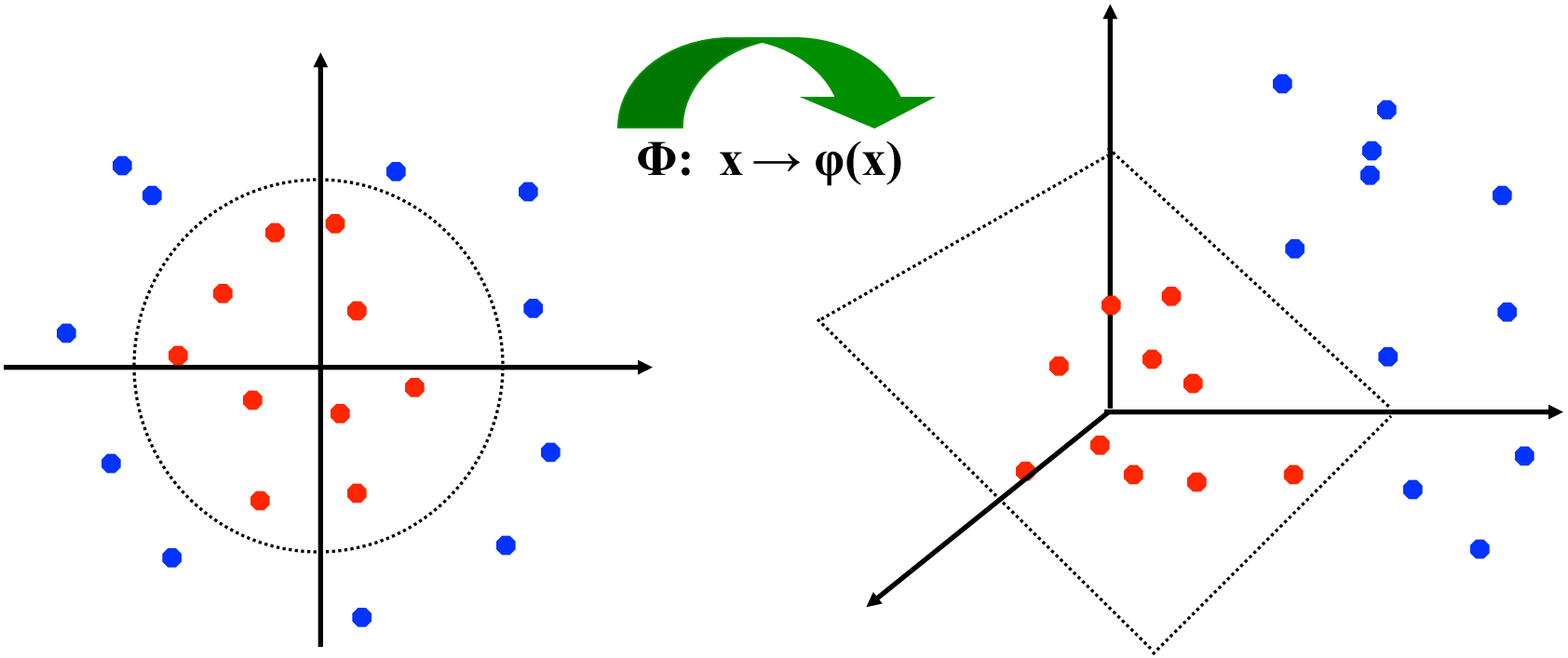
Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Beyond linear boundaries

- Weeks 2-4: More features & regularization



Beyond linear boundaries

- Week 5: Kernel Trick - SVM approach

Kernel: $$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

Classifier: $$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha^i y^i K(\mathbf{x}^i, \mathbf{x}) + b$$

Beyond linear boundaries

- Week 6: Weak Learners & Ensembling: Adaboost approach

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

Strong classifier

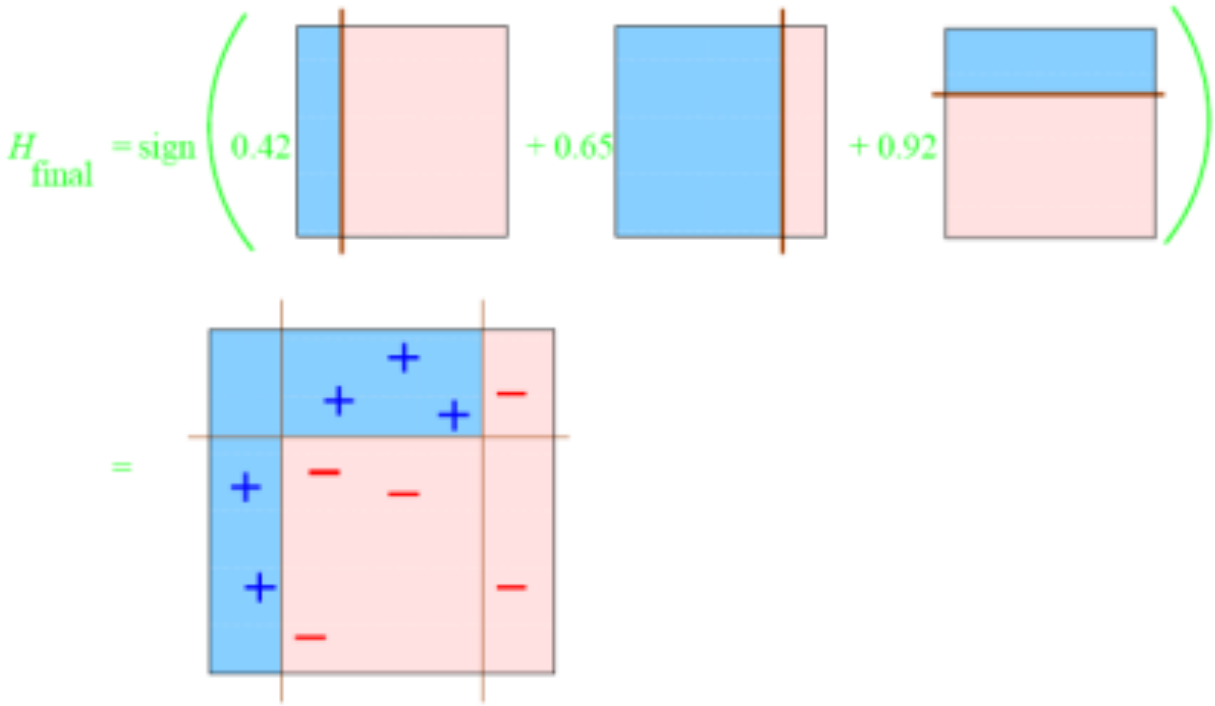
Feature vector

Weight

Weak classifier

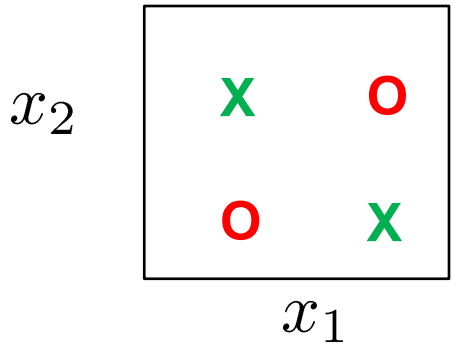
Beyond linear boundaries

- Week 6: Weak Learners & Ensembling: Adaboost approach

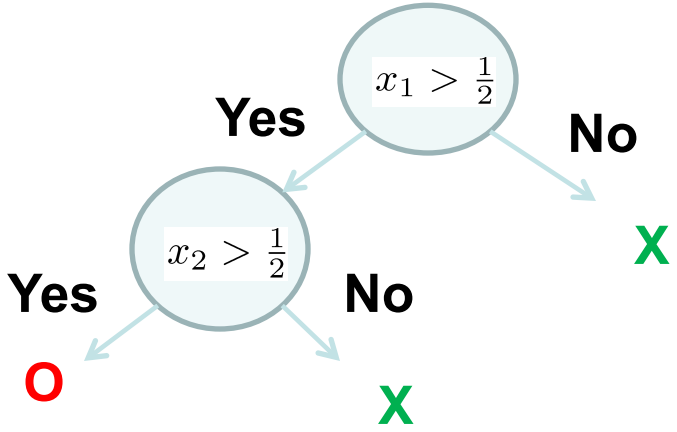


Last lecture: decision trees & random forests

- No decision stump can separate these data with error less than 1/2

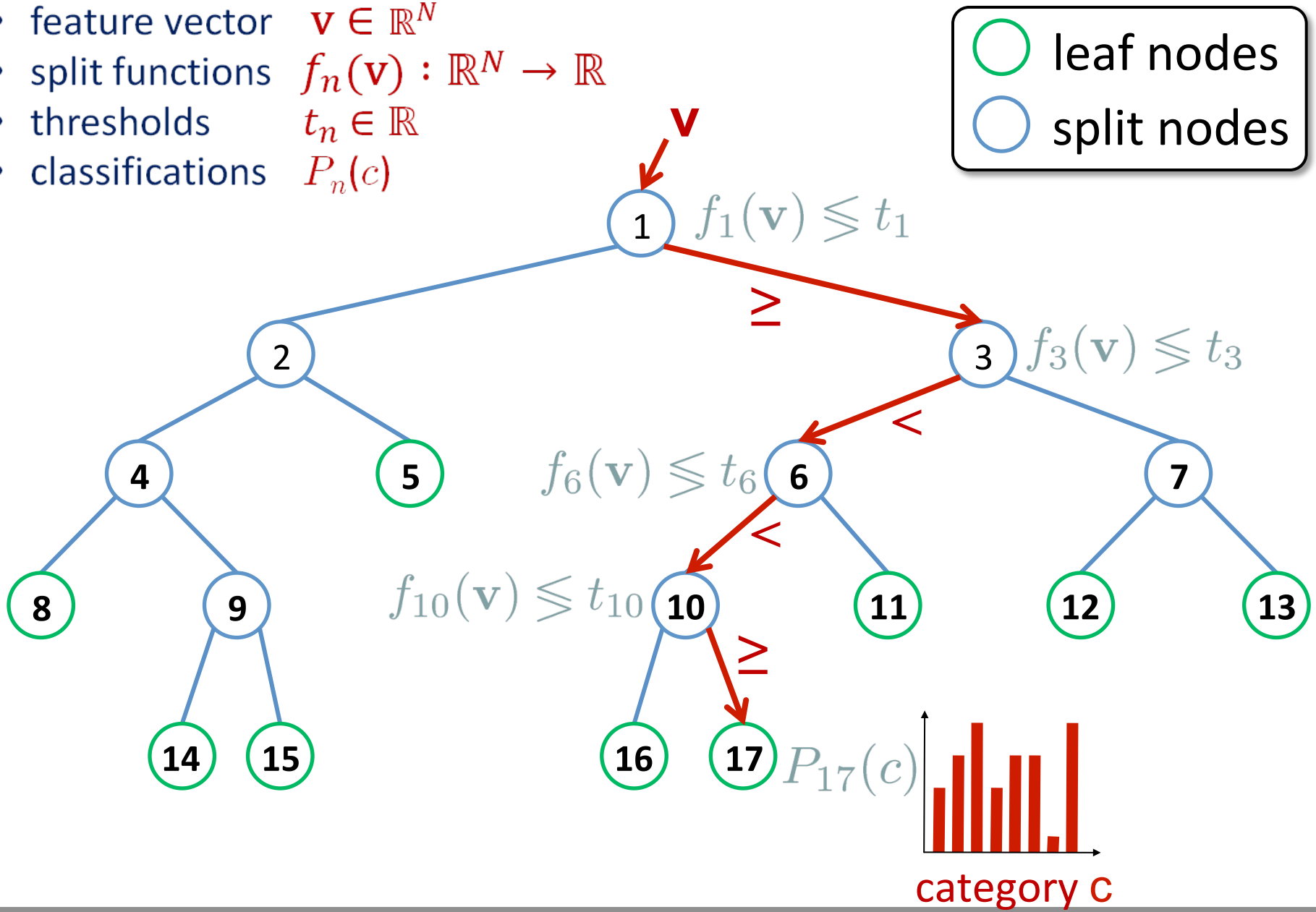


- In two steps: error 1/4



Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



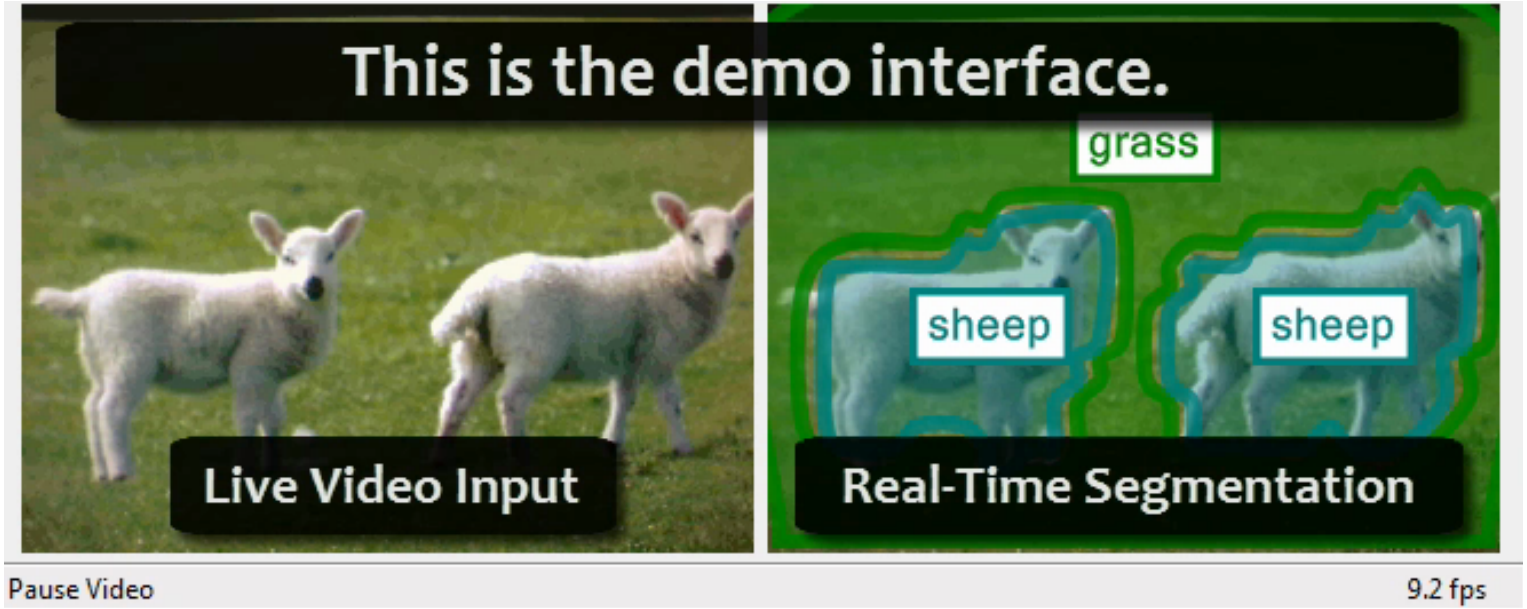
Example Semantic Texton Forest

Input Image

Ground Truth

Real-Time Object Segmentation [Shotton *et al.* 2008]

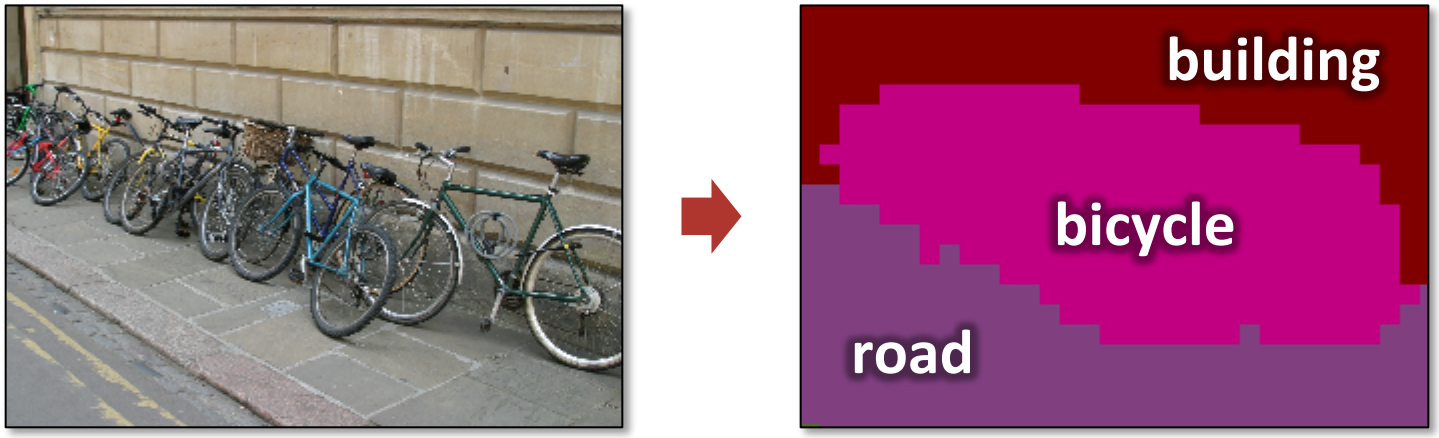
- Segment image and label segments in real-time



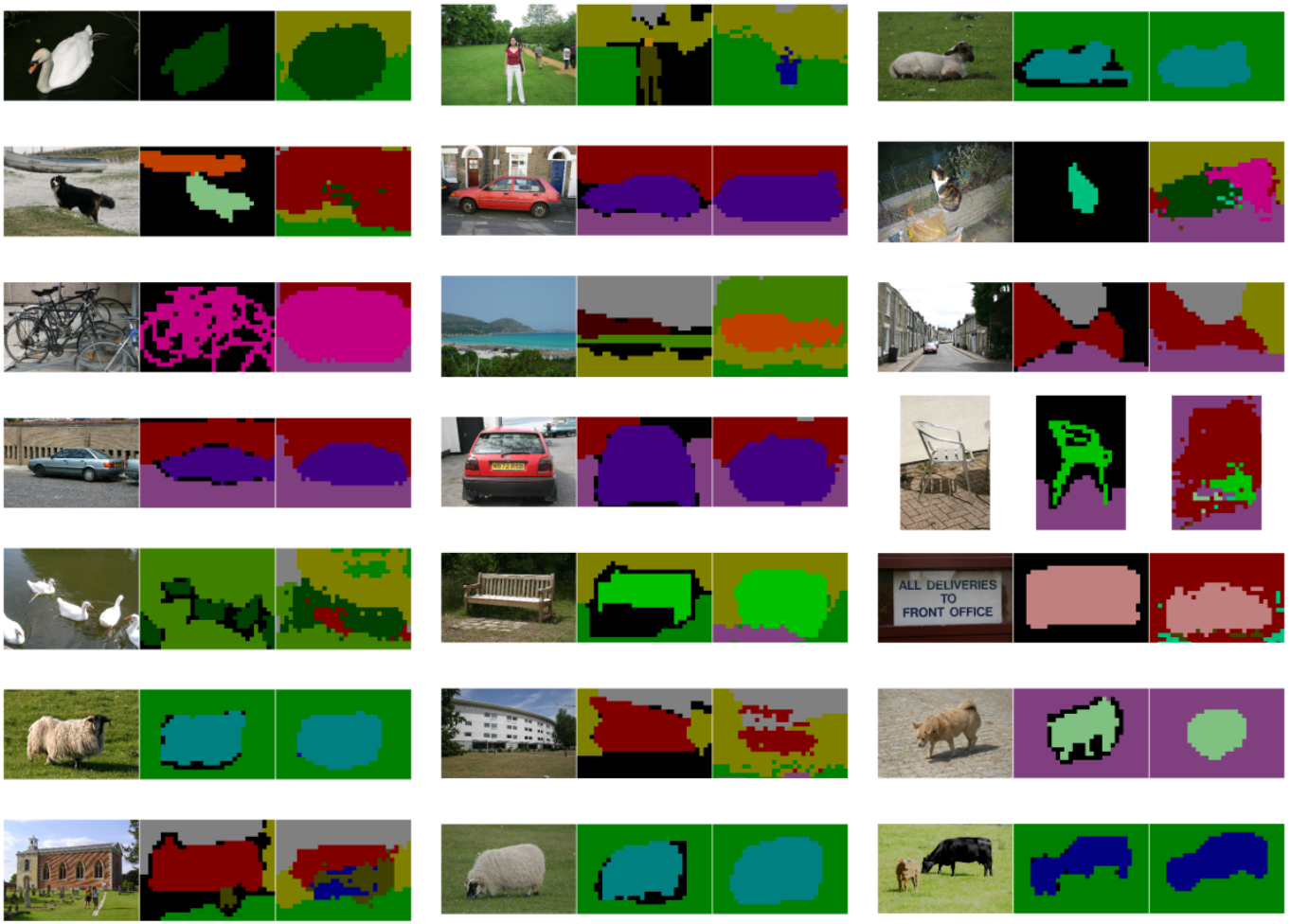
CVPR 2008 Best Demo Award

Segmentation Forest

- Object segmentation



MSRC Dataset Results



building	grass	tree	cow	sheep	sky	airplane	water	face	car		boat
bicycle	flower	sign	bird	book	chair	road	cat	dog	body		

$UCCE$$ $TORY: Kinect

12.1 million sold by end 2011

GUINNESS WORLD RECORDS™

Kinect's pose estimation pipeline

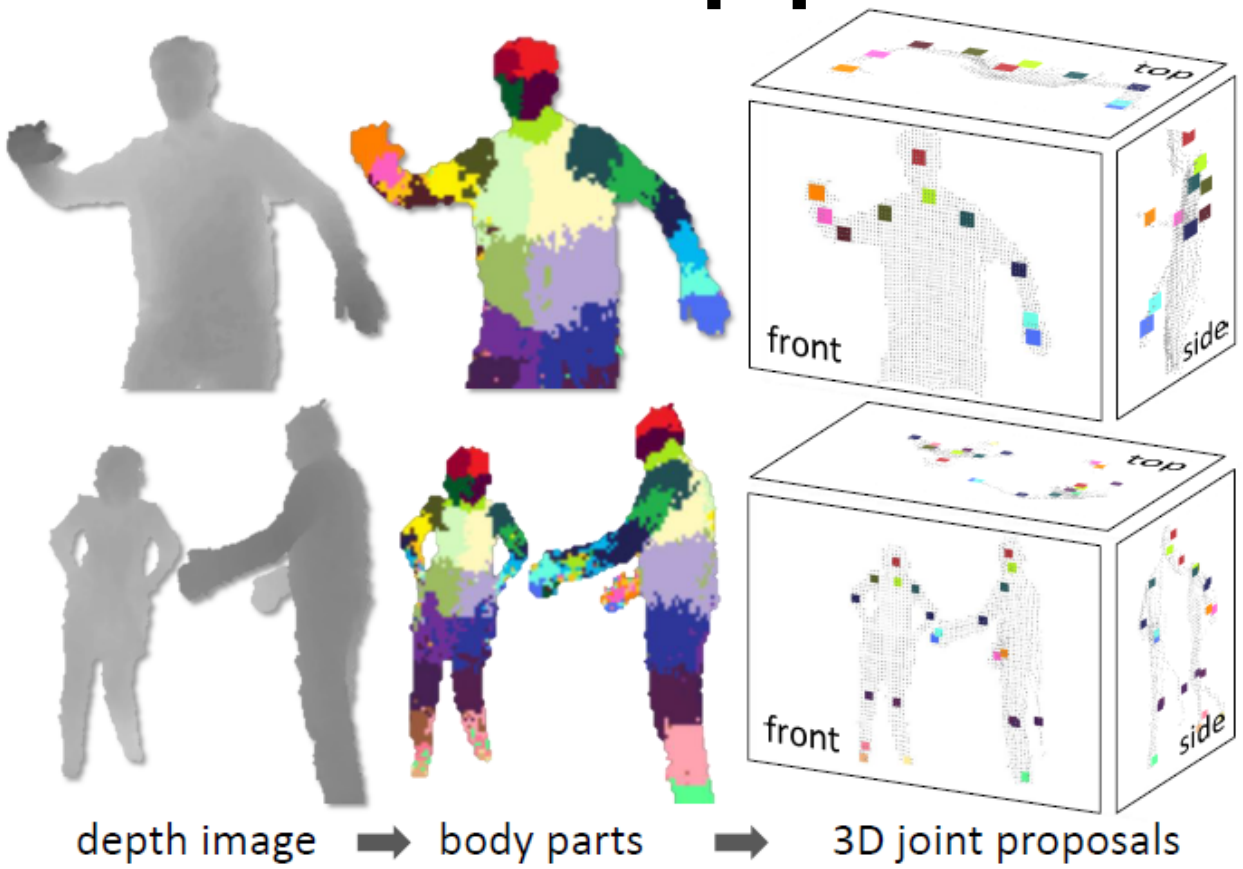


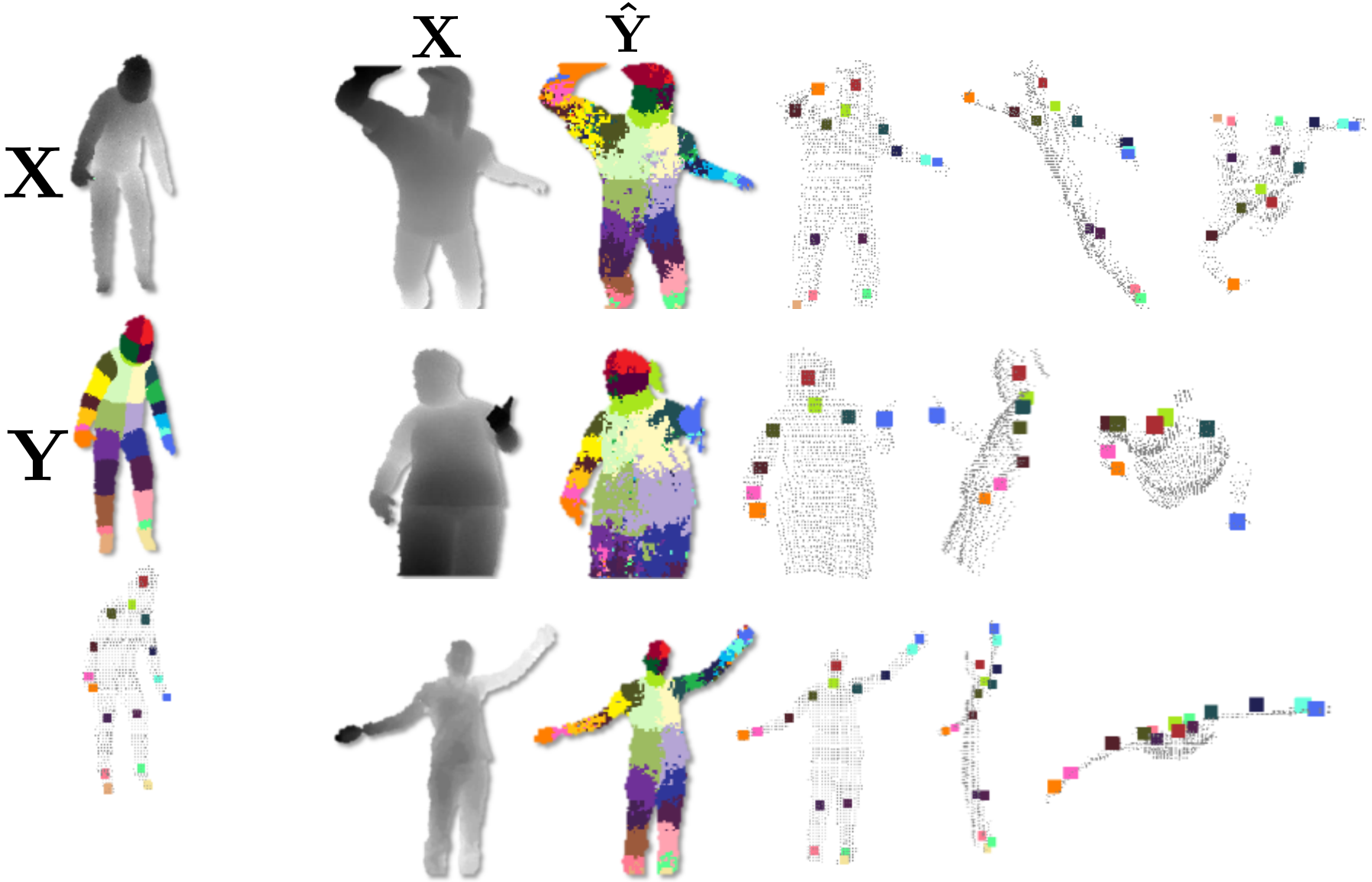
Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

Kinect's pose estimation system

Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake, Real-Time Human Pose Recognition in Parts from a Single Depth Image,in *CVPR*, IEEE, June 2011

- **Abstract:** We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that **maps the difficult pose estimation problem into a simpler per-pixel classification problem.** Our **large and highly varied training dataset** allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.
- **The system runs at 200 frames per second on consumer hardware**. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

Training & testing a 3D pose estimation system



Training data

Test time

Decision functions on tree nodes

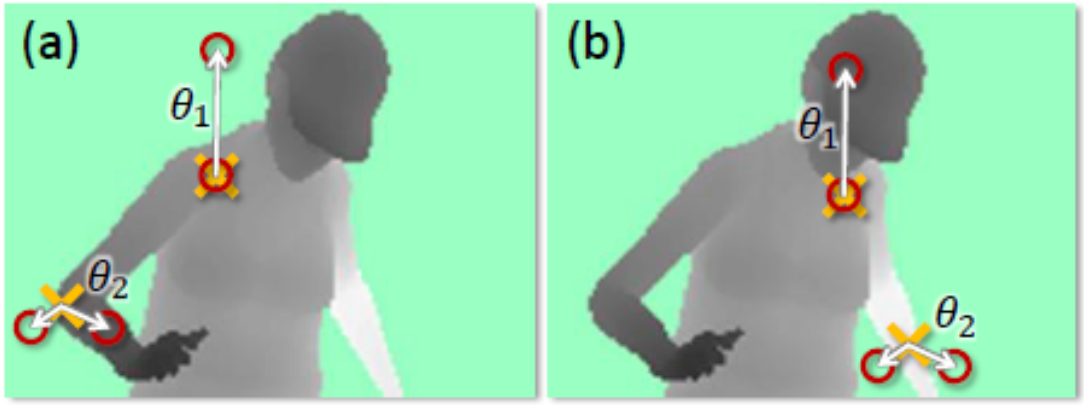


Figure 3. **Depth image features.** The yellow crosses indicates the pixel x being classified. The red circles indicate the offset pixels as defined in Eq. 1. In (a), the two example features give a large depth difference response. In (b), the same two features at new image locations give a much smaller response.

Average depth of patches traversing a node

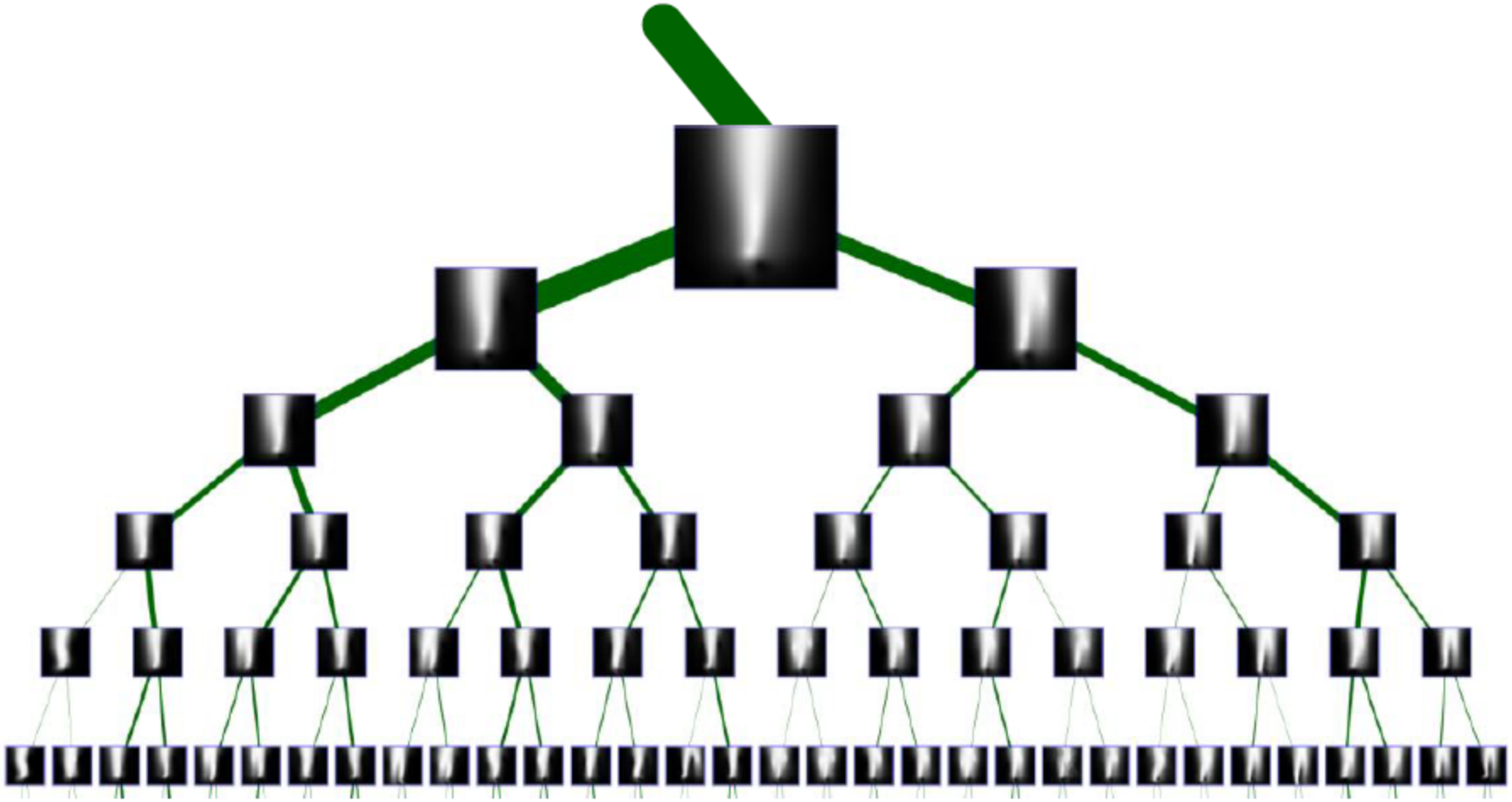
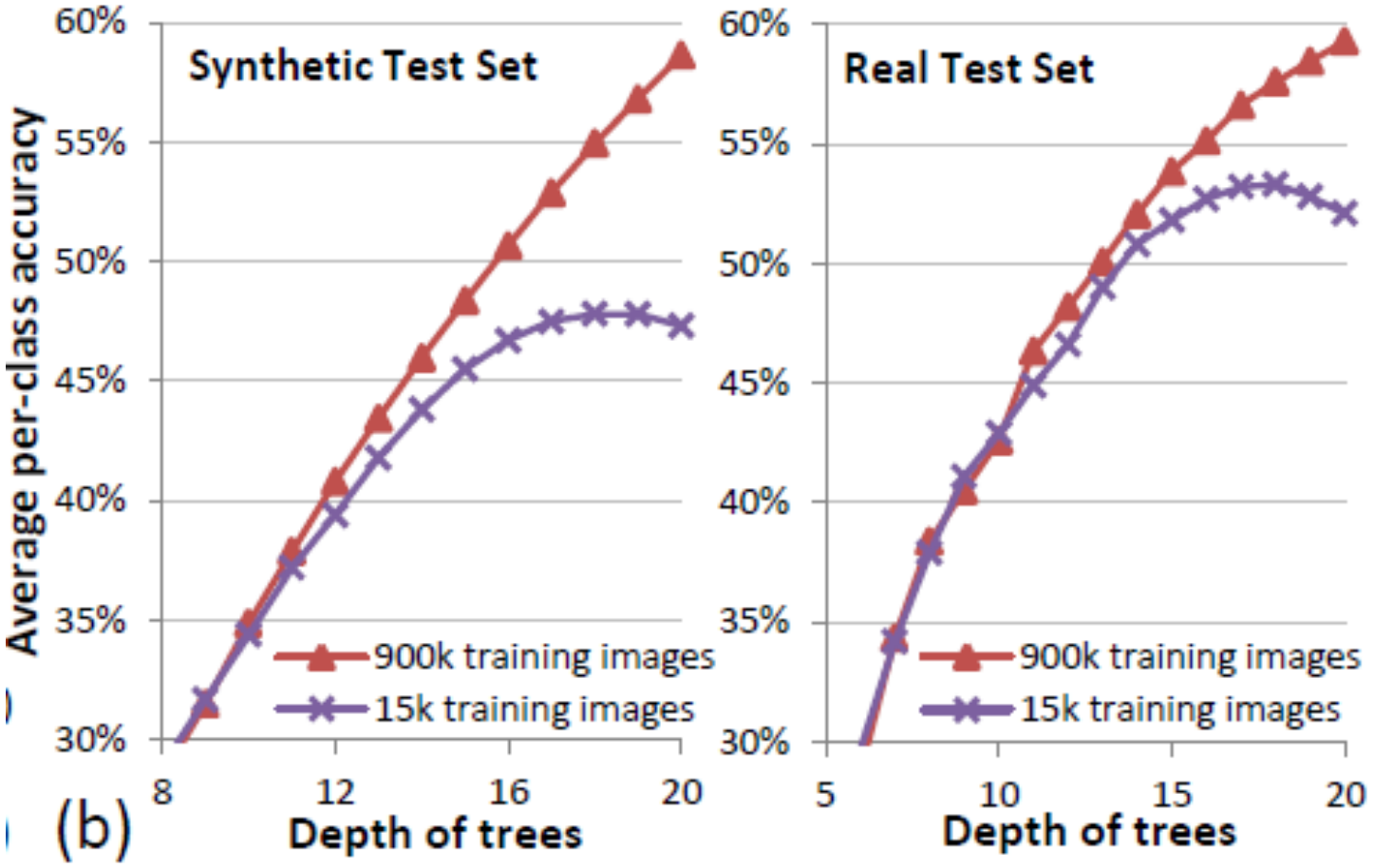


Figure 11. **Visualization of a trained decision tree.** Two separate subtrees are shown. A depth image patch centered on each pixel is taken, depth normalized, and binarized to a foreground/background silhouette. The patches are averaged across all pixels that reached any given tree node. The thickness of the edges joining the tree nodes is proportional to the number of pixels, and here shows fairly balanced trees. All pixels from 15k images are used to build the visualization shown.

POWER OF FUNCTION COMPOSITION

Accuracy as function of tree complexity & dataset size



POWER OF FUNCTION COMPOSITION

Decision Trees/Forests

Extremely flexible classifiers (power of composition)

Low-cost (simple comparisons, no kernel evaluations)

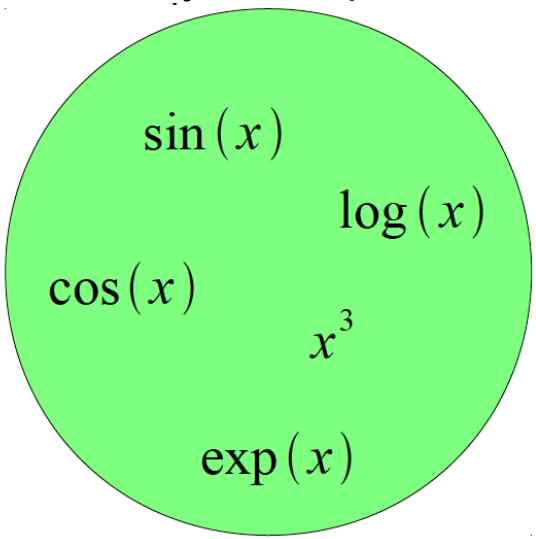
But: learned greedily

Once root has been found, there's no turning back

Can we optimize a composition of functions?

Building A Complicated Function

Given a library of simple functions

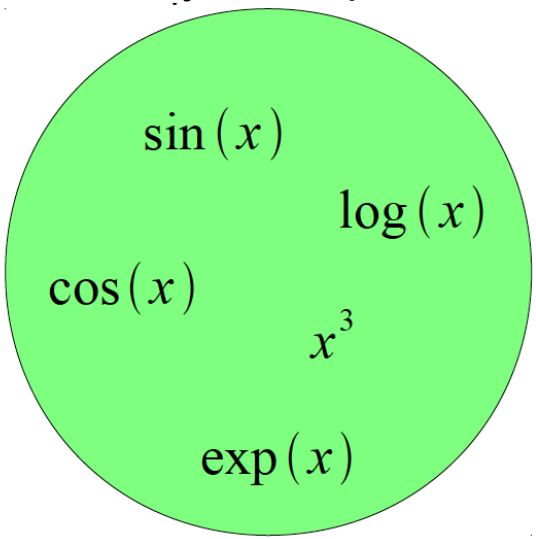


Compose into a

complicated function

Building A Complicated Function

Given a library of simple functions



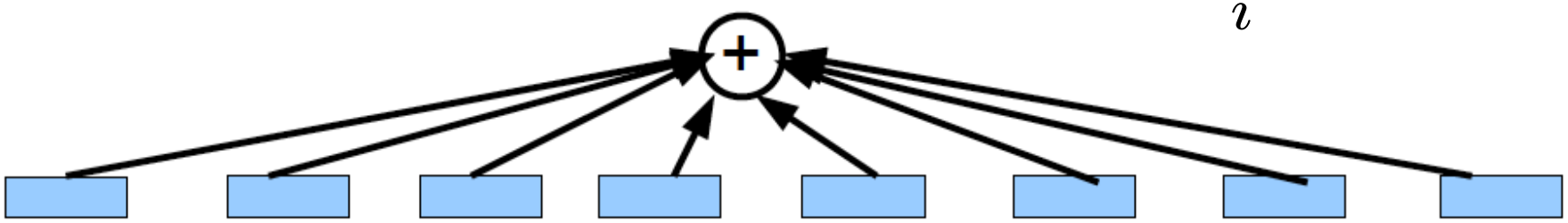
Compose into a

complicated function

Idea 1: Linear Combinations

- Boosting
- Kernels
- …

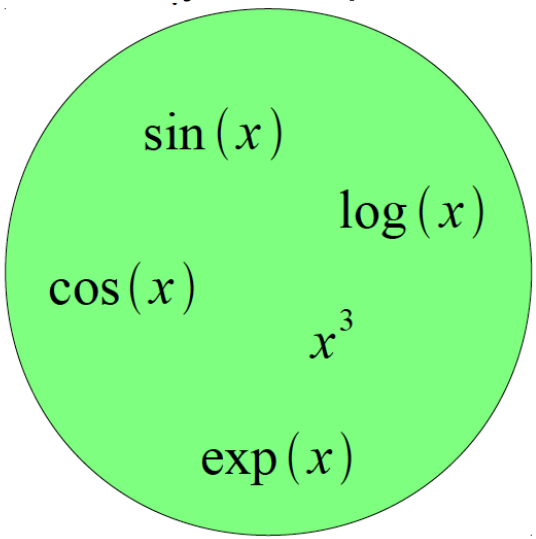
$$f(x) = \sum_i \alpha_i g_i(x)$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



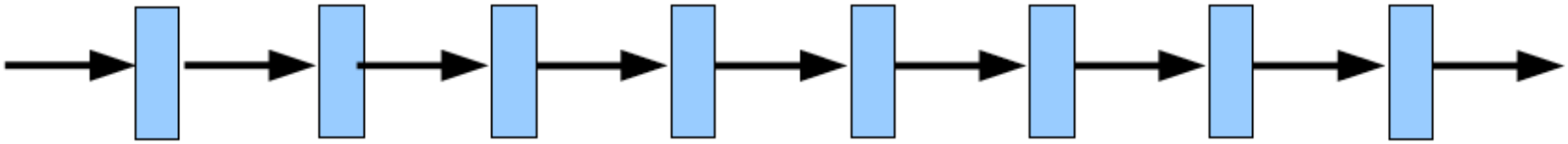
Compose into a

complicated function

Idea 2: Compositions

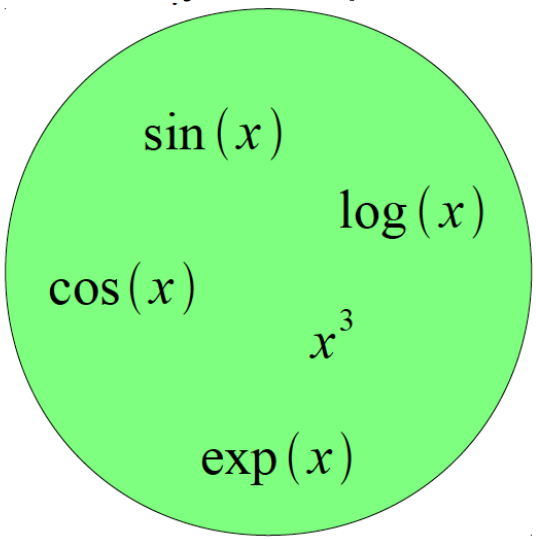
- Decision Trees
- Grammar models
- Deep Learning

$$f(x) = g_1(g_2(\ldots(g_n(x)\ldots))$$



Building A Complicated Function

Given a library of simple functions



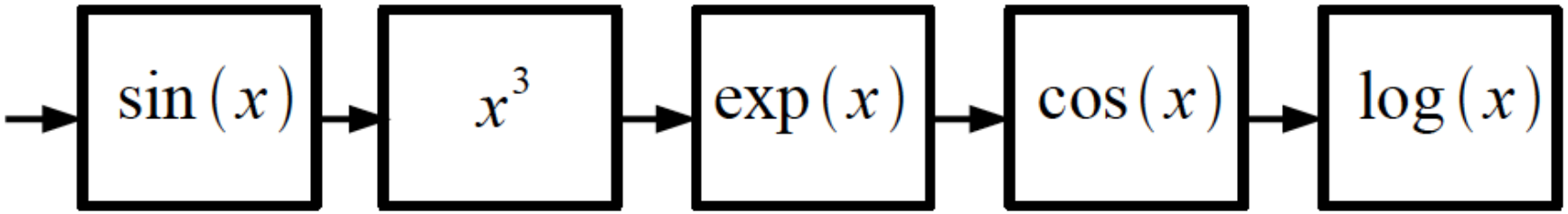
Compose into a

complicate function

Idea 2: Compositions

- Decision Trees
- Grammar models
- Deep Learning

$$f(x) = \log(\cos(\exp(\sin^3(x))))$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Idea: build complex functions out of simple ones

Analogy with calculus

Power series: flat representation of functions

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots,$$

one power series per function

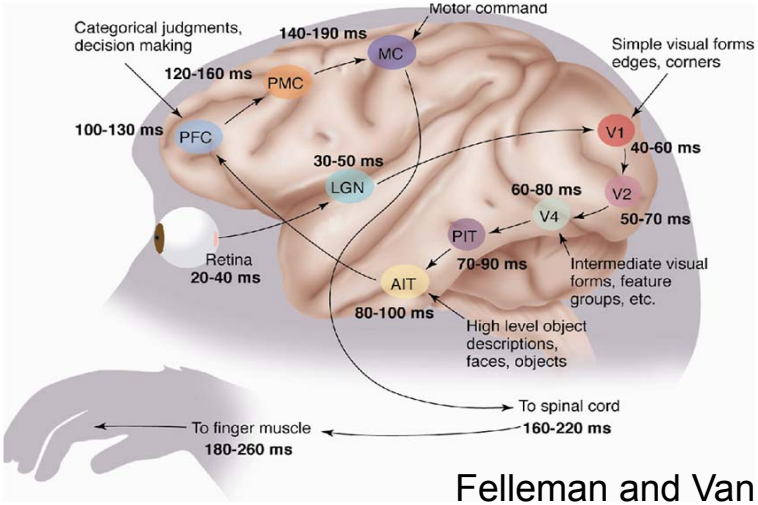
Function composition: deep representation of functions

$$e^{\sin(x)}, e^{2\sin(x)}, e^{2\sin(2x)}, e^{\sin(e^x)} \ldots$$

‘Neuron’: basic building block

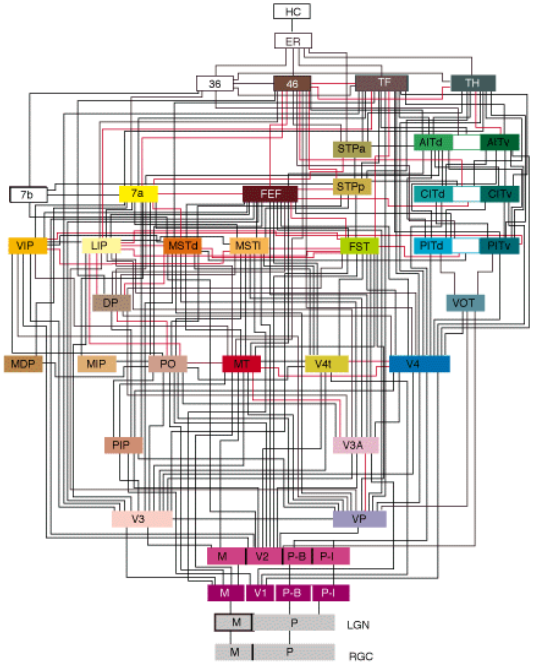
Analogy with the brain

Hierarchical organization of the cortex

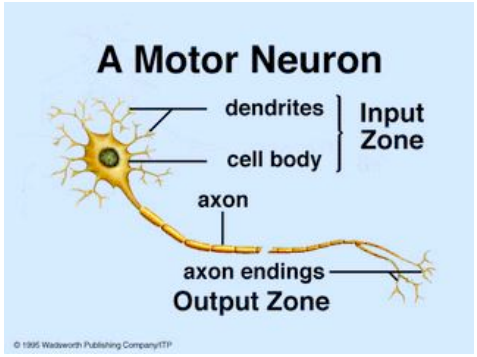


Felleman and Van Essen, 1991

Diagram of the visual system



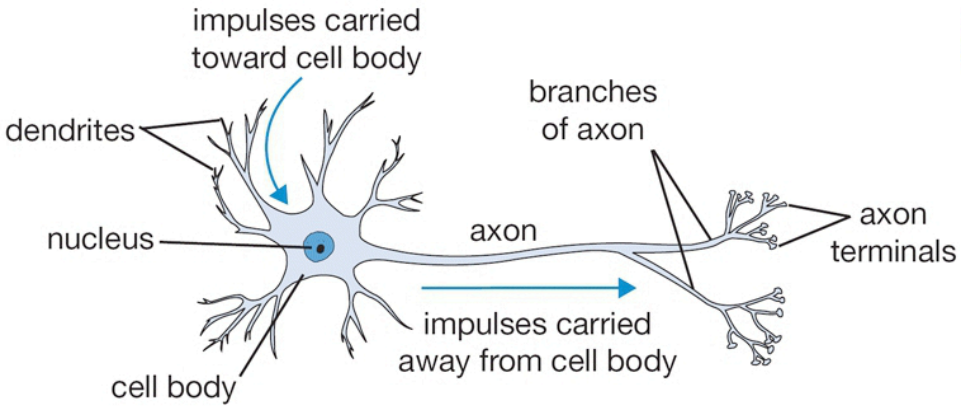
Building blocks: neurons



same neuron dynamics throughout

The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Artificial neuron
  - Multiply inputs by weights along edges
  - Apply some function to the set of inputs at each node



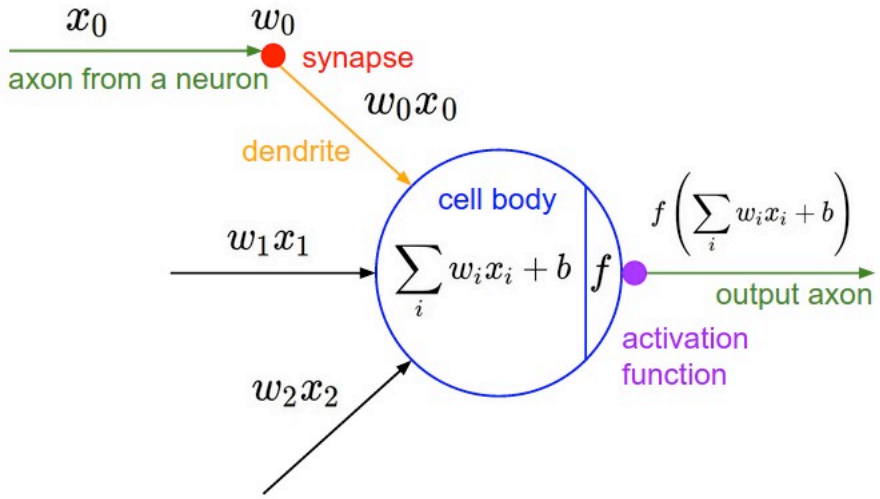
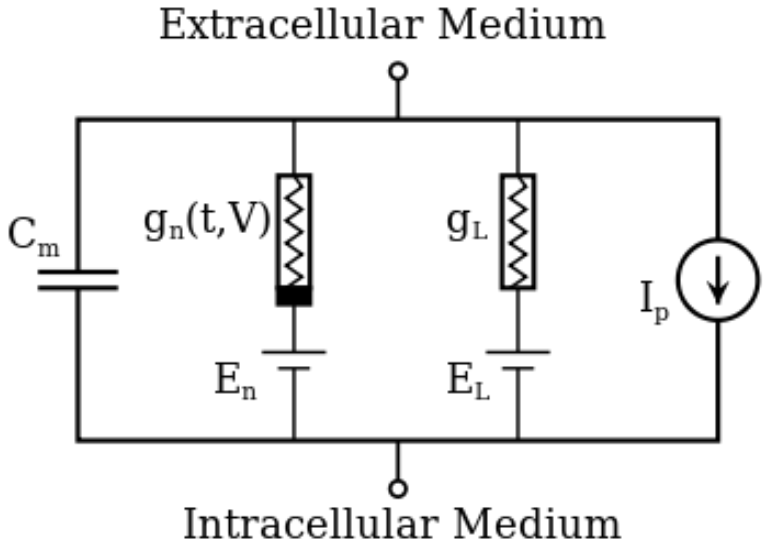


Image Credit: Andrej Karpathy, CS231n

Side comment: real neurons are much more complicated

- Hodgkin-Huxley equation

1963 Nobel prize
in Physiology

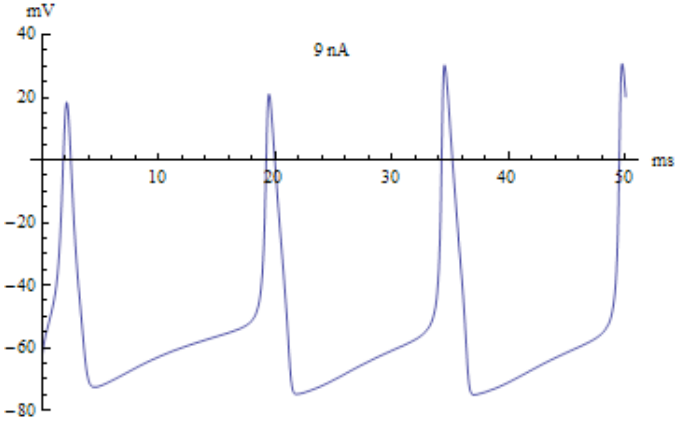


$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_K) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

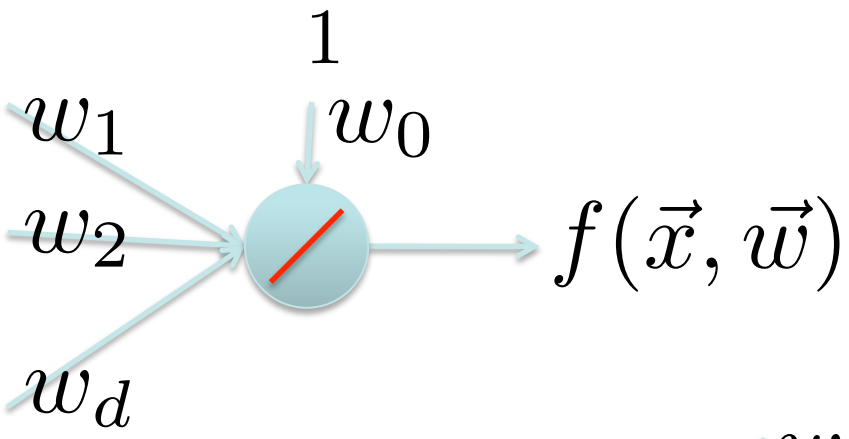
$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

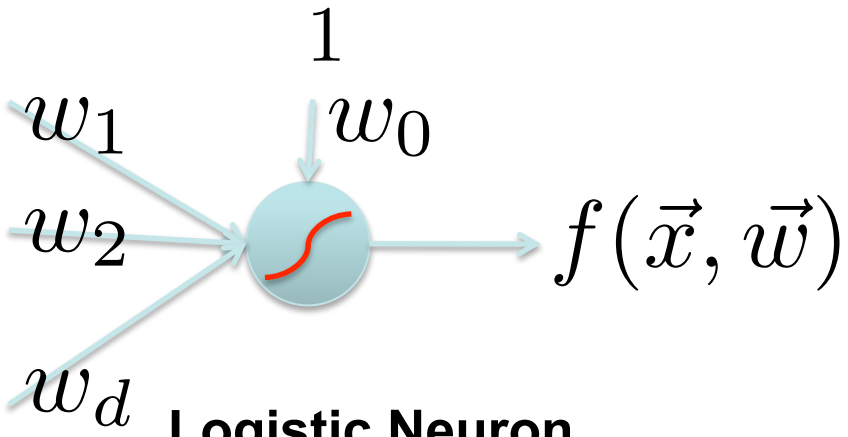


Abstraction: average firing rate

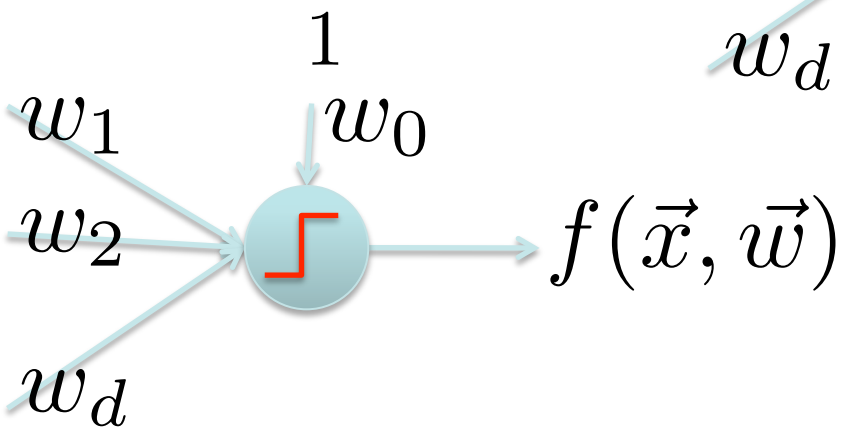
Types of Neurons



Linear Neuron



Logistic Neuron



Perceptron

Activation functions

Step ("perceptron")

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

Sigmoidal ("logistic")

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

Hyperbolic tangent

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

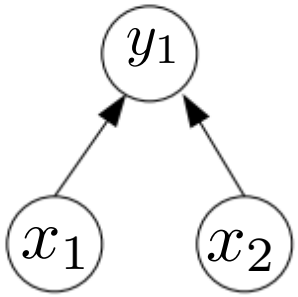
Rectified Linear Unit (RELU)

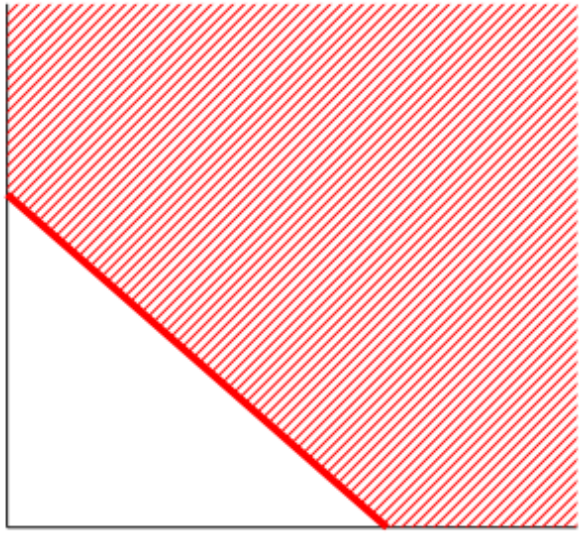
$$g(a) = \max(0, a)$$

Beyond linear boundaries

- Today: ‘deep learning’ (a.k.a. neural network) approach

1 layer of trainable weights



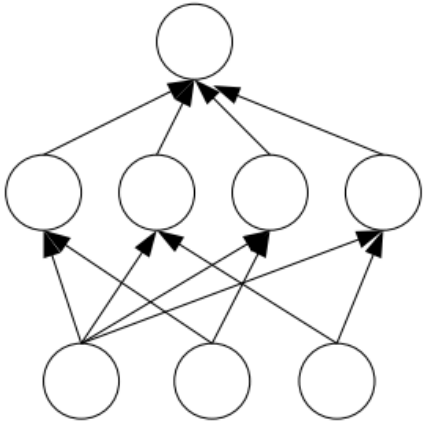


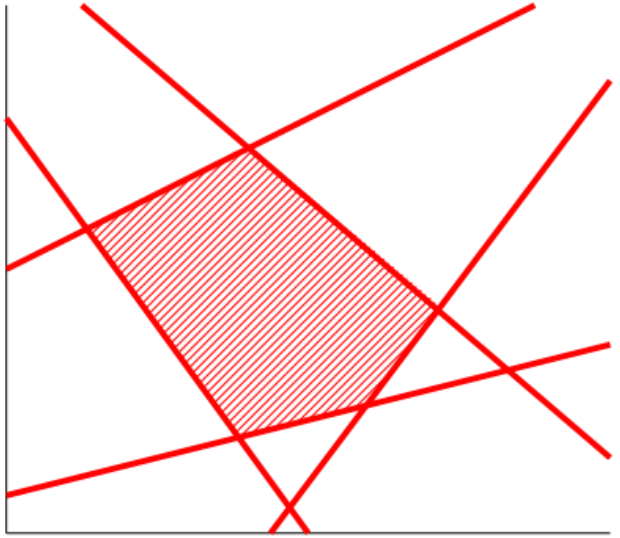
separating hyperplane

Beyond linear boundaries

- Today: ‘deep learning’ (a.k.a. neural network) approach

2 layers of trainable weights



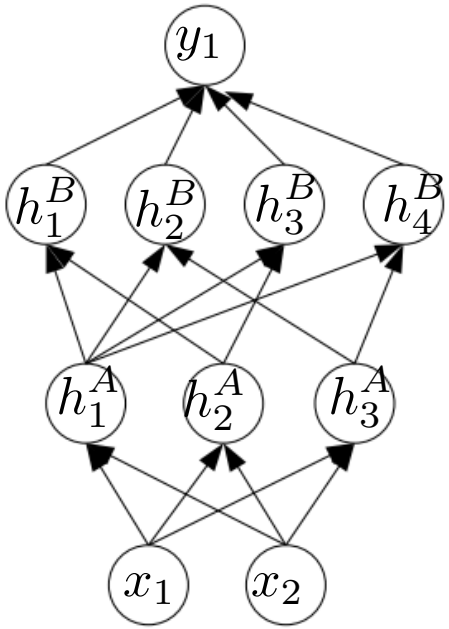


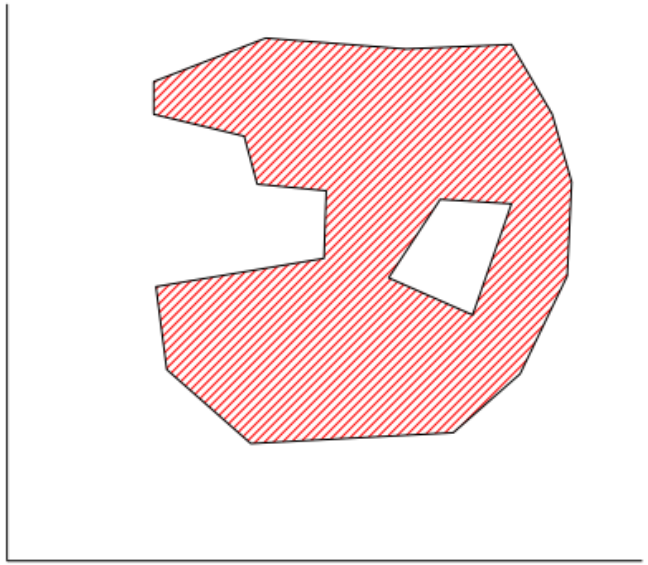
convex polygon region

Beyond linear boundaries

- Today: ‘deep learning’ (a.k.a. neural network) approach

3 layers of trainable weights





Idea: build complex functions out of simple ones

Analogy with calculus

Power series: flat representation of functions

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots,$$

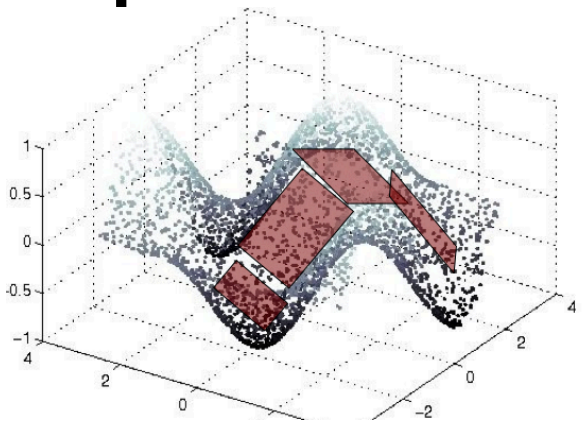
one power series per function

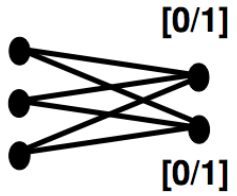
Function composition: deep representation of functions

$$e^{\sin(x)}, e^{2\sin(x)}, e^{2\sin(2x)}, e^{\sin(e^x)} \ldots$$

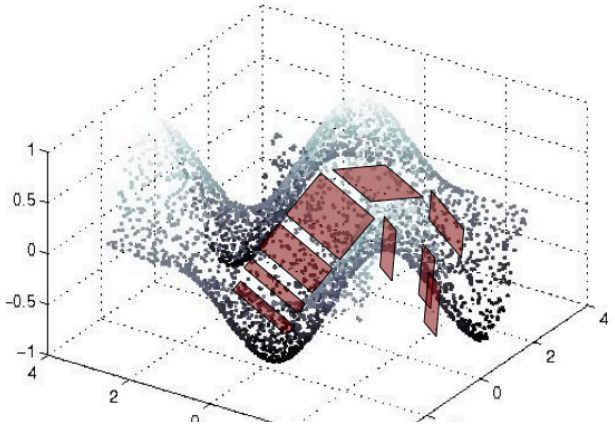
Deep = rich, but cheap

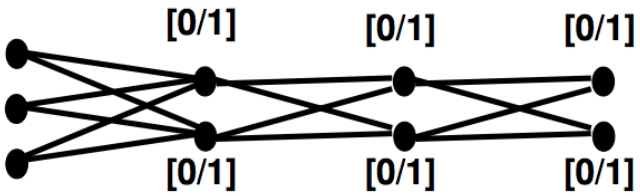
$$\boldsymbol{h}^1 = max(0, W^1 \boldsymbol{x} + \boldsymbol{b}^1)$$





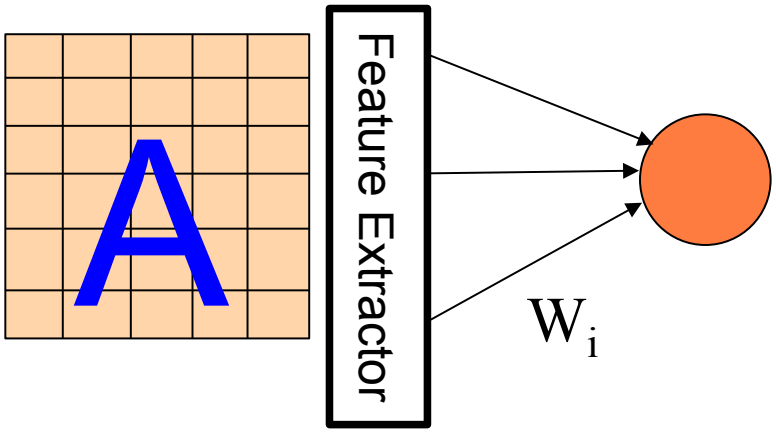
ReLU layers do local linear approximation. Number of planes grows exponentially with number of hidden units. Multiple layers yeild exponential savings in number of parameters (parameter sharing).



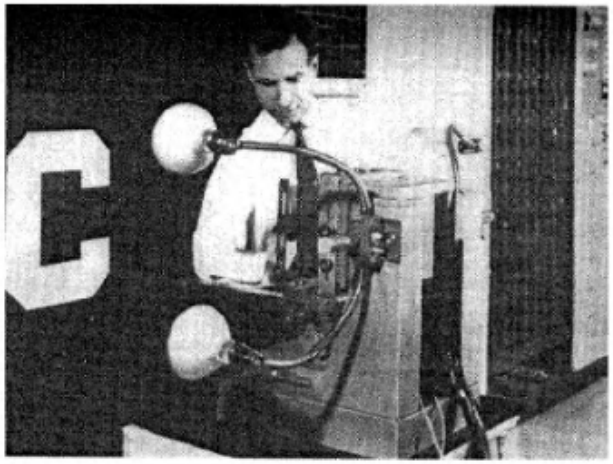


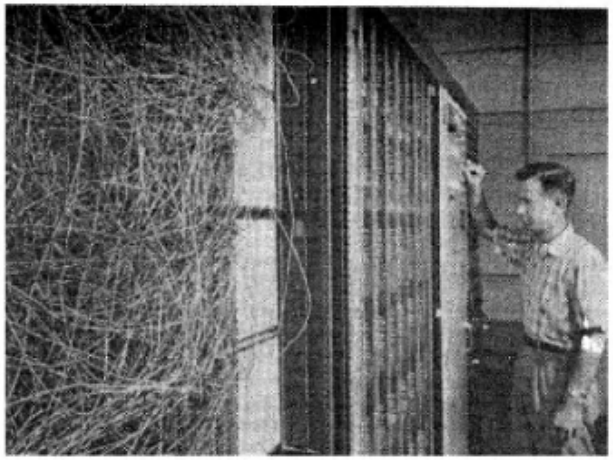
Quite an old paradigm

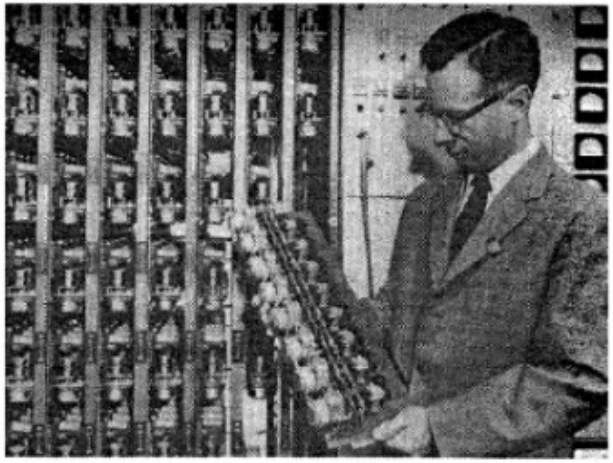
- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



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

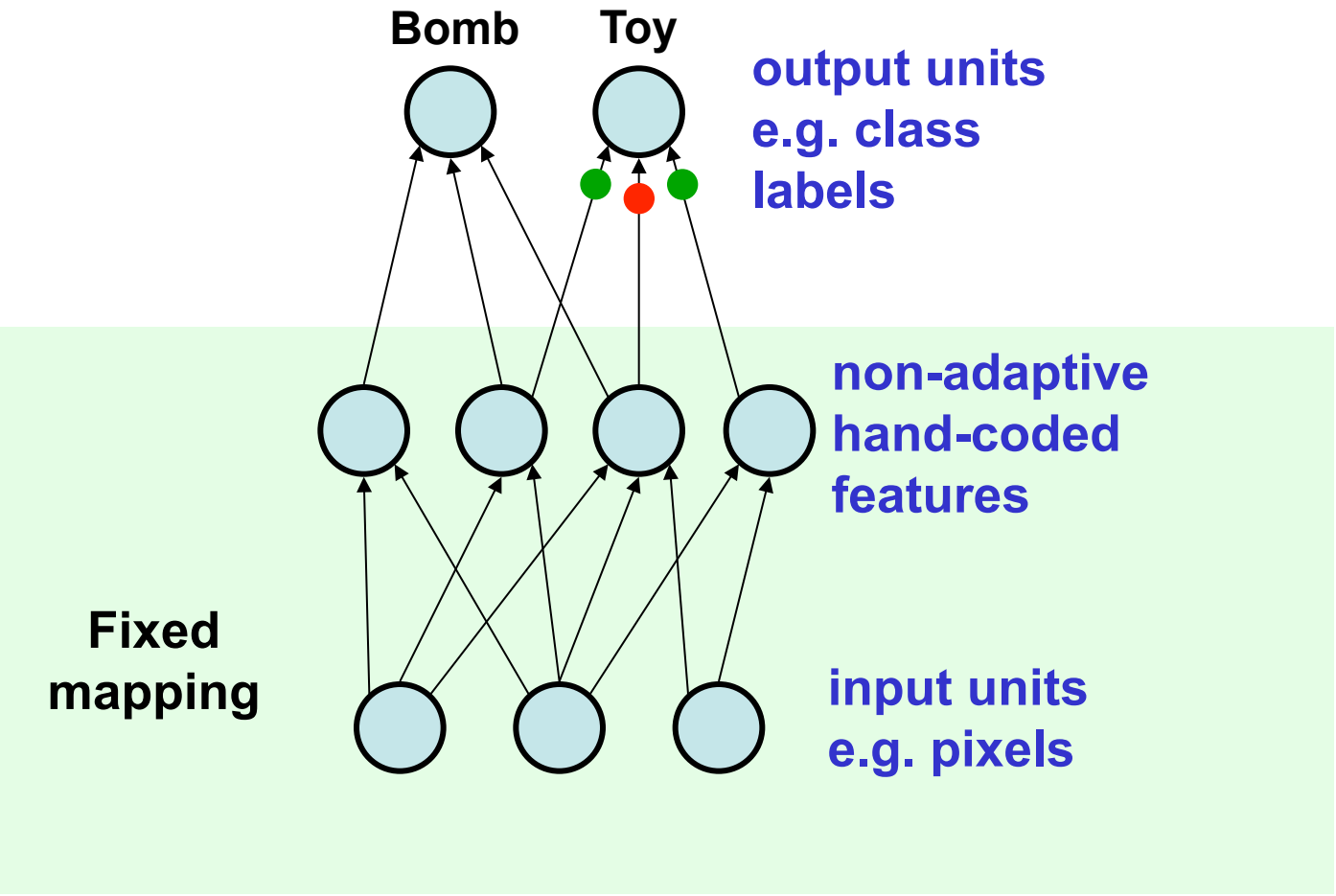






Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Perceptron, ‘60s

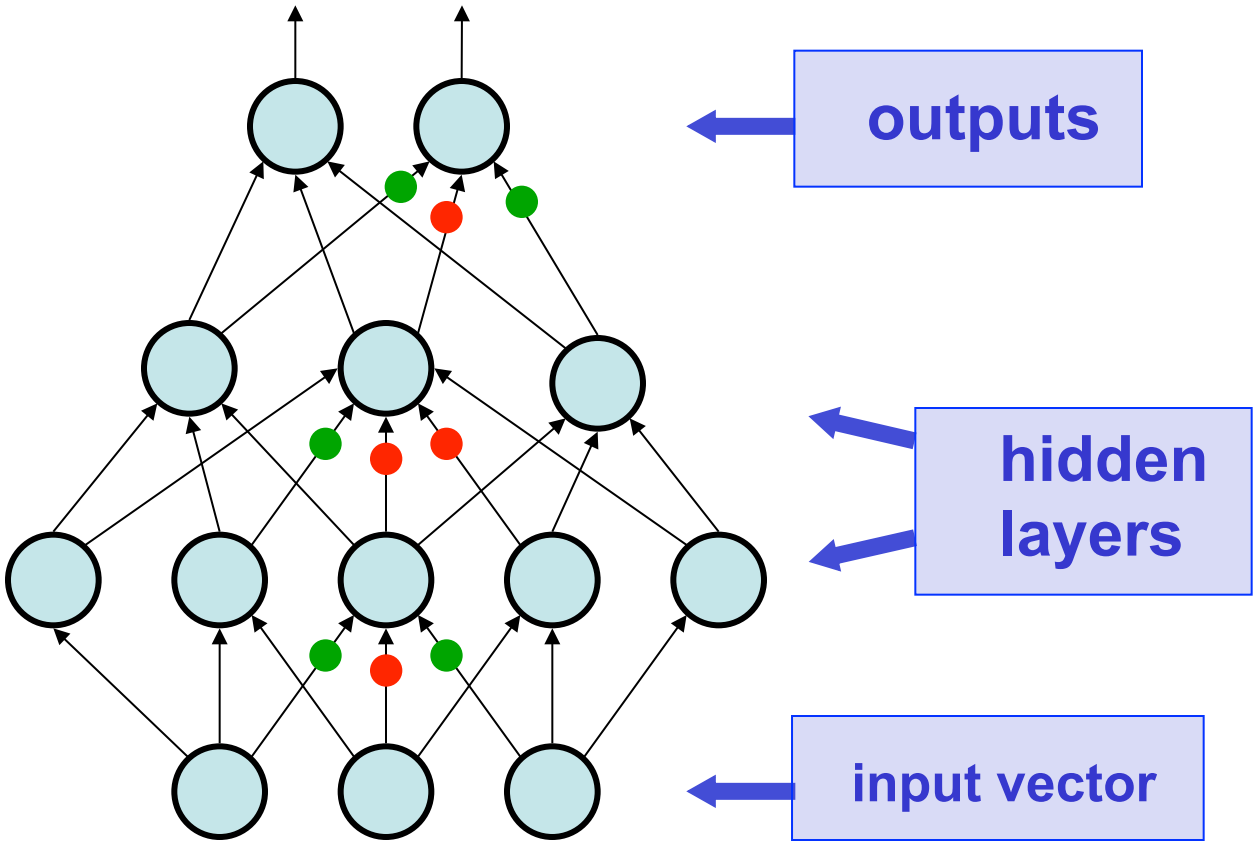


One of the first data-driven models in AI

Slide credits: G. Hinton

Multi-Layer Perceptrons (~1985)

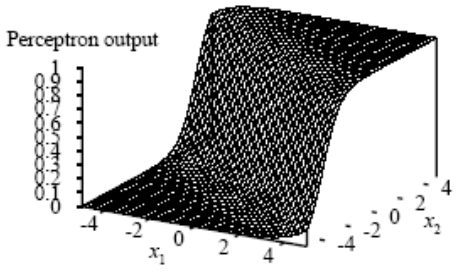
$$u_i = g\left(\sum_{k\in\mathcal{N}(i)} w_{k,i}\, g\left(\sum_{m\in\mathcal{N}(k)} w_{m,k}u_m + b_k\right) + b_i\right)$$



Slide credits: G. Hinton

Expressiveness of perceptrons

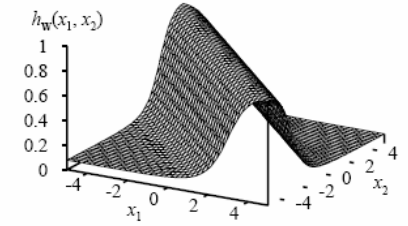
Single layer perceptron:
Linear classifier



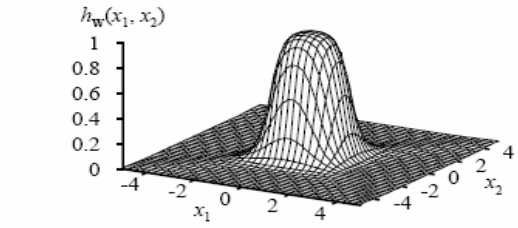
`soft threshold function'

(b)

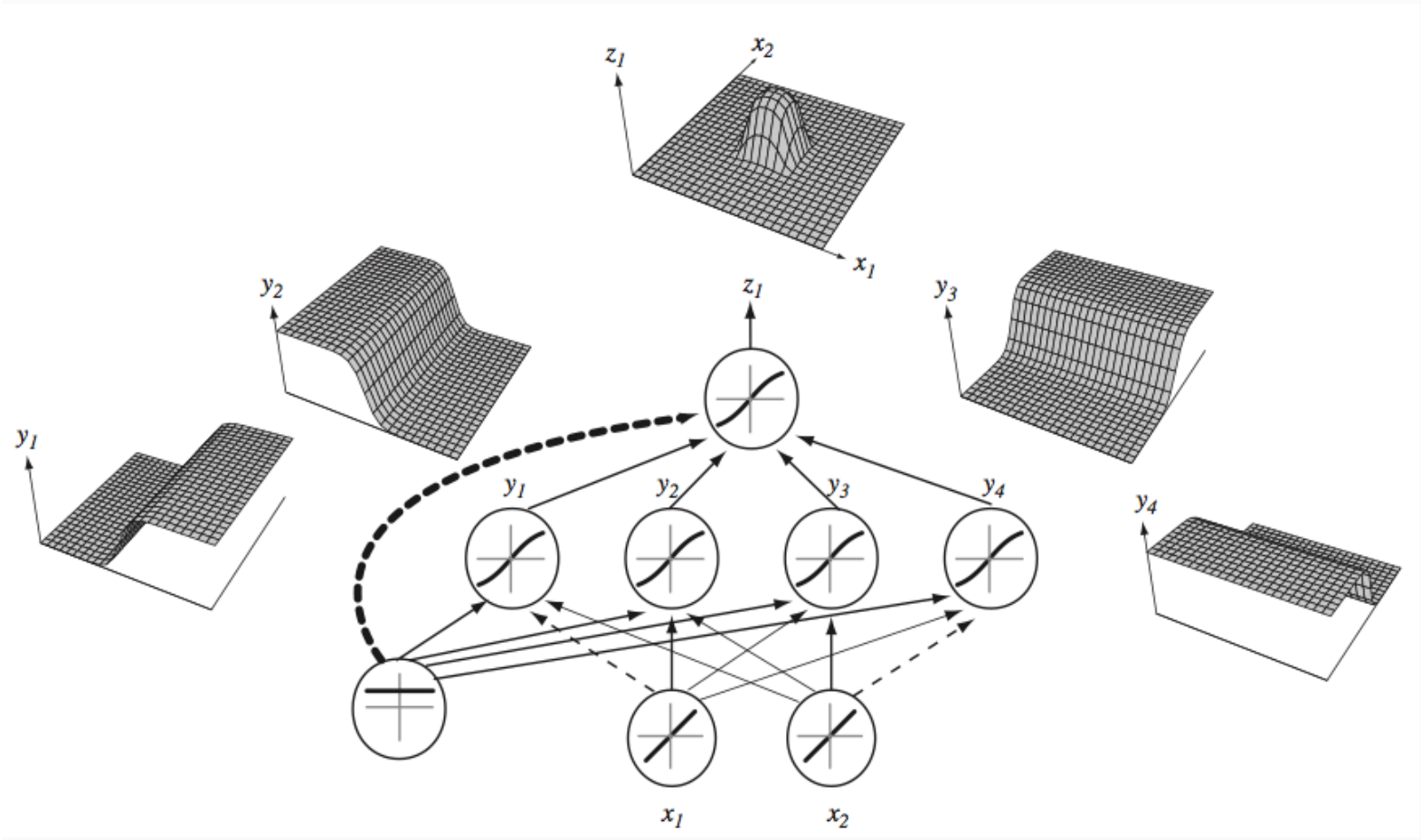
Two opposite `soft threshold' functions: a ridge



Two ridges: a bump



A network for a single bump

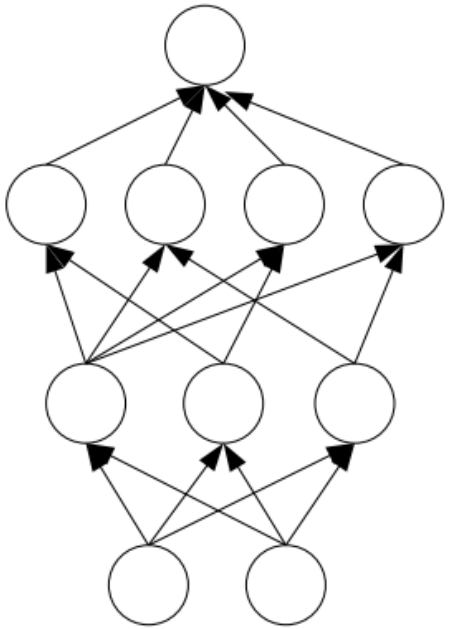


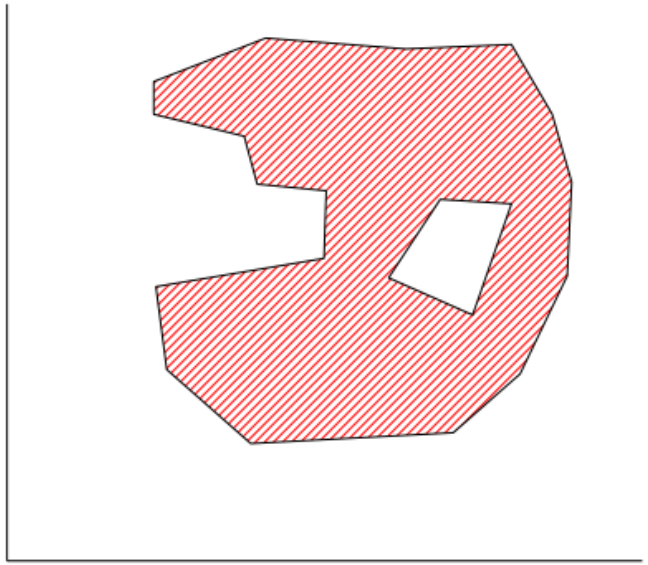
Any function: sum of bumps

Beyond linear boundaries

- Today: ‘deep learning’ (a.k.a. neural network) approach

3 layers of trainable weights

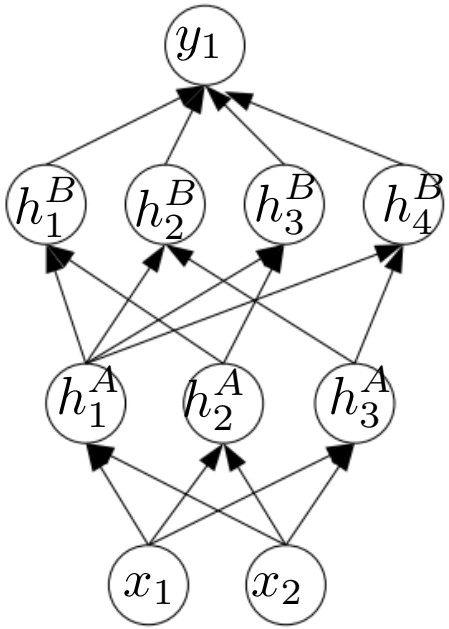


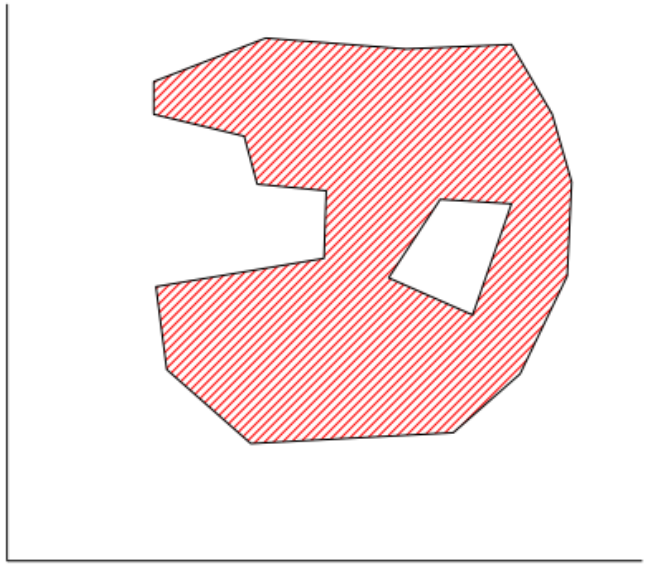


Beyond linear boundaries

- Today: ‘deep learning’ (a.k.a. neural network) approach

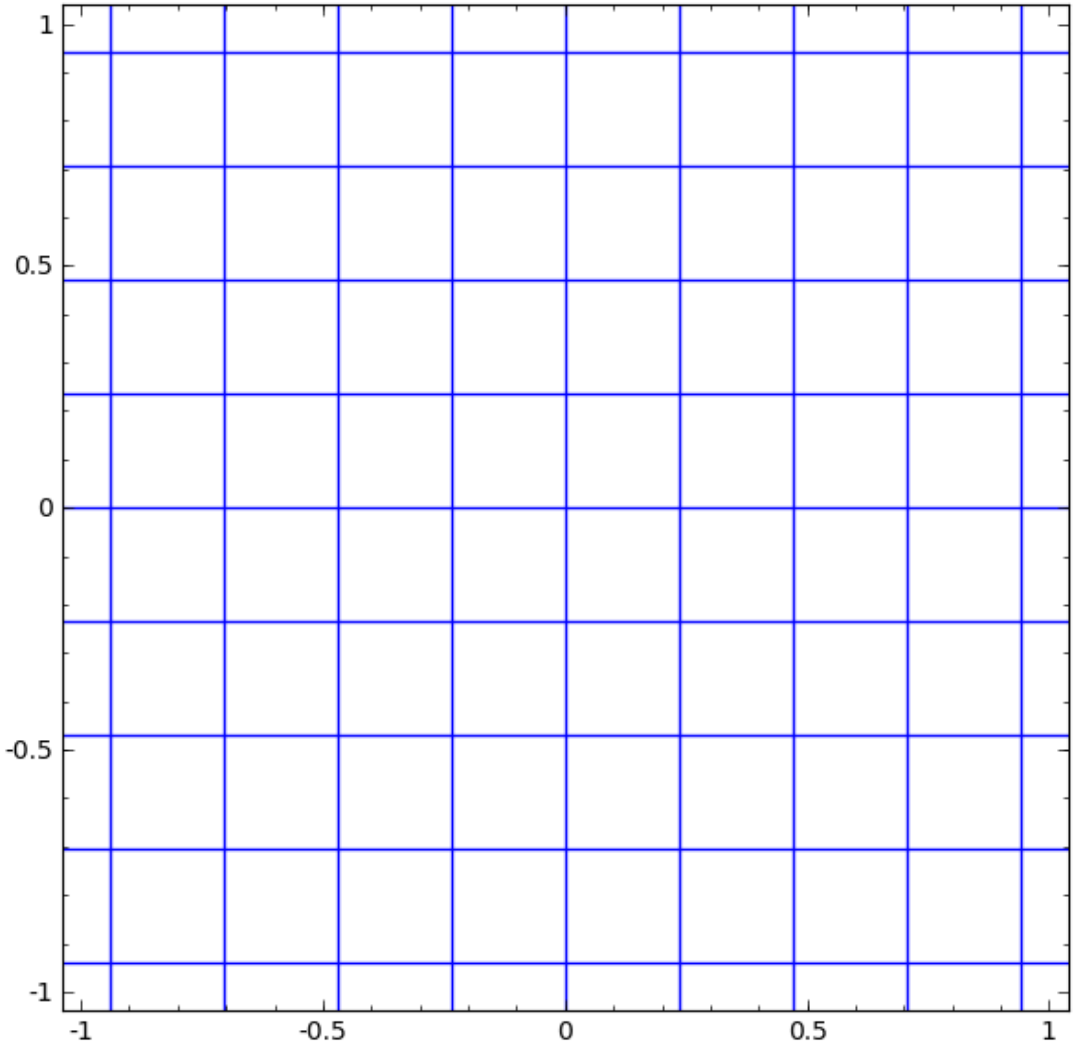
3 layers of trainable weights





neuron-based mapping $\mathbb{R}^2 \rightarrow \mathbb{R}^2$

Evolution of isocontours as parameters change



$$y_1 = g(w_1^1 x_1 + w_1^1 x_2 + w_3^1))$$

$$y_2 = g(w_1^2 x_1 + w_2^2 x_2 + w_3^2))$$

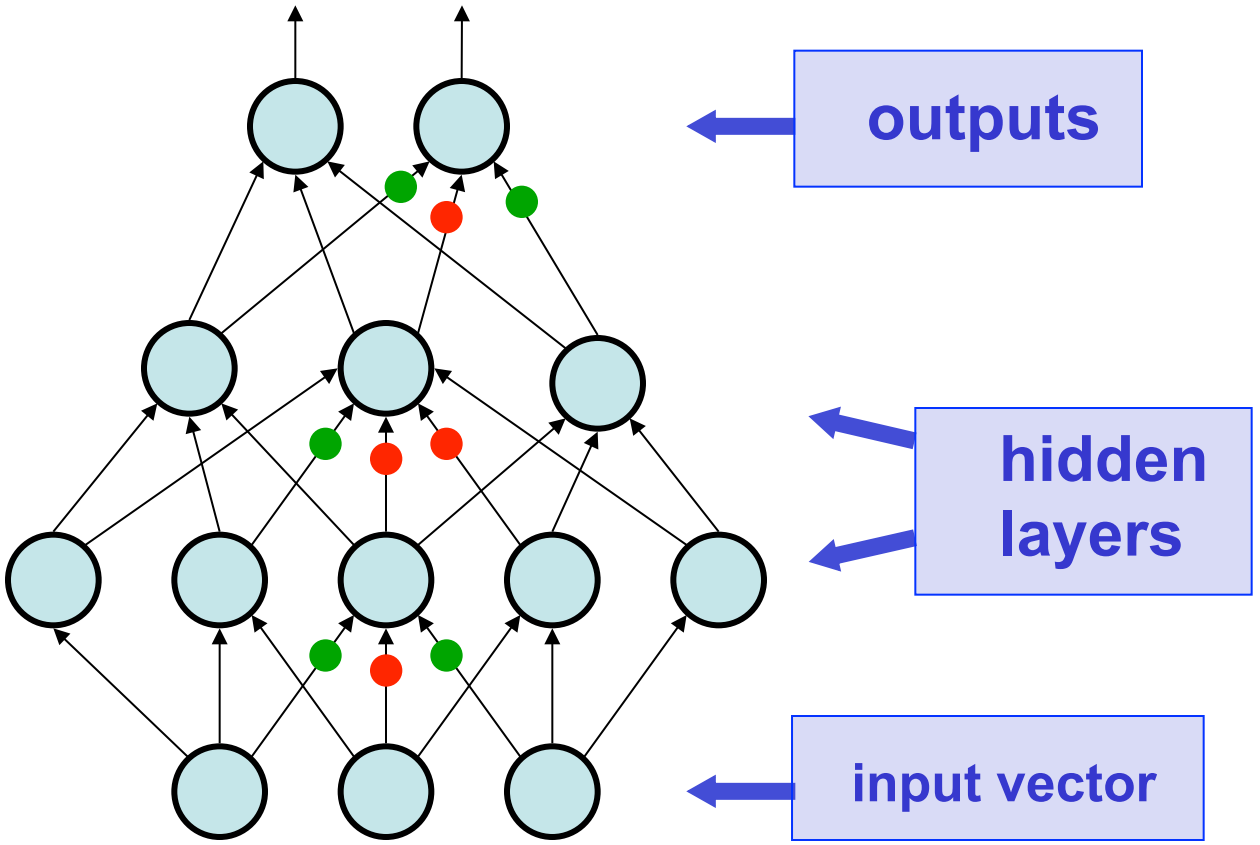
$$\mathbf{y} = g(\mathbf{W}\mathbf{x})$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Multi-Layer Perceptrons (~1985)

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



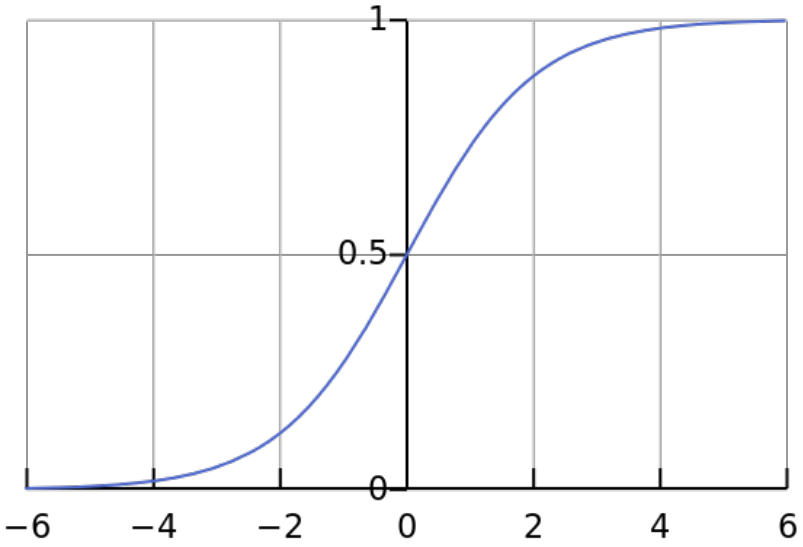
Slide credits: G. Hinton

Input-output at circles: activation functions

Sigmoidal ("logistic")

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

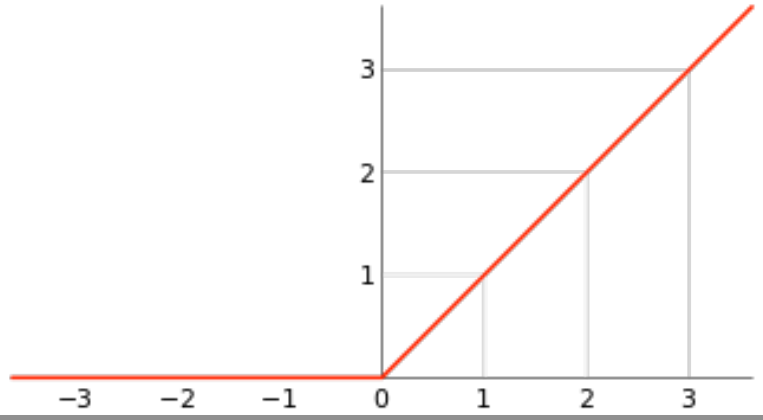
$$g'(a) = g(a)(1 - g(a))$$



Rectified Linear Unit (RELU)

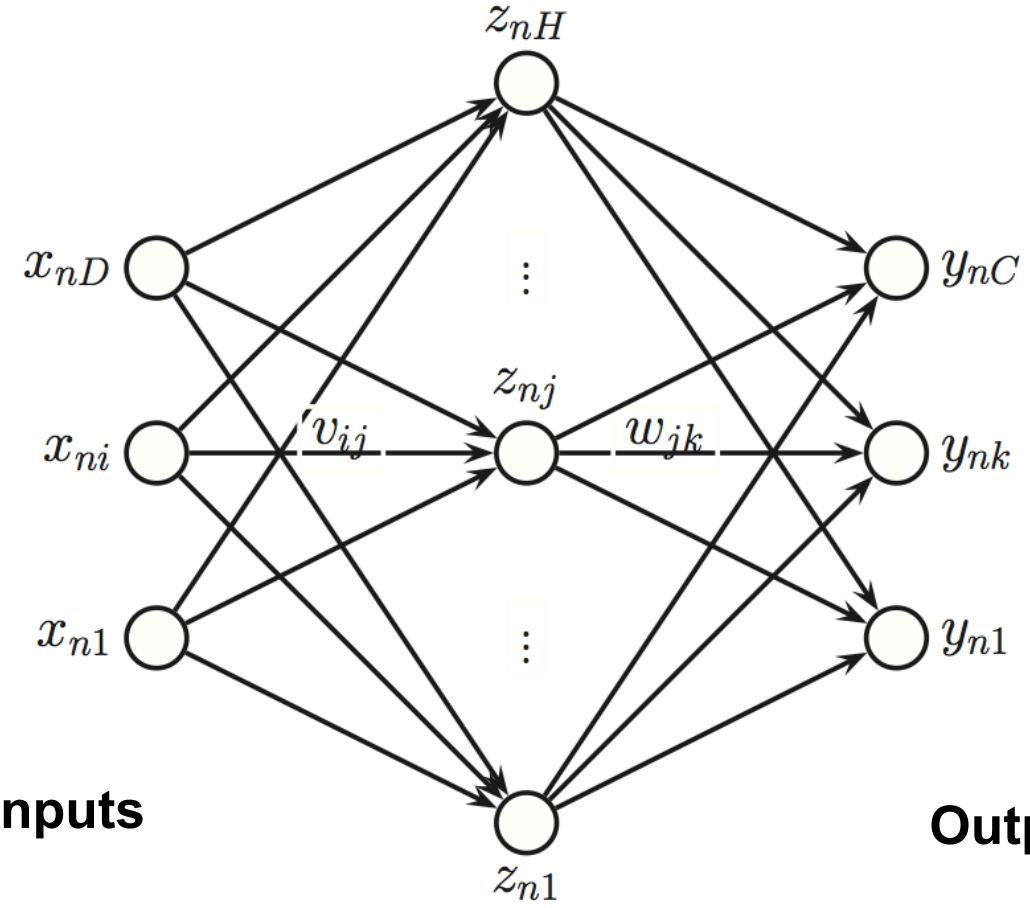
$$g(a) = \max(0, a)$$

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



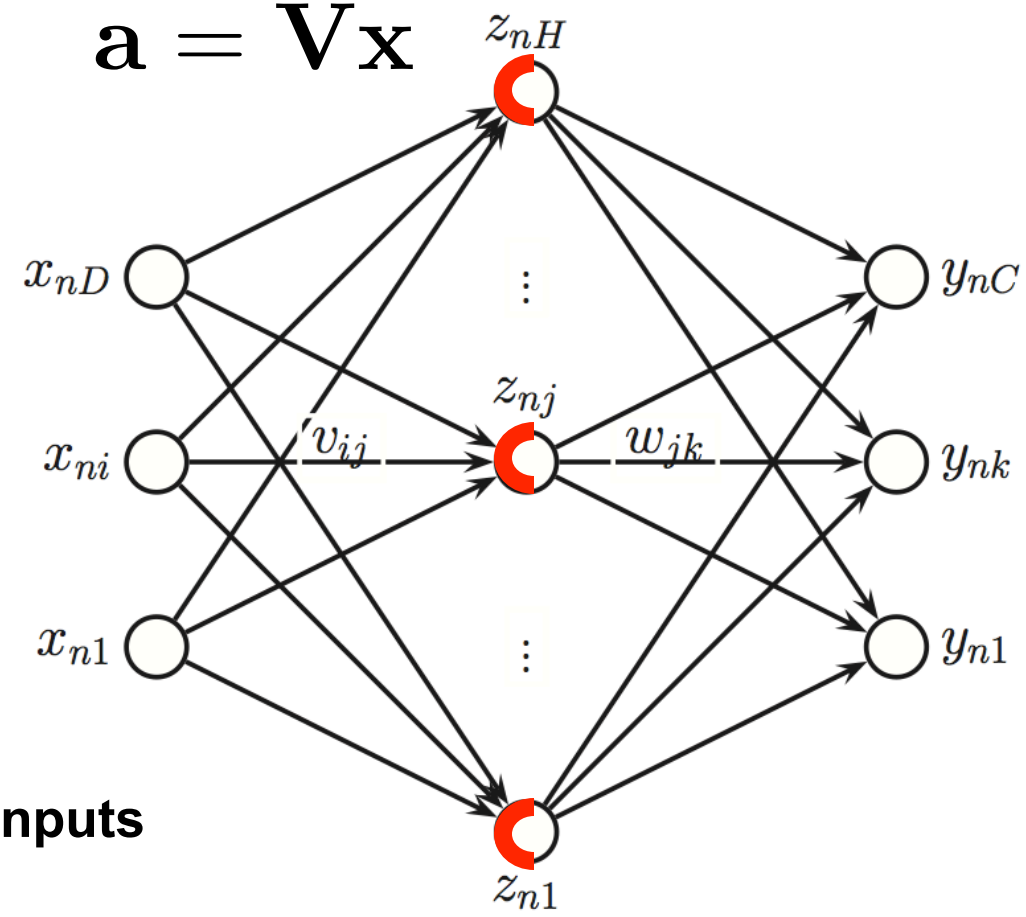
A neural network for multi-way classification

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



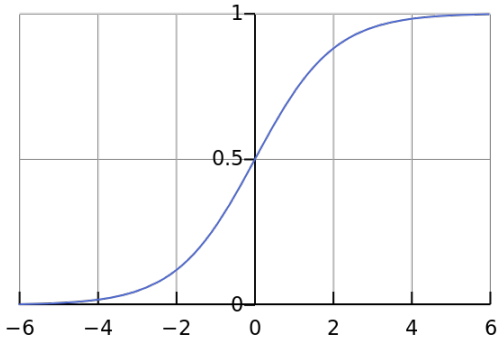
A neural network:

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

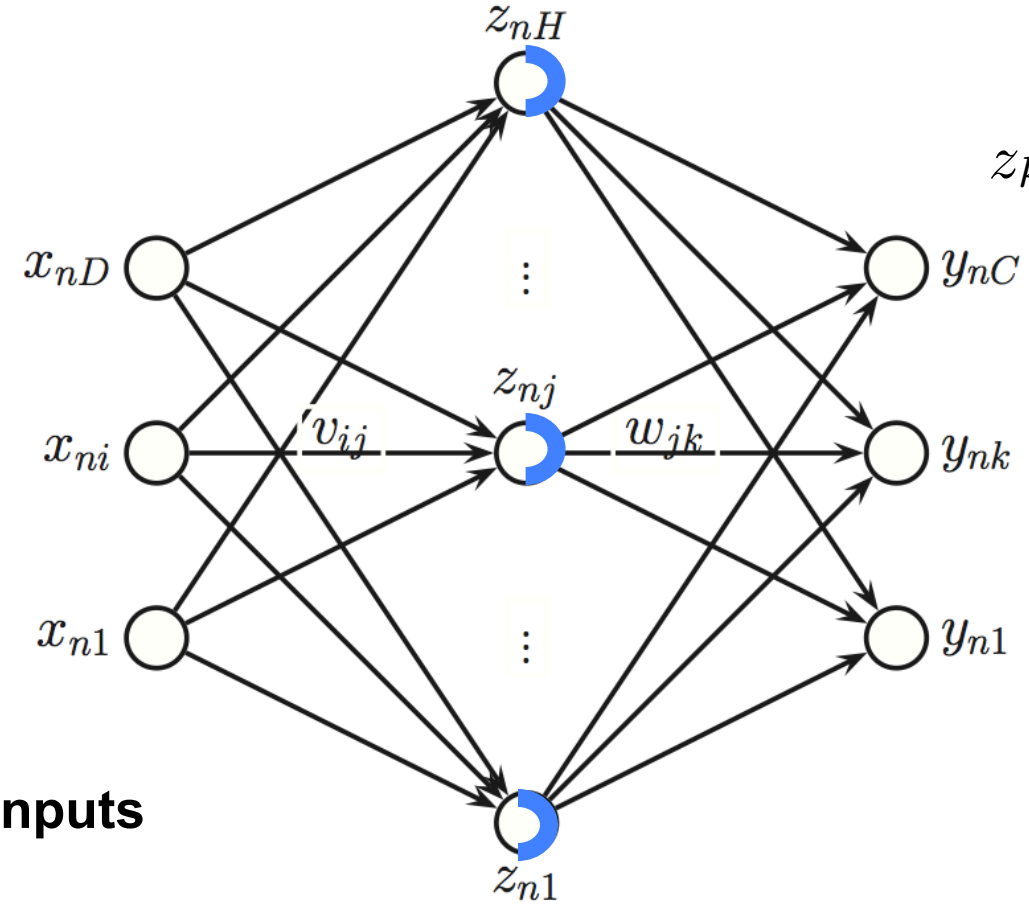


A neural network:

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



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

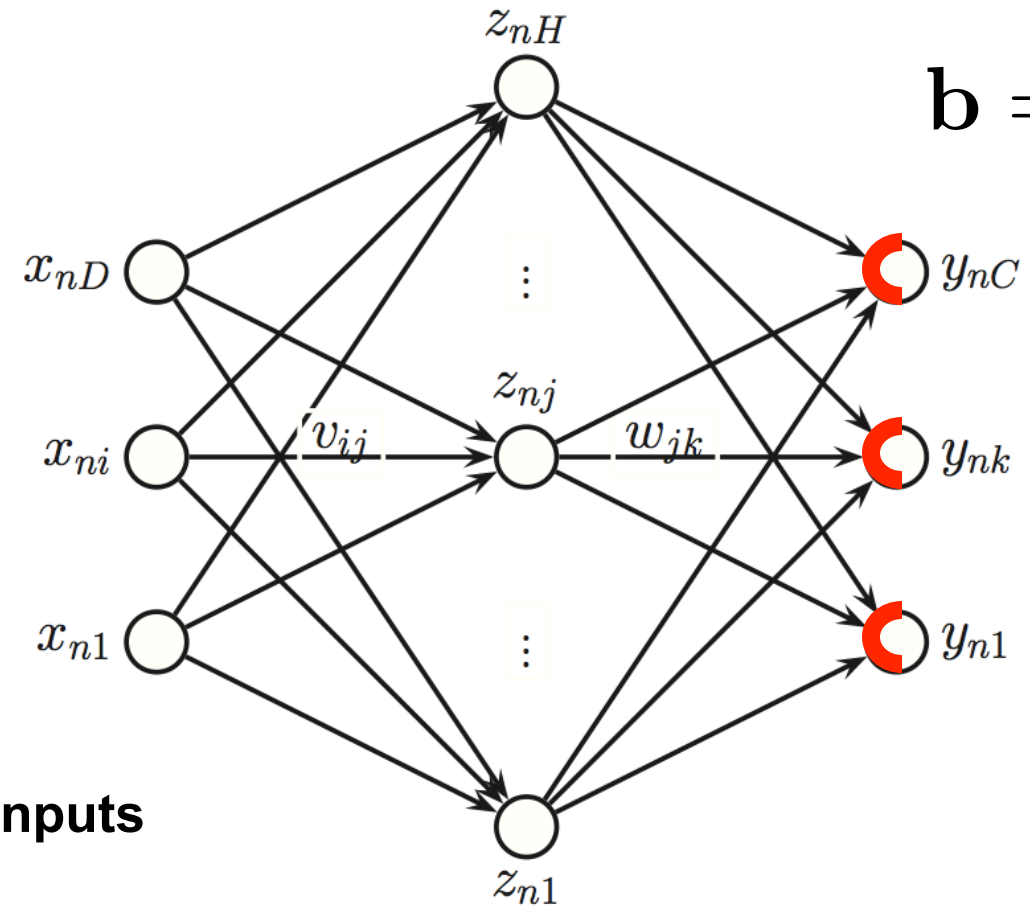


Inputs

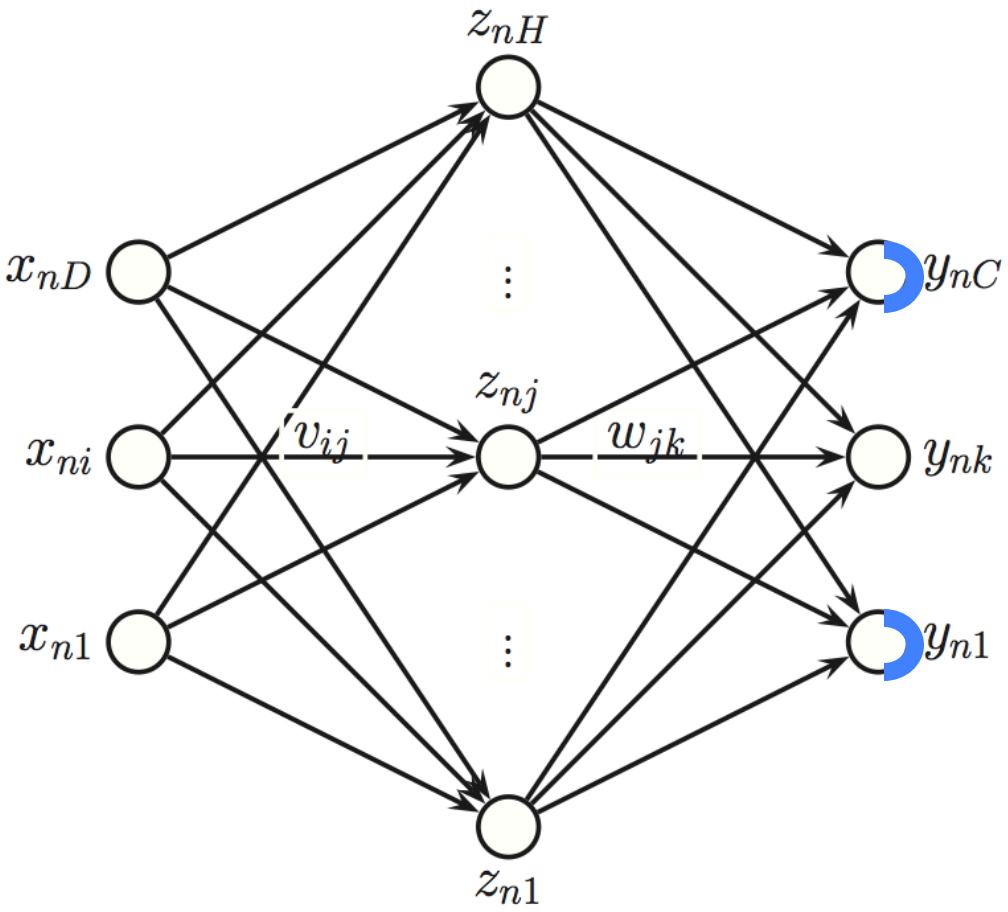
Hidden layer

A neural network:

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



A neural network:



Hidden layer

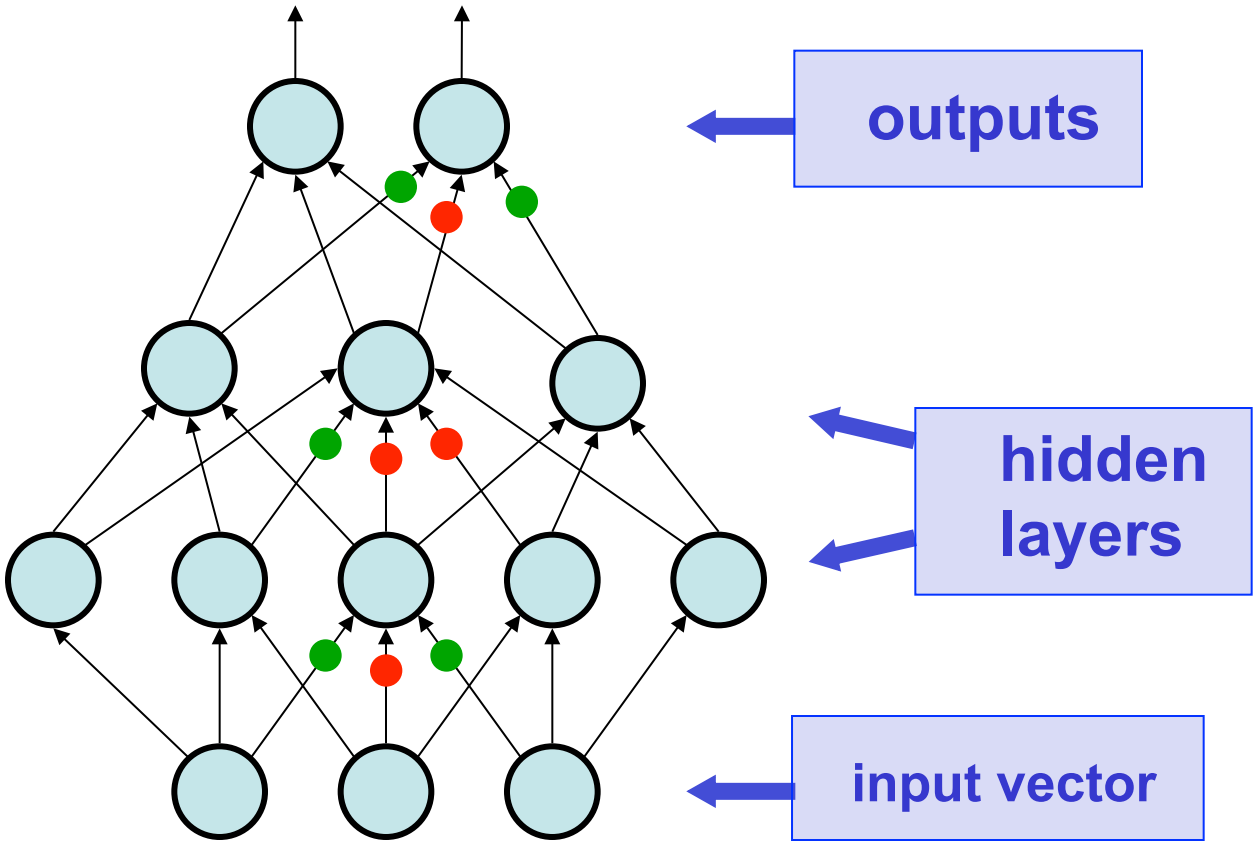
$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$

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

Outputs

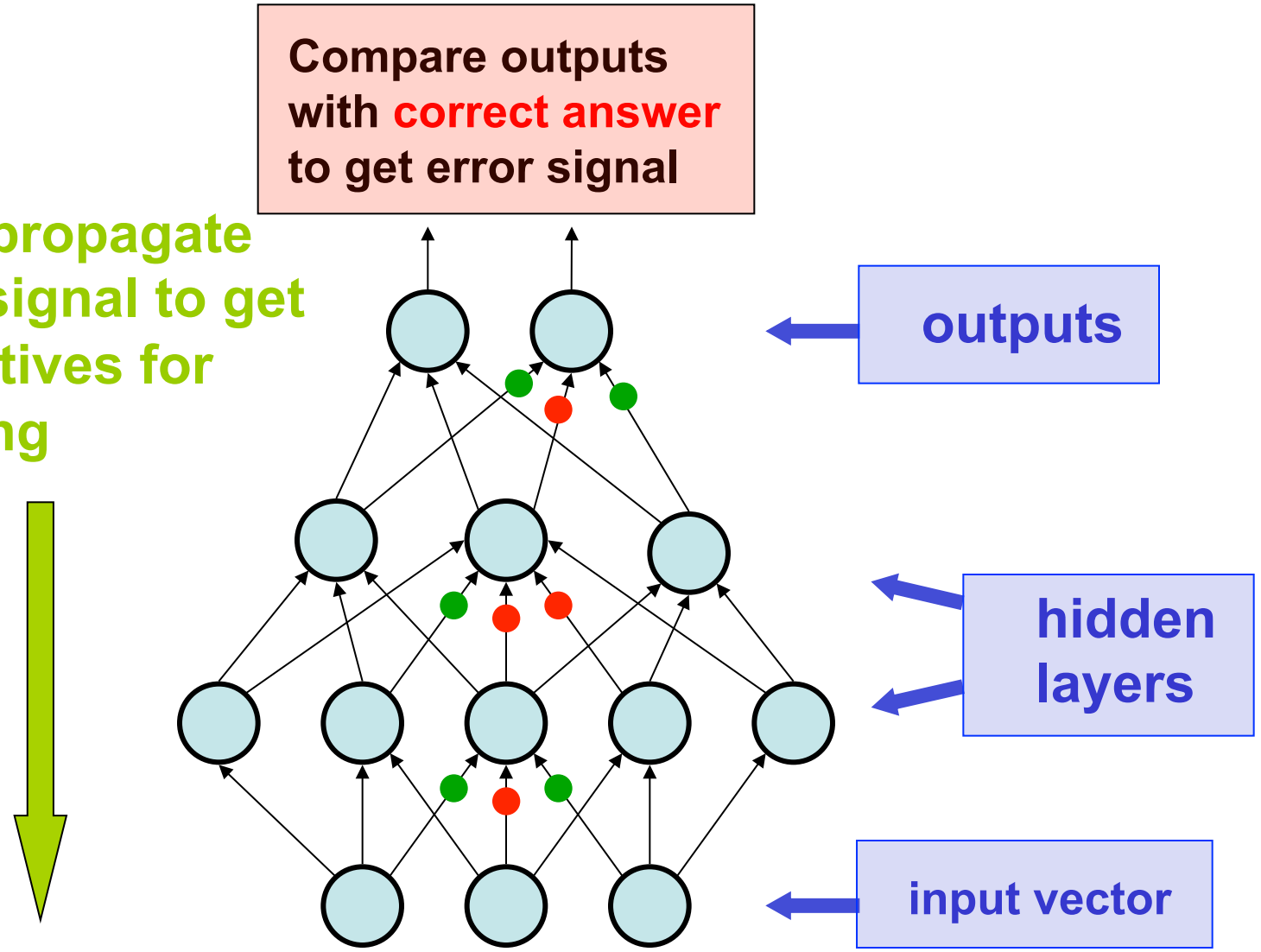
Multi-Layer Perceptrons (~1985)

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



Slide credits: G. Hinton

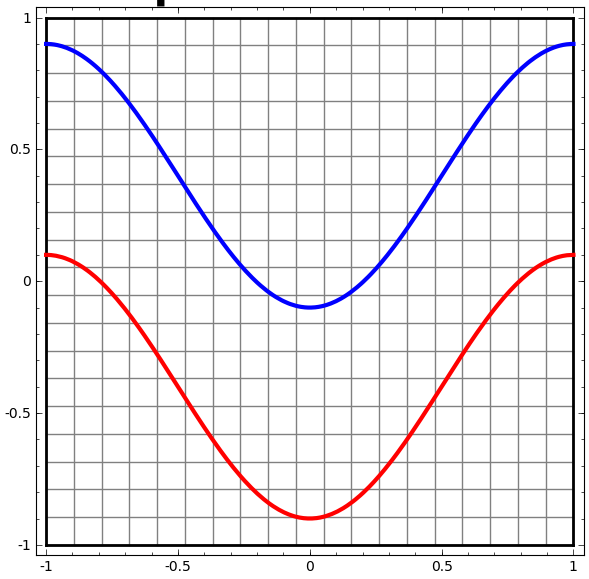
Multi-Layer Perceptrons (~1985)



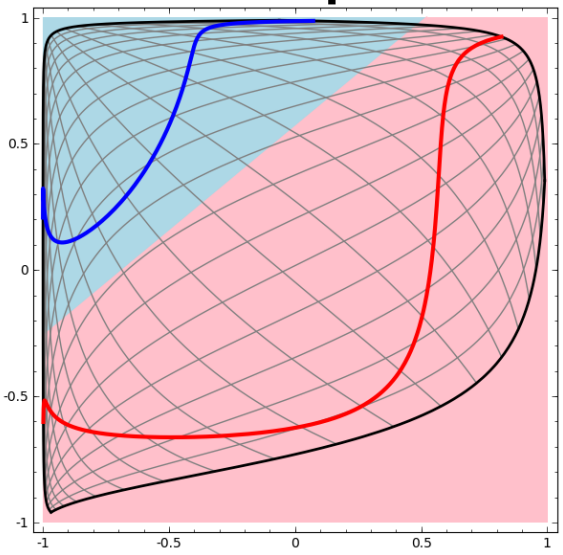
Slide credits: G. Hinton

From non-separable to linearly separable

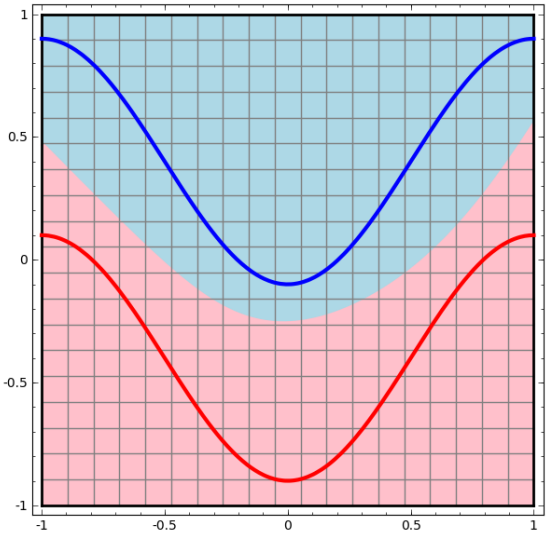
Non-linearly separable data



Data mapped to learnt space

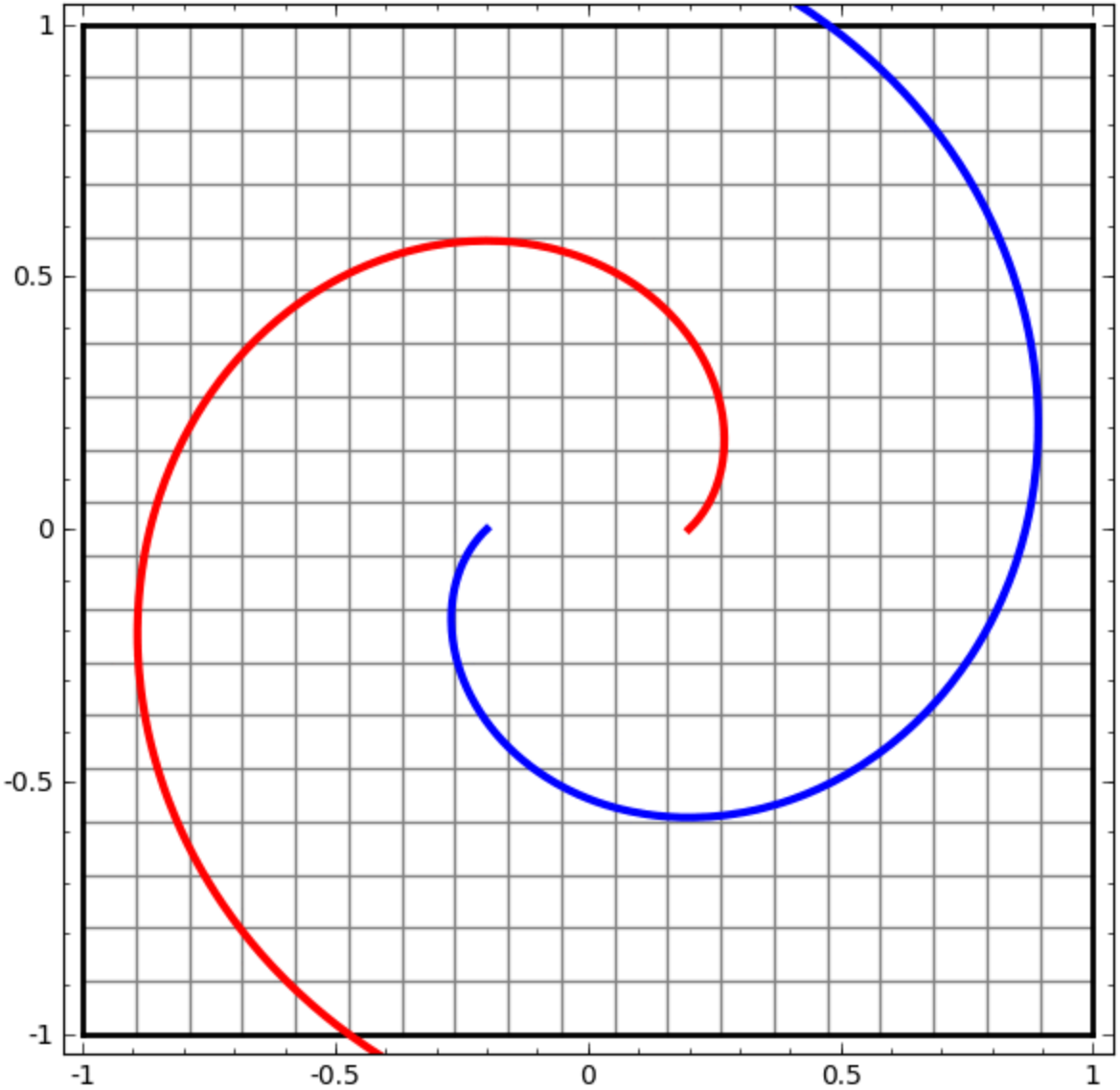


Decision function



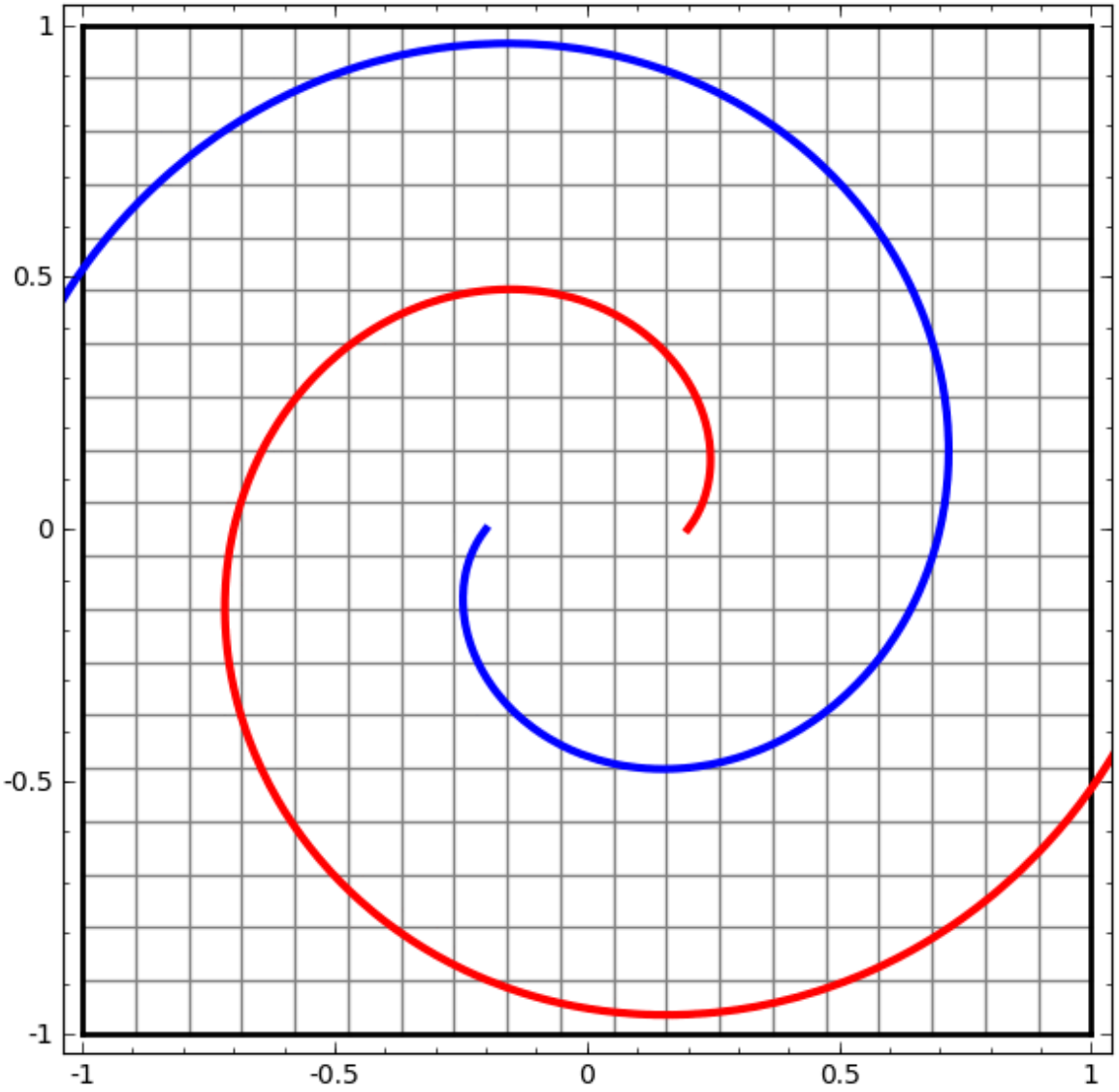
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Linearizing a 2D classification task (4 hidden layers)



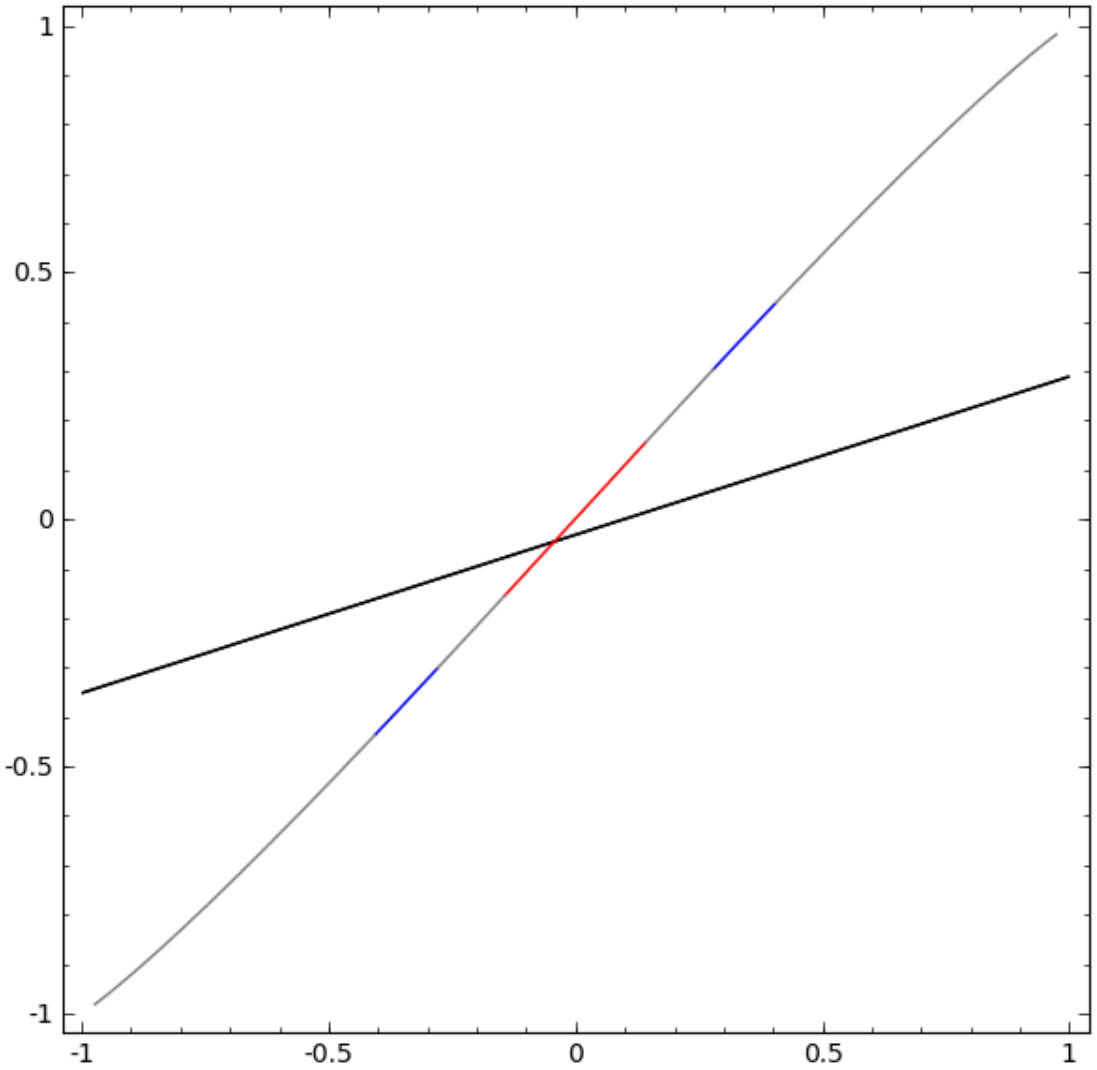
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Linearization: will not always work



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

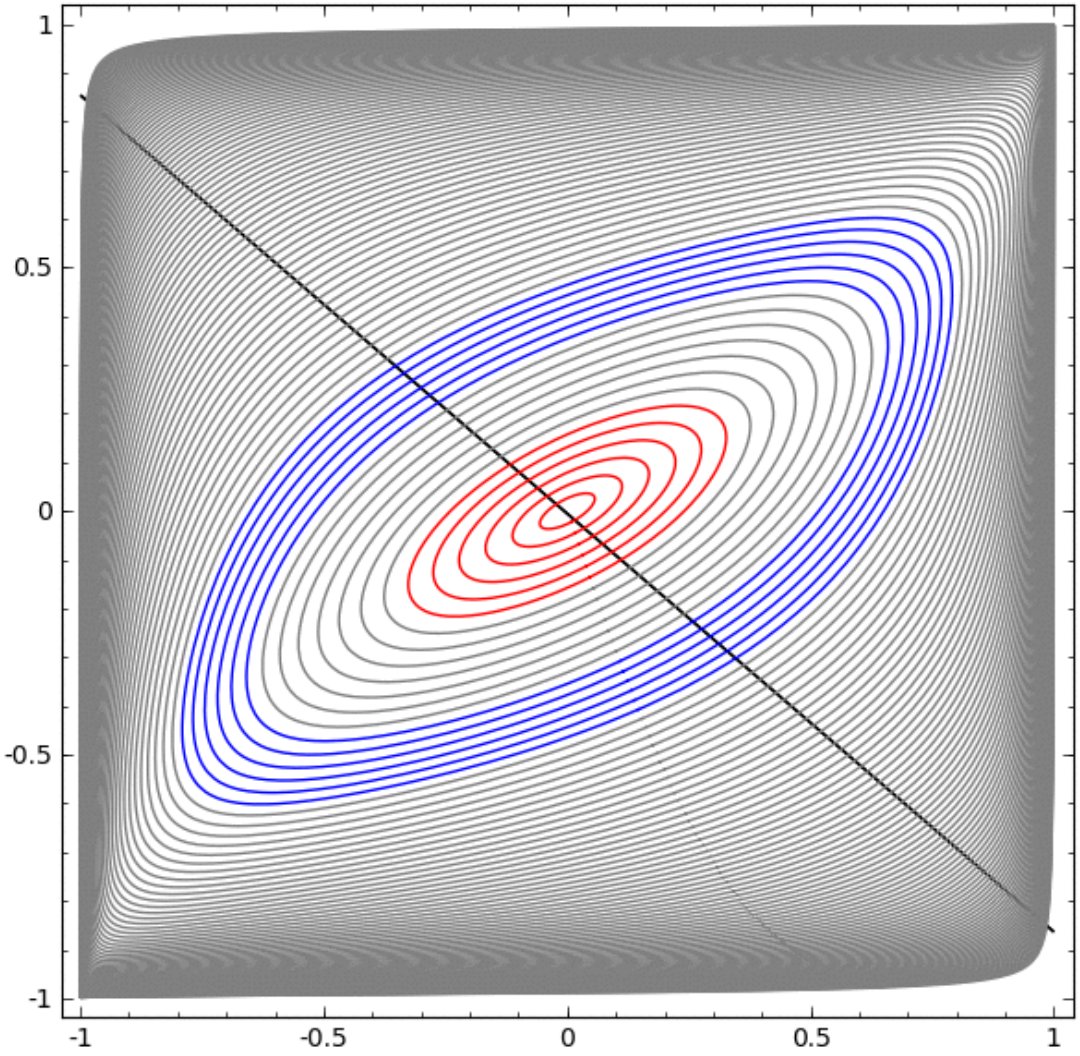
Linearization: may need higher dimensions



Points in 1D,
Decision in 2D

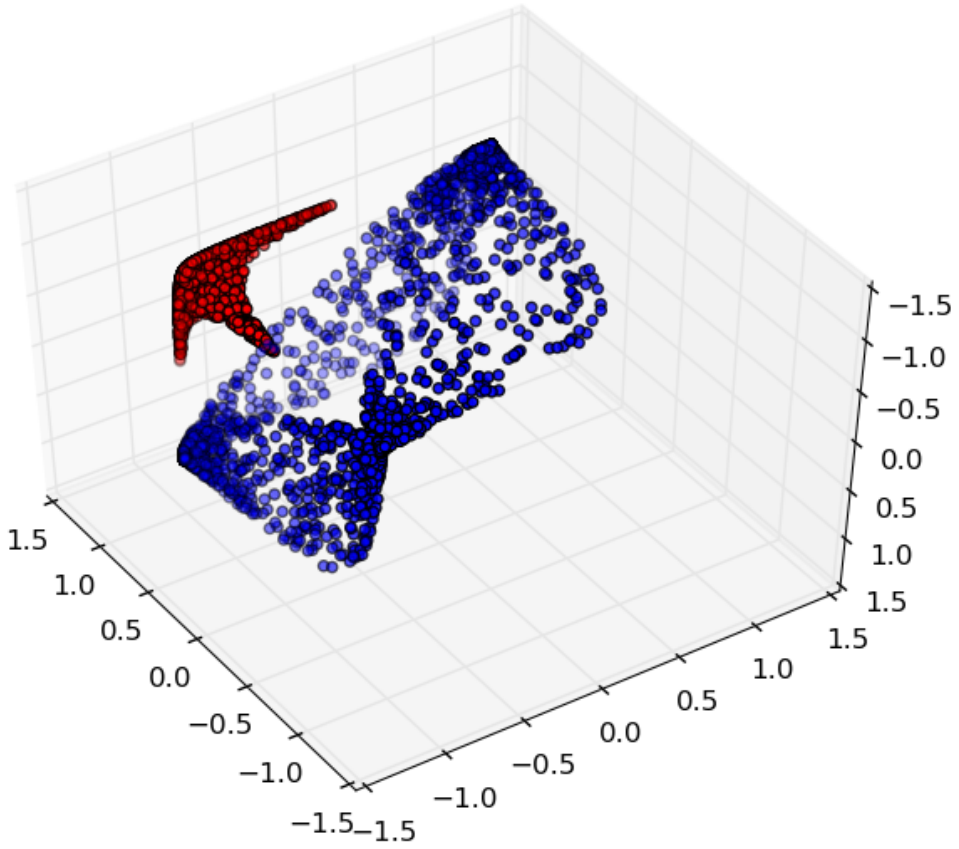
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Linearization: may need higher dimensions



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

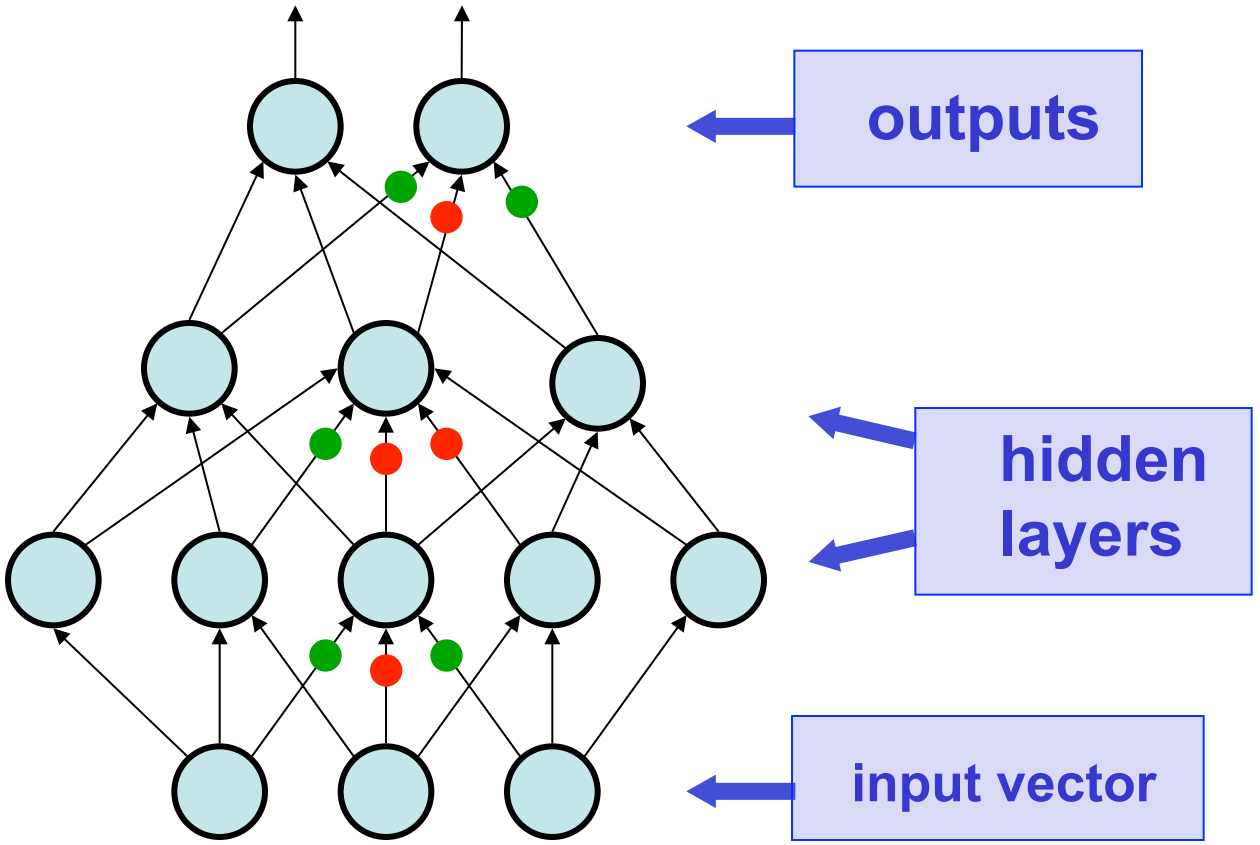
Linearization: may need higher dimensions



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

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

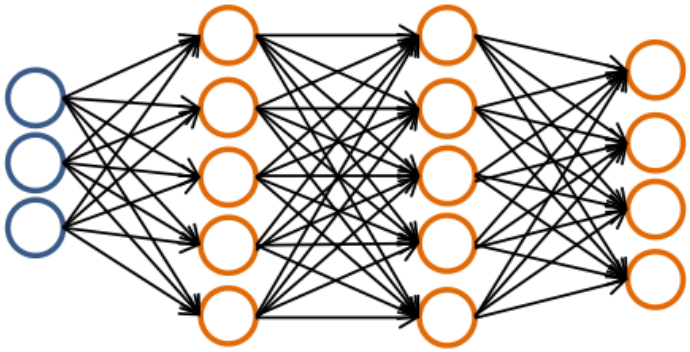
Multiple output units: One-vs-all.

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

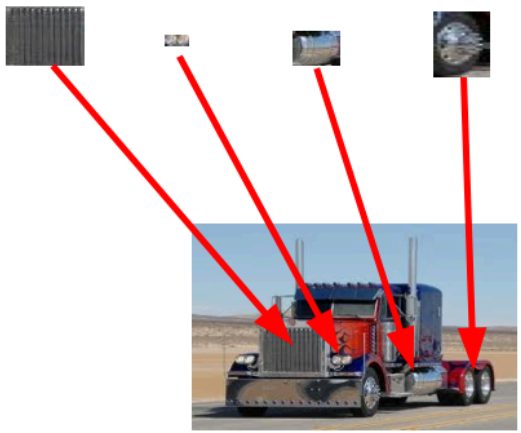
Slide credits: A. Ng

Hidden Layers: what do they do?

Intuition: learn "dictionary" for objects

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

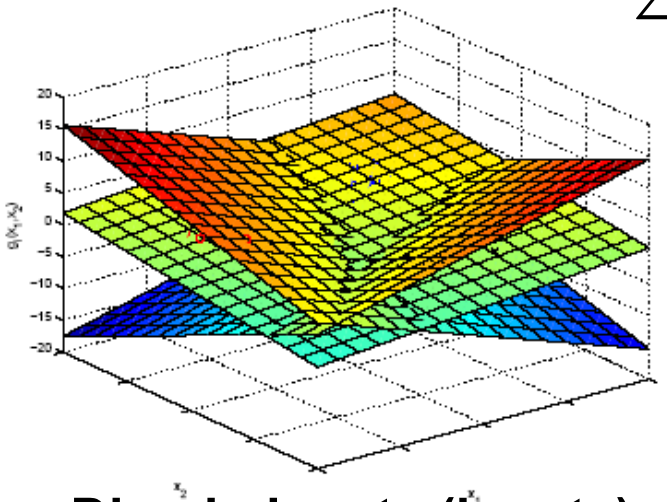
[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 …] truck feature



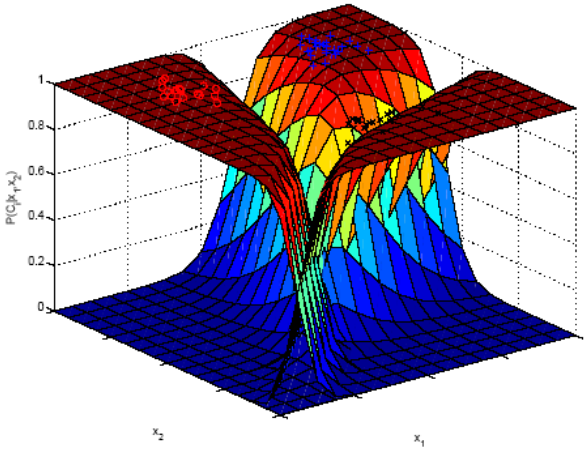
Reminder (W4): multiple classes & logistic regression

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)

Parameter estimation, multi-class case

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

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

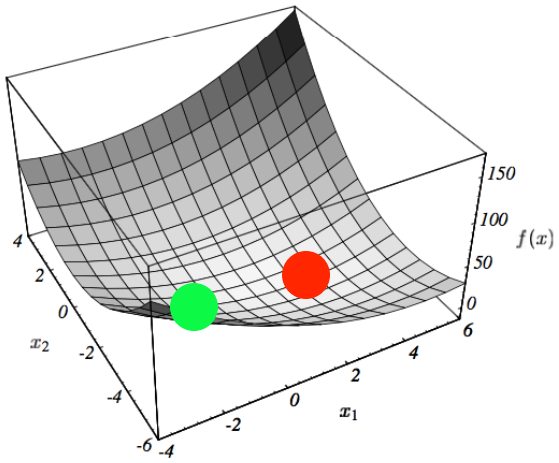
$$P(\mathbf{y}^i | \mathbf{x}^i; \mathbf{w}) = \prod_{i=1}^{N} \prod_{c=1}^{C} \left(g_c(\mathbf{x}, \mathbf{W})\right)^{\mathbf{y}_c^i}$$

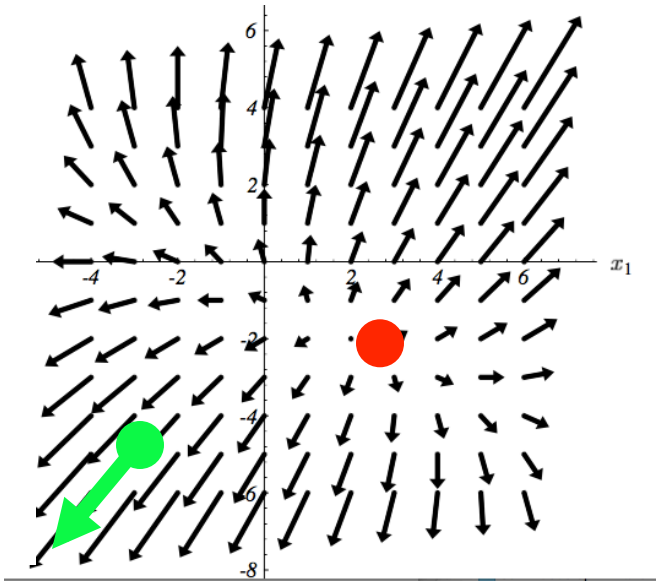
Optimization criterion:

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

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

Gradient-based minimization

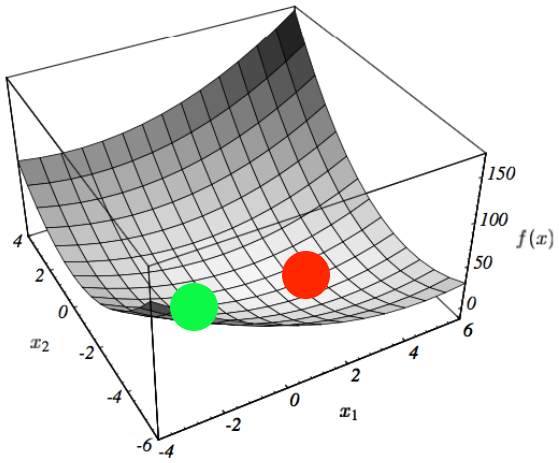


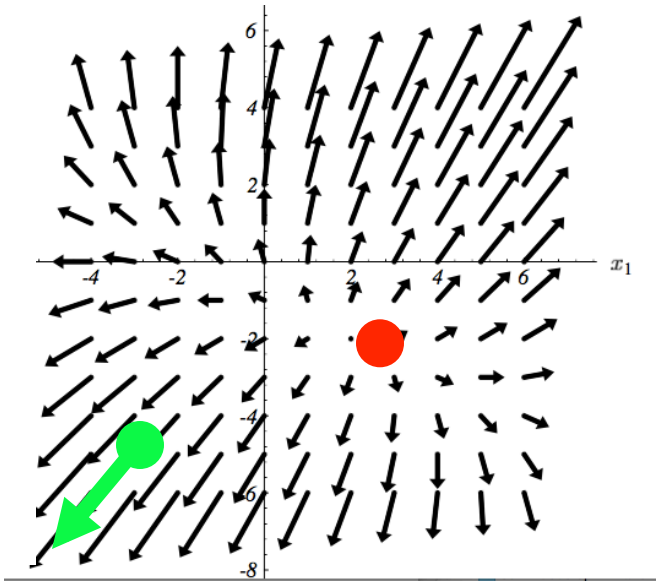


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Gradient-based minimization



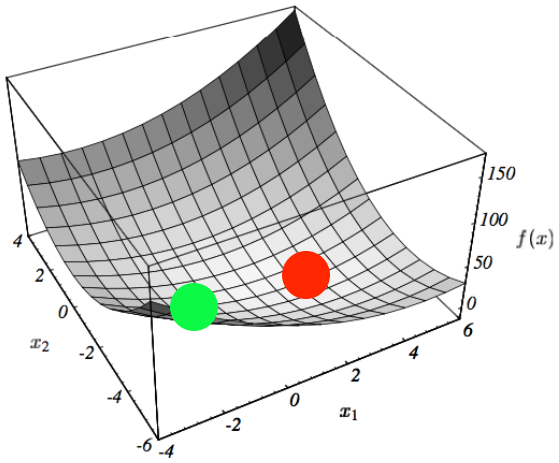


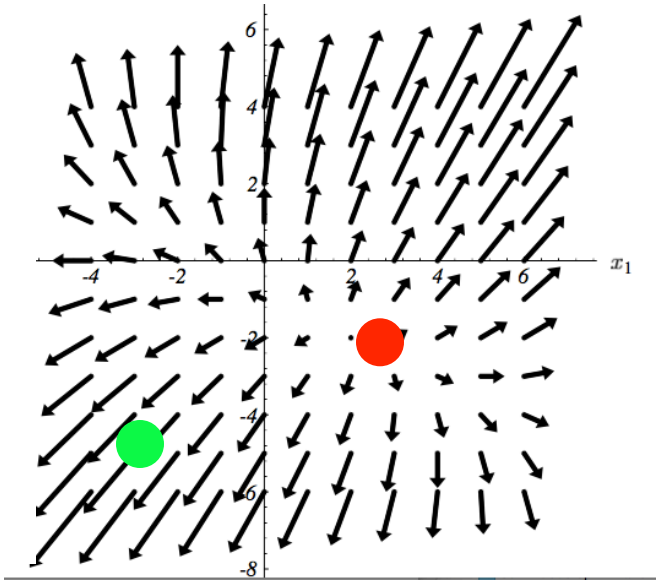
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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



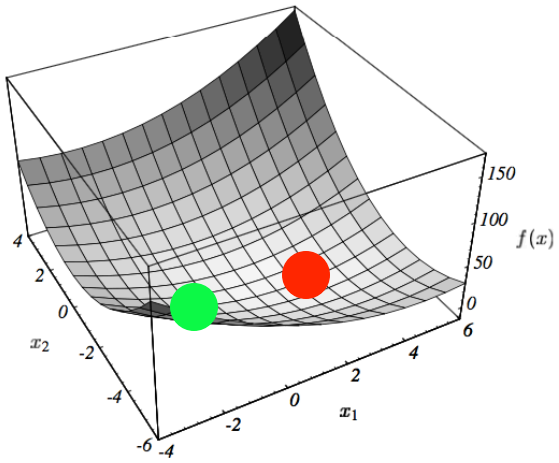


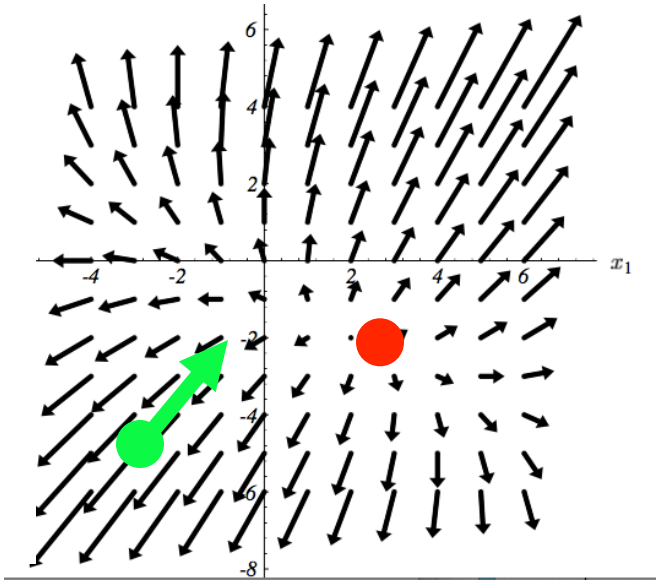
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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



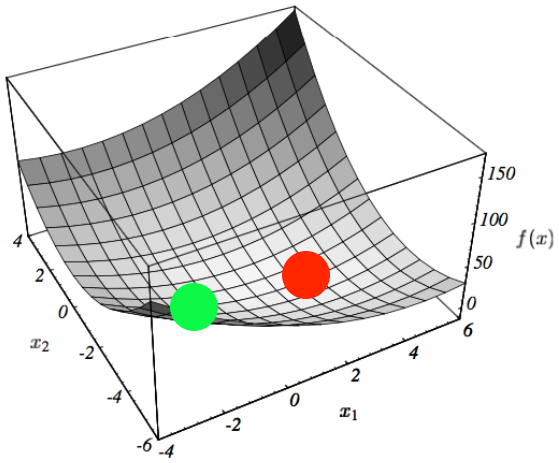


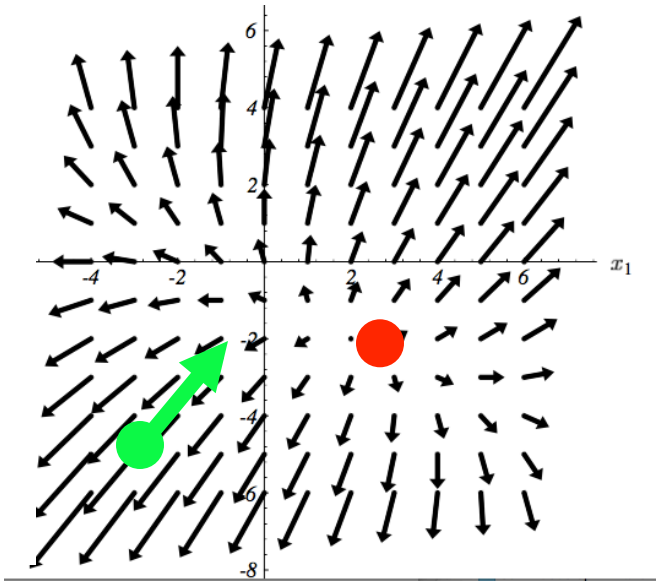
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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





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

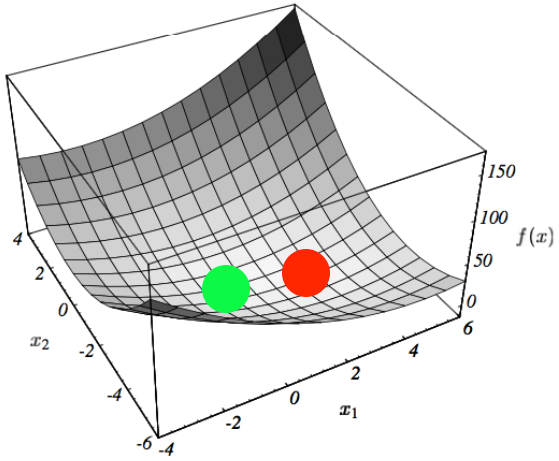
Fact: gradient at any point gives direction of fastest increase

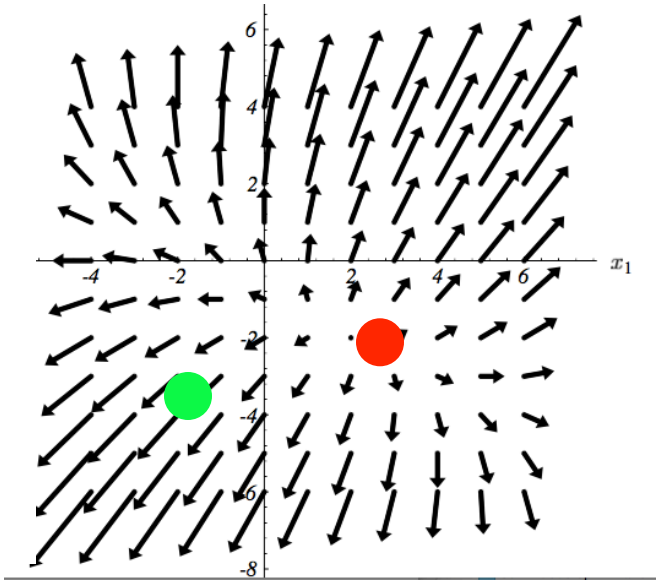
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

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

Gradient-based minimization

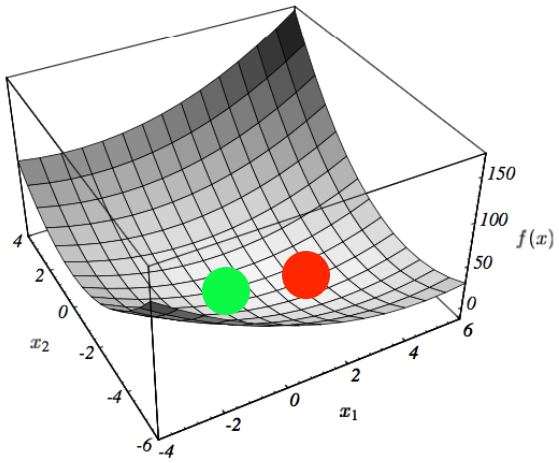


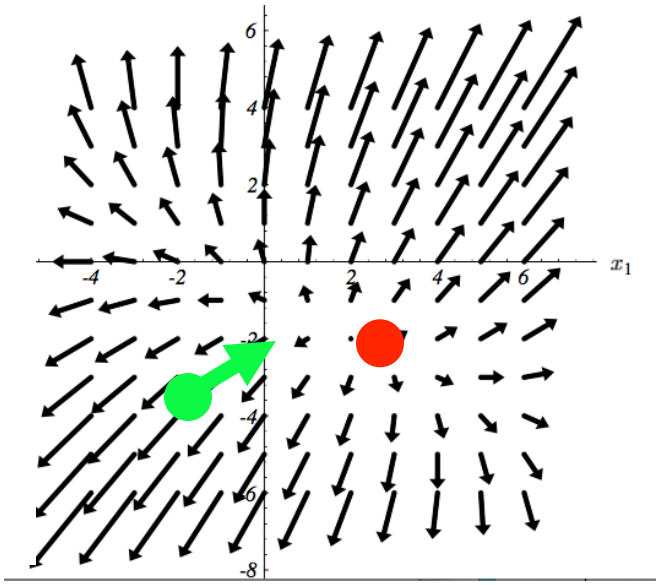


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient-based minimization

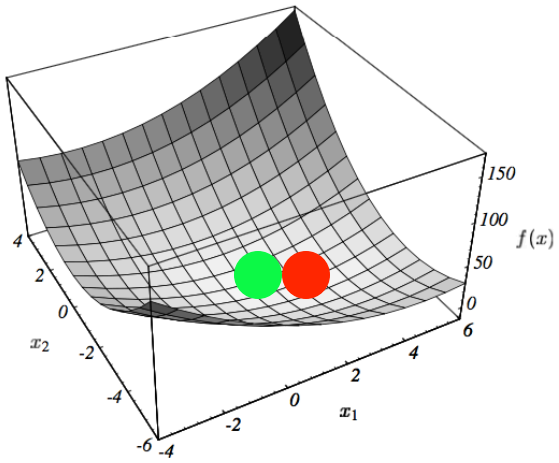


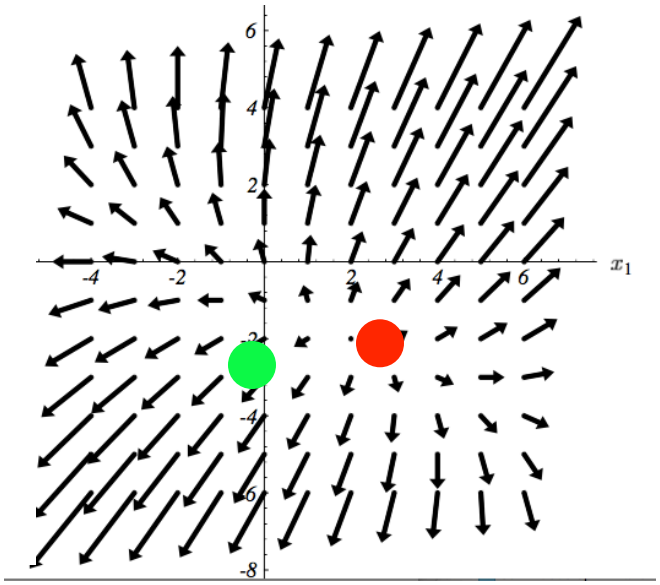


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient-based minimization

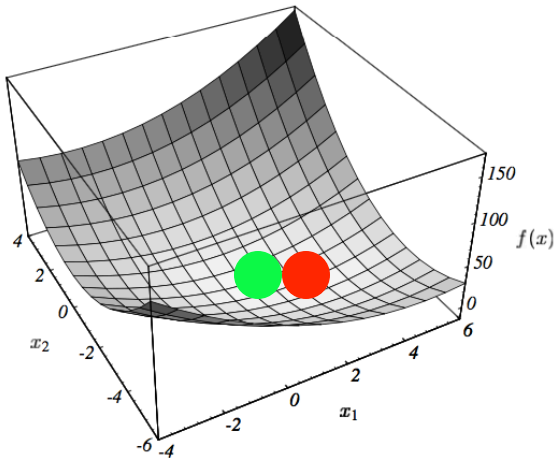


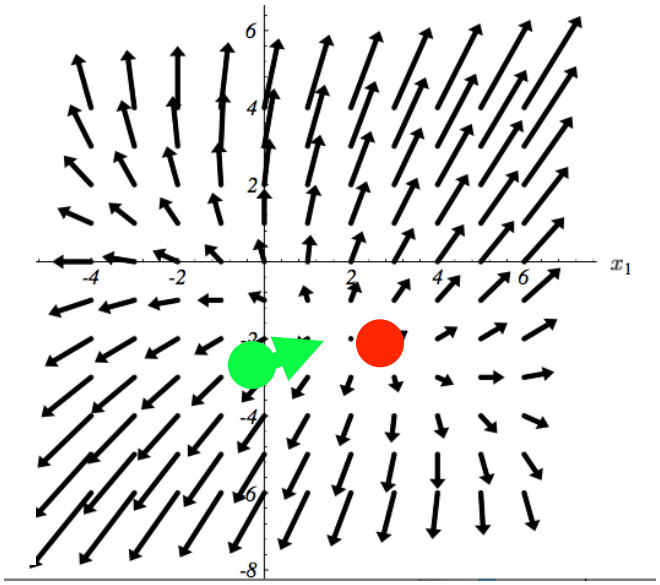


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient-based minimization

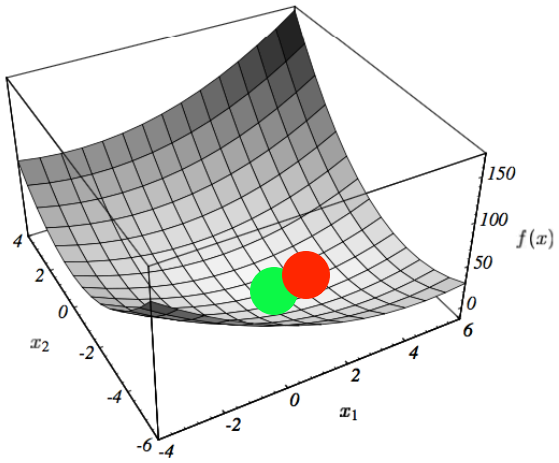


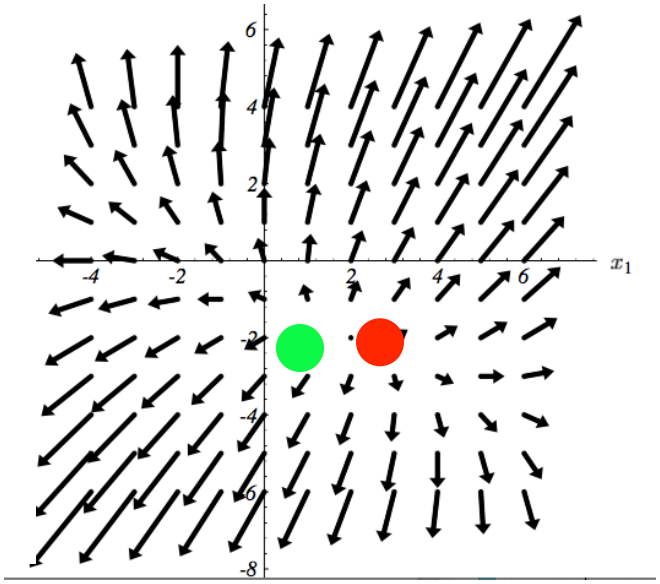


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient-based minimization

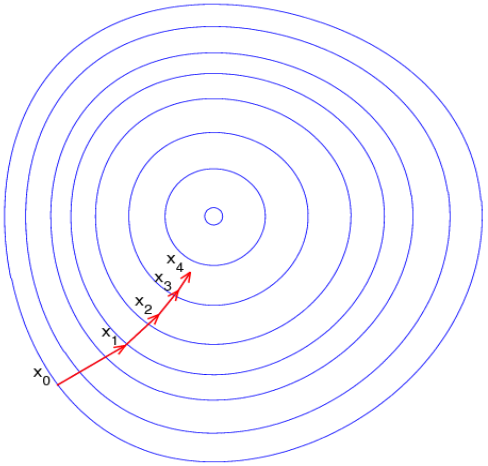




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

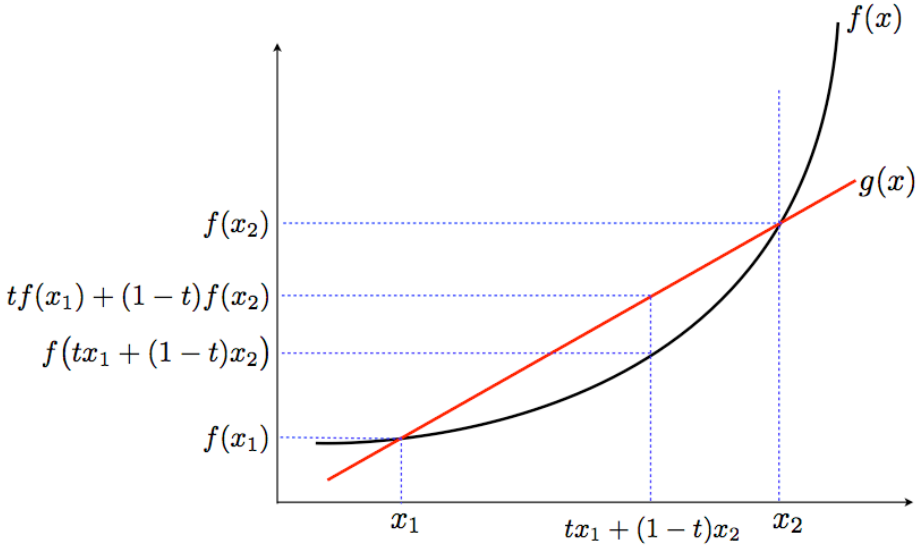
Gradient descent minimization method

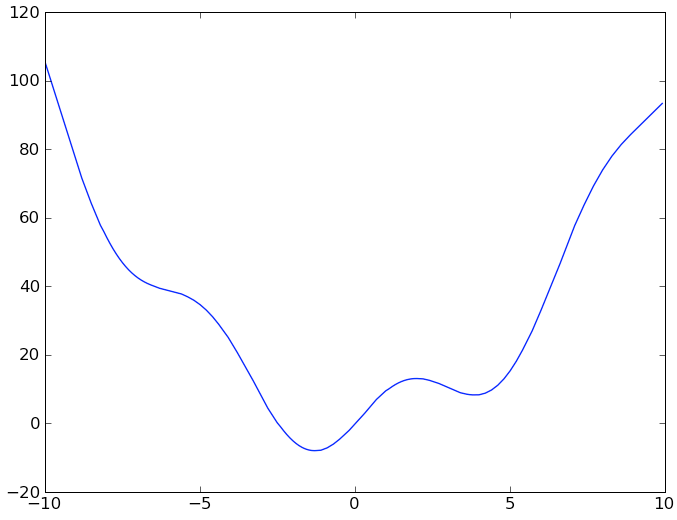


Initialize: $\mathbf{x}_0$

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

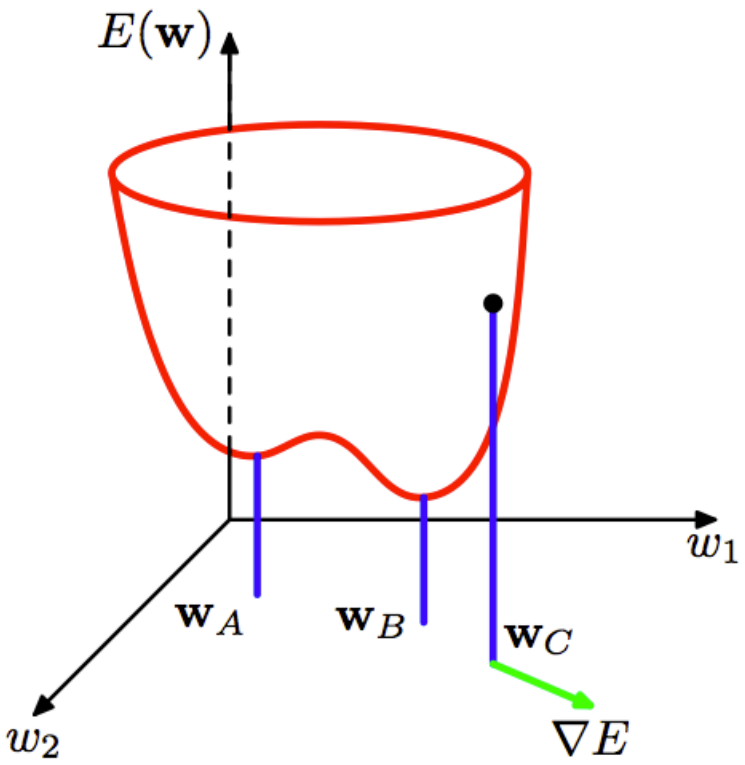
We can always make it converge for a **convex** function



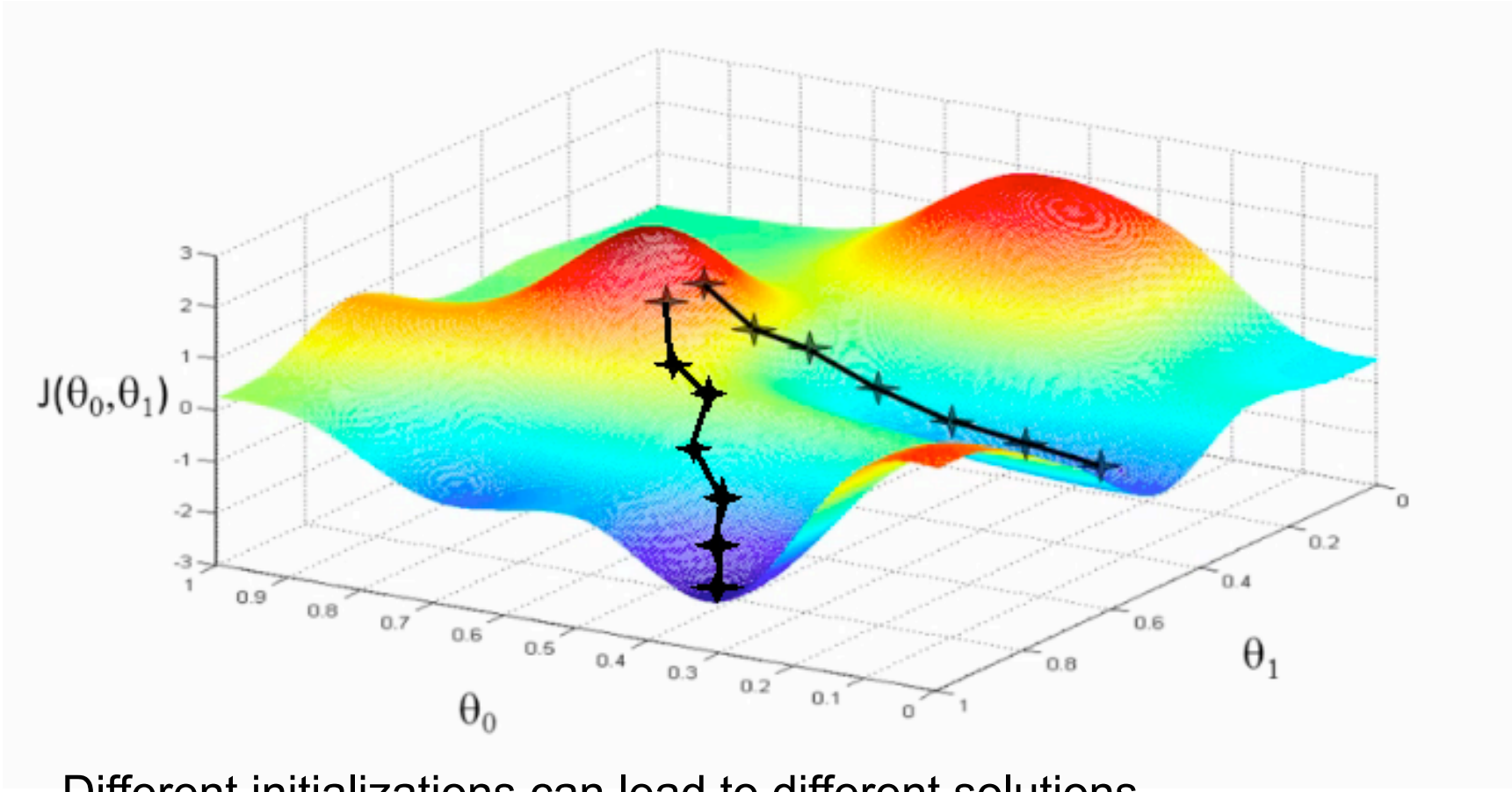


Problems: multiple local minima

Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.



Problems: multiple local minima



Different initializations can lead to different solutions

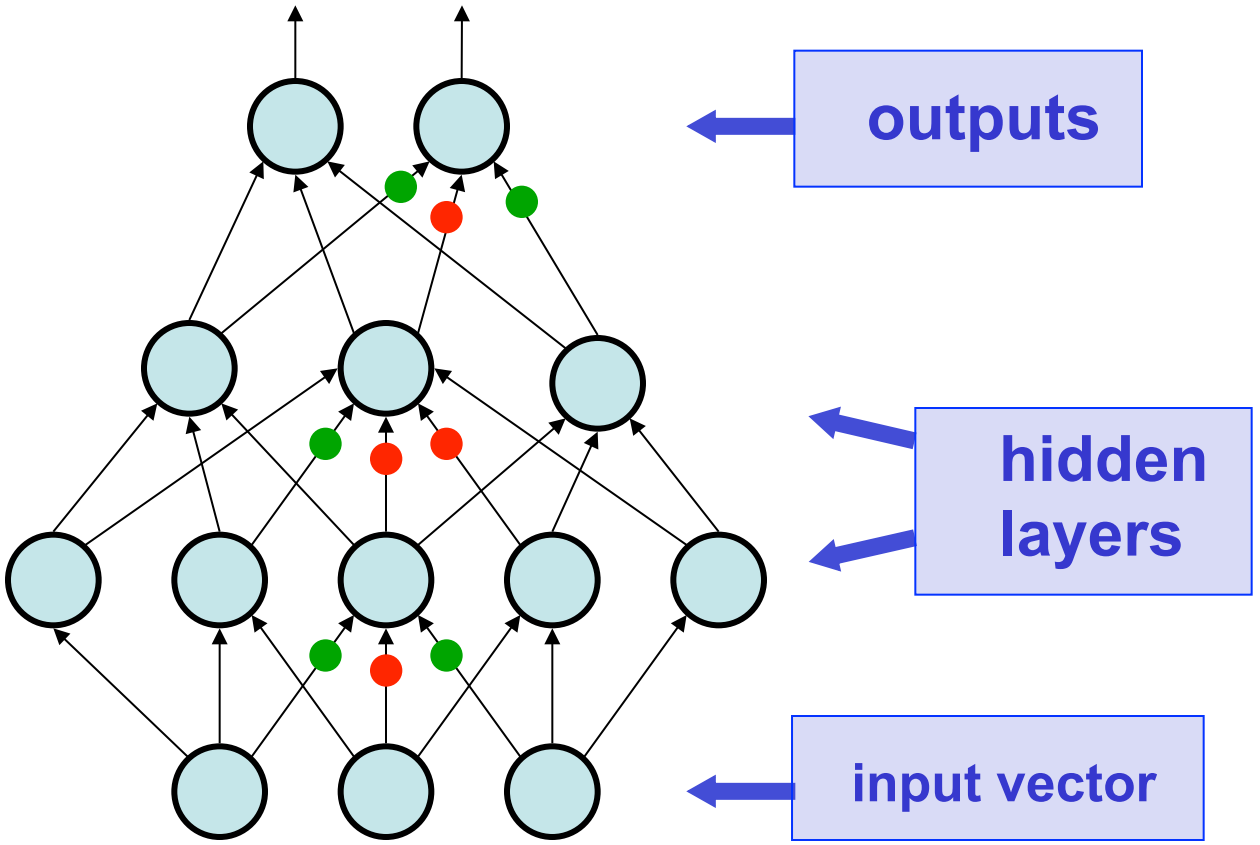
-Empirically all are almost equally good

-Empirically all are better than flat counterparts

On to the gradients!

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

Training a multi-class classifier

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

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

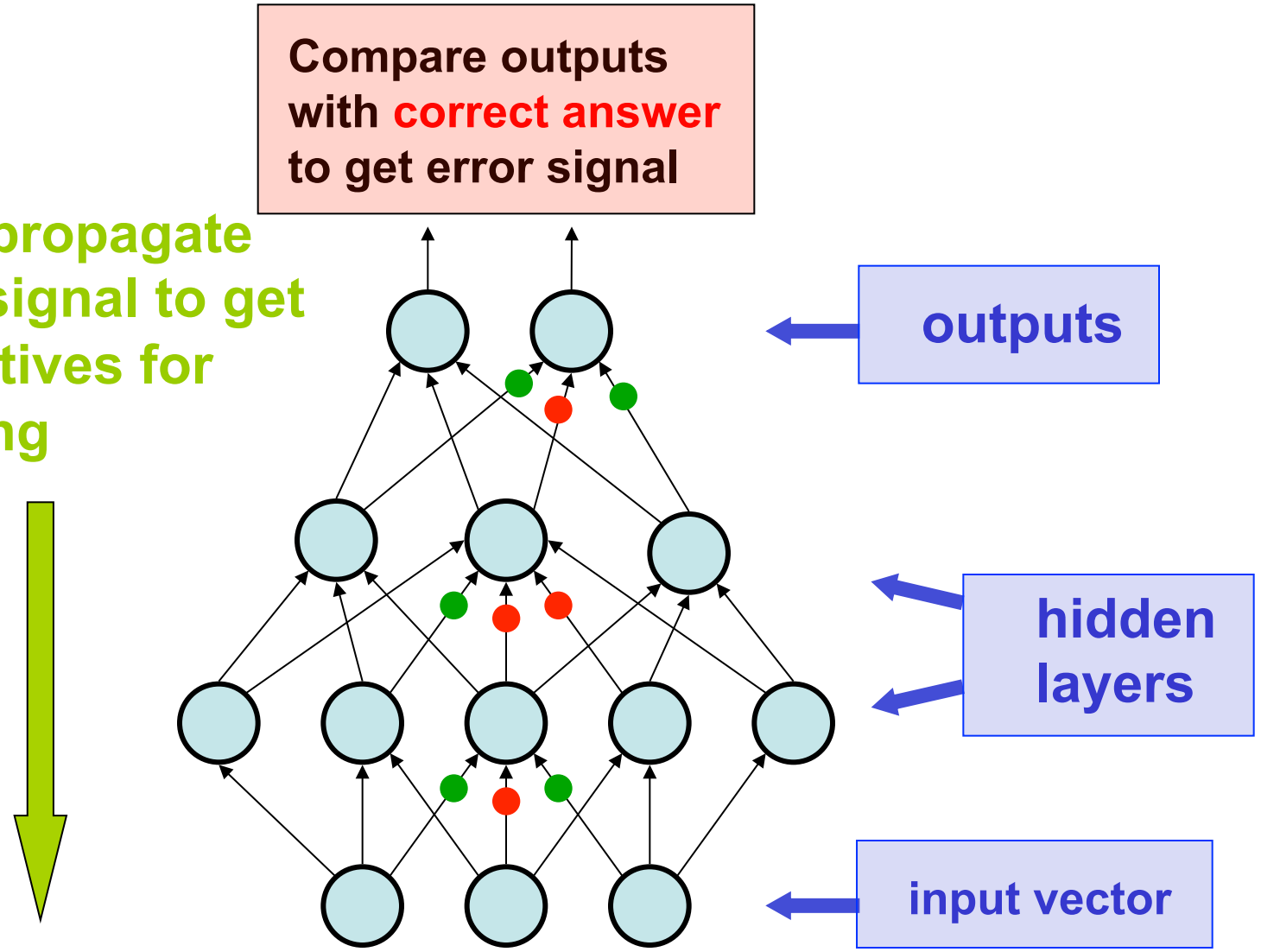
neural network

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

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

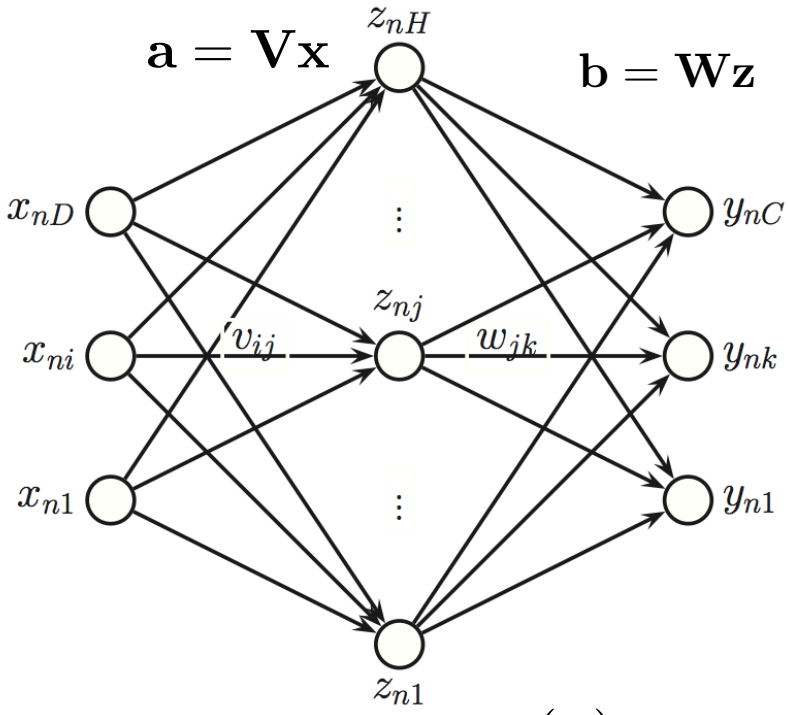
Back-propagation algorithm

Multi-Layer Perceptrons (~1985)



Slide credits: G. Hinton

Training a neural network



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

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

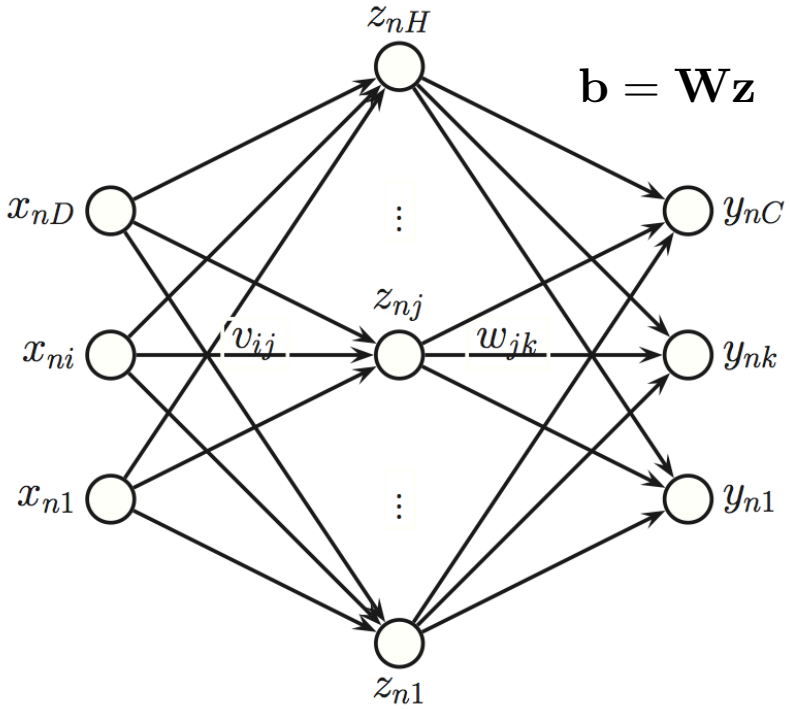
Training objective as a function of the parameters:

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

Gradient Descent

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

Training a neural network



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

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

$$\hat{\mathbf{y}} \leftrightarrow \mathbf{y}$$

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

What is the gradient of the loss?

Chain rule

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

y is affected by x through intermediate quantity, u:

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

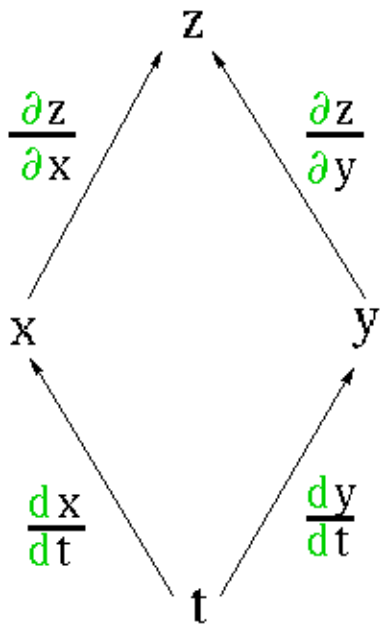
Rewrite:

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

Chain rule of derivative – 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

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

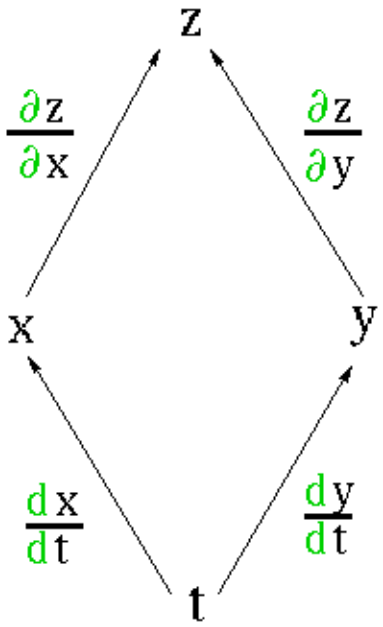


https://www.math.hmc.edu/calculus/tutorials/multichainrule/

Chain rule of derivative – 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

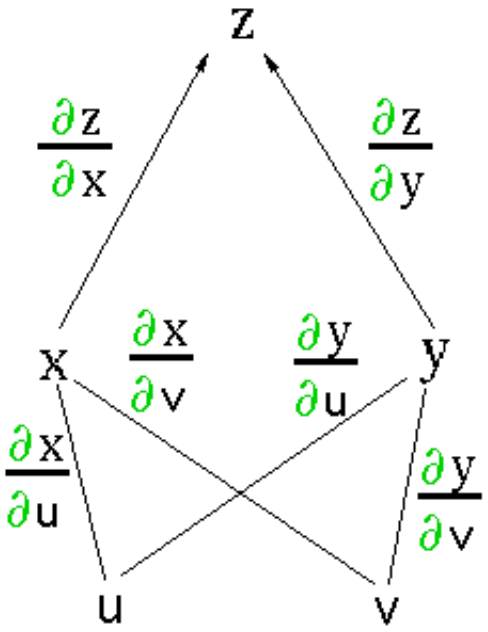
$$\begin{aligned}
\frac{dz}{dt} &= \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \\
&= (2xy)(2t) + (x^2 - 2y)(2) \\
&= (2t^2 \cdot 2t)(2t) + \left((t^2)^2 - 2(2t)\right)(2) \\
&= 8t^4 + 2t^4 - 8t \\
&= 10t^4 - 8t.
\end{aligned}$$

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/

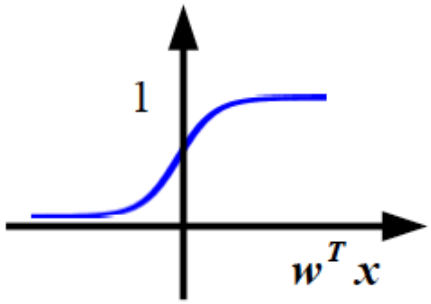
Linear Classifier: Logistic Regression

Input: $\boldsymbol{x} \in R^D$

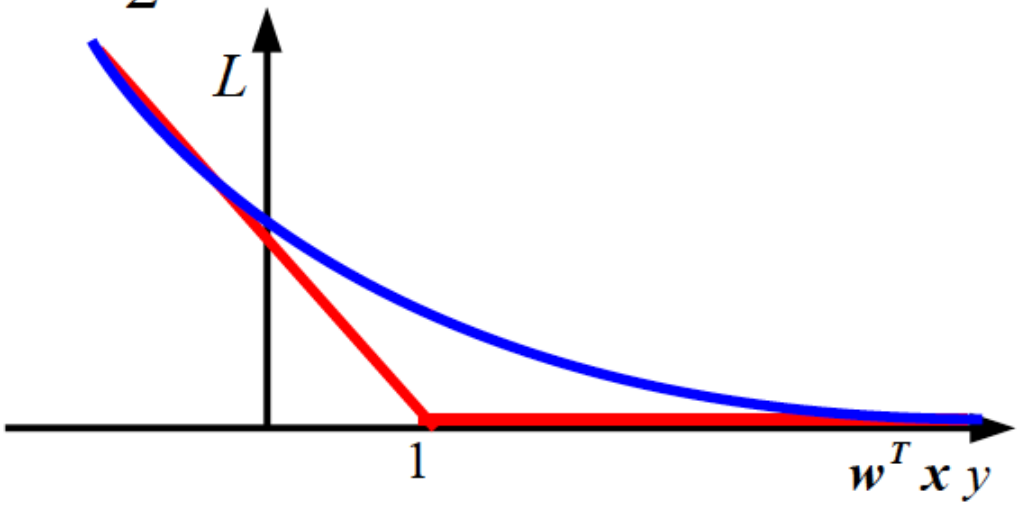
Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $p(y=1|\boldsymbol{x}) = \dfrac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}}}$



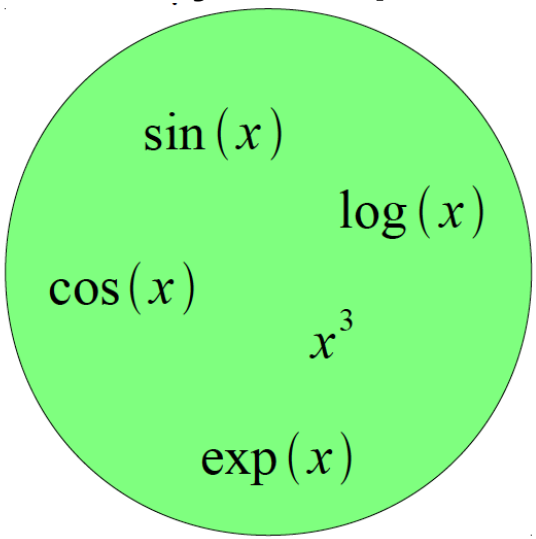
Loss: $L = \dfrac{1}{2}\|\boldsymbol{w}\|^2 - \lambda \log(p(y|\boldsymbol{x}))$



Log Loss

Logistic Regression as a Cascade

Given a library of simple functions

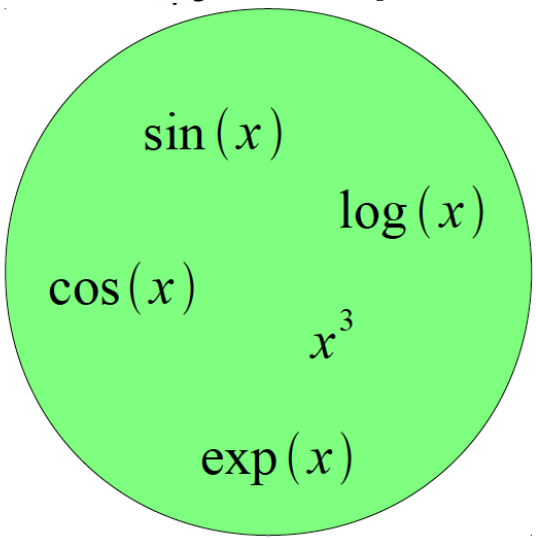


Compose into a complicated function

$$-\log\left(\frac{1}{1+e^{-\mathbf{w}^\intercal \mathbf{x}}}\right)$$

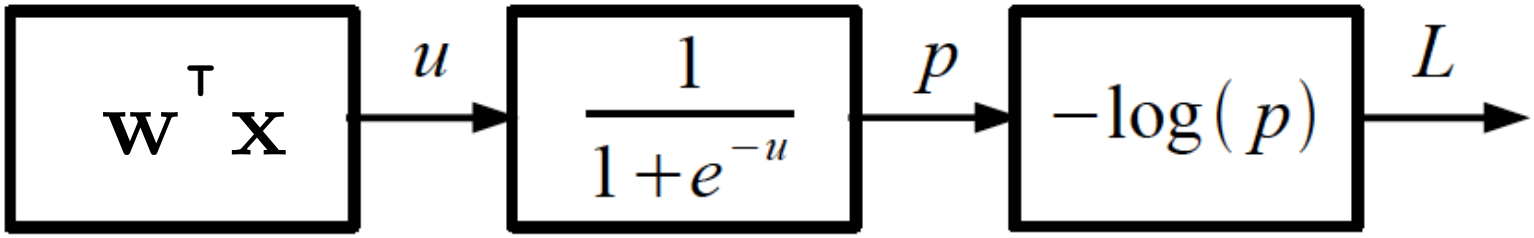
Logistic Regression as a Cascade

Given a library of simple functions



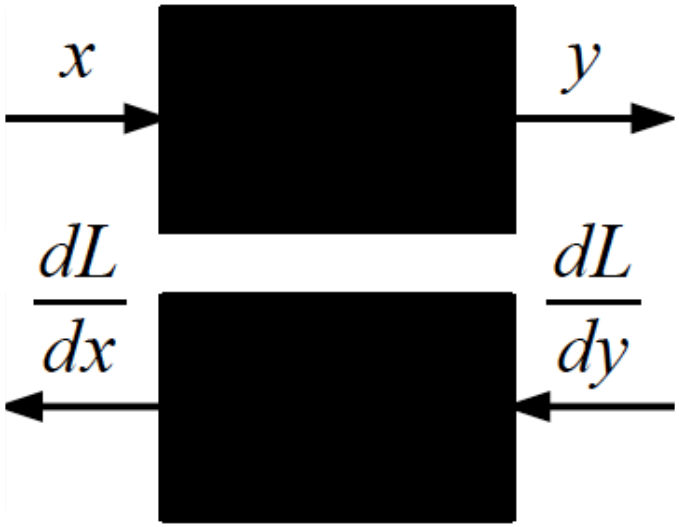
Compose into a complicate function

$$-\log\left(\frac{1}{1+e^{-\mathbf{w}^\intercal \mathbf{x}}}\right)$$



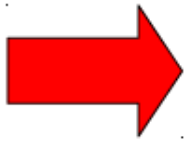
Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Chain Rule

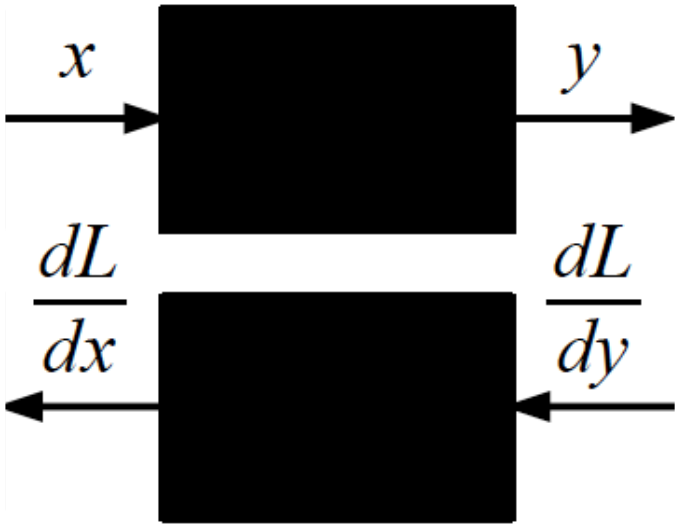


Given $y(x)$ and dL/dy,

What is dL/dx ?



Chain Rule



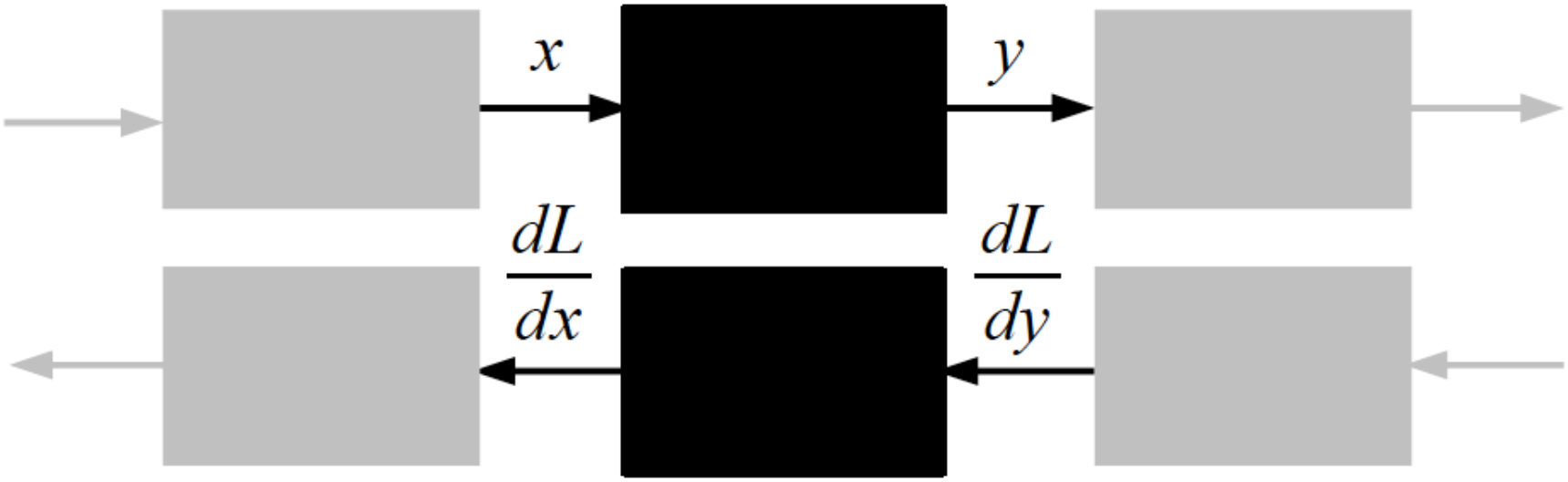
Given $y(x)$ and dL/dy,

What is dL/dx ?

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

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Chain Rule: All local



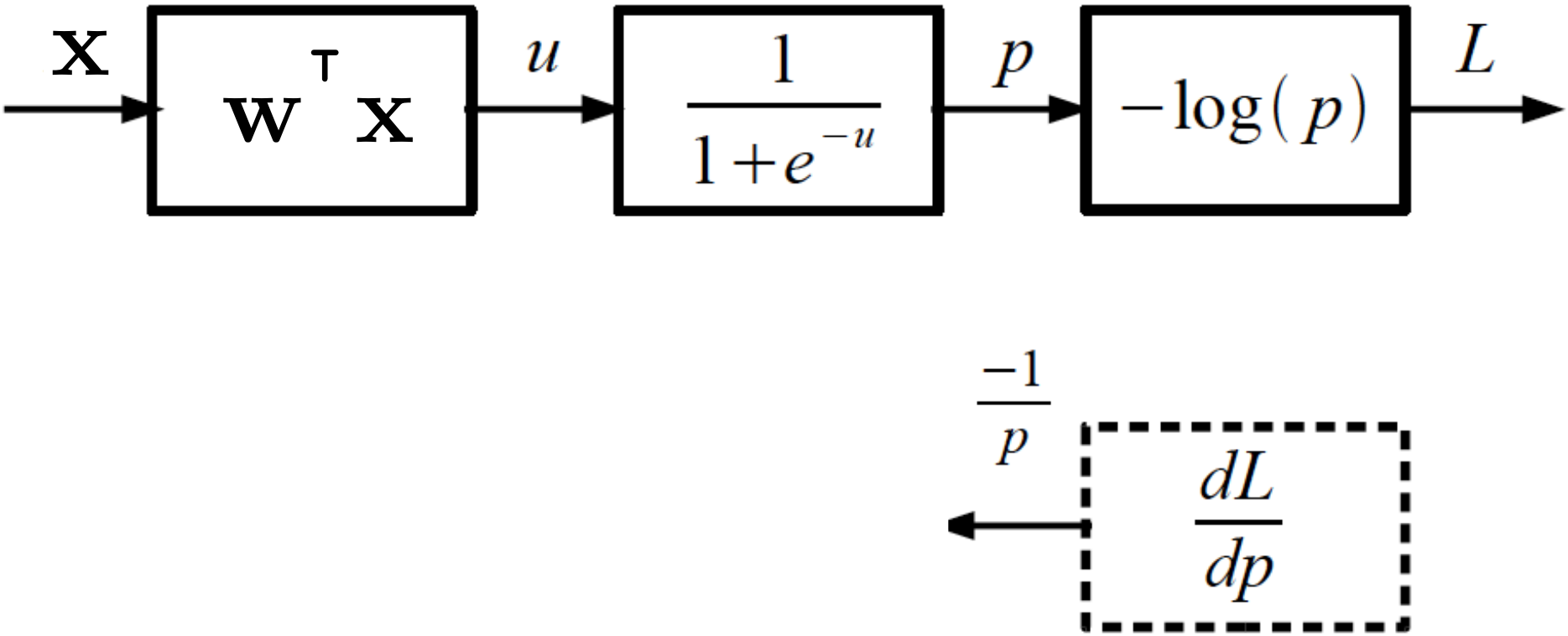
Given $y(x)$ and dL/dy,

What is dL/dx ?

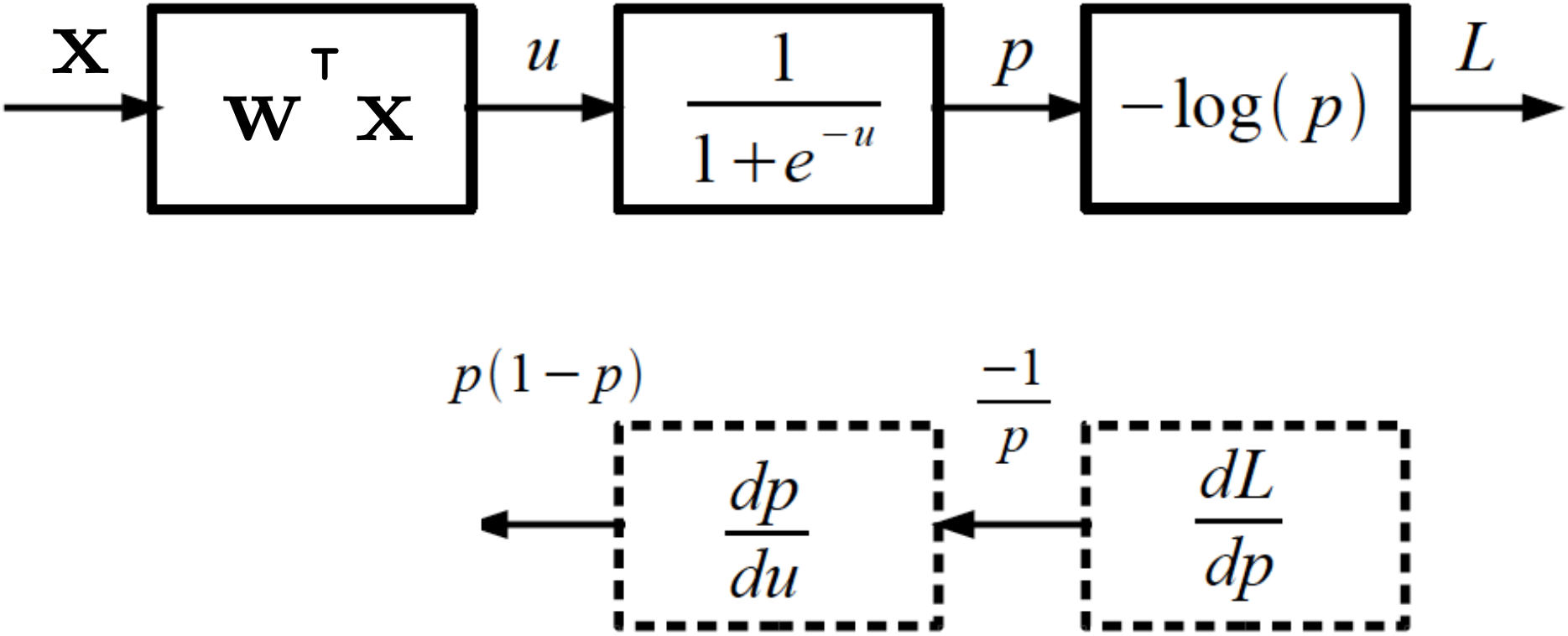
$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Logistic Regression as a Cascade

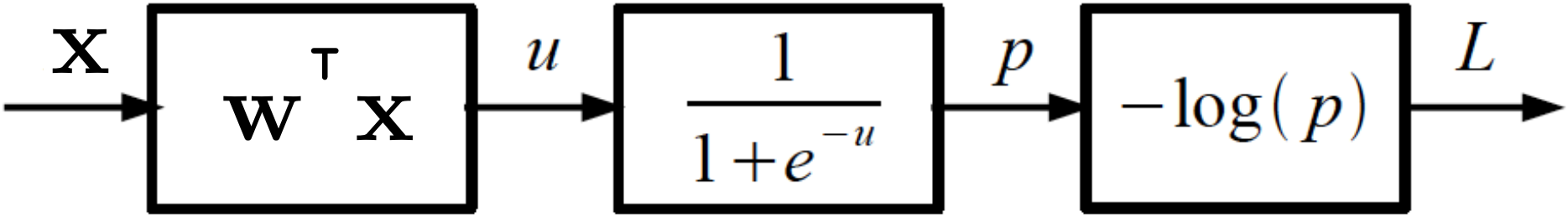


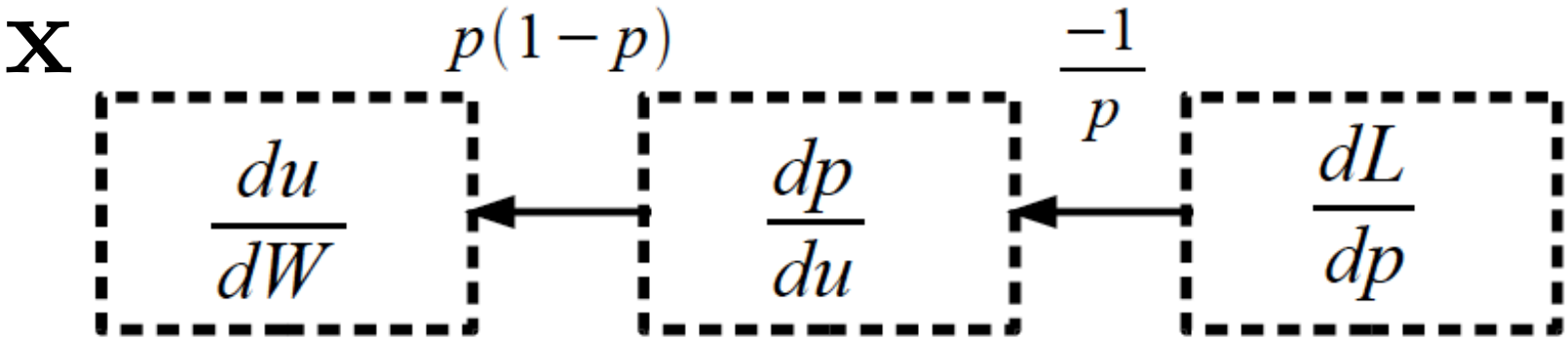
Logistic Regression as a Cascade



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Logistic Regression as a Cascade

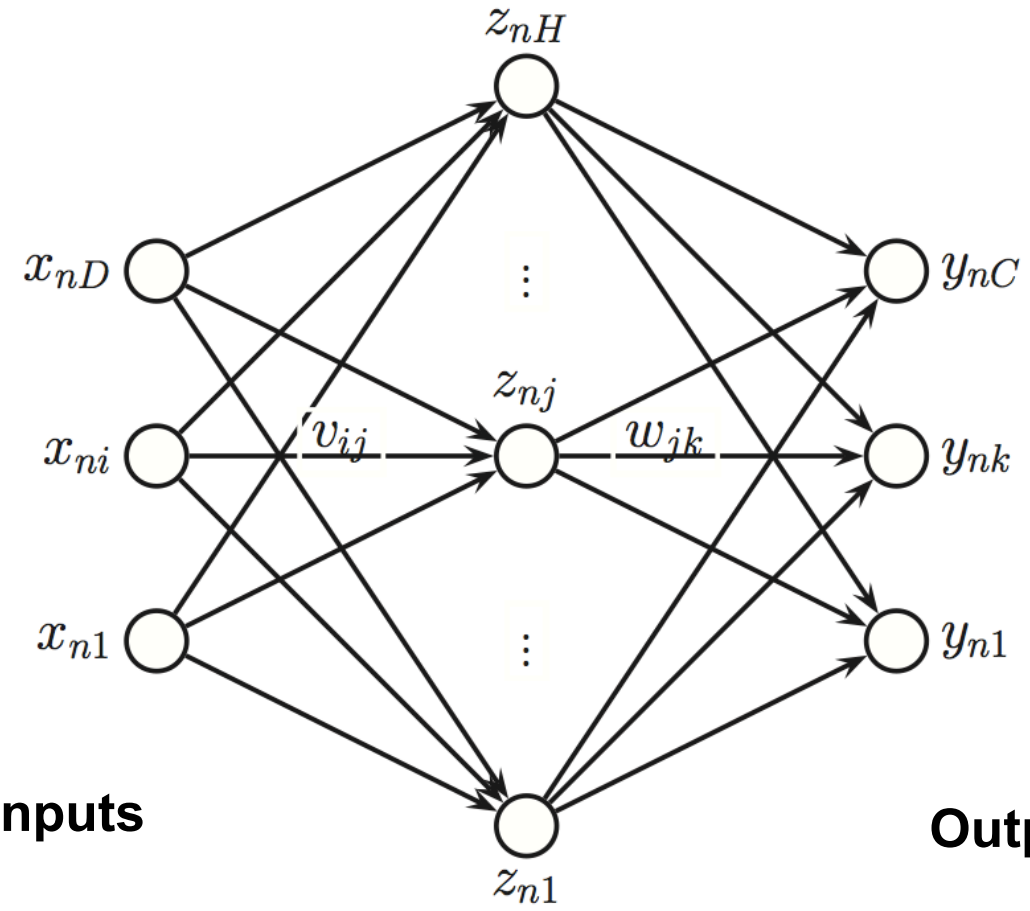




$$\frac{dL}{dW} = \frac{dL}{dp} \cdot \frac{dp}{du} \cdot \frac{du}{dW} = (p-1)\mathbf{x}$$

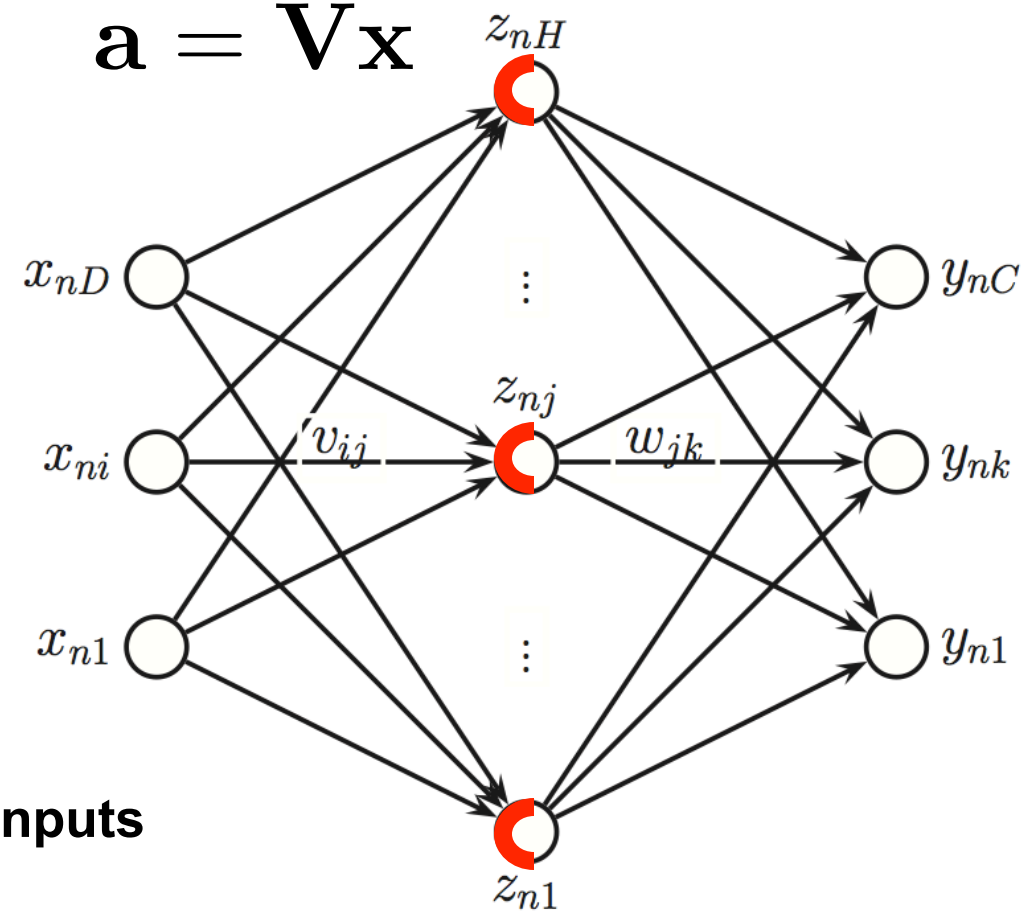
A neural network for multi-way classification

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



A neural network in forward mode:

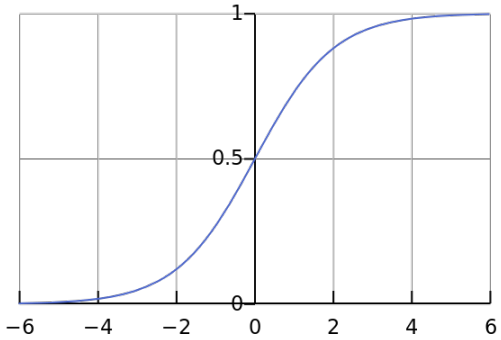
$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

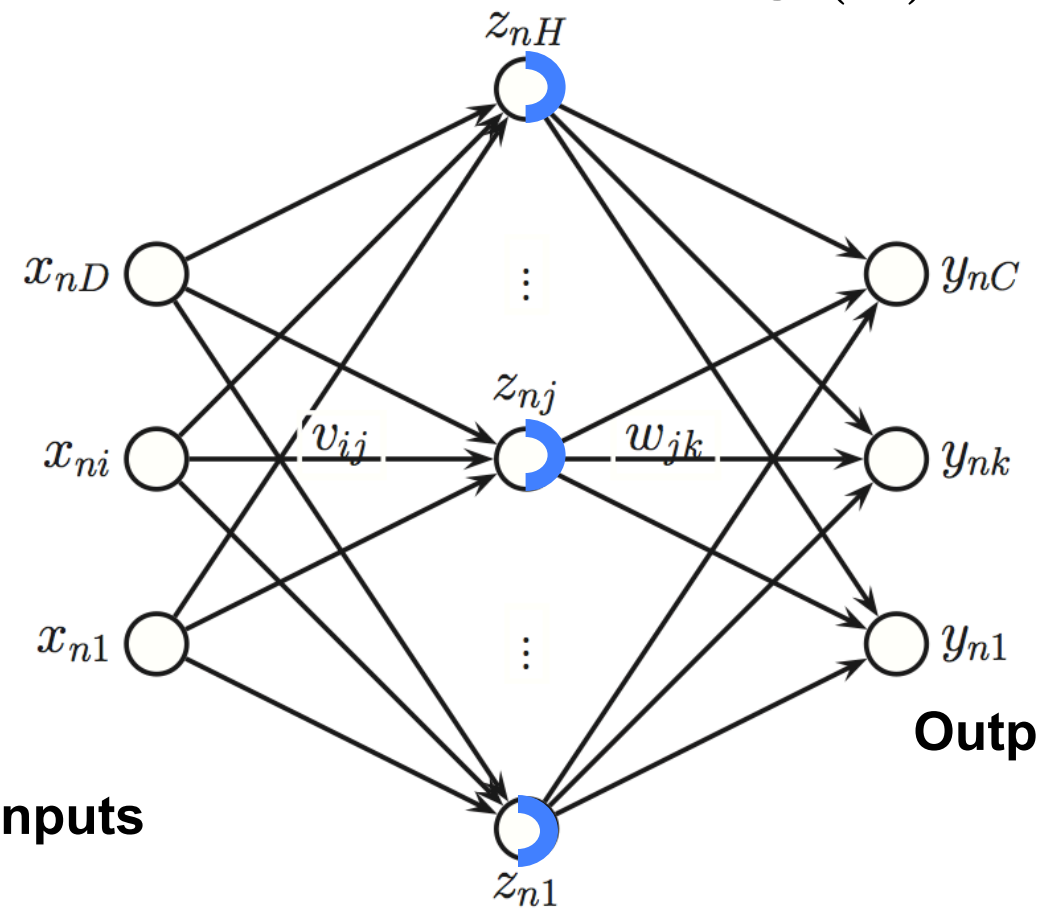


A neural network in forward mode:

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

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



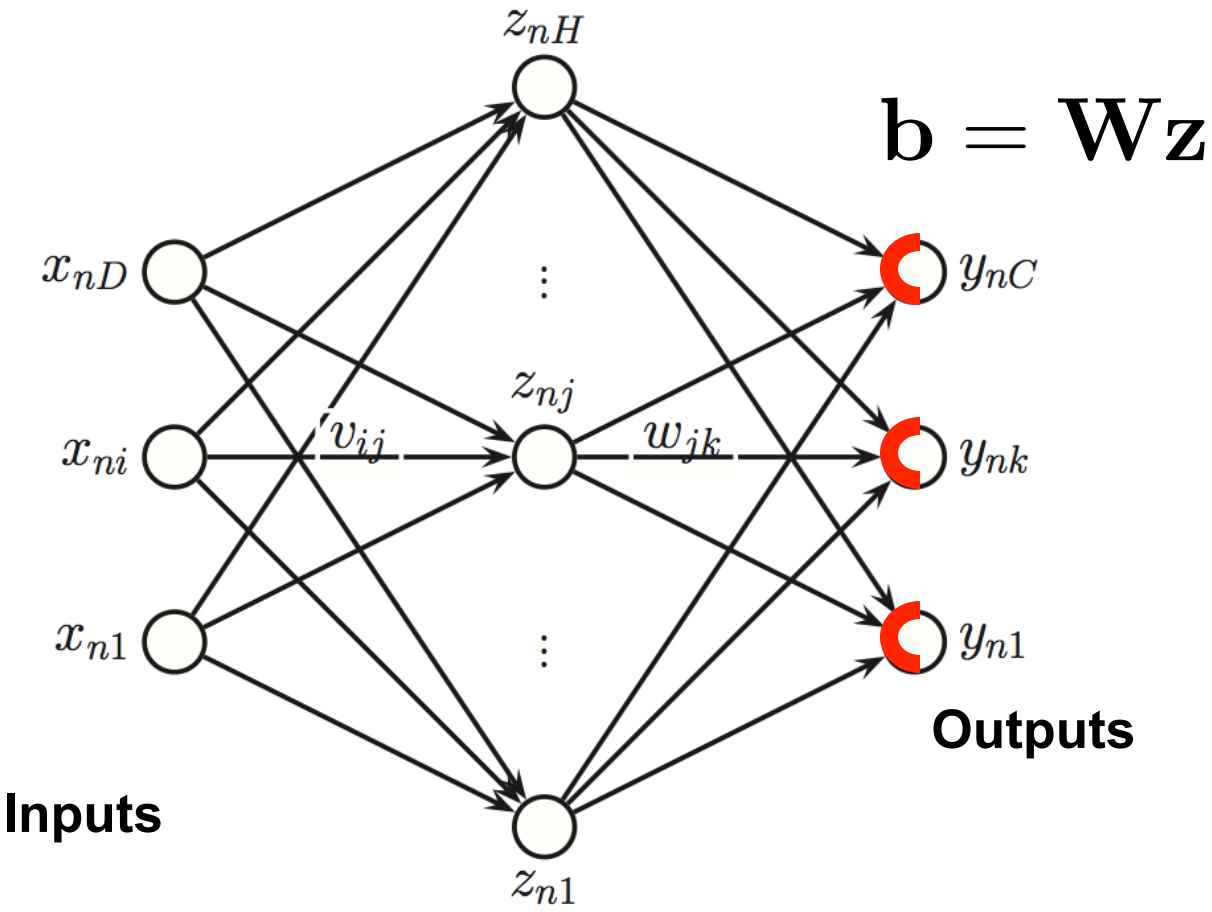


Outputs

Inputs

Hidden layer

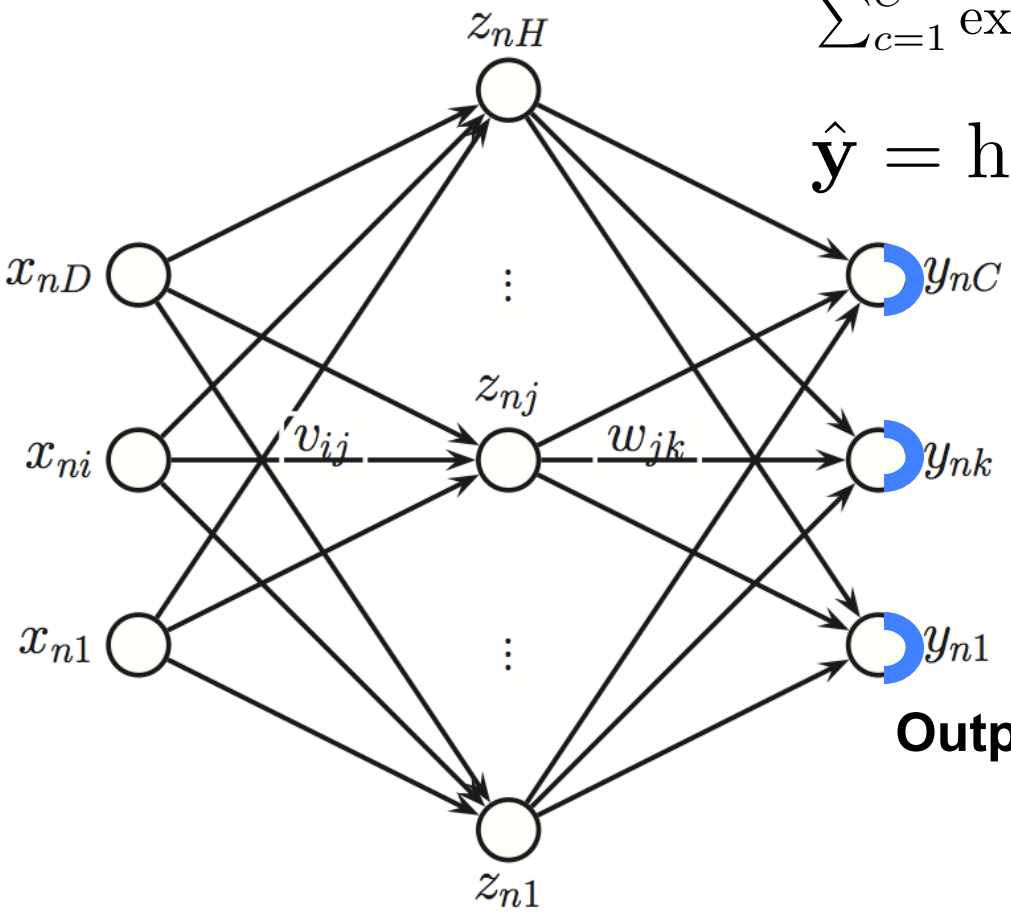
A neural network in forward mode:



A neural network in forward mode:

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

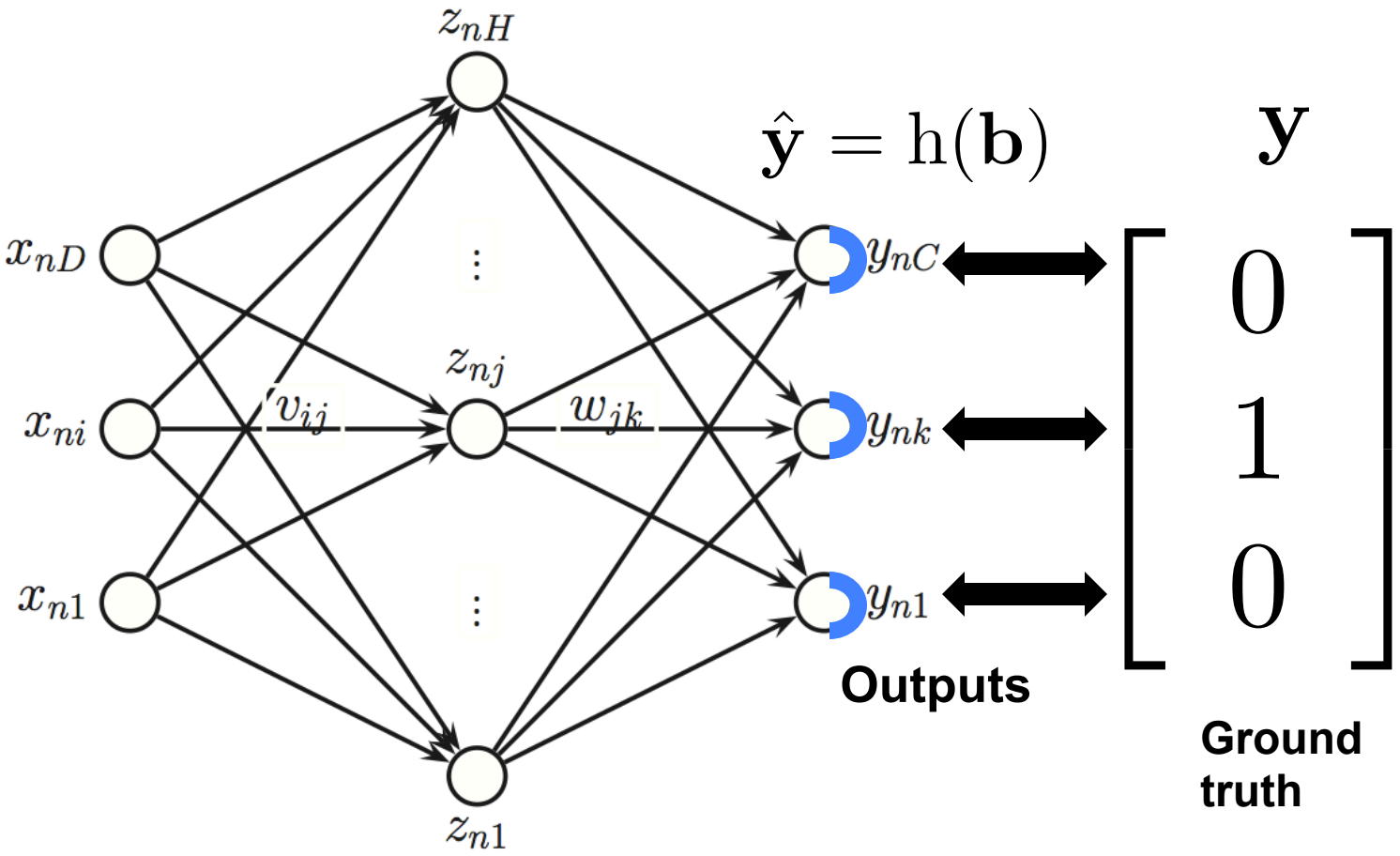
$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$



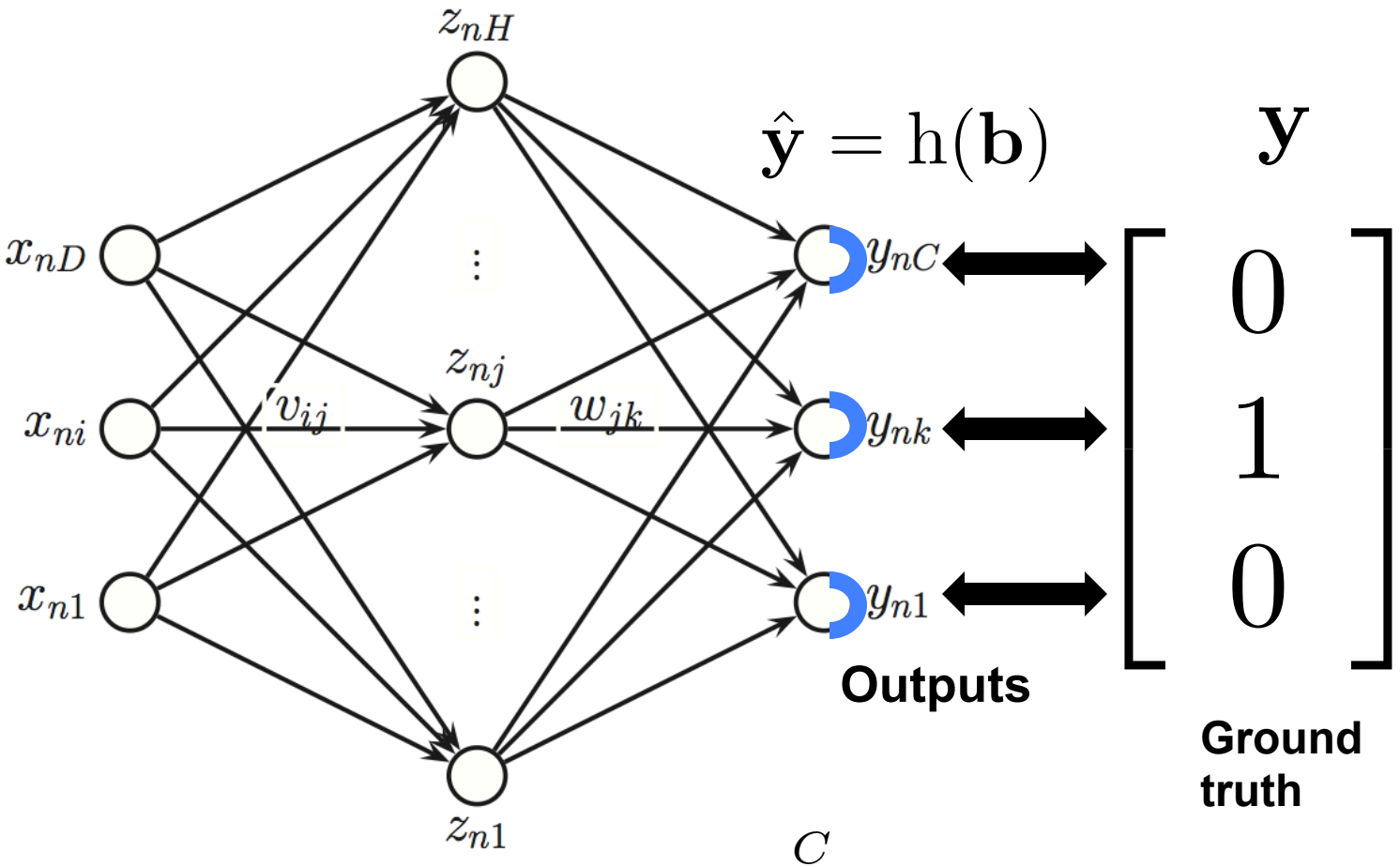
Outputs

Hidden layer

Objective for multi-class classification

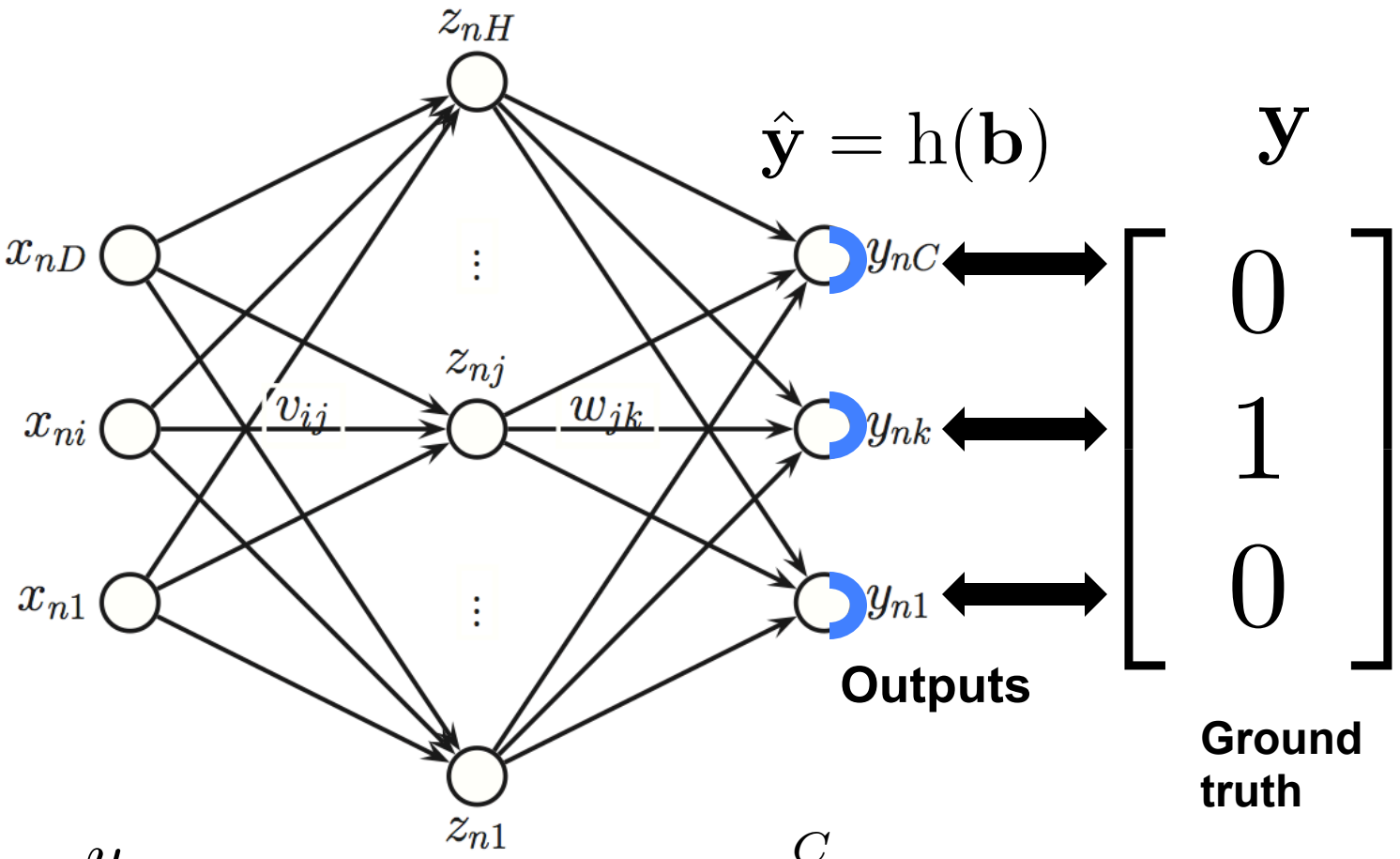


Objective for multi-class classification



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

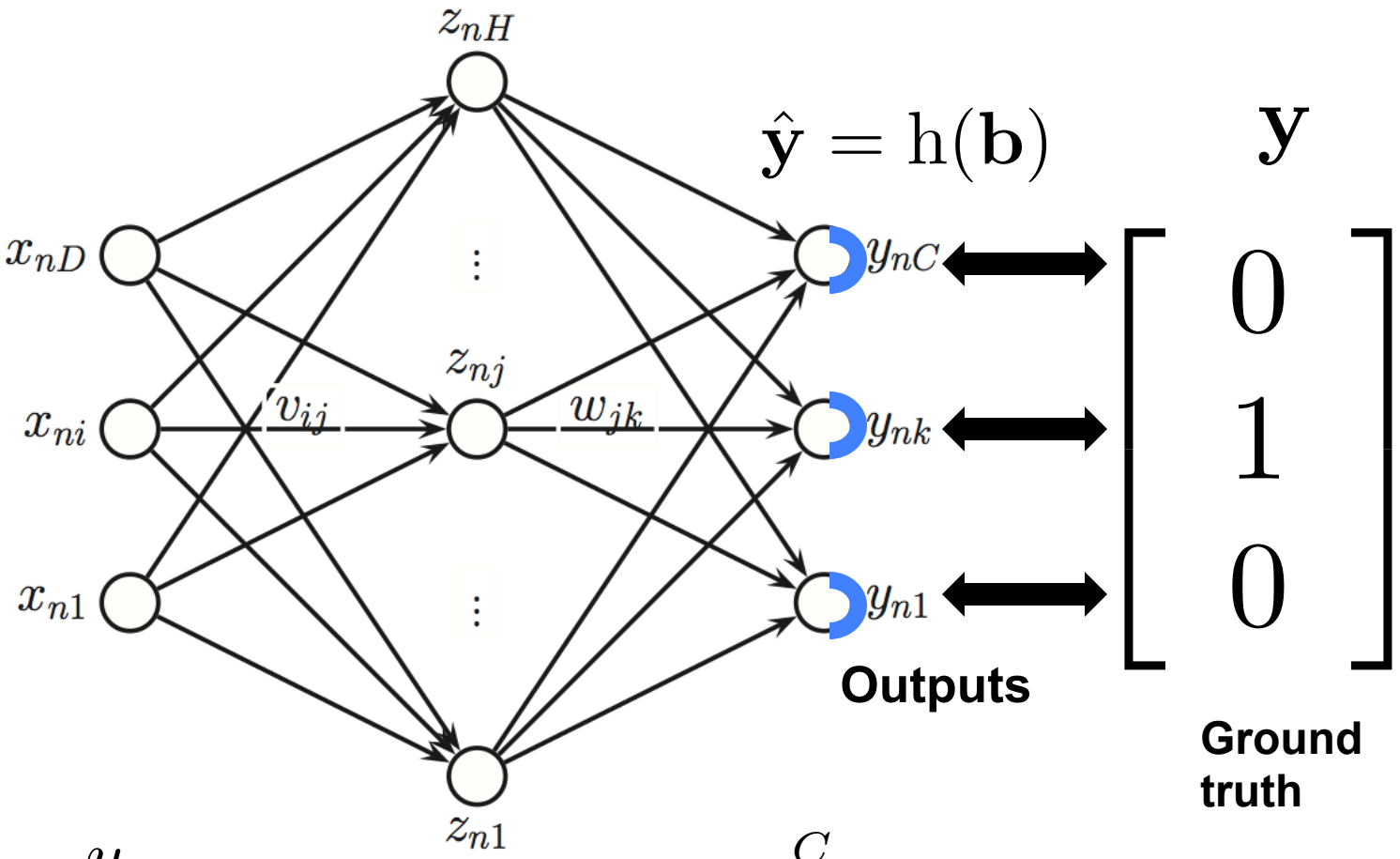
Derivative of loss w.r.t. top-layer neurons



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

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

A neural network in backward mode:



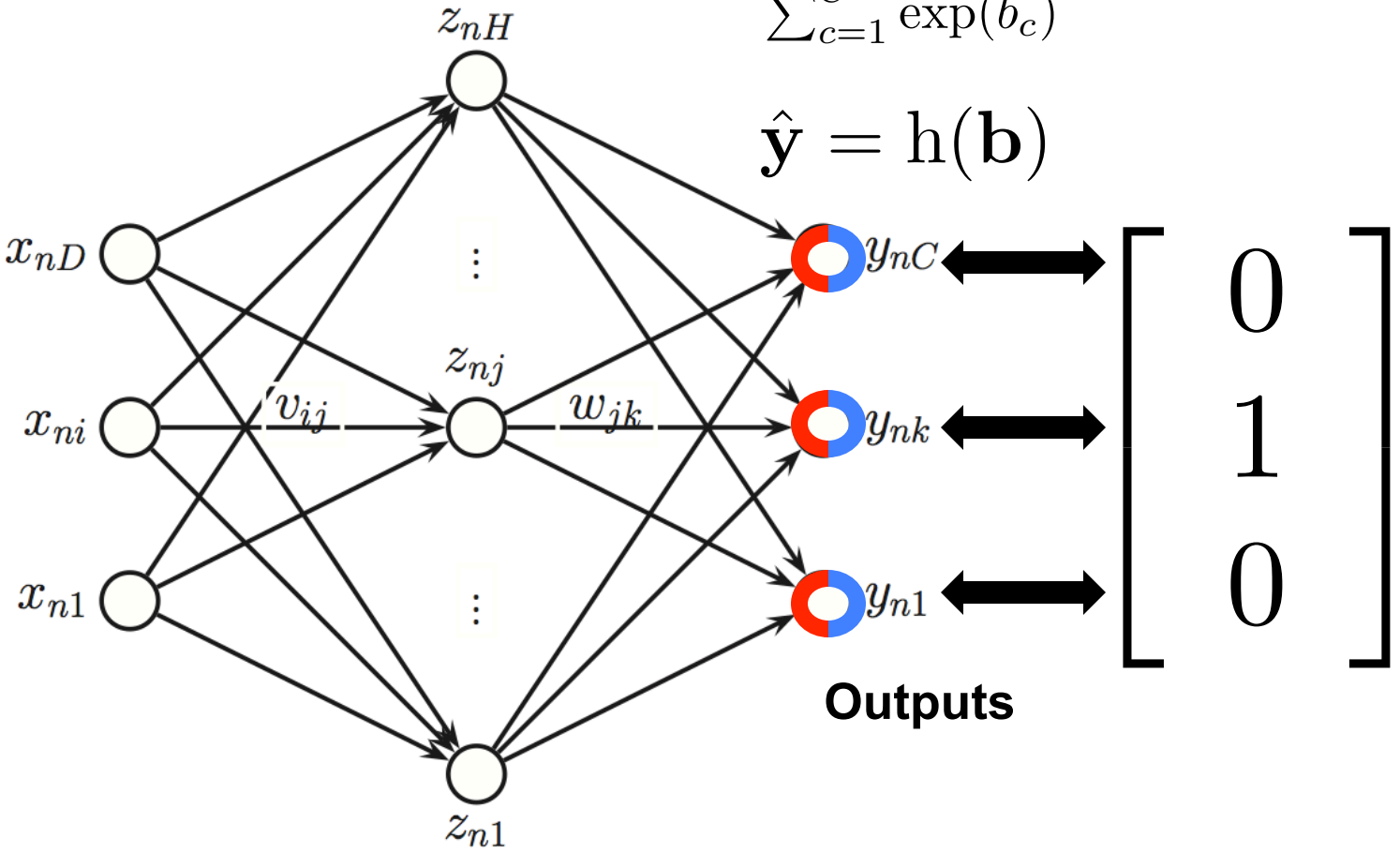
$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

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

A neural network in backward mode:

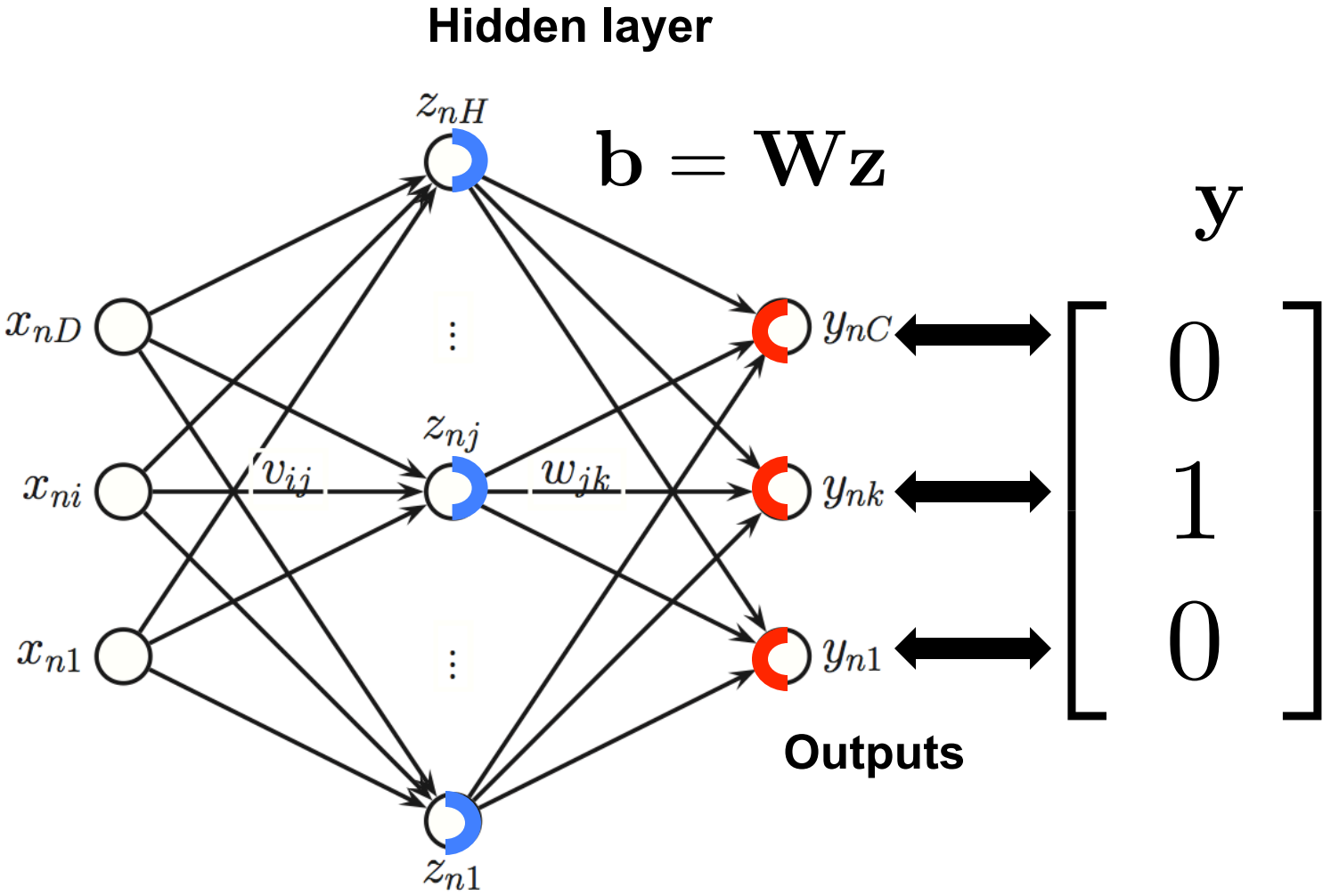
$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{c=1}^{C} \exp(b_c)}$$

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



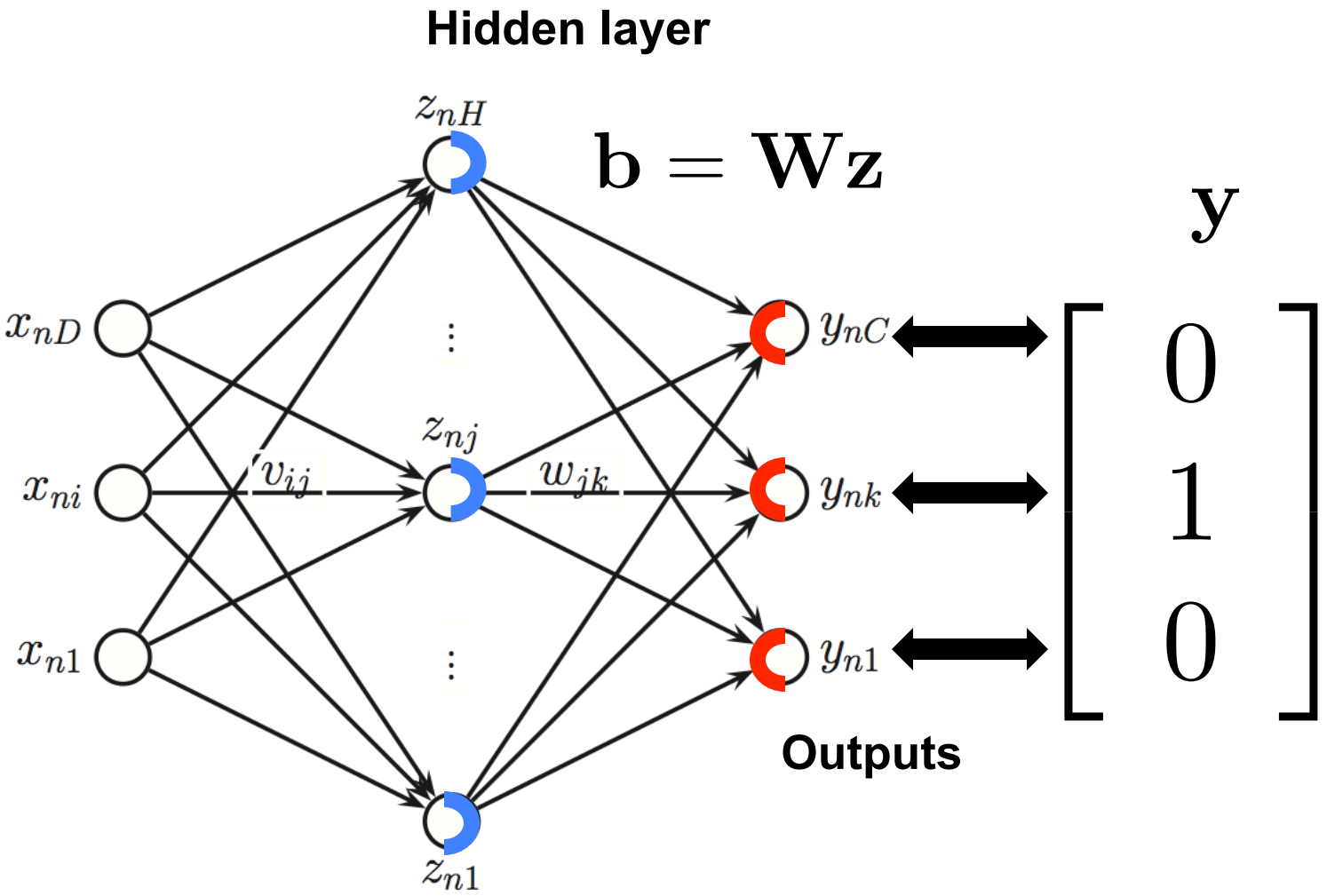
$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c} \qquad \frac{\partial L}{\partial b_c} = \frac{\partial L}{\partial \hat{y}_c}\frac{\partial \hat{y}_c}{\partial b_c} \quad \ldots = (\hat{y}_c - y_c)b_c$$

A neural network in backward mode:



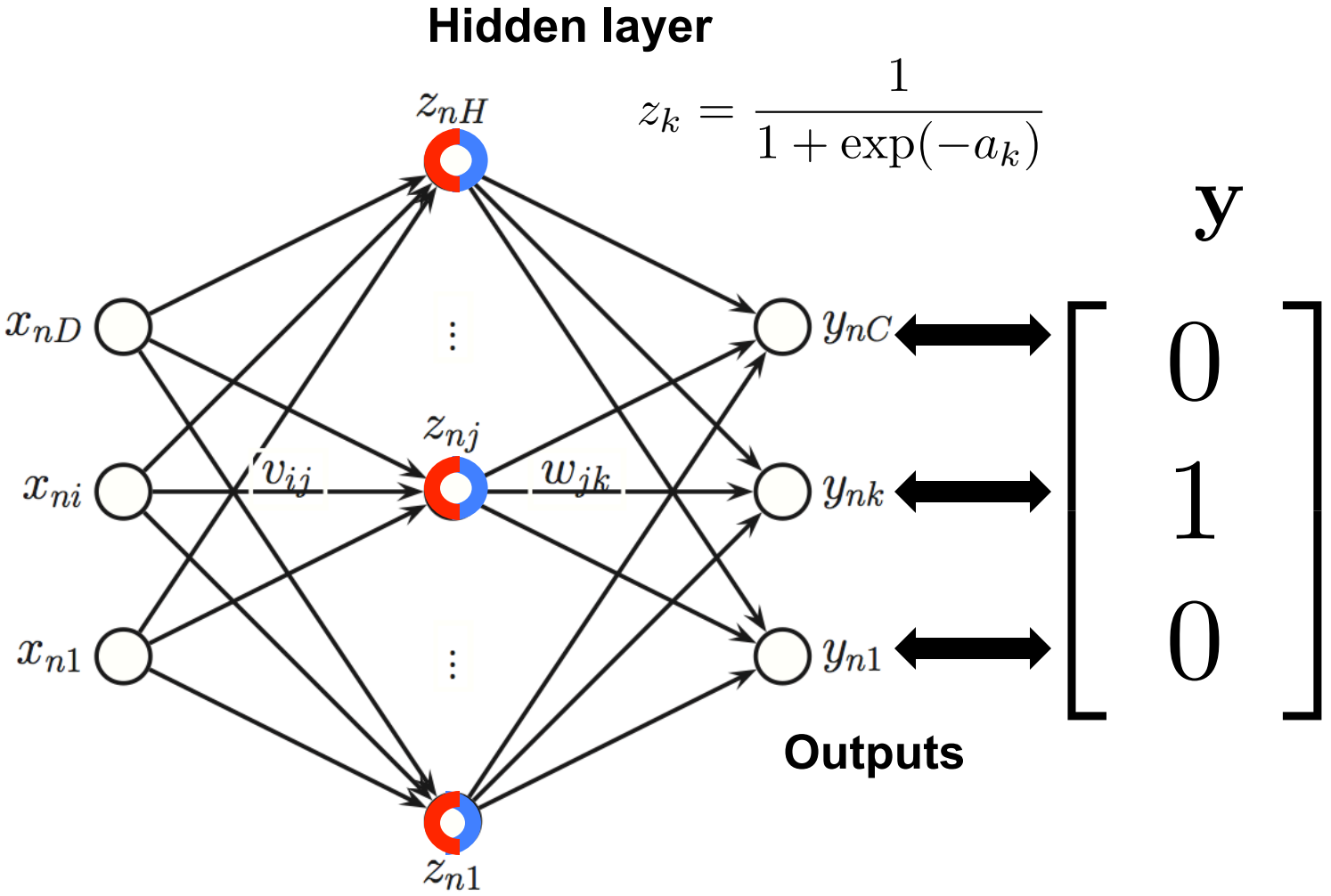
$$\frac{\partial l}{\partial w_{jk}} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial w_{jk}} = \frac{\partial l}{\partial b_k} z_j$$

A neural network in backward mode:



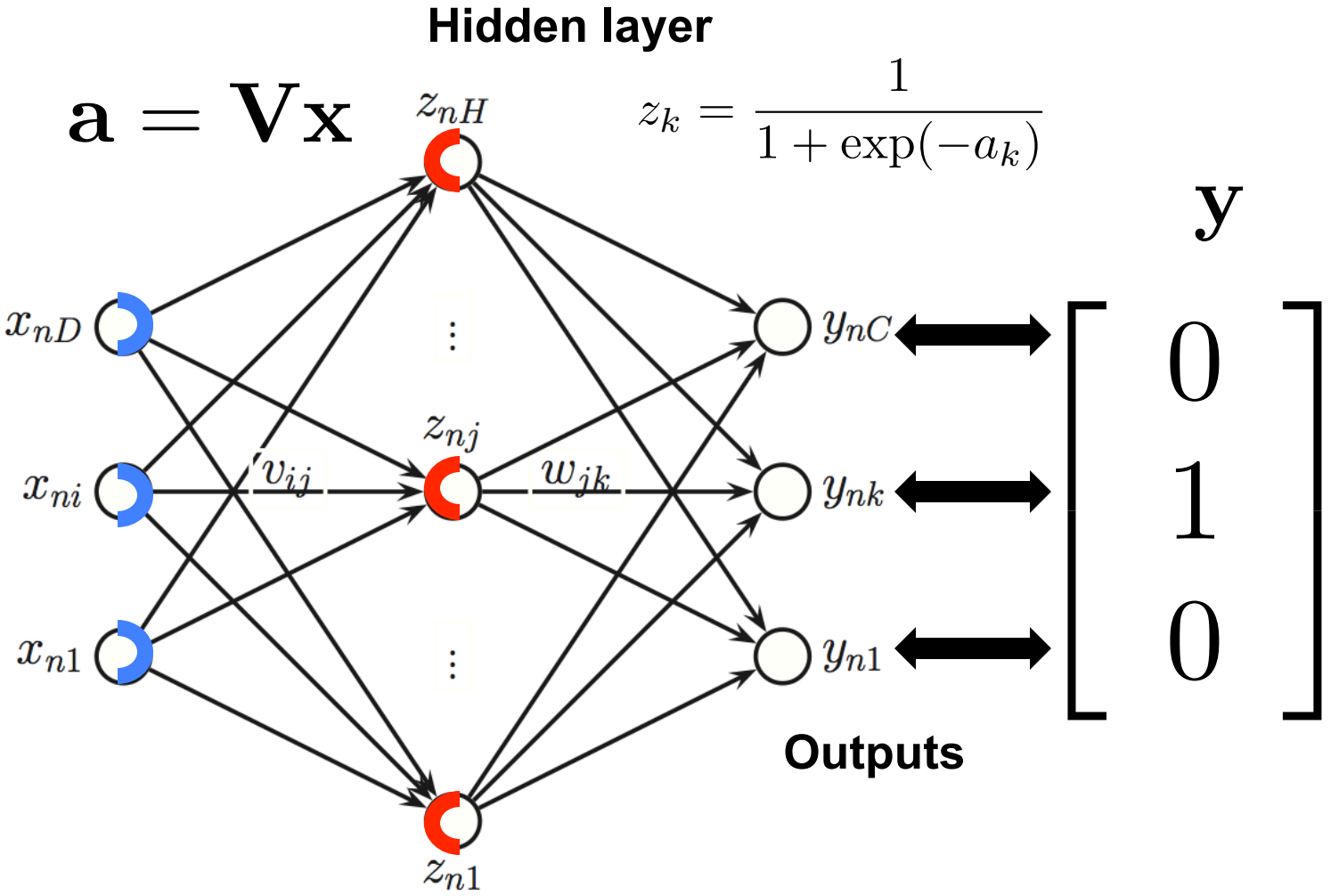
$$\frac{\partial l}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} w_{j,m}$$

A neural network in backward mode:



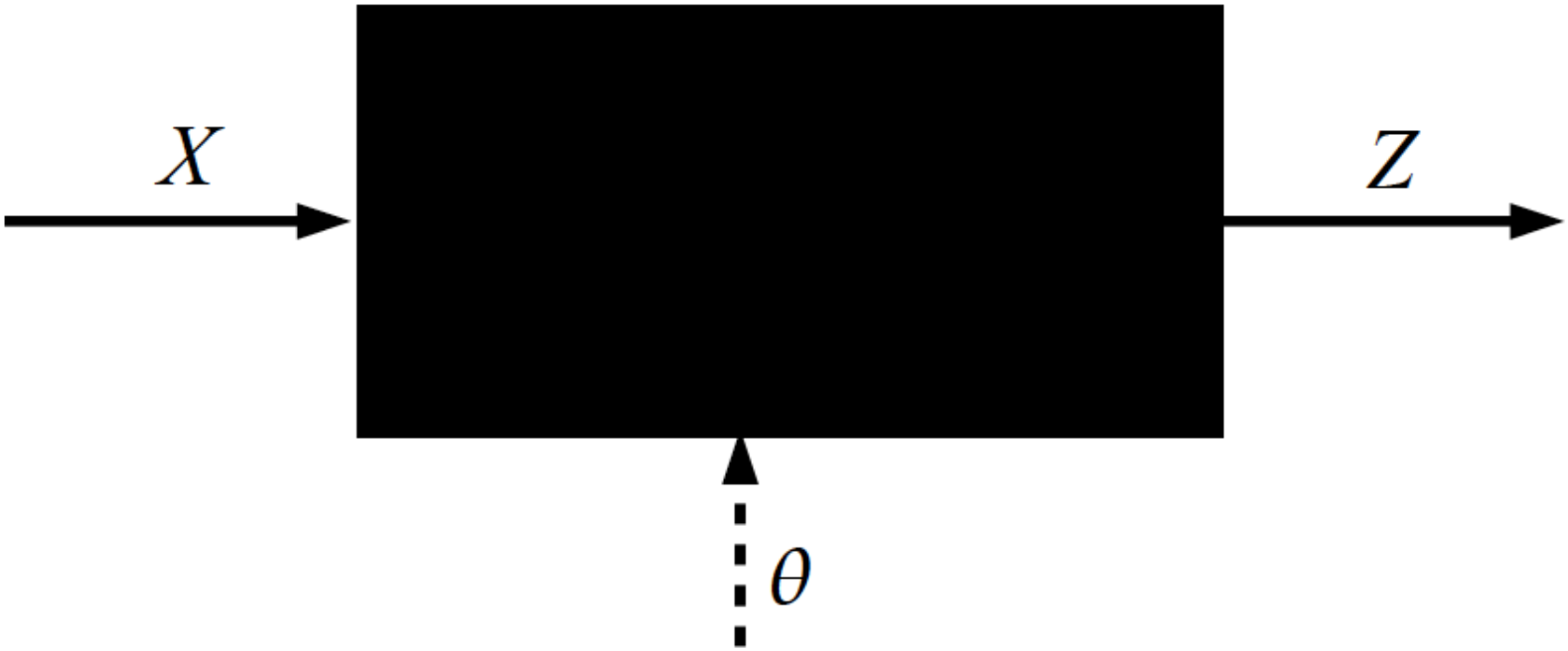
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1-g(a_k))$$

A neural network in backward mode:



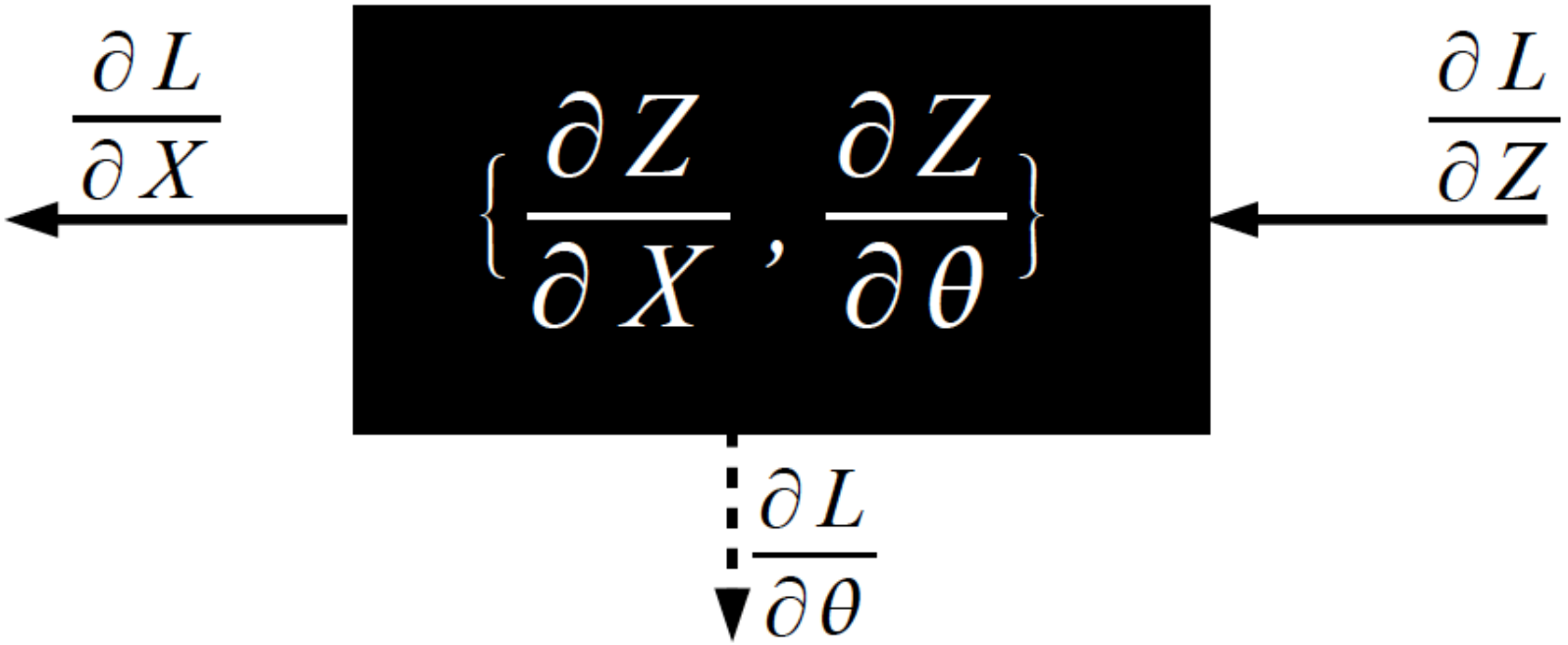
$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k}\frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

Key Computation: Forward-Prop



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Key Computation: Back-Prop

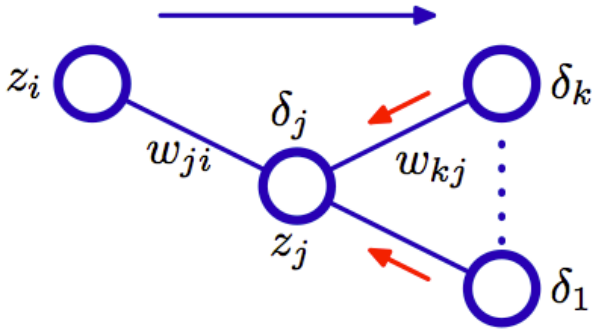


Back-propagation in a nutshell

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

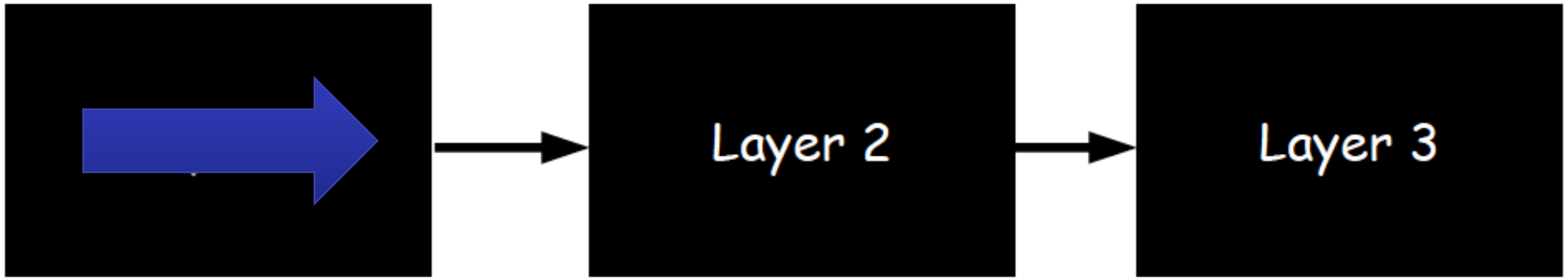
Every node estimates its error message by forming a weighted sum of the error messages of its recipients

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.



Neural Network Training

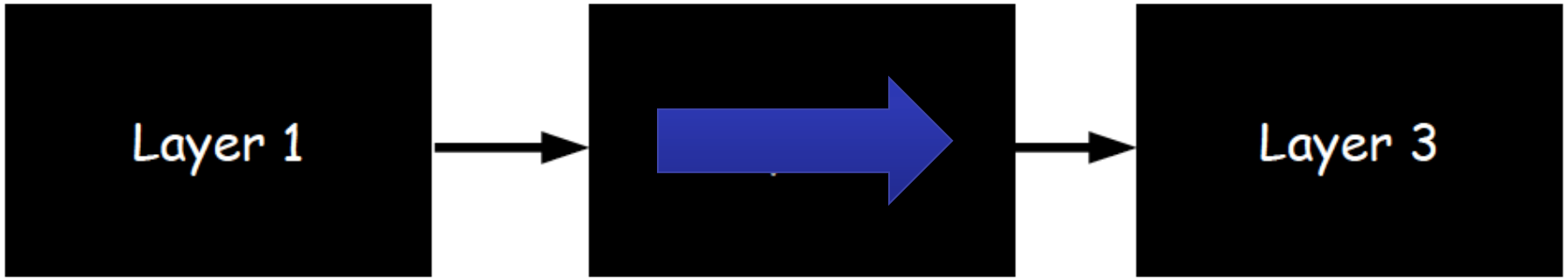
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

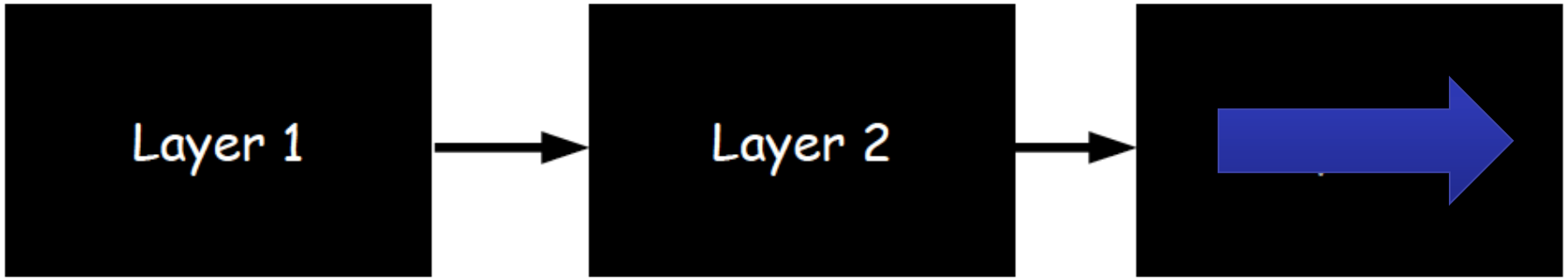
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

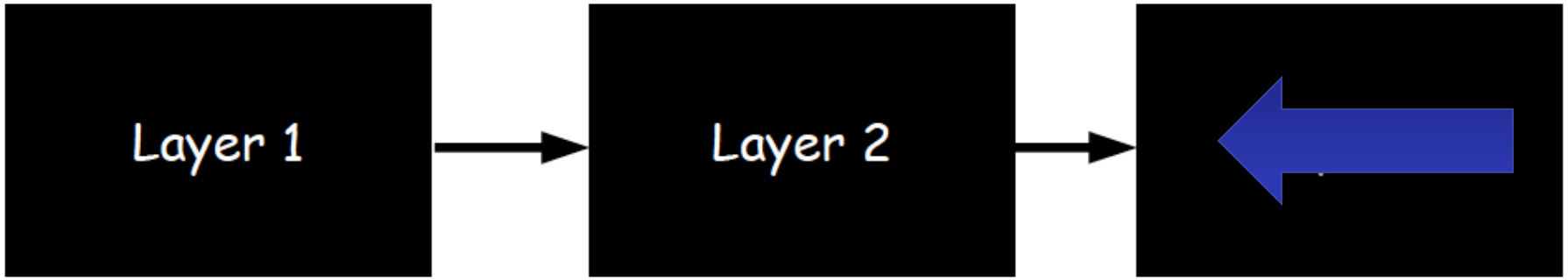
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

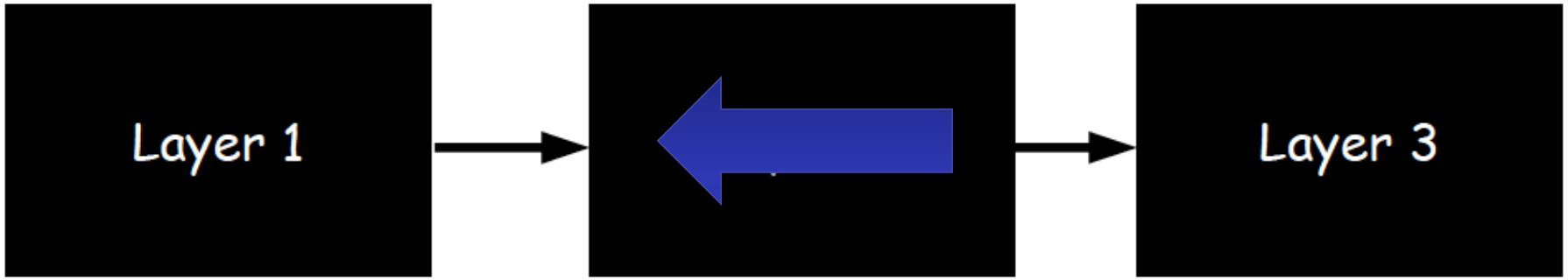
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

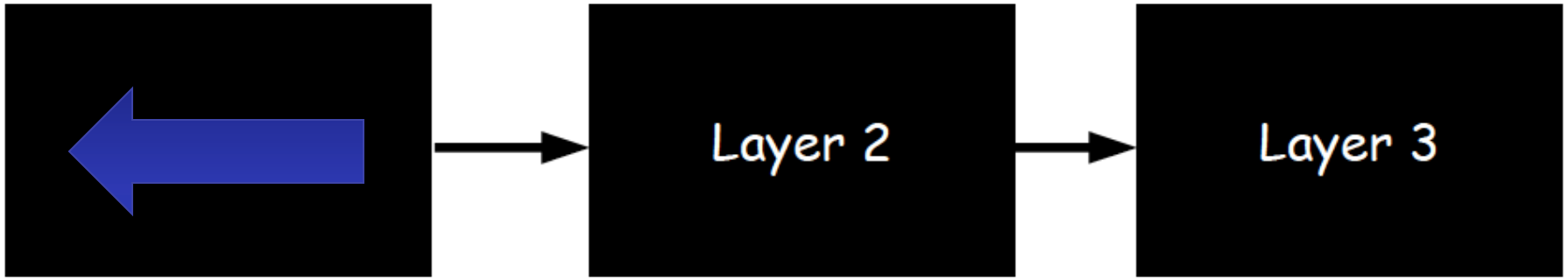
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

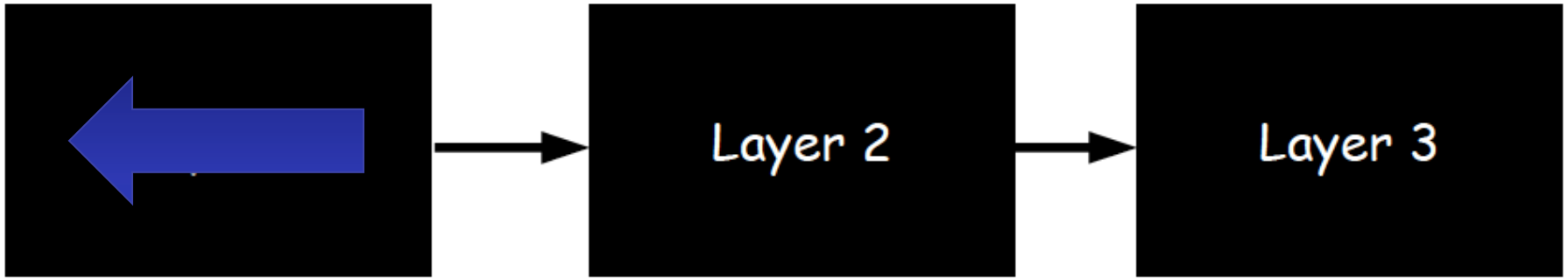
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
- Step 3: Use gradient to update parameters

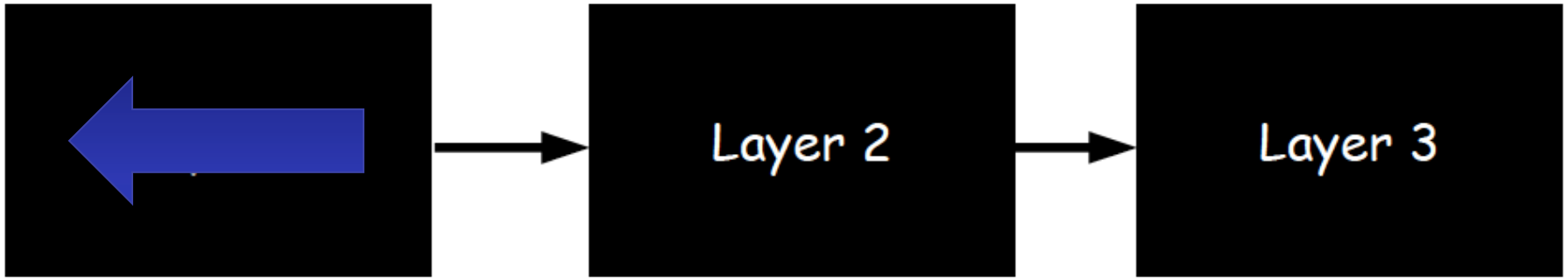


$$\theta \leftarrow \theta - \eta \frac{dL}{d\theta}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
- Step 3: Use gradient to update parameters
  - With momemtum

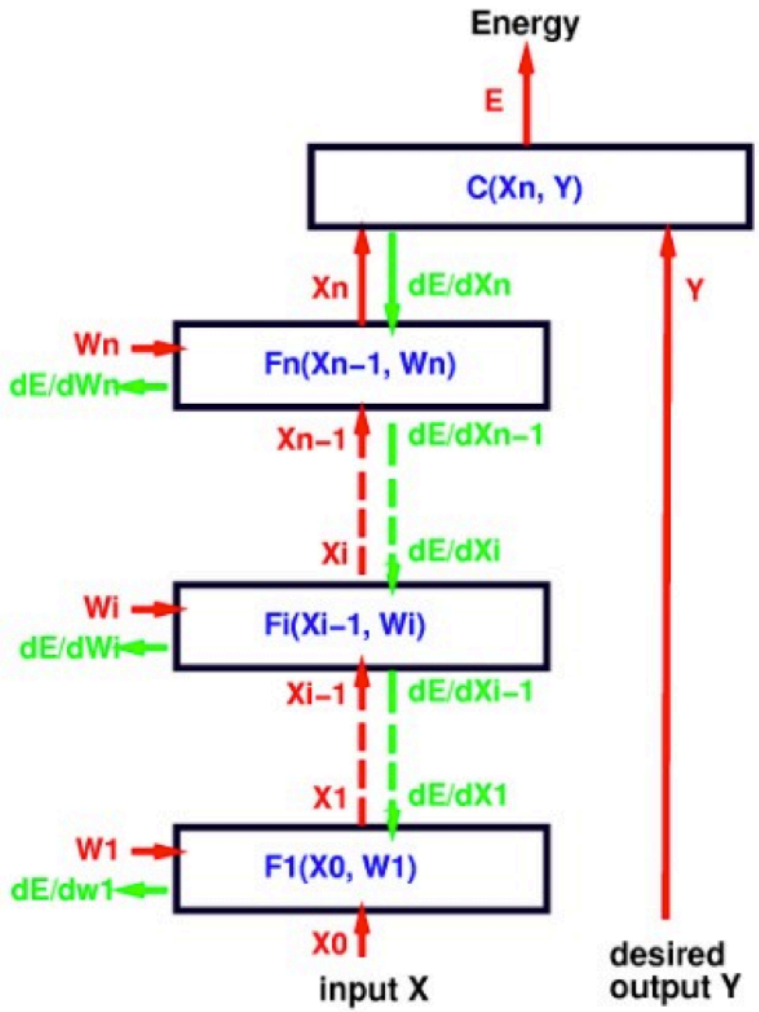


$$\theta \leftarrow \theta - \eta \Delta$$

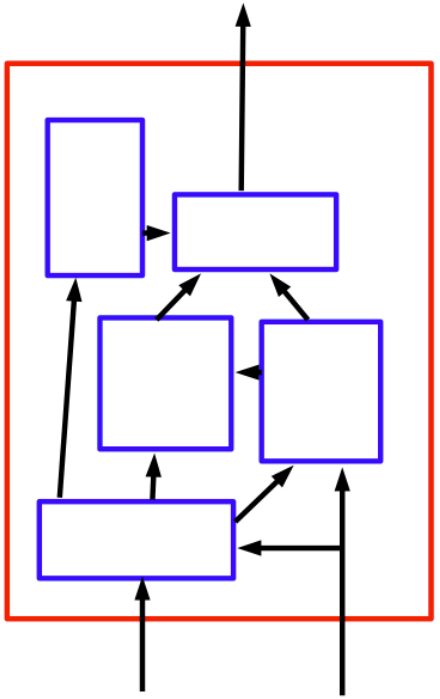
$$\Delta \leftarrow 0.9\,\Delta + \frac{\partial L}{\partial \theta}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Forward/backward information flow

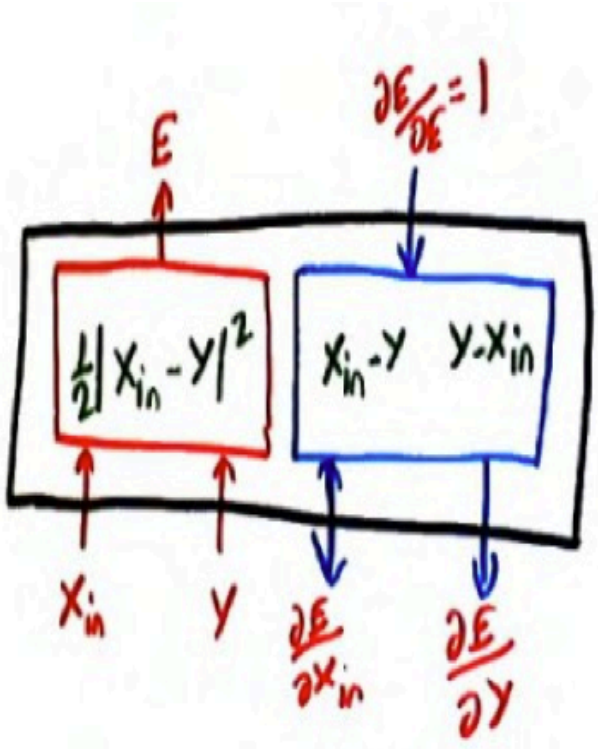


any DAG is OK!



Slide credit: Le Cun/Ranzatto

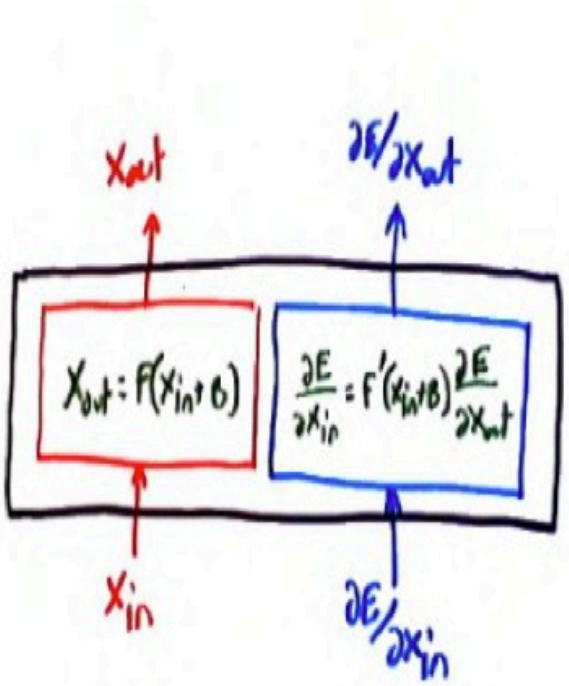
Any differentiable module is OK!



- fprop: $X_{\text{out}} = \frac{1}{2}||X_{\text{in}} - Y||^2$
- bprop to X input: $\frac{\partial E}{\partial X_{\text{in}}} = X_{\text{in}} - Y$
- bprop to Y input: $\frac{\partial E}{\partial Y} = Y - X_{\text{in}}$

Slide credit: Le Cun/Ranzatto

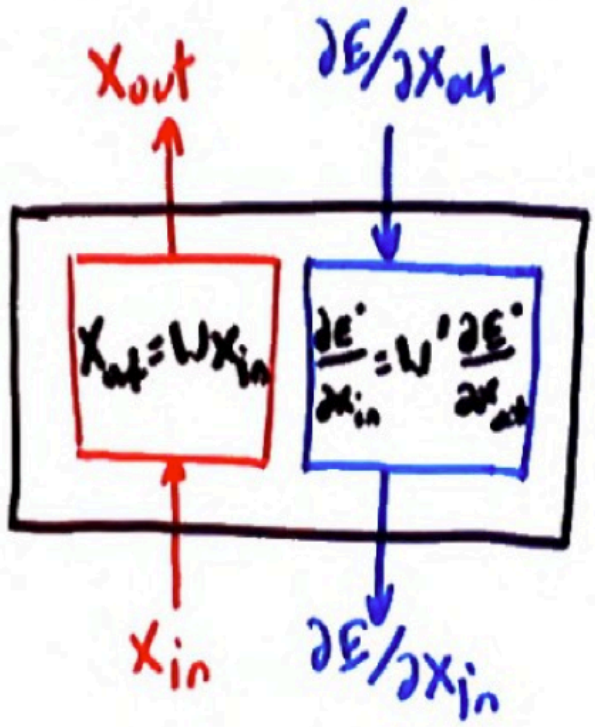
Any differentiable module is OK!



- fprop: $(X_{\text{out}})_i = \tanh((X_{\text{in}})_i + B_i)$
- bprop to input:
  $(\frac{\partial E}{\partial X_{\text{in}}})_i = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- bprop to bias:
  $\frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- $\tanh(x) = \frac{2}{1+\exp -x} - 1 = \frac{1-\exp(-x)}{1+\exp(-x)}$

Slide credit: Le Cun/Ranzatto

Any differentiable module is OK!



- fprop: $X_{\text{out}} = W X_{\text{in}}$
- bprop to input:
  $\frac{\partial E}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{out}}}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} W$
- by transposing, we get column vectors:
  $\frac{\partial E}{\partial X_{\text{in}}}' = W' \frac{\partial E}{\partial X_{\text{out}}}'$
- bprop to weights:
  $\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_{\text{out}i}} \frac{\partial X_{\text{out}i}}{\partial W_{ij}} = X_{\text{in}j} \frac{\partial E}{\partial X_{\text{out}i}}$
- We can write this as an outer-product:
  $\frac{\partial E}{\partial W}' = \frac{\partial E}{\partial X_{\text{out}}}' X_{in}'$

Slide credit: Le Cun/Ranzatto