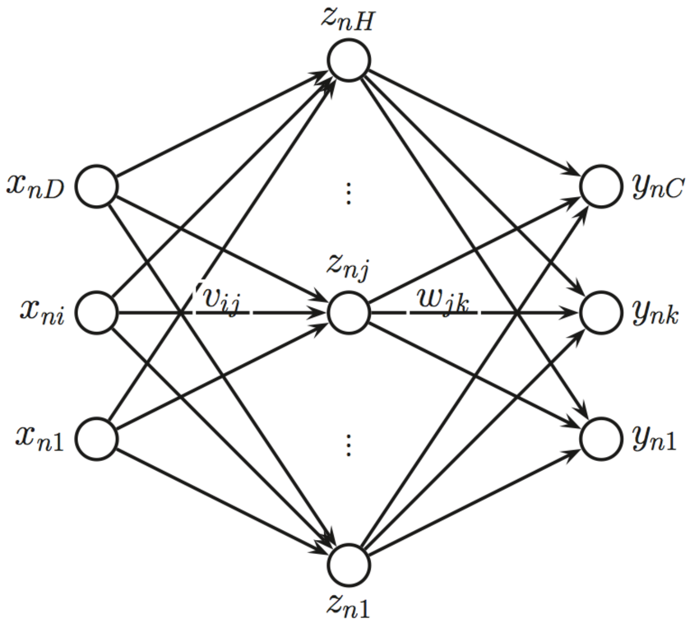
Introduction to Machine Learning



Week 5
Learning with neural networks

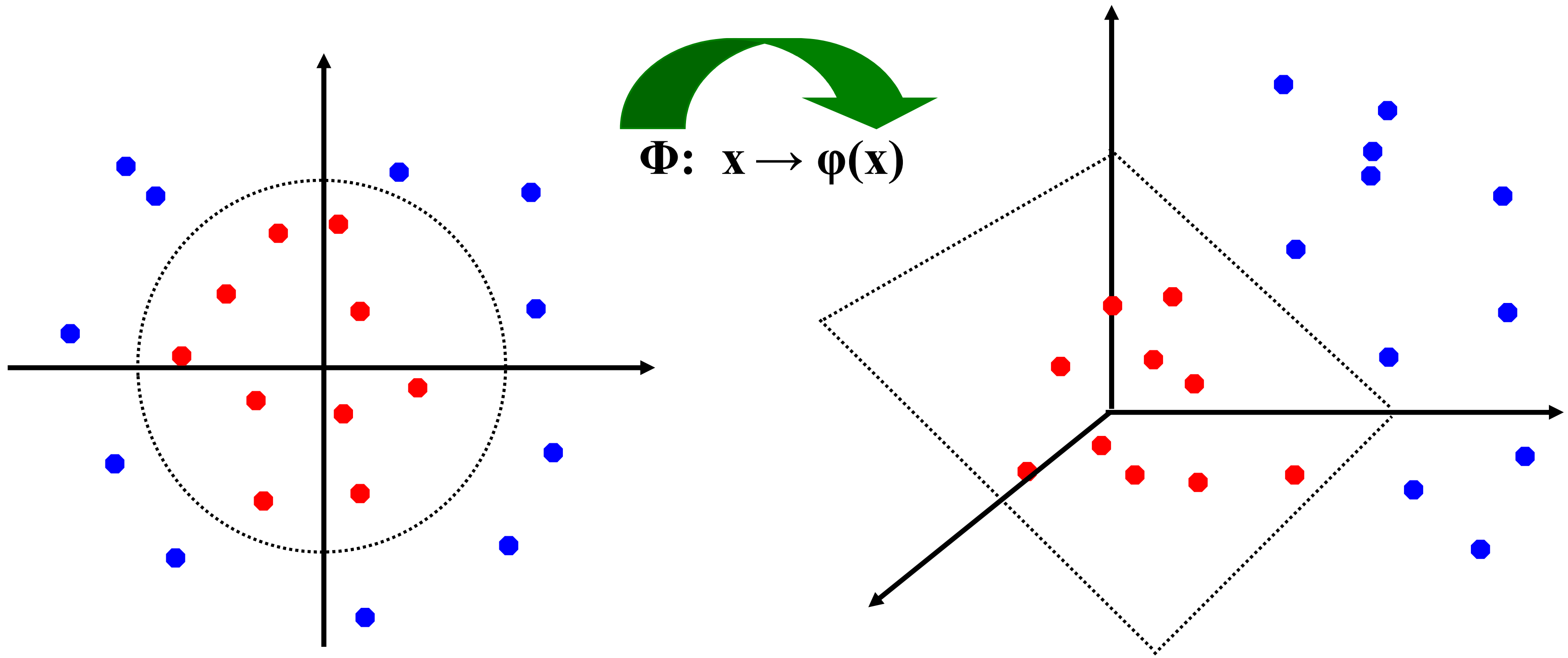
Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Beyond linear boundaries

- Weeks 1-2: More features & regularization



Beyond linear boundaries

- Week 3: 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 4: 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

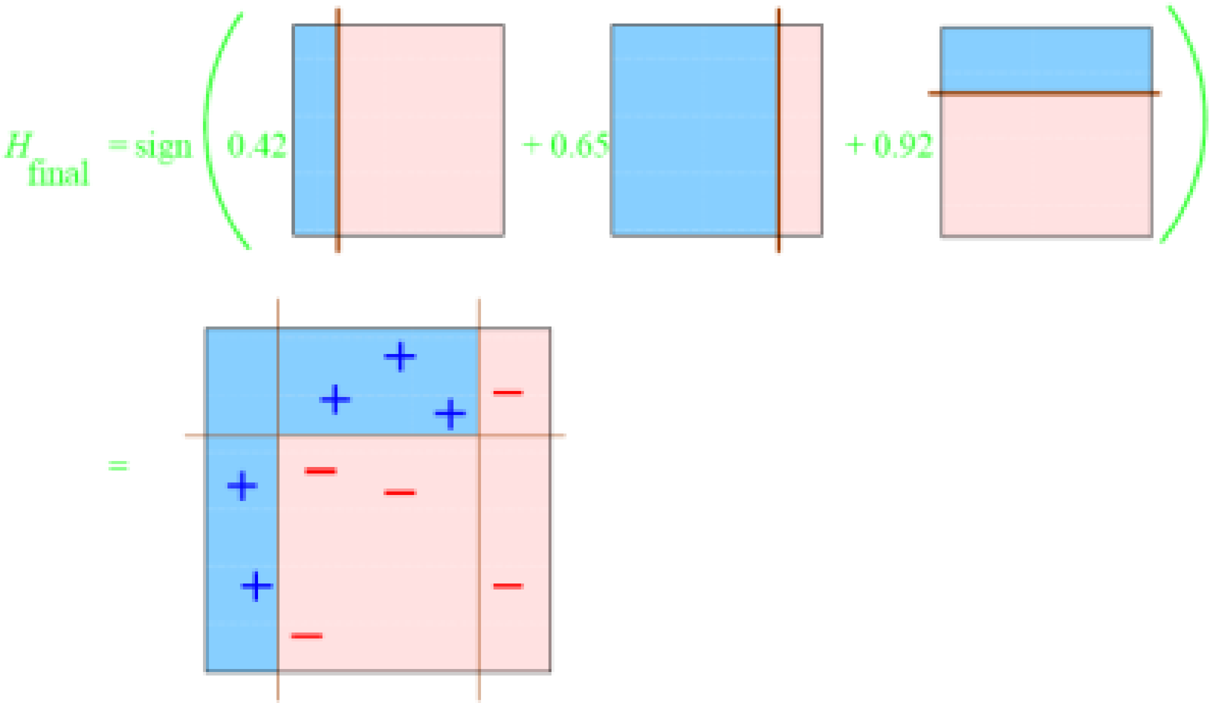
Weak classifier

Weight

Feature vector

Beyond linear boundaries

- Week 6: Weak Learners & Ensembling: Adaboost approach



Decision Trees/Forests

Extremely flexible classifiers (power of composition)

Low-cost (simple comparisons, no kernel evaluations)

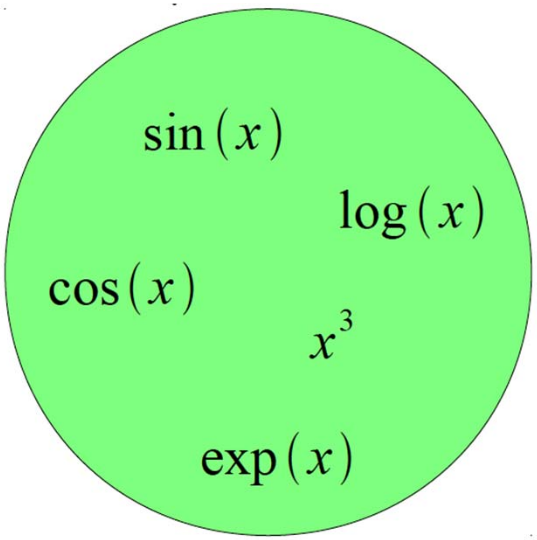
But: learned greedily

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

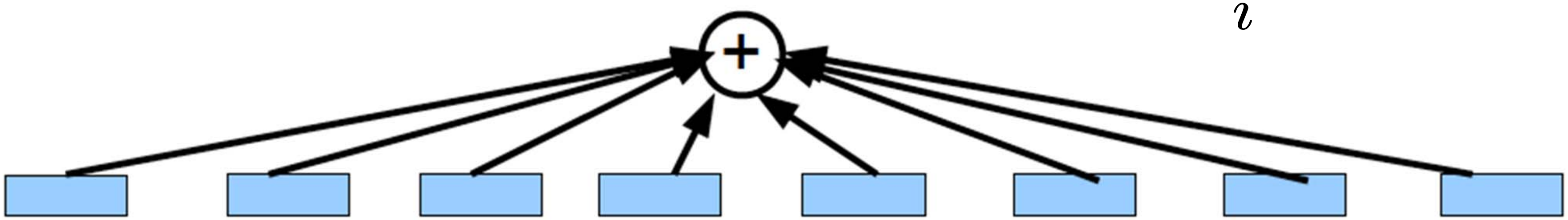


Compose into a complicated function

Idea 1: Linear Combinations

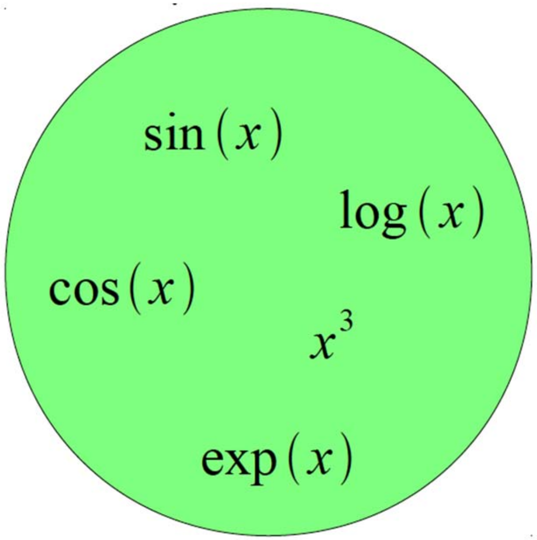
- Boosting
- Kernels
- …

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



Building A Complicated Function

Given a library of simple functions



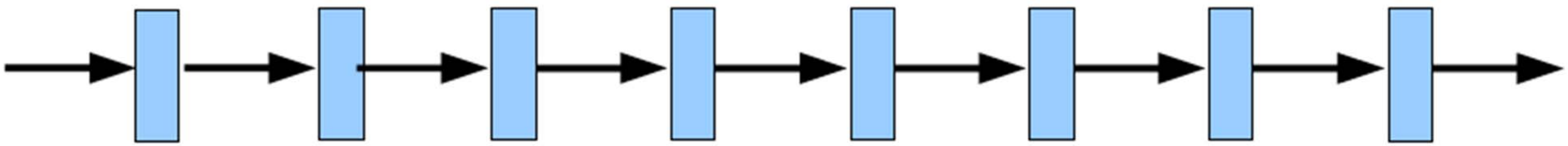
Compose into a

complicated function

Idea 2: Compositions

- Decision Trees
- Grammar models
- Deep Learning

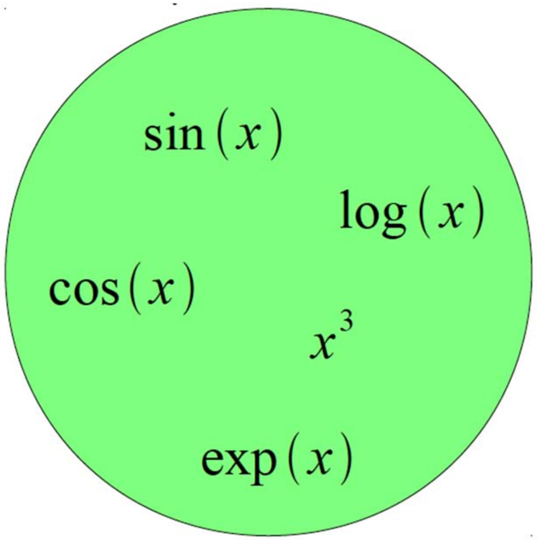
$$f(x) = g_1(g_2(\ldots(g_n(x)\ldots))$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



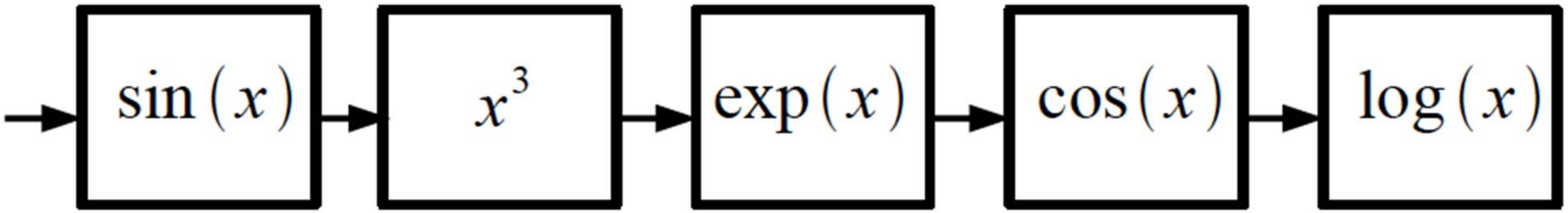
Compose into a

complicated function

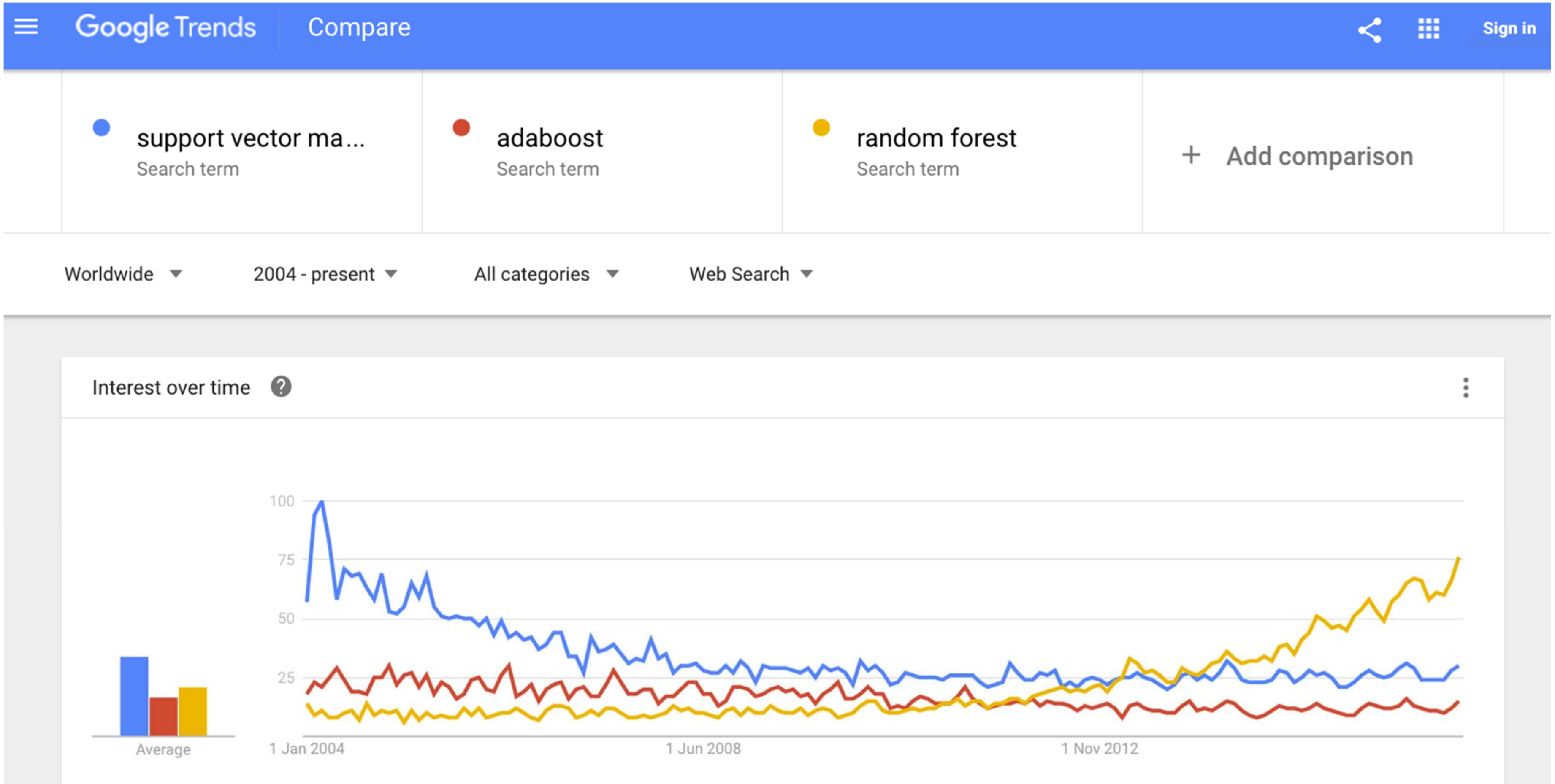
Idea 2: Compositions

- Decision Trees
- Grammar models
- Deep Learning

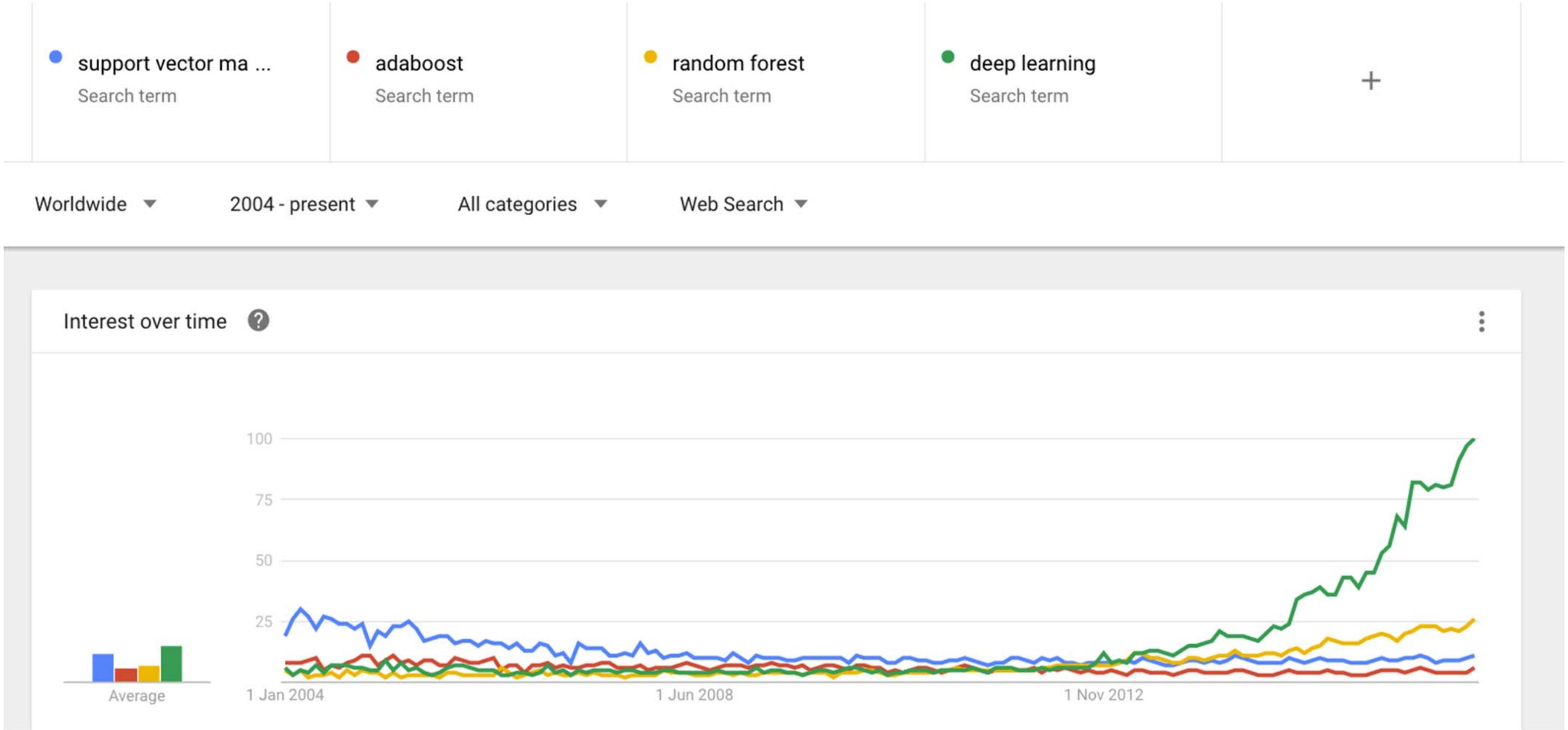
$$f(x) = \log(\cos(\exp(\sin^3(x))))$$



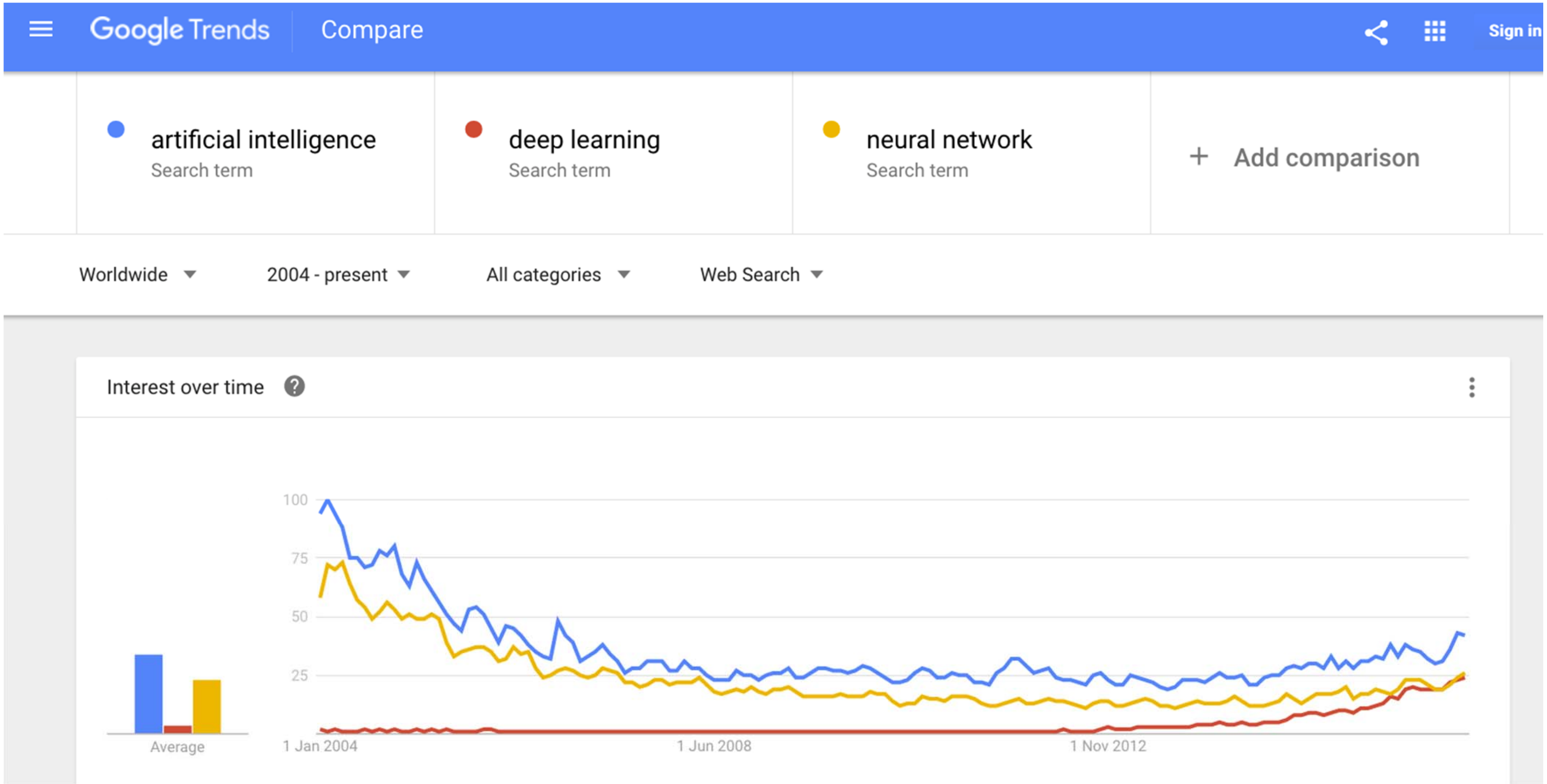
Trends, 2004-now (Google trends)



Trends, 2004-now (Google trends)

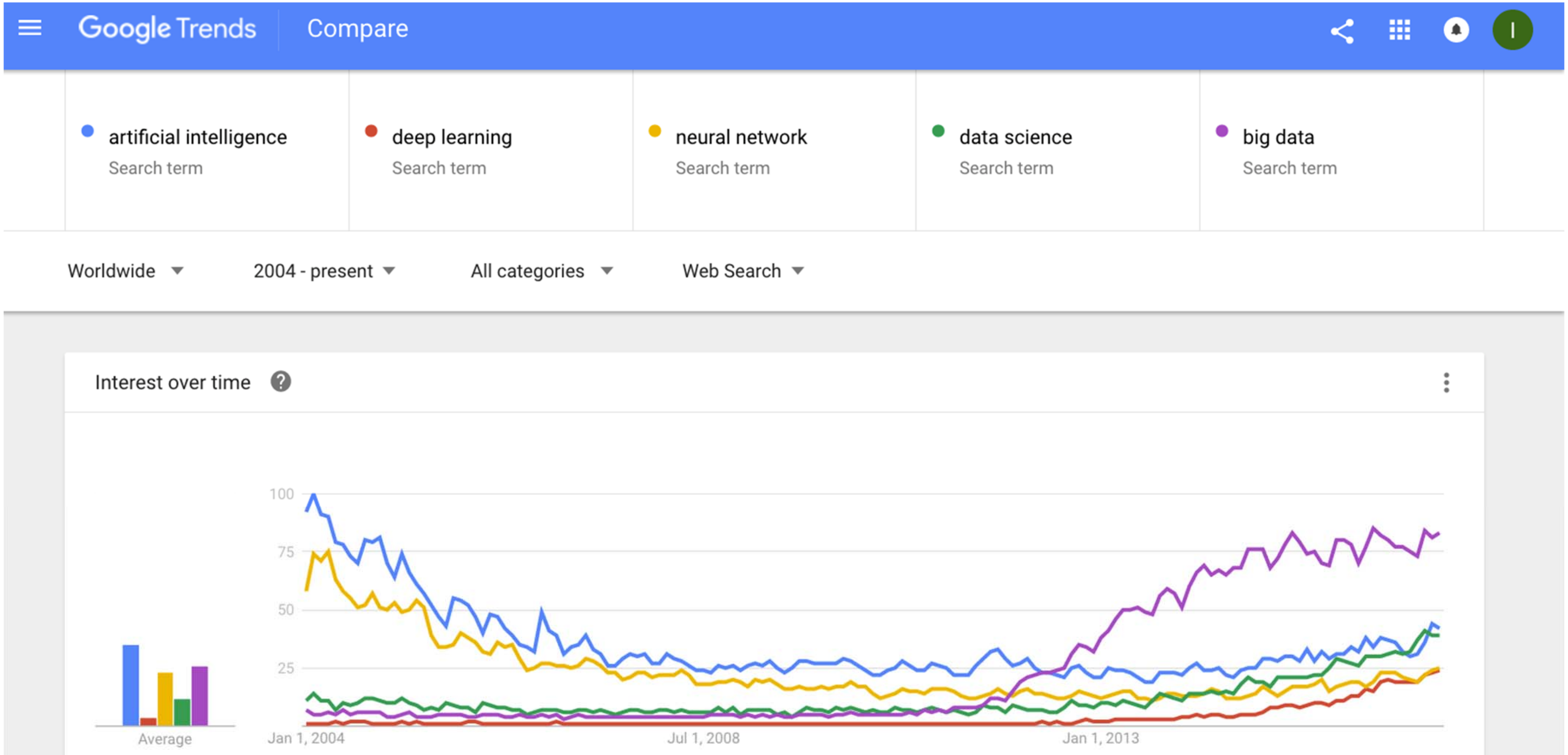


Trends, 2004-now (Google trends)



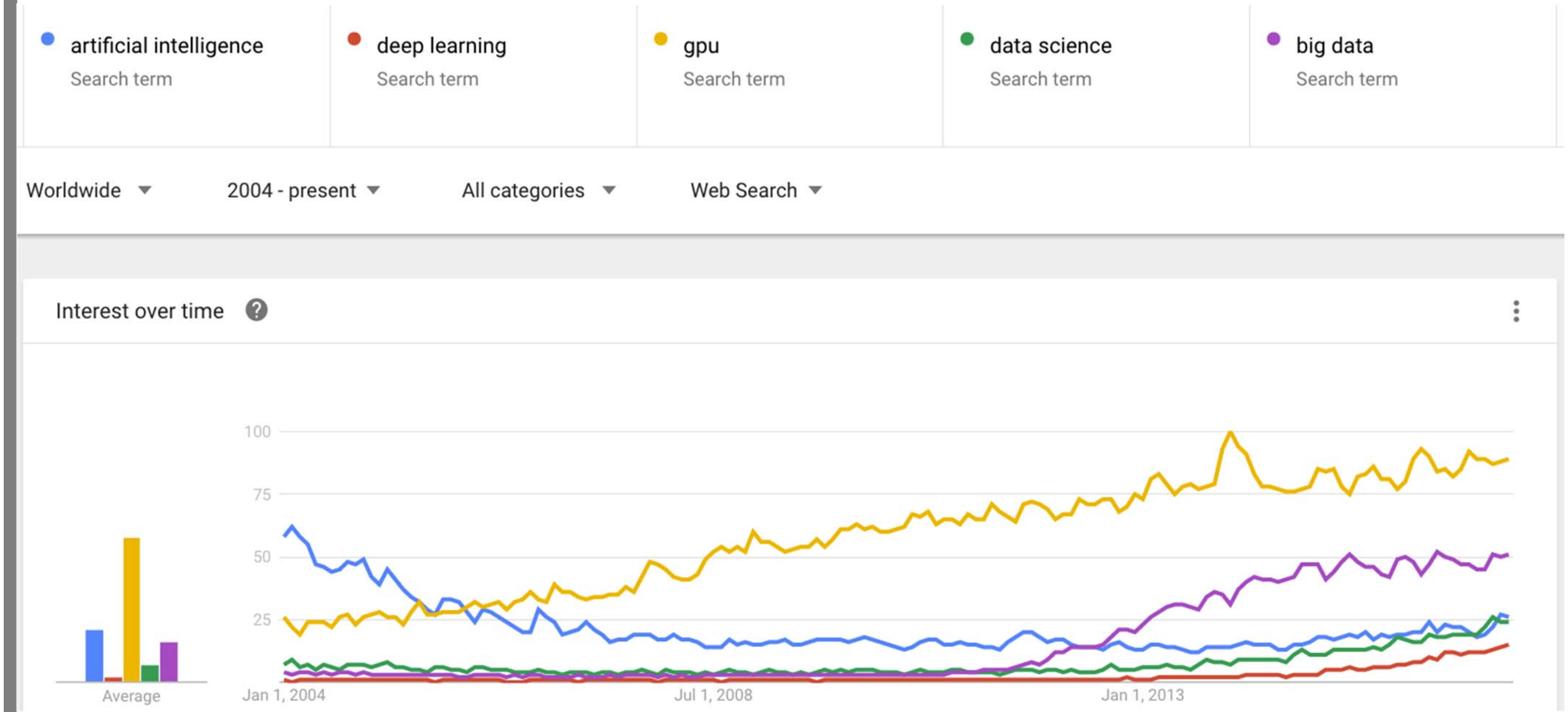
Useful to keep in mind:
deep learning = neural networks

Trends, 2014-now (Google trends)



Useful to keep in mind:
deep learning = neural networks (+ big data)

Trends, 2014-now (Google trends)

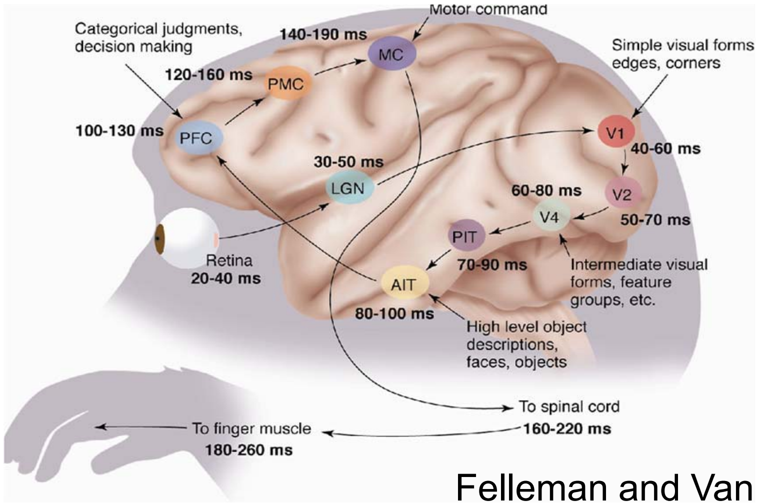


Useful to keep in mind:
deep learning = neural networks (+ big data + GPUs)

‘Neuron’: basic building block

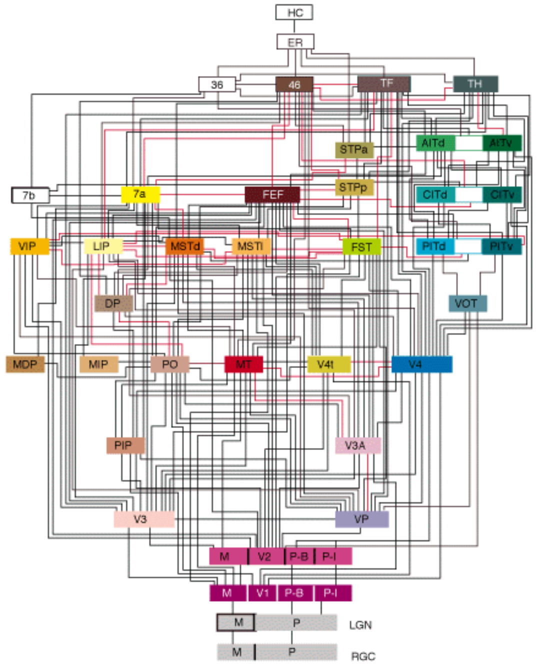
Analogy with the brain

Hierarchical organization of the cortex

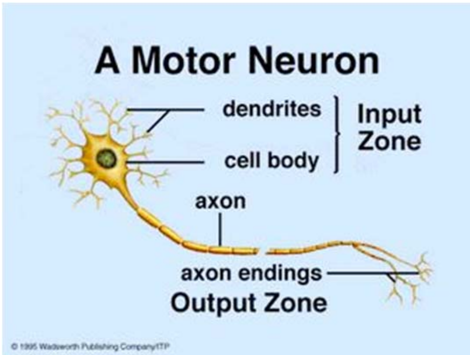


Felleman and Van Essen, 1991

Diagram of the visual system

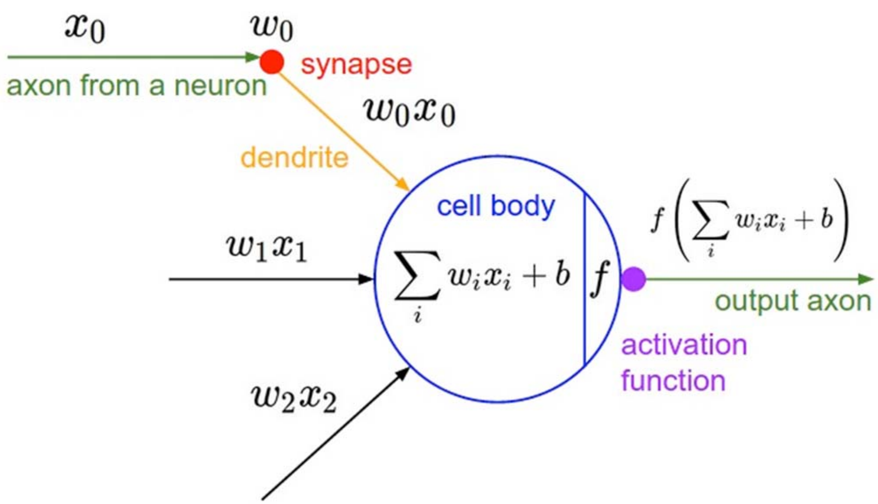


Building blocks: neurons



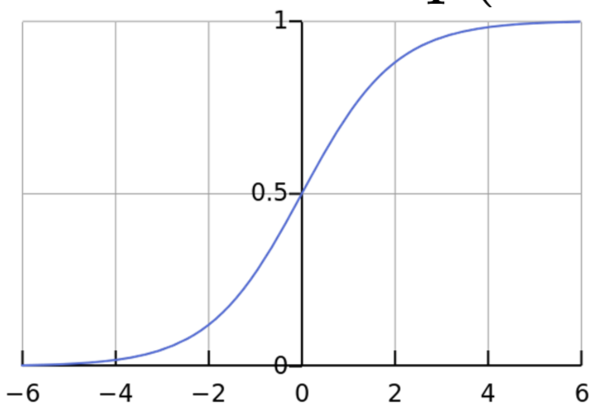
same neuron dynamics throughout

‘Neuron’: cascade of linear and nonlinear function



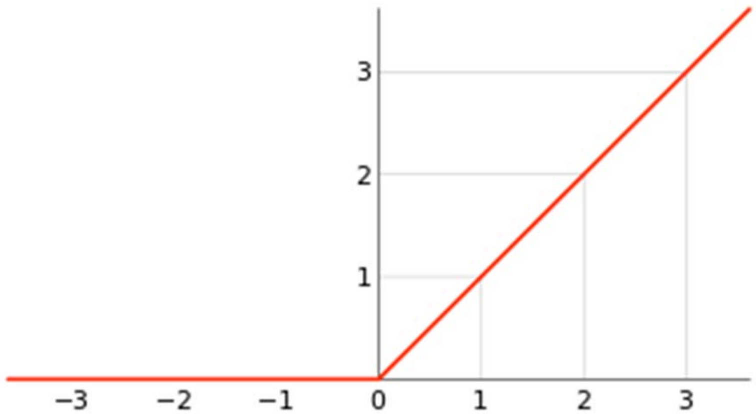
Sigmoidal (“logistic”)

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



Rectified Linear Unit (RELU)

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



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

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

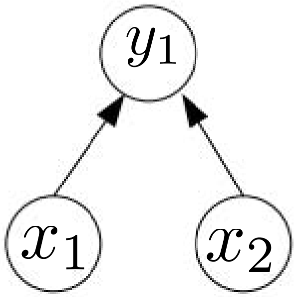
Rectified Linear Unit (RELU)

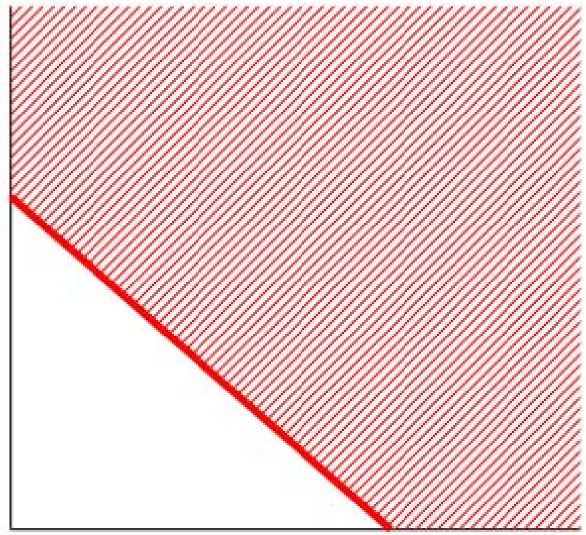
$$g(a) = \max(0, a)$$

Beyond linear boundaries

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

1 layer of trainable weights

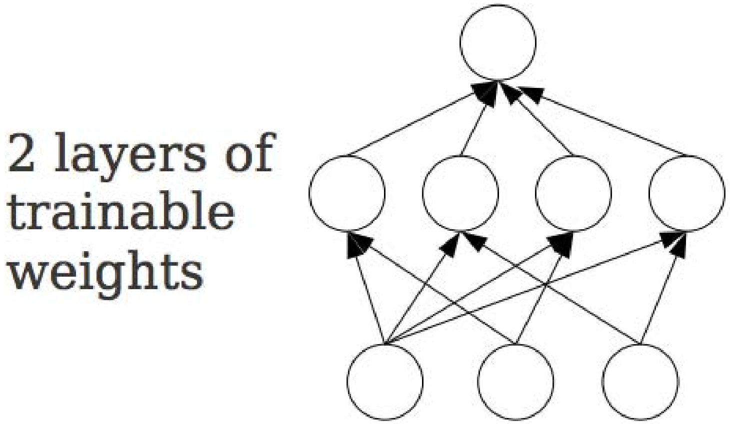


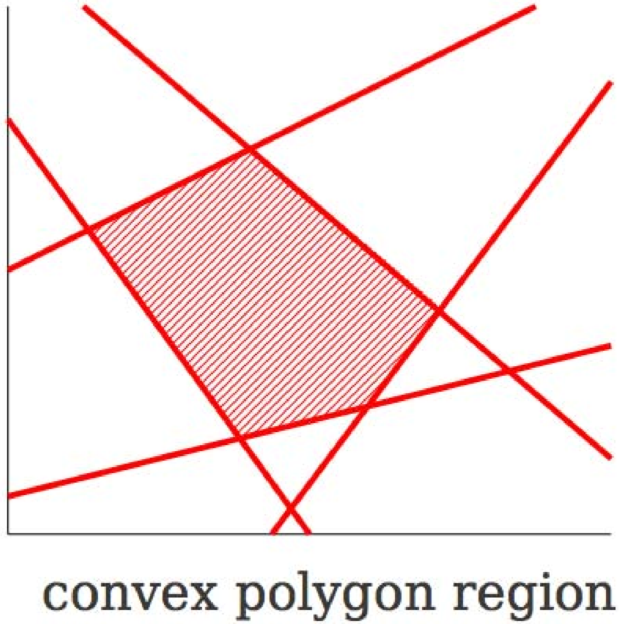


separating hyperplane

Beyond linear boundaries

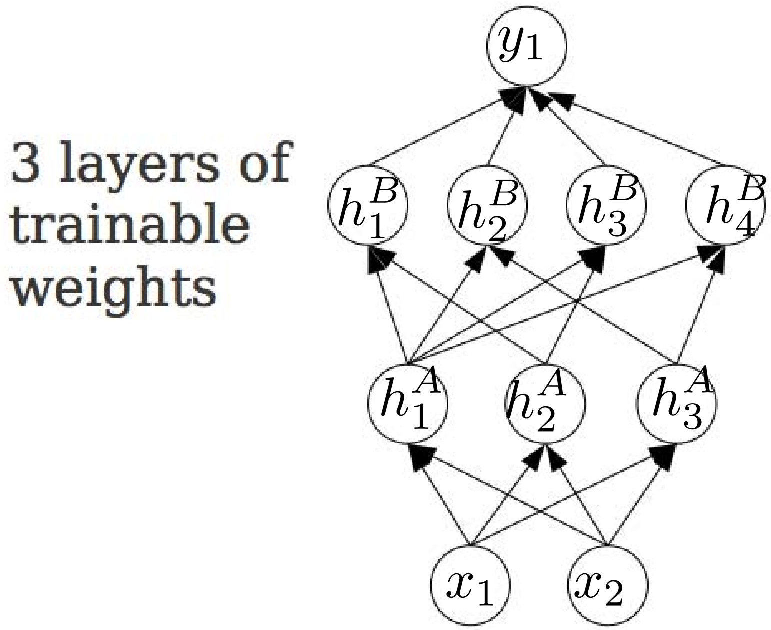
- Today: ‘deep learning’ (a.k.a. neural network) approach

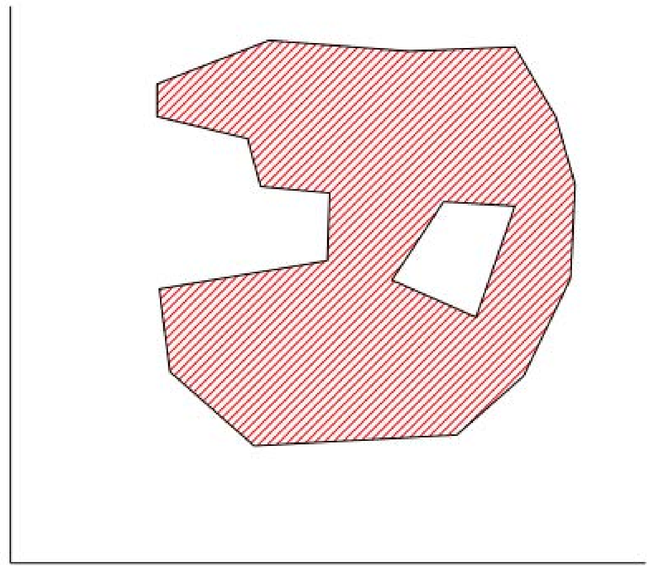




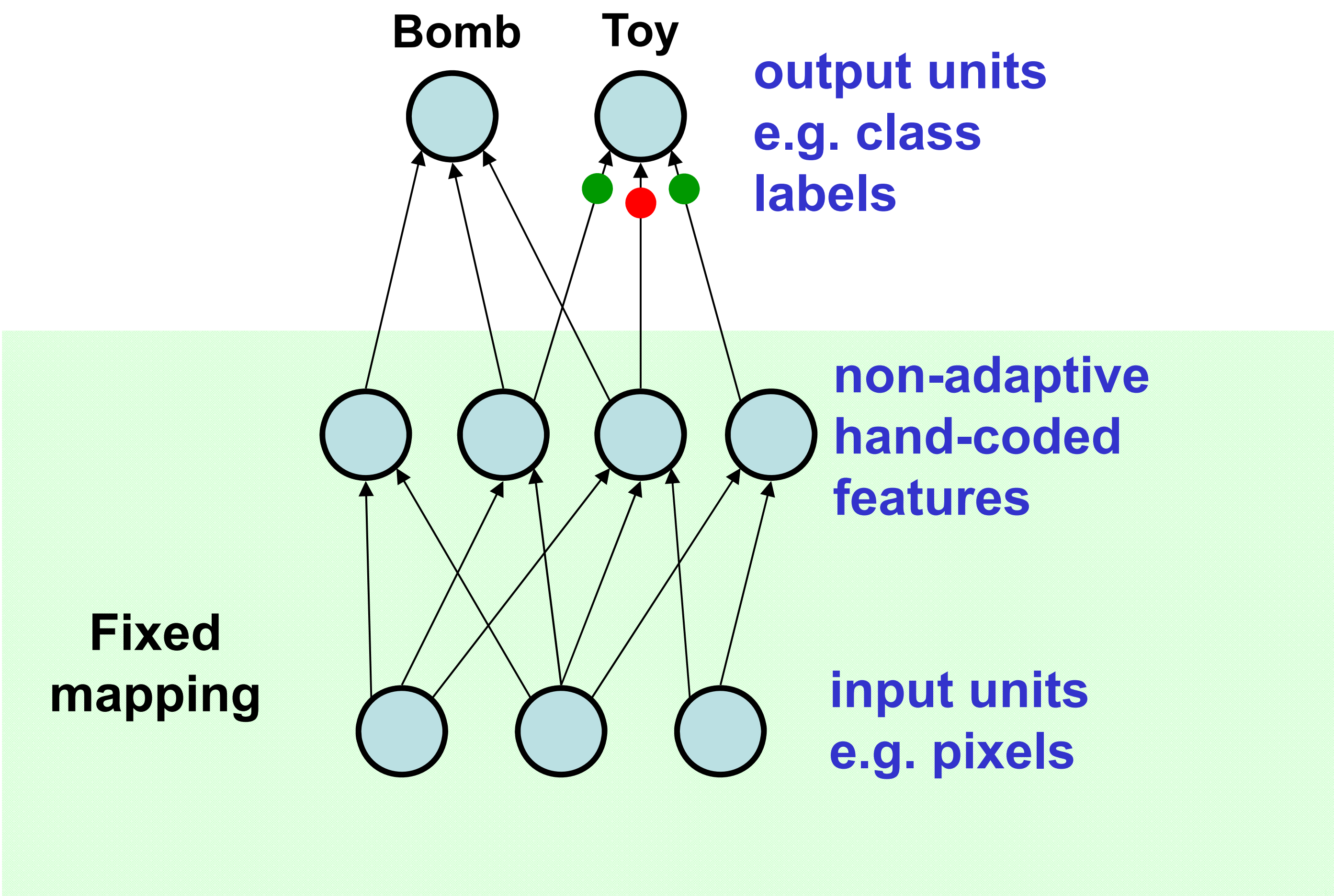
Beyond linear boundaries

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





Perceptron, ‘60s

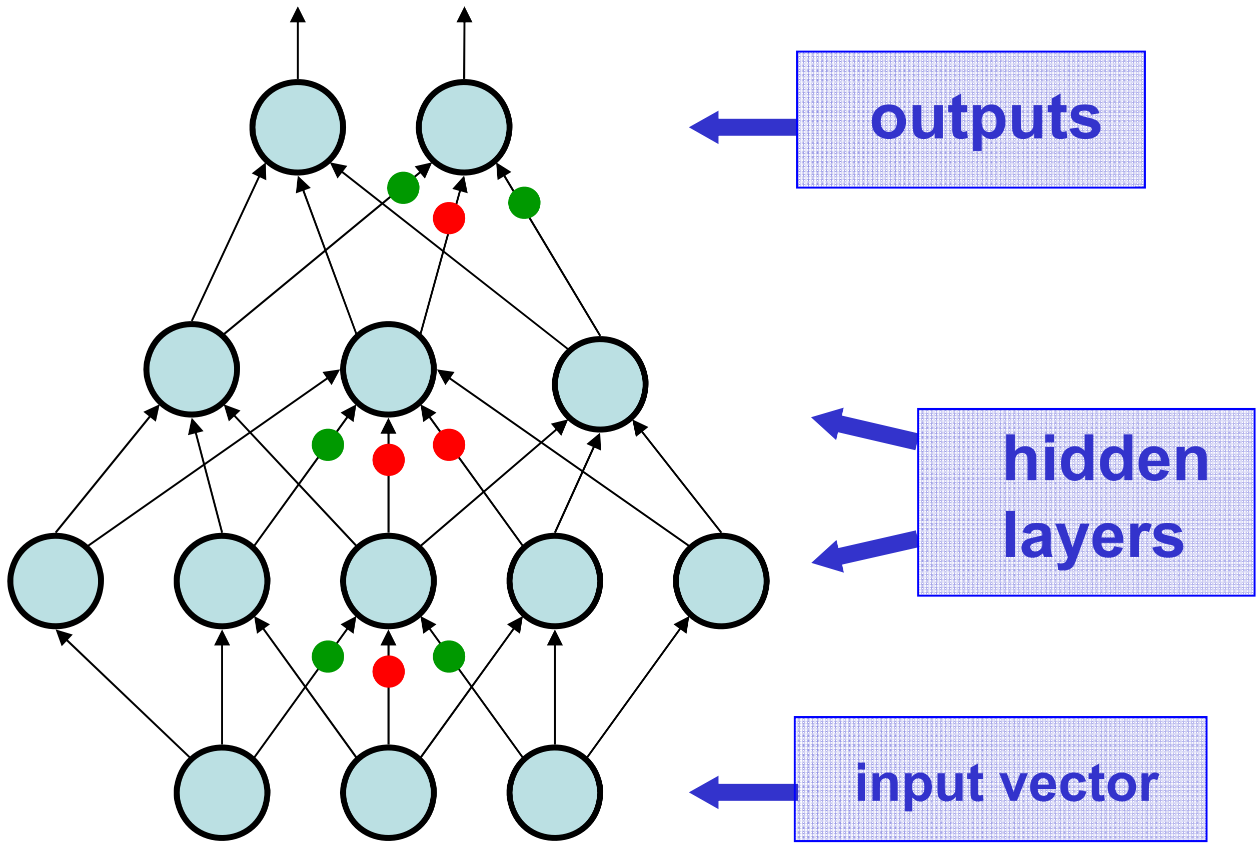


One of the first data-driven models in AI

Slide credits: G. Hinton

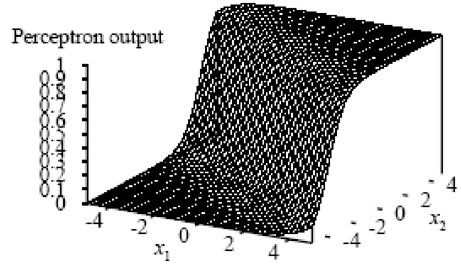
Multi-Layer Perceptrons (~1985)

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



Slide credits: G. Hinton

Expressiveness of perceptrons

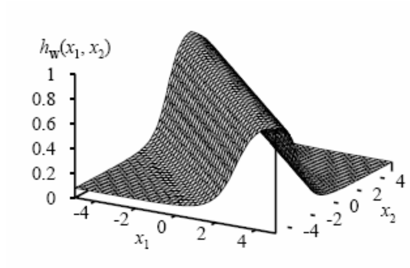


Single layer perceptron:
Linear classifier

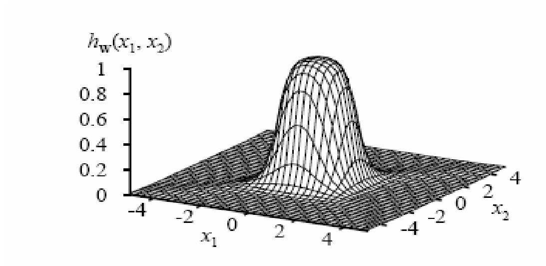
`soft threshold function'

(b)

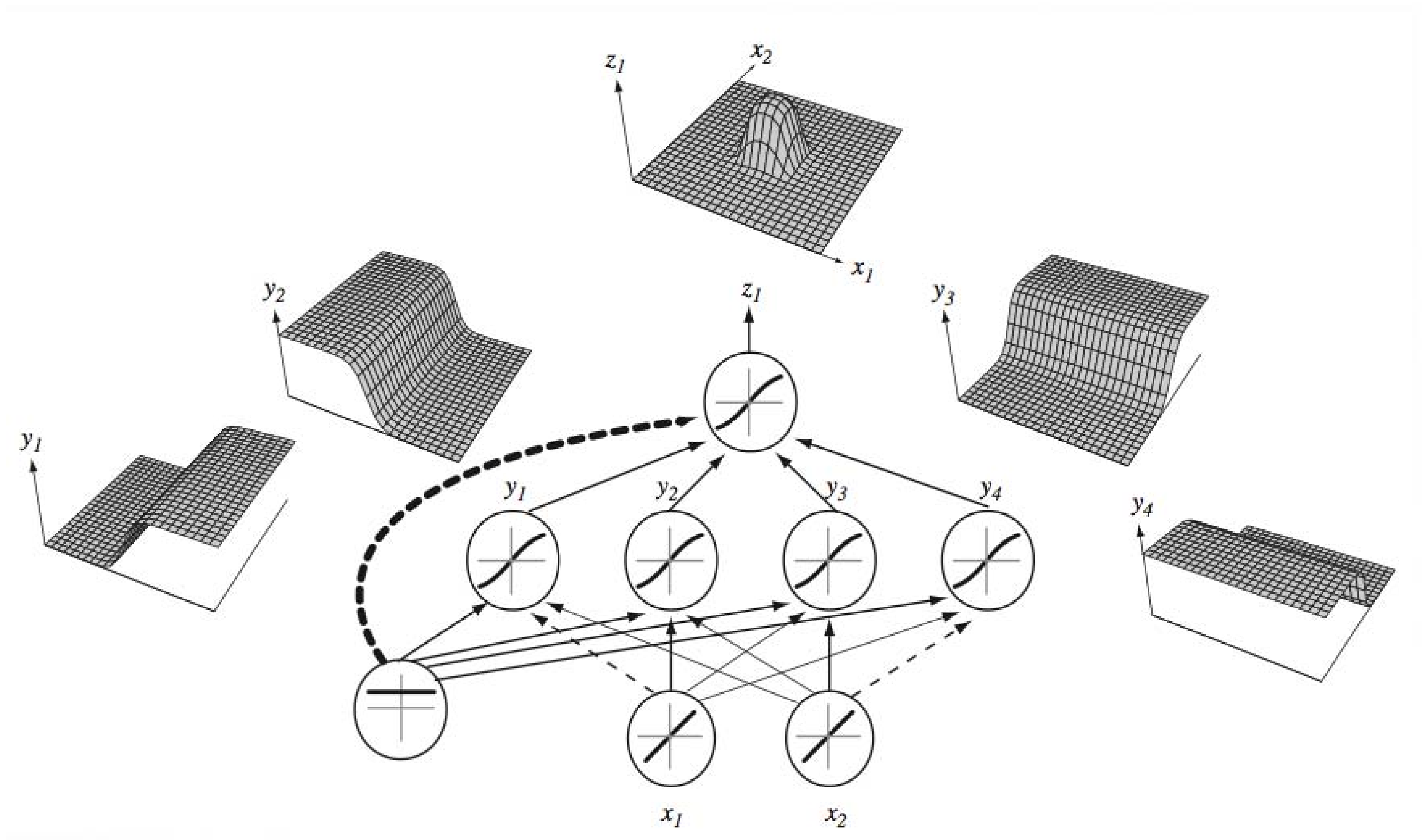
Two opposite `soft threshold' functions: a ridge



Two ridges: a bump

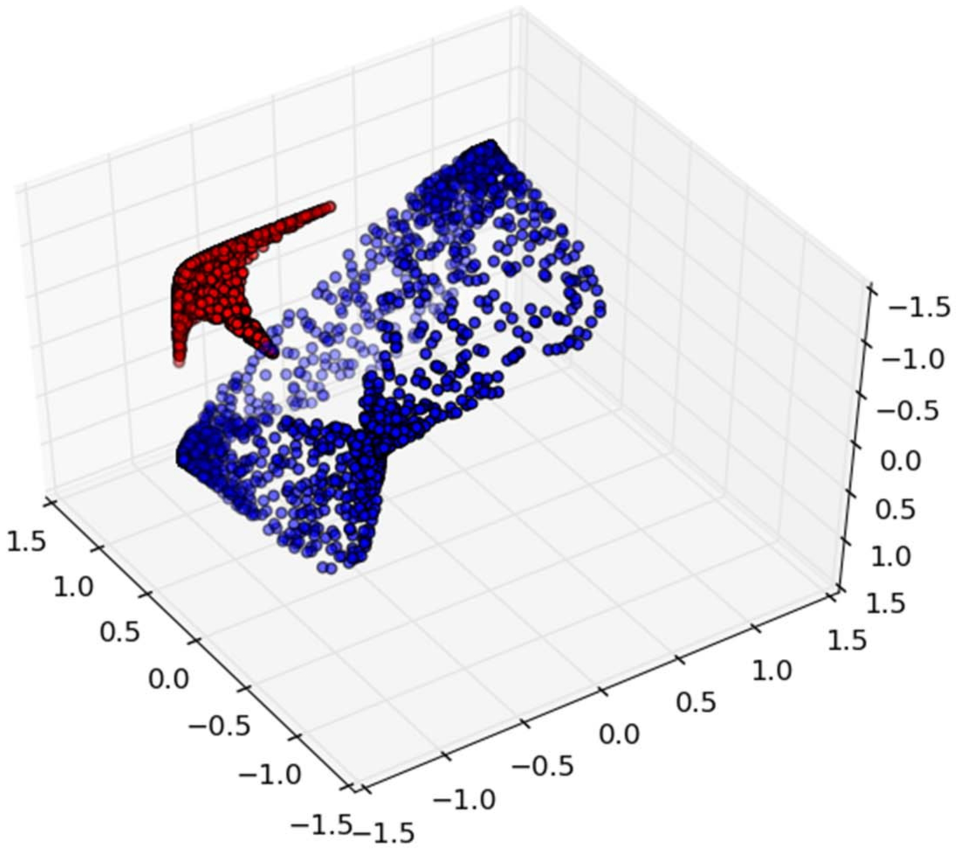


A network for a single bump



Any function: sum of bumps

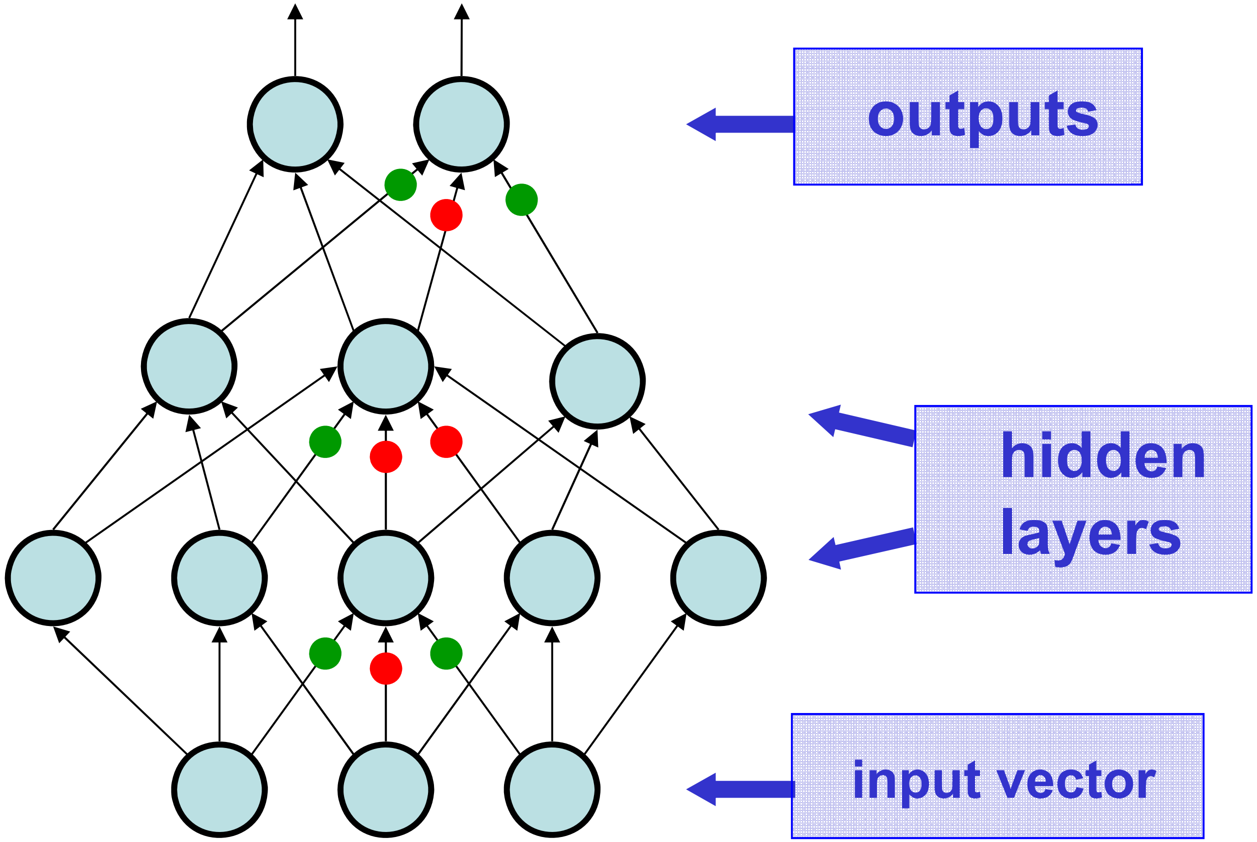
Linearization: may need higher dimensions



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

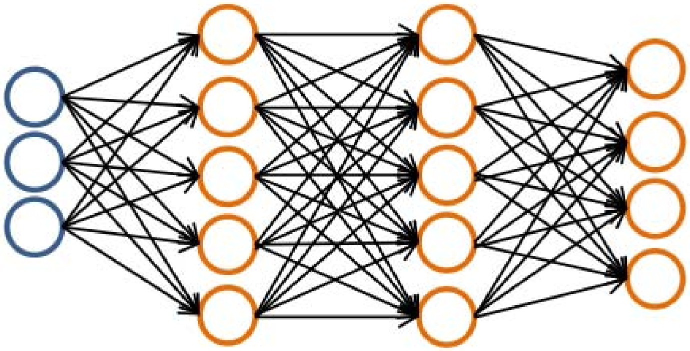
Multiple output units: One-vs-all.

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

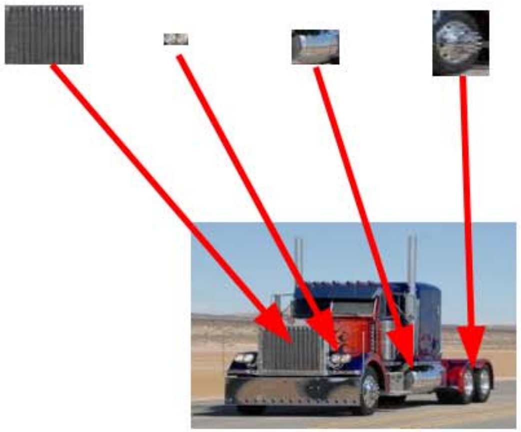
Slide credits: A. Ng

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

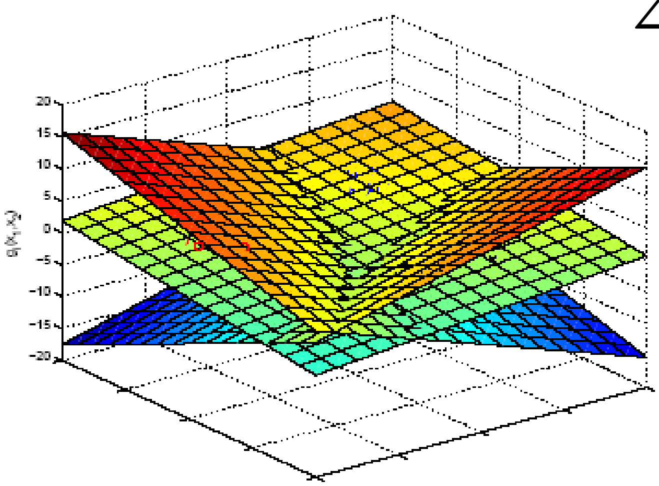


Ranzato

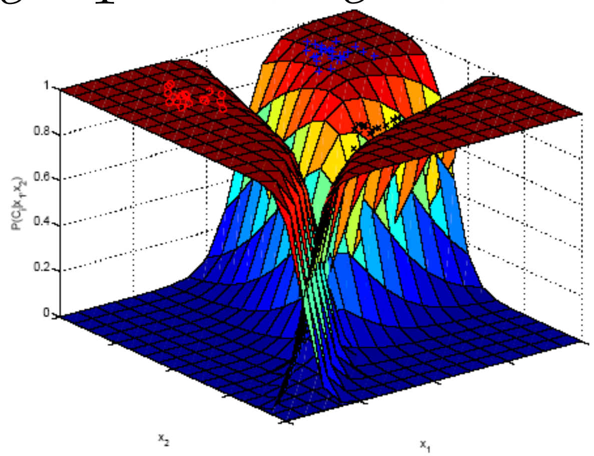
Reminder (W4): multiple classes & logistic regression

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: $(\mathbf{y}^i, \mathbf{x}^i)$

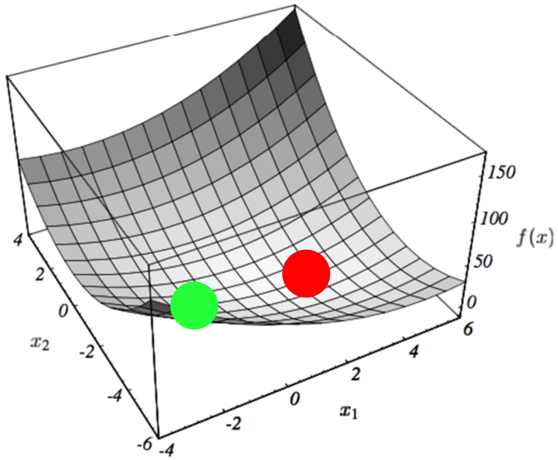
$$P(\mathbf{y}^i | \mathbf{x}^i; \mathbf{w}) = \prod_{i=1}^{N} \prod_{c=1}^{C} (g_c(\mathbf{x}, \mathbf{W}))^{\mathbf{y}_c^i}$$

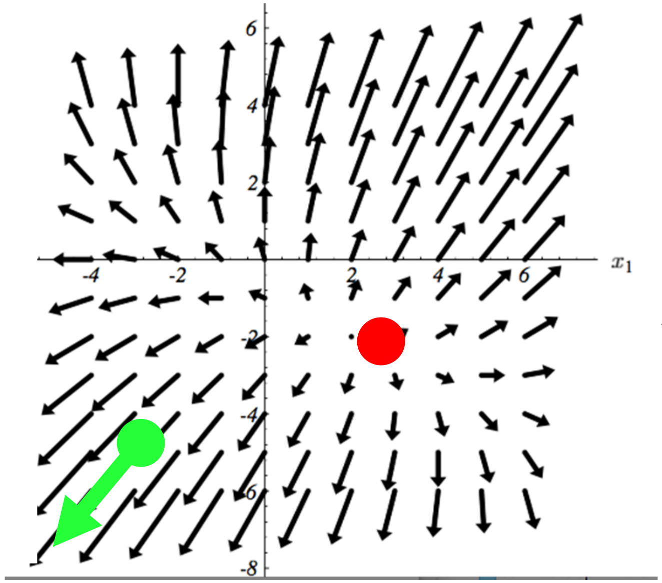
Optimization criterion:

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

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

Gradient-based minimization

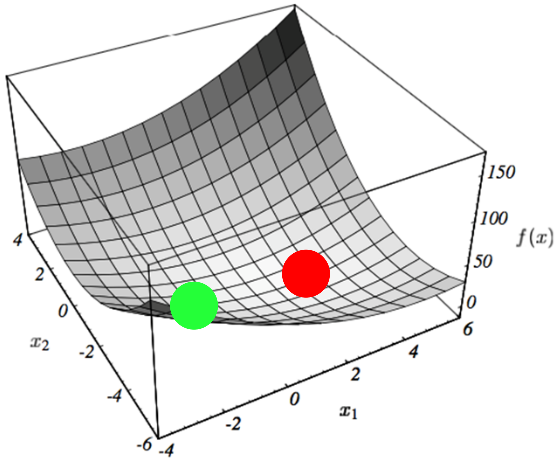


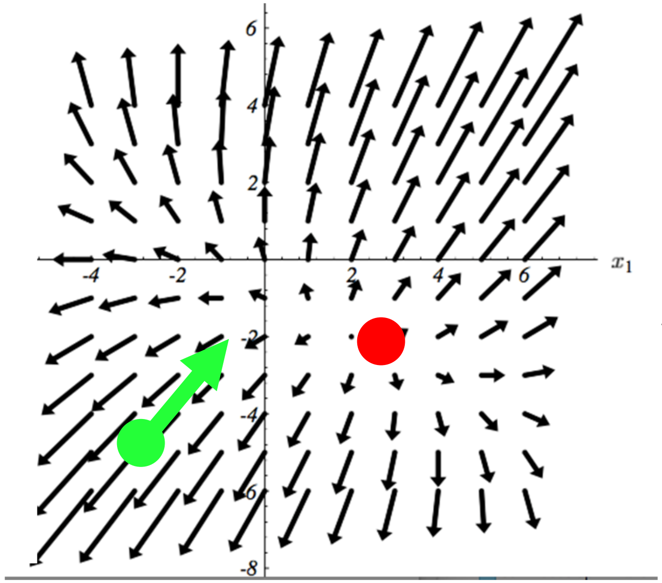


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Gradient-based minimization





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

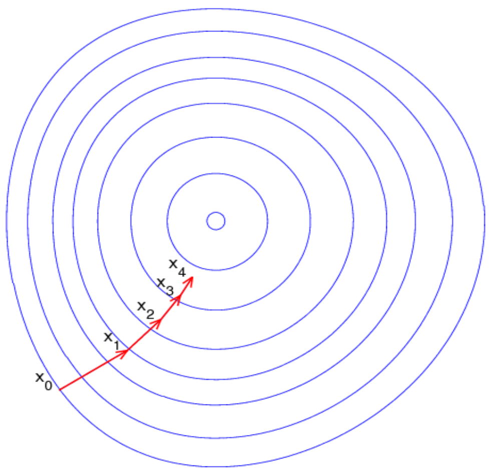
Fact: gradient at any point gives direction of fastest increase

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

Initialize: $\mathbf{x}_0$

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

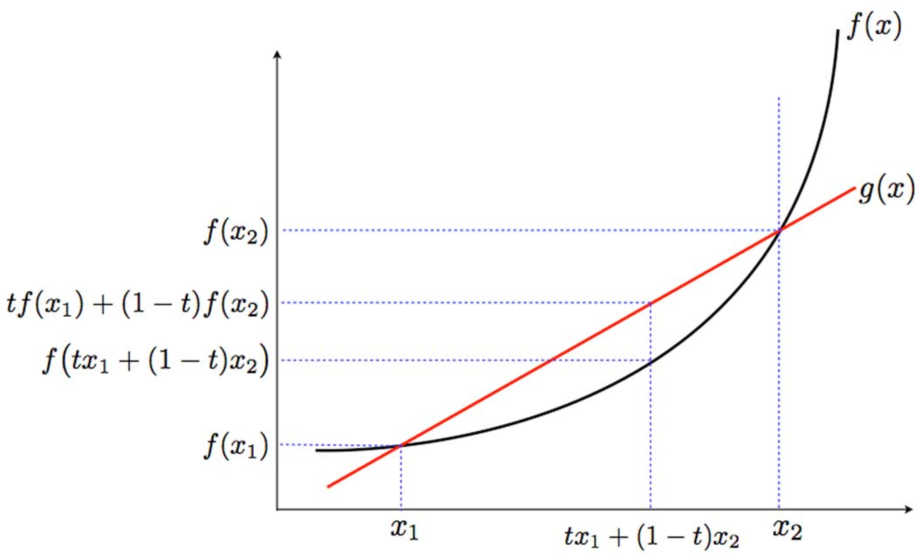
Gradient descent minimization method

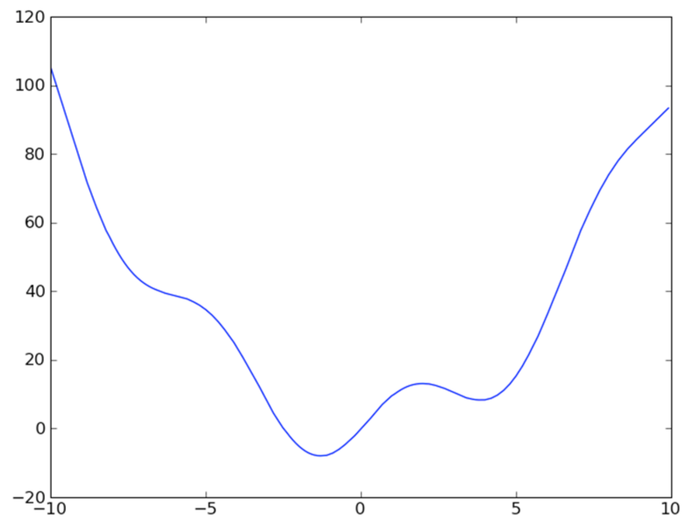


Initialize: $\mathbf{x}_0$

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

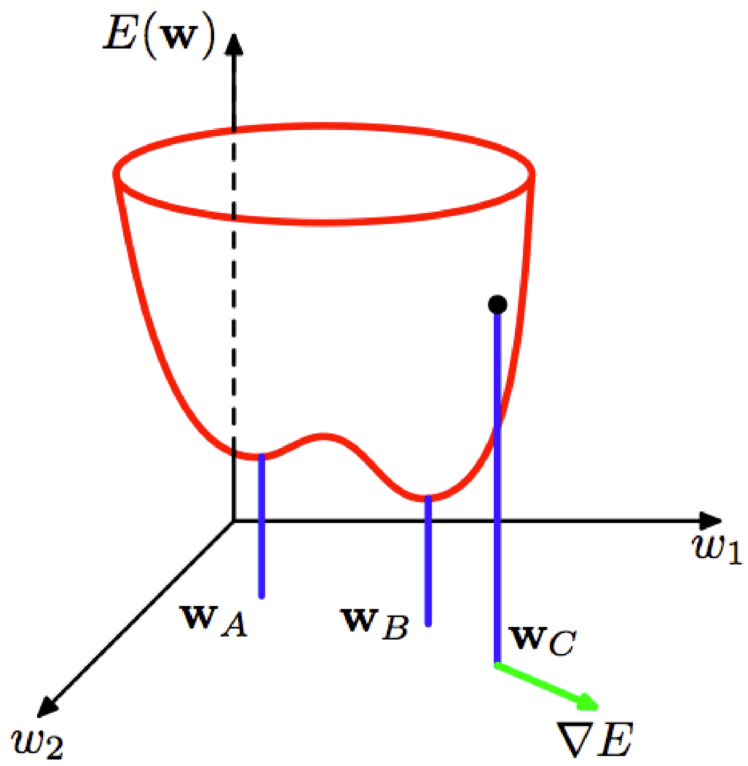
We can always make it converge for a **convex** function



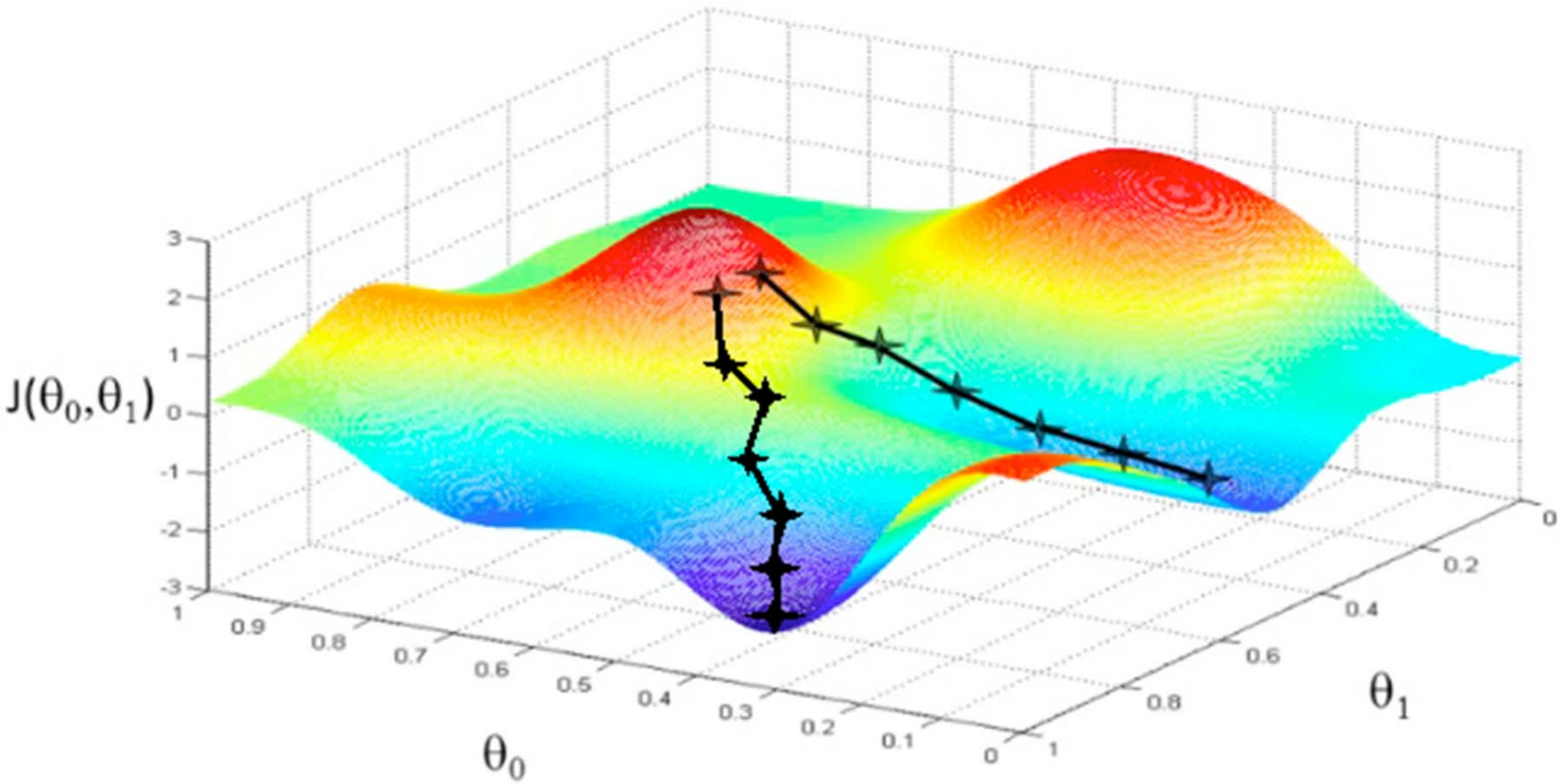


Problems: multiple local minima

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!

Back-propagation algorithm

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:

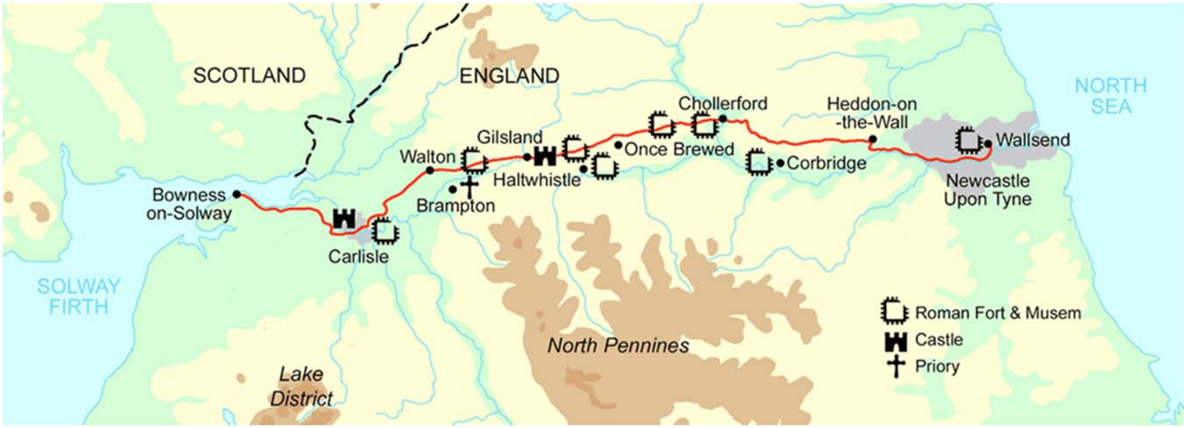
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Chain rule of differentiation– multiple variables

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

z=f(x,y) given by map

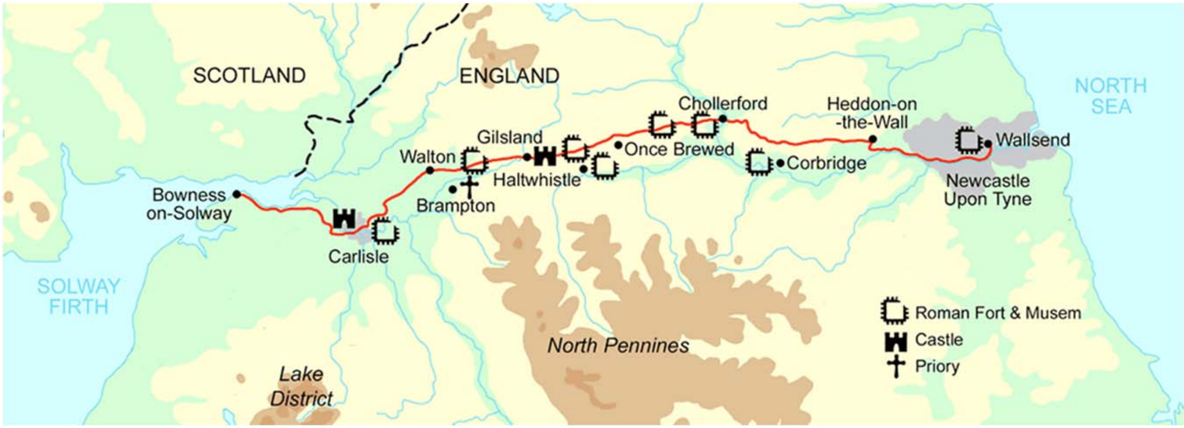
Q: what is your speed in the vertical direction?

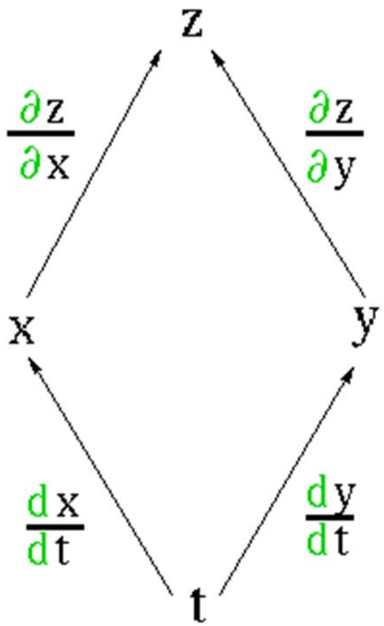


https://www.math.hmc.edu/calculus/tutorials/multichainrule/

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

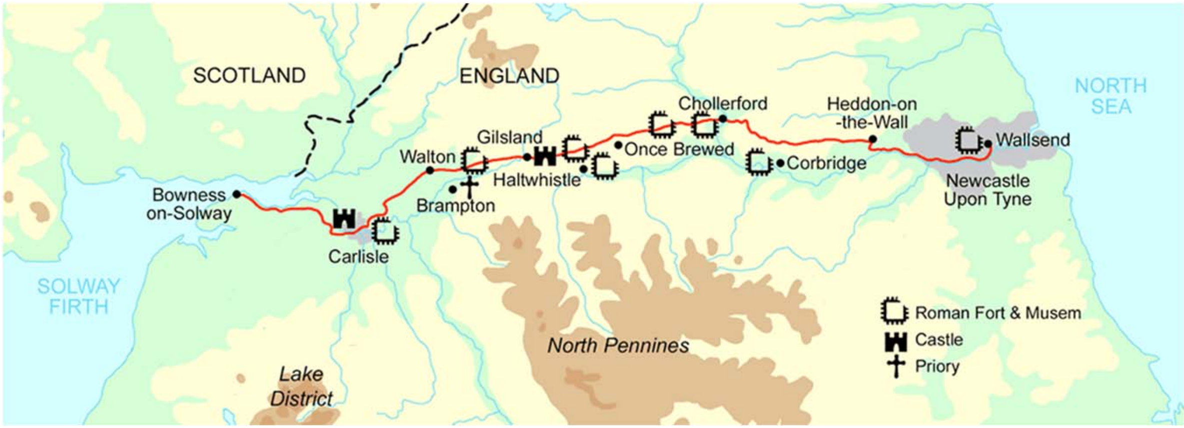
Let $x = x(t)$ and $y = y(t)$ be differentiable at t and suppose that $z = f(x, y)$ is differentiable at $(x(t), y(t))$. Then $z = f(x(t), y(t))$ is differentiable at t and

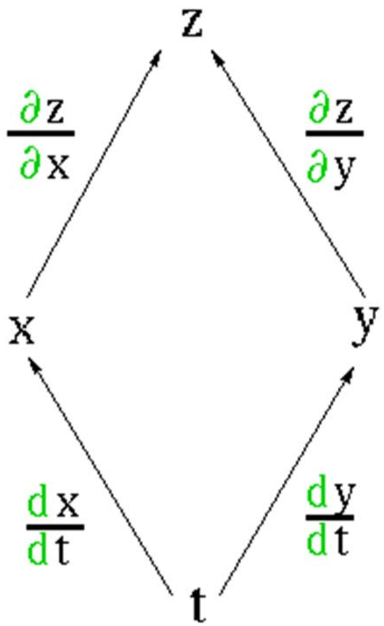
$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}.$$

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

z=f(x,y) given by map

Q: what is your speed in the vertical direction?



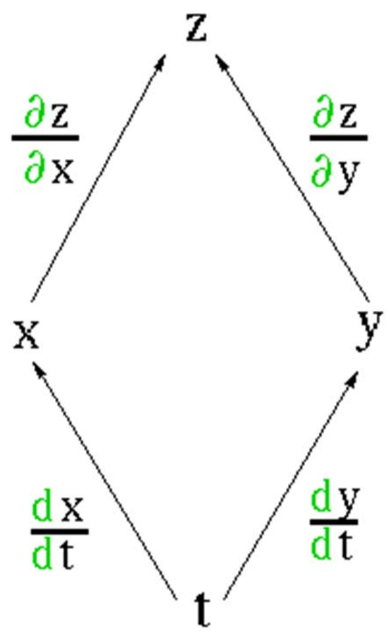


https://www.math.hmc.edu/calculus/tutorials/multichainrule/

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

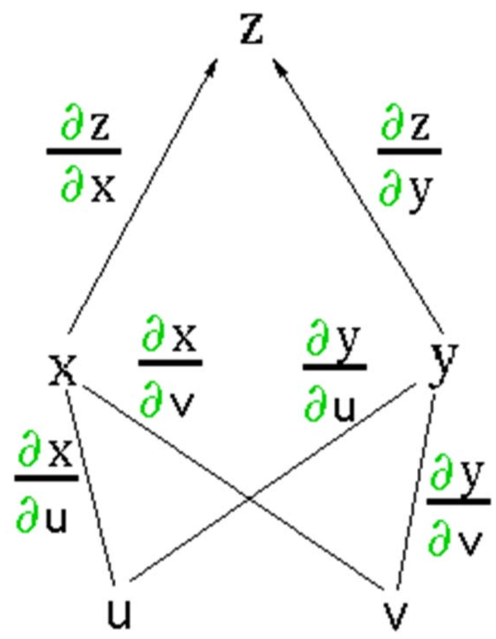
$$\begin{aligned}
\frac{dz}{dt} &= \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \\
&= (2xy)(2t) + (x^2 - 2y)(2) \\
&= (2t^2 \cdot 2t)(2t) + \left((t^2)^2 - 2(2t)\right)(2) \\
&= 8t^4 + 2t^4 - 8t \\
&= 10t^4 - 8t.
\end{aligned}$$

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

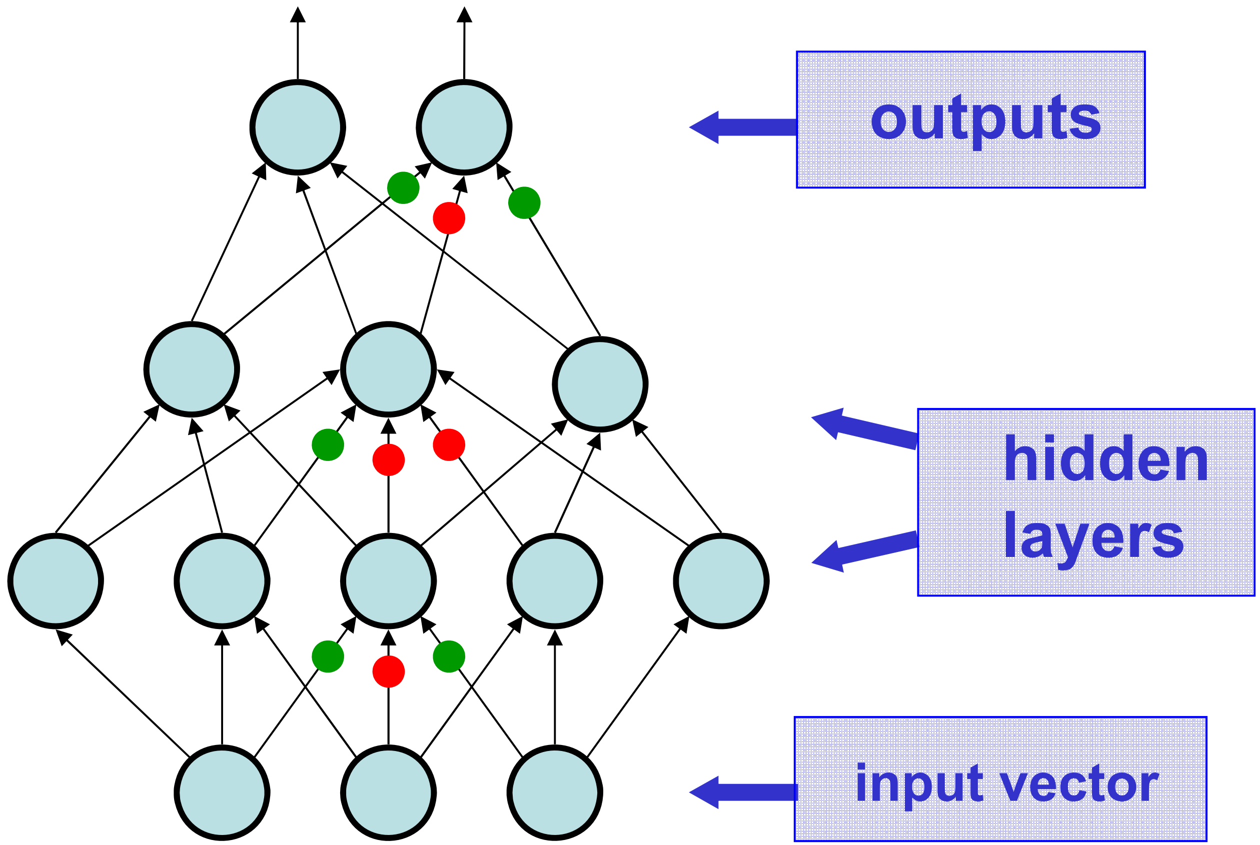
$$\begin{aligned}\frac{\partial z}{\partial u} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}\\ \frac{\partial z}{\partial v} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}.\end{aligned}$$



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

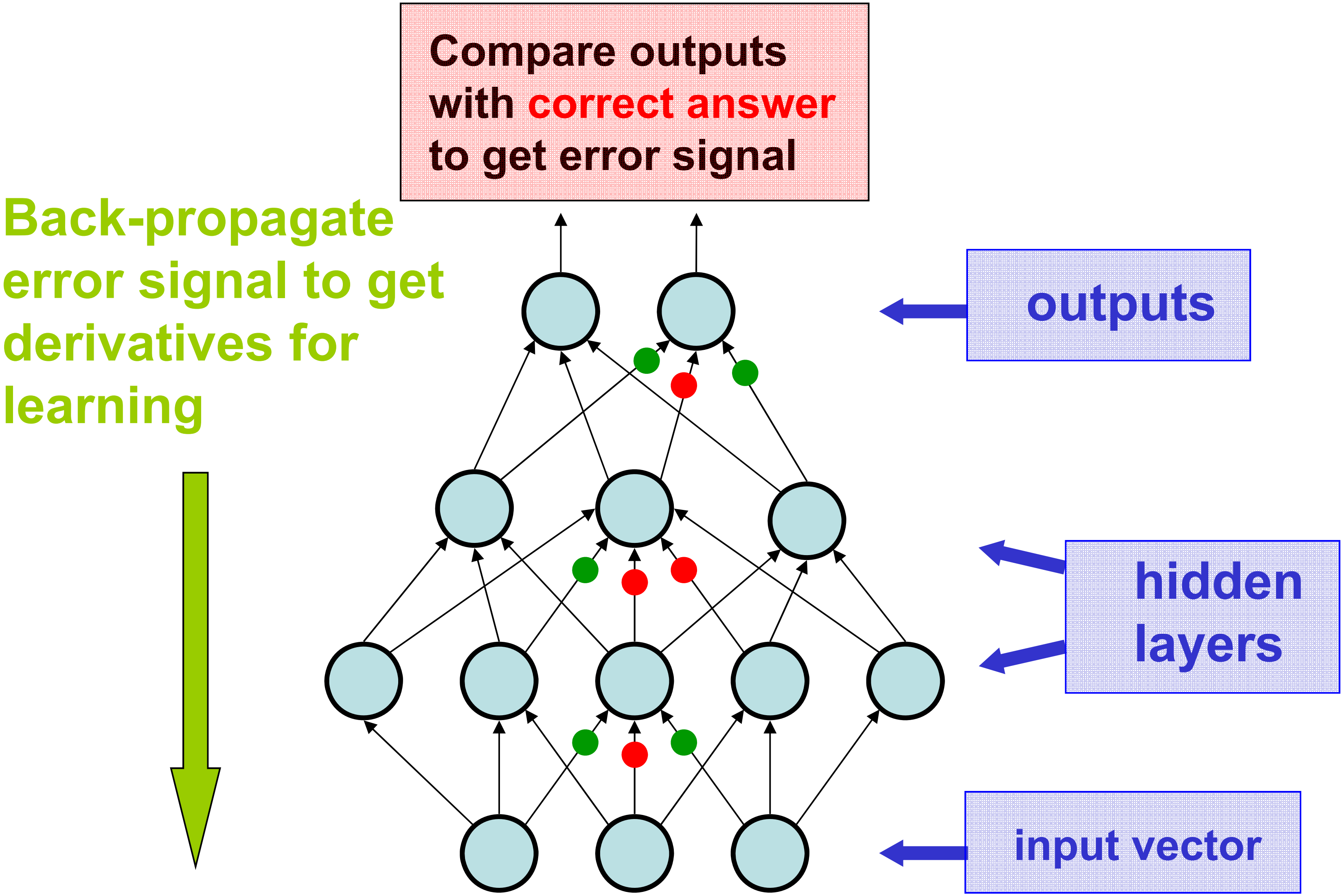
Multi-Layer Perceptrons

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



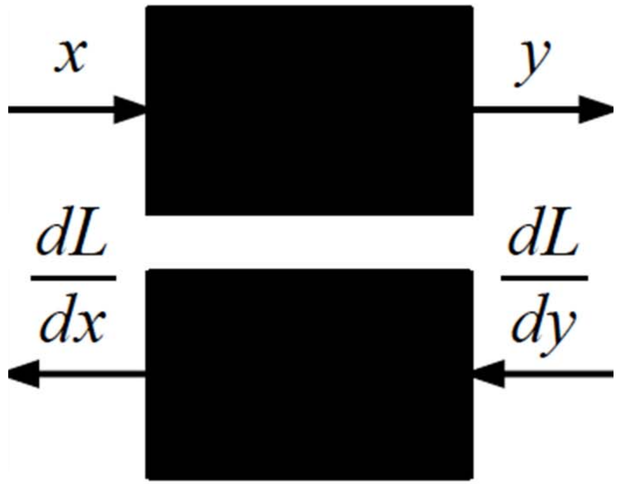
Slide credits: G. Hinton

Multi-Layer Perceptrons (~1985)



Slide credits: G. Hinton

Chain Rule

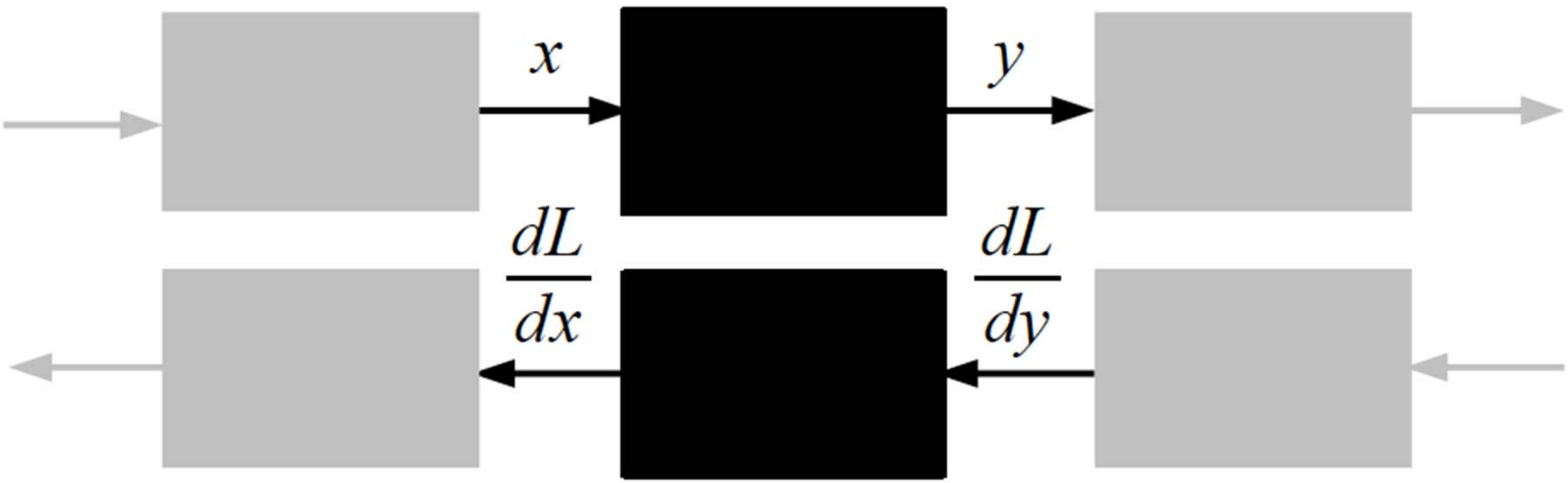


Given $y(x)$ and dL/dy,

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,

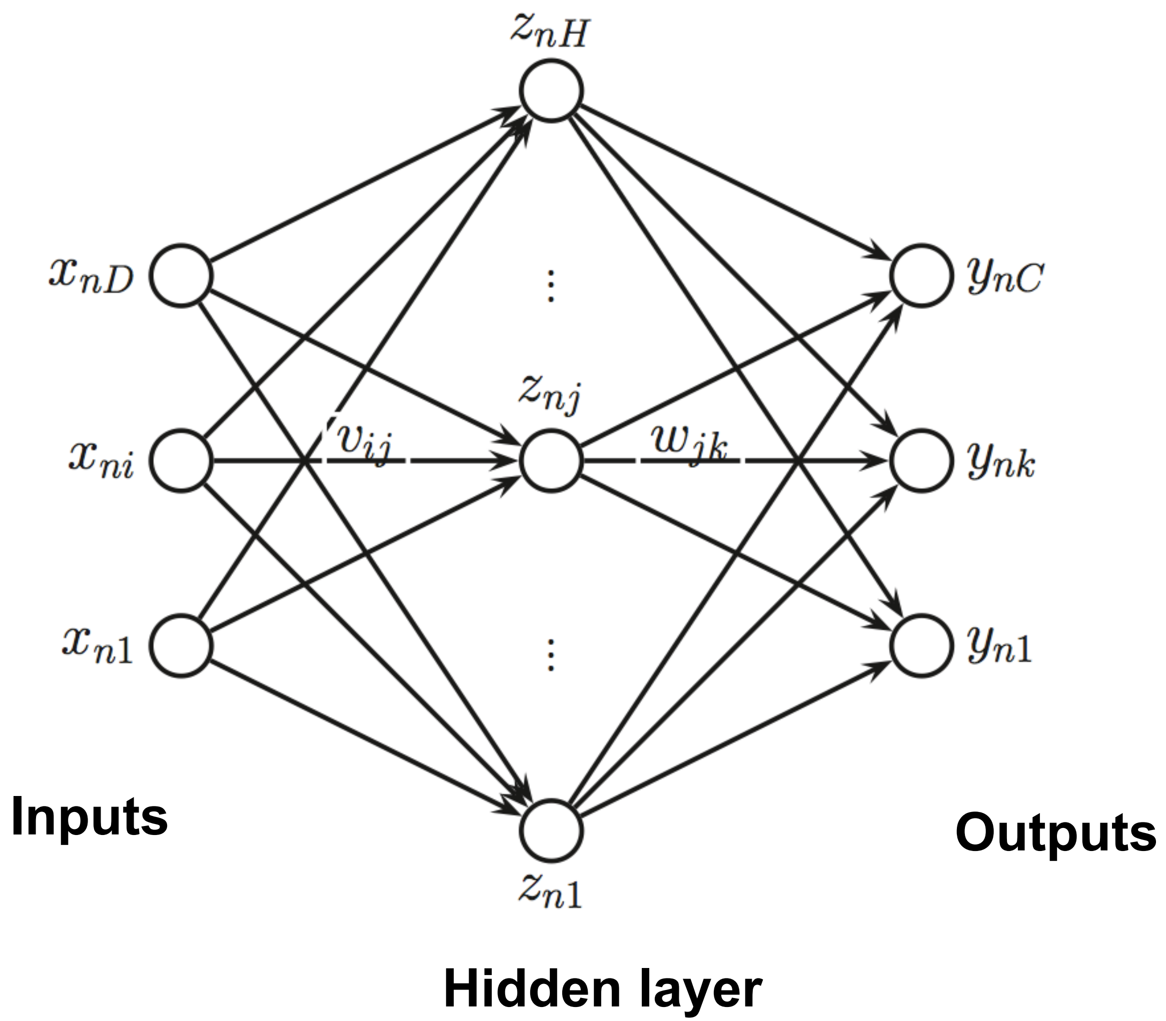
What is dL/dx ?

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

Slide Credit: Marc'Aurelio Ra

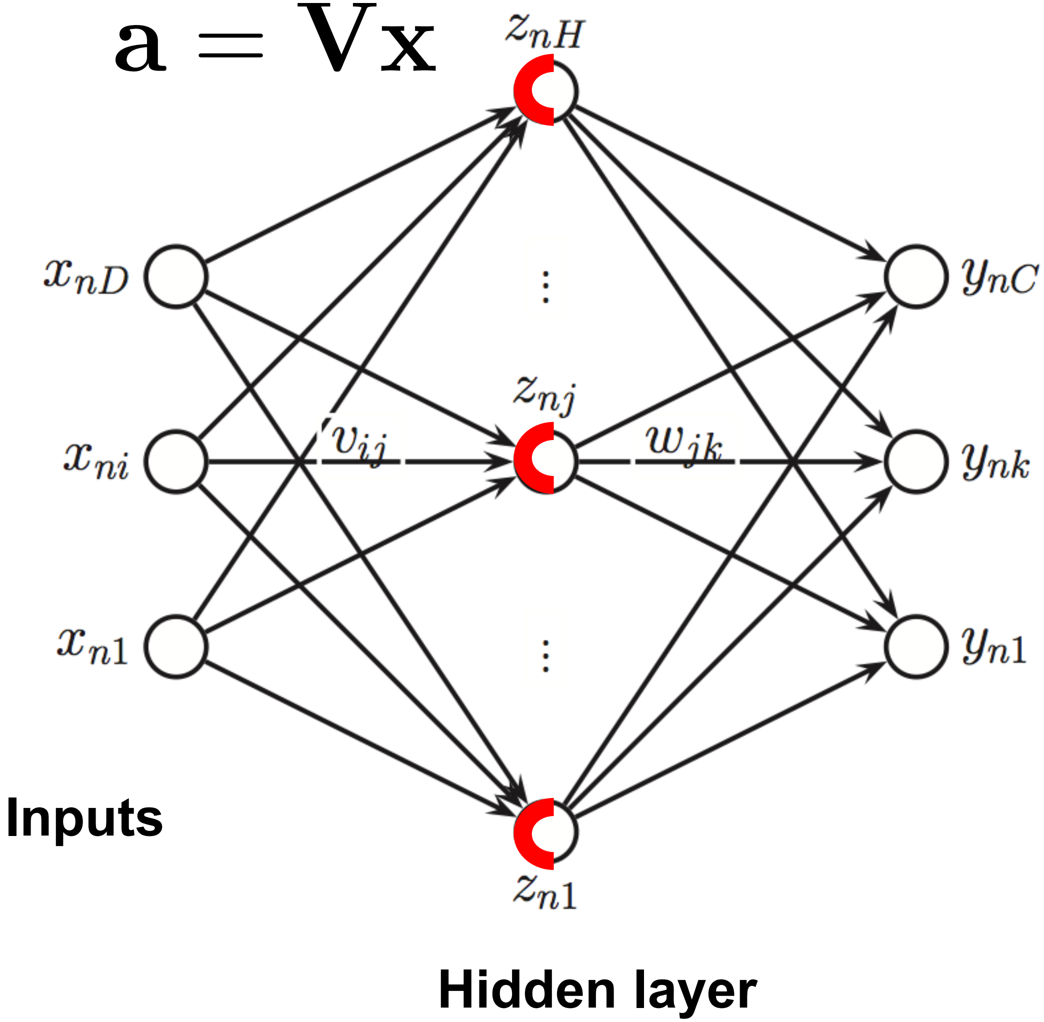
A neural network for multi-way classification

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



A neural network in forward mode:

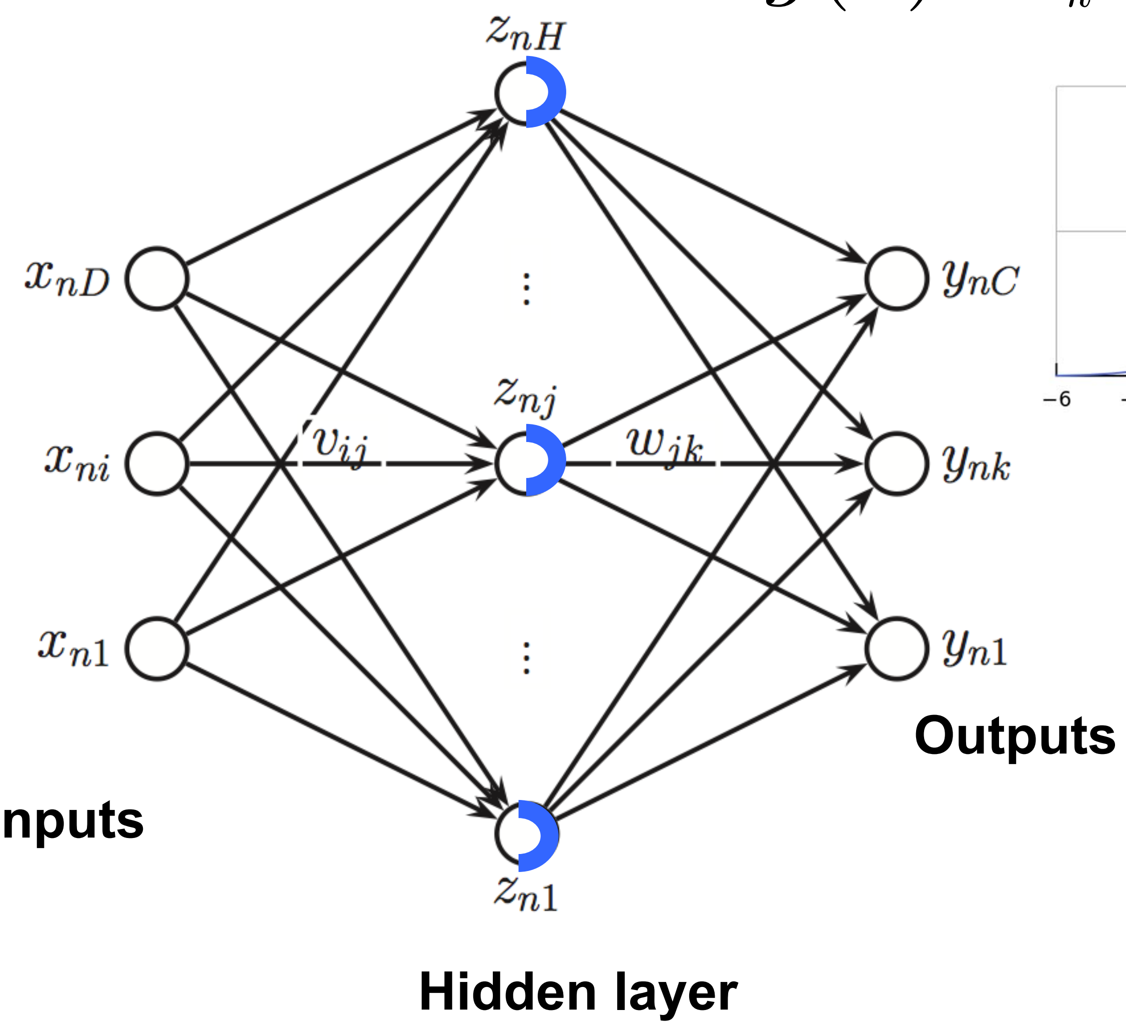
$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

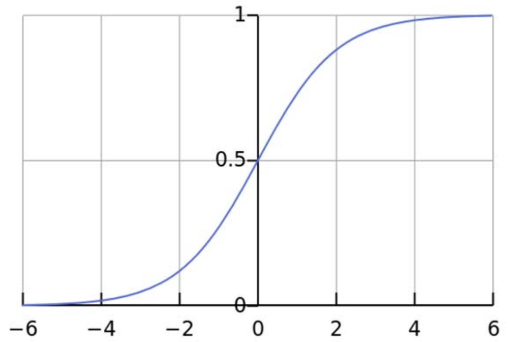


A neural network in forward mode:

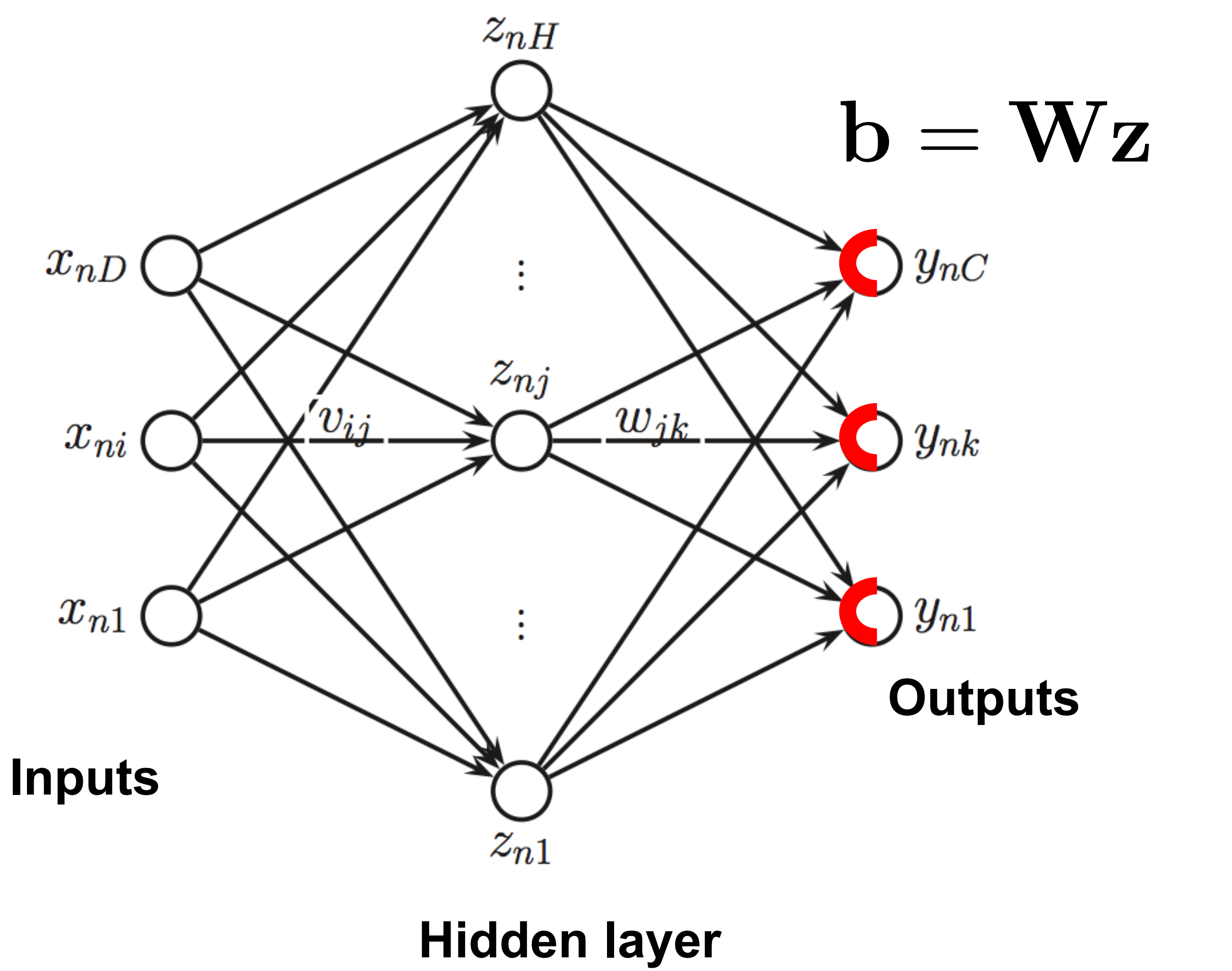
$$\mathbf{z} = g(\mathbf{a})$$

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





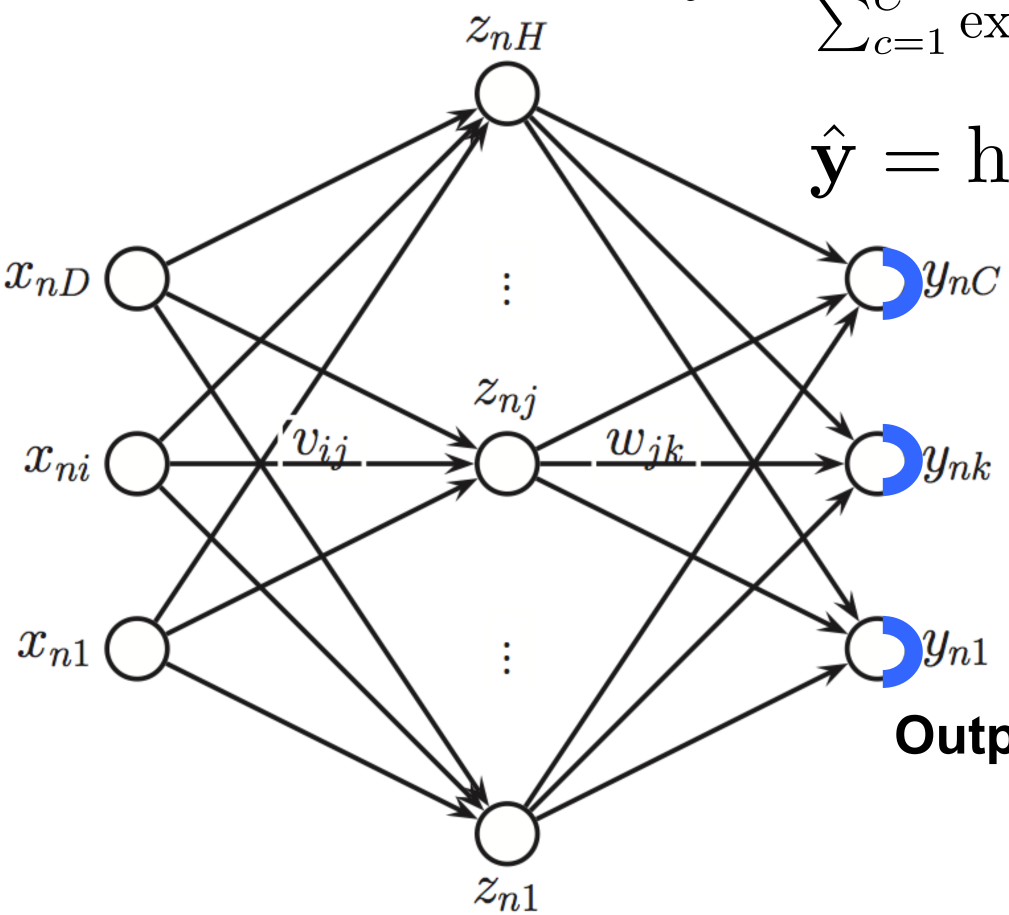
A neural network in forward mode:

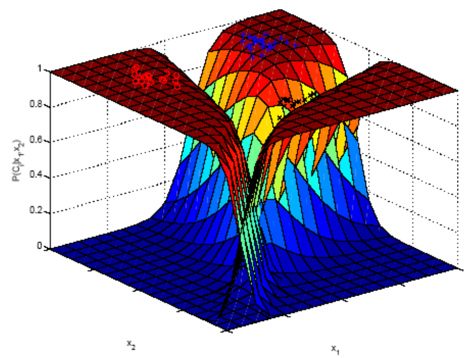


A neural network in forward mode:

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

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





Outputs

Hidden layer

Training objective, multi-class classification

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

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

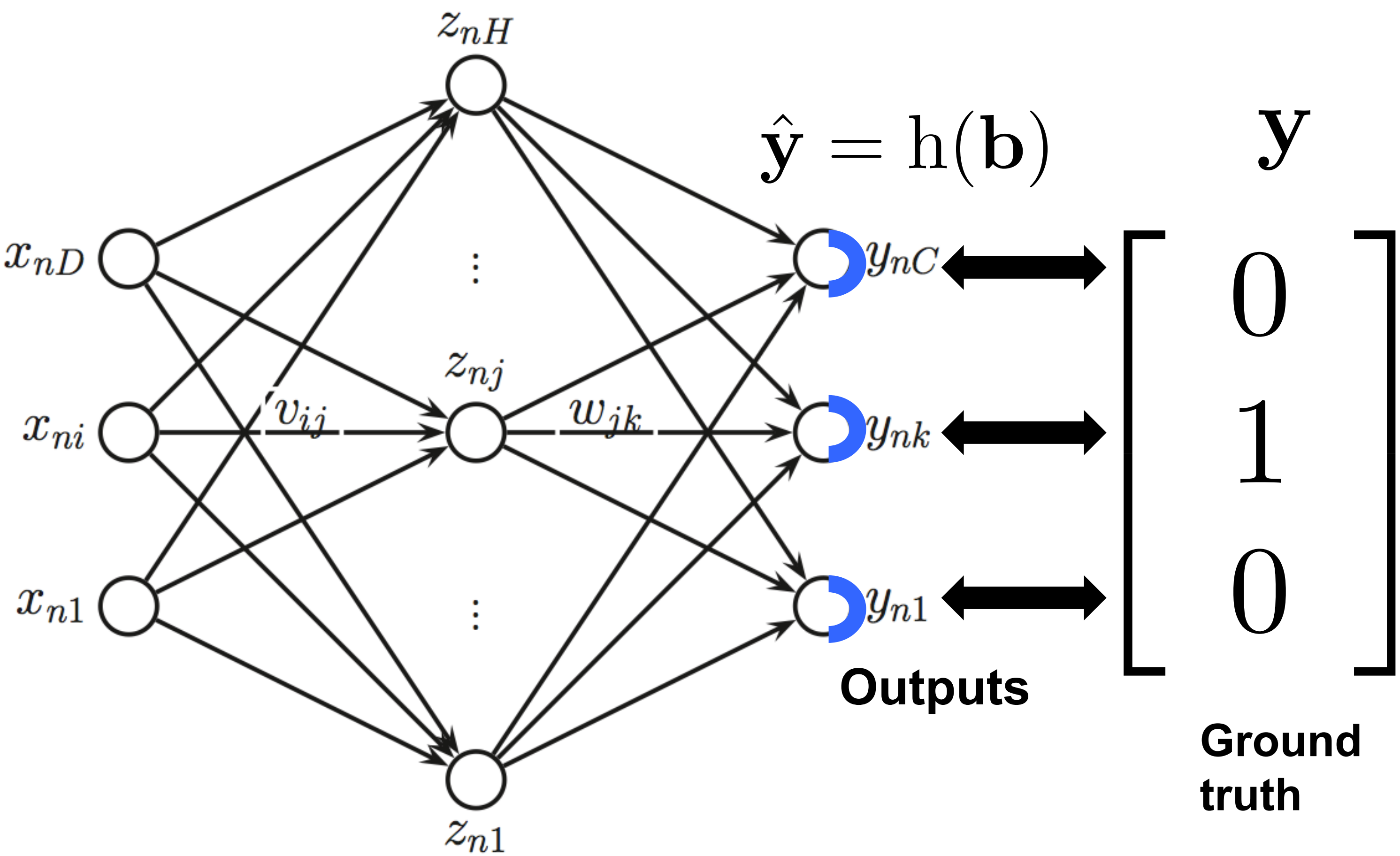
$$P(\mathbf{y}^i | \mathbf{x}^i; \mathbf{w}) = \prod_{c=1}^{C} (g_c(\mathbf{x}, \mathbf{W}))^{\mathbf{y}_c^i}$$

Optimization criterion:

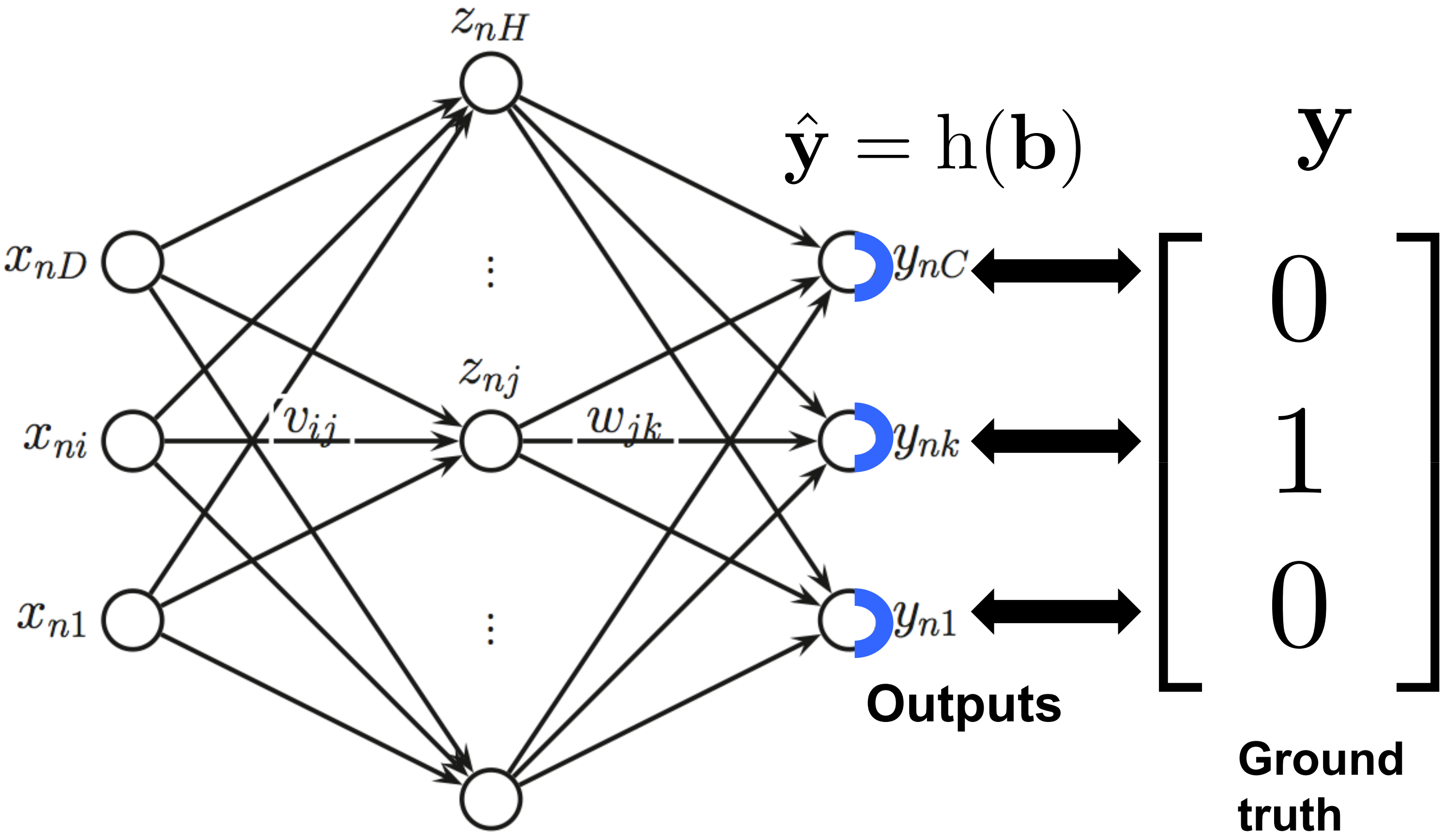
$$L(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} \mathbf{y}_c^i \log (g_c(\mathbf{x}, \mathbf{W}))$$

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

Objective for multi-class classification

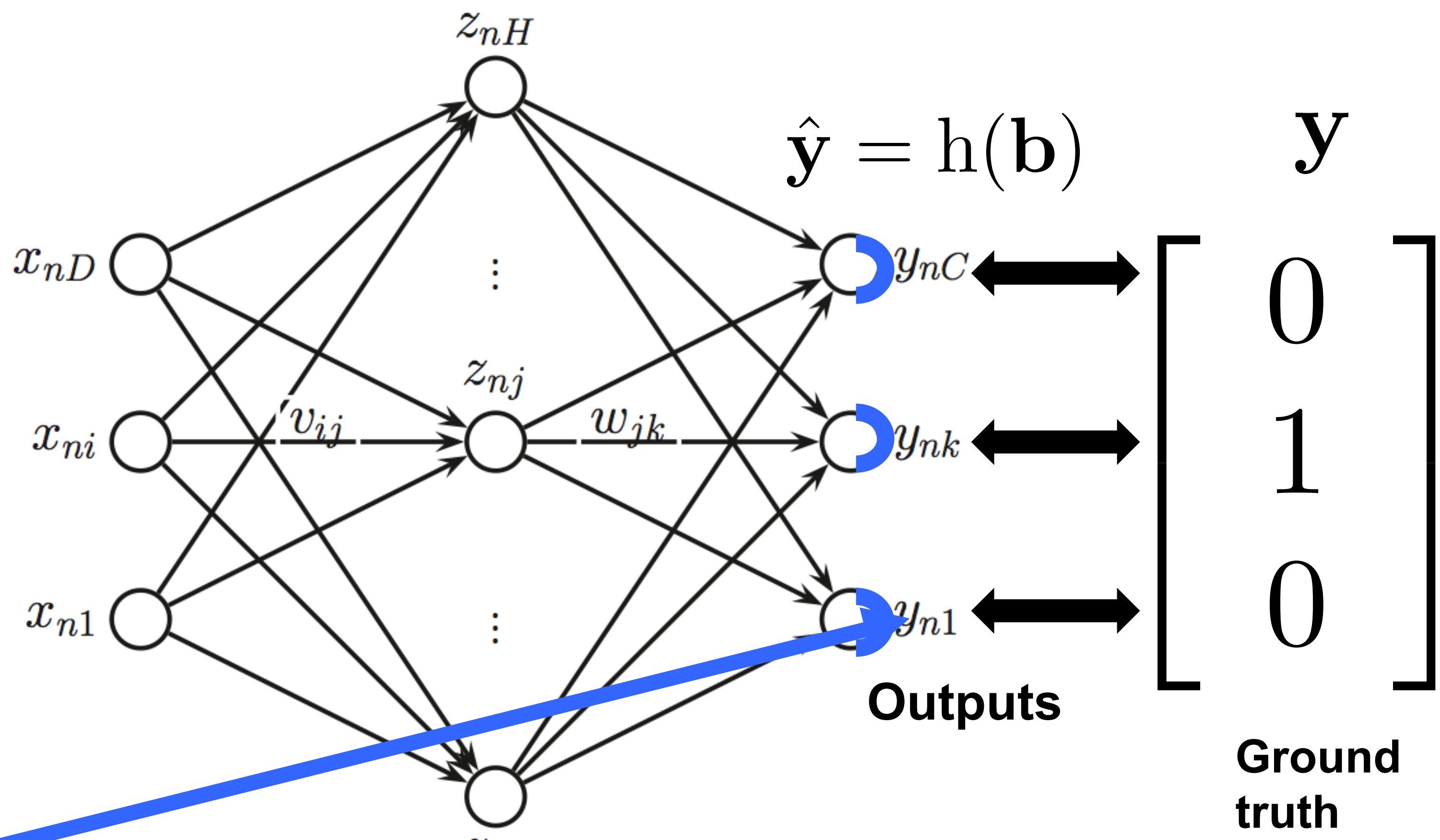


Objective for multi-class classification



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

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

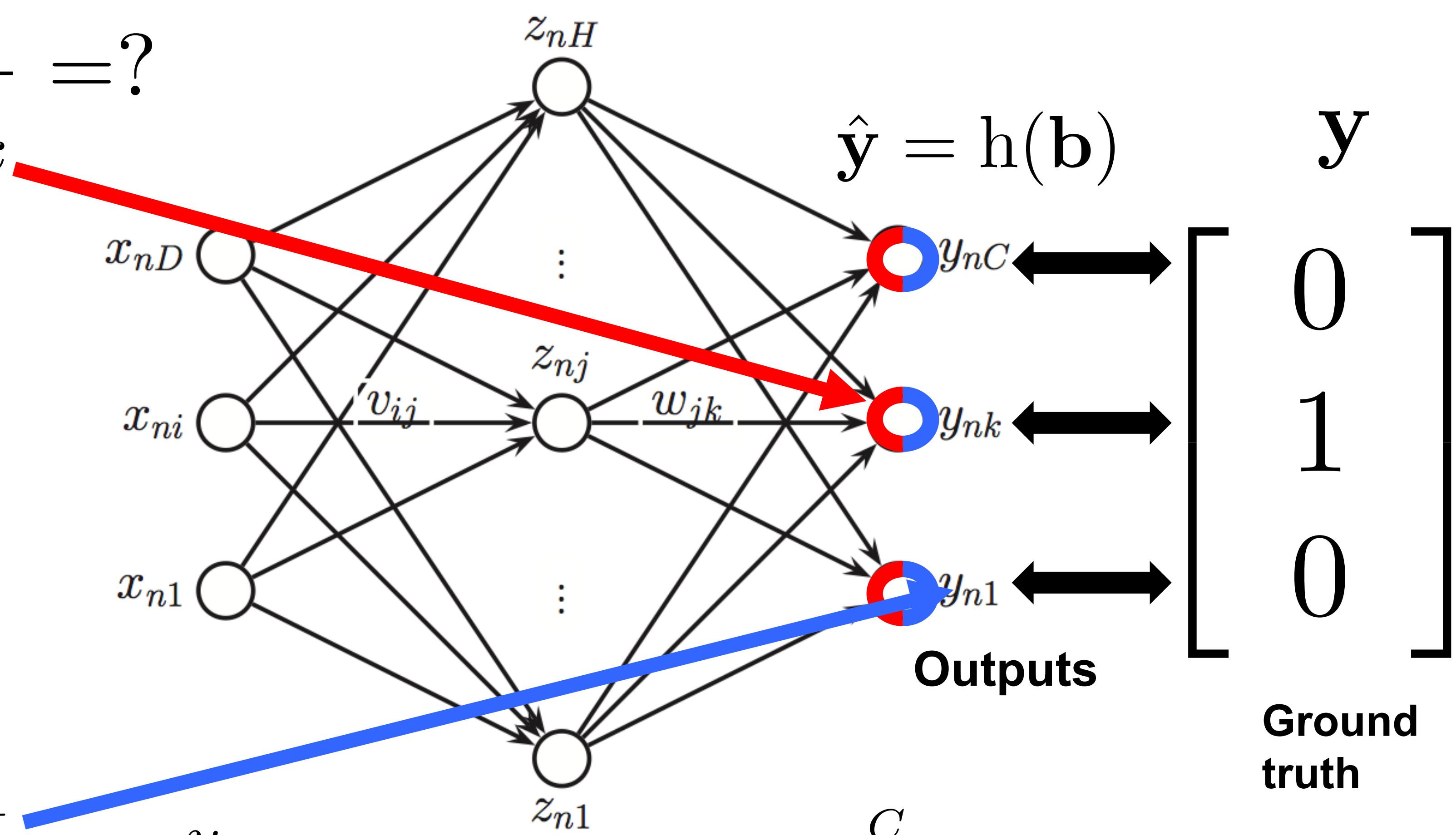


$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

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

A neural network in backward mode:

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

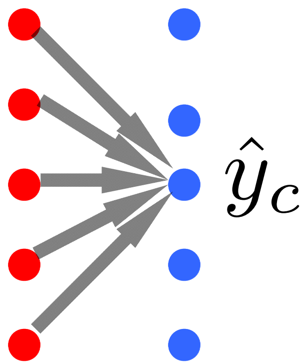


$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

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

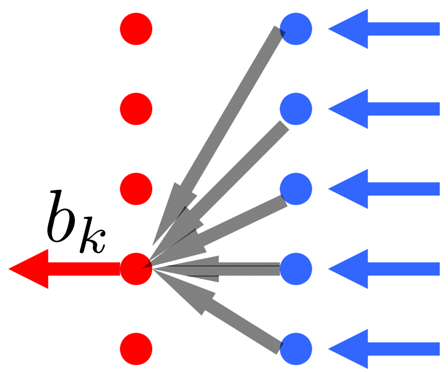
Softmax in forward mode: all for one

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



Softmax in backward mode: one from all

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



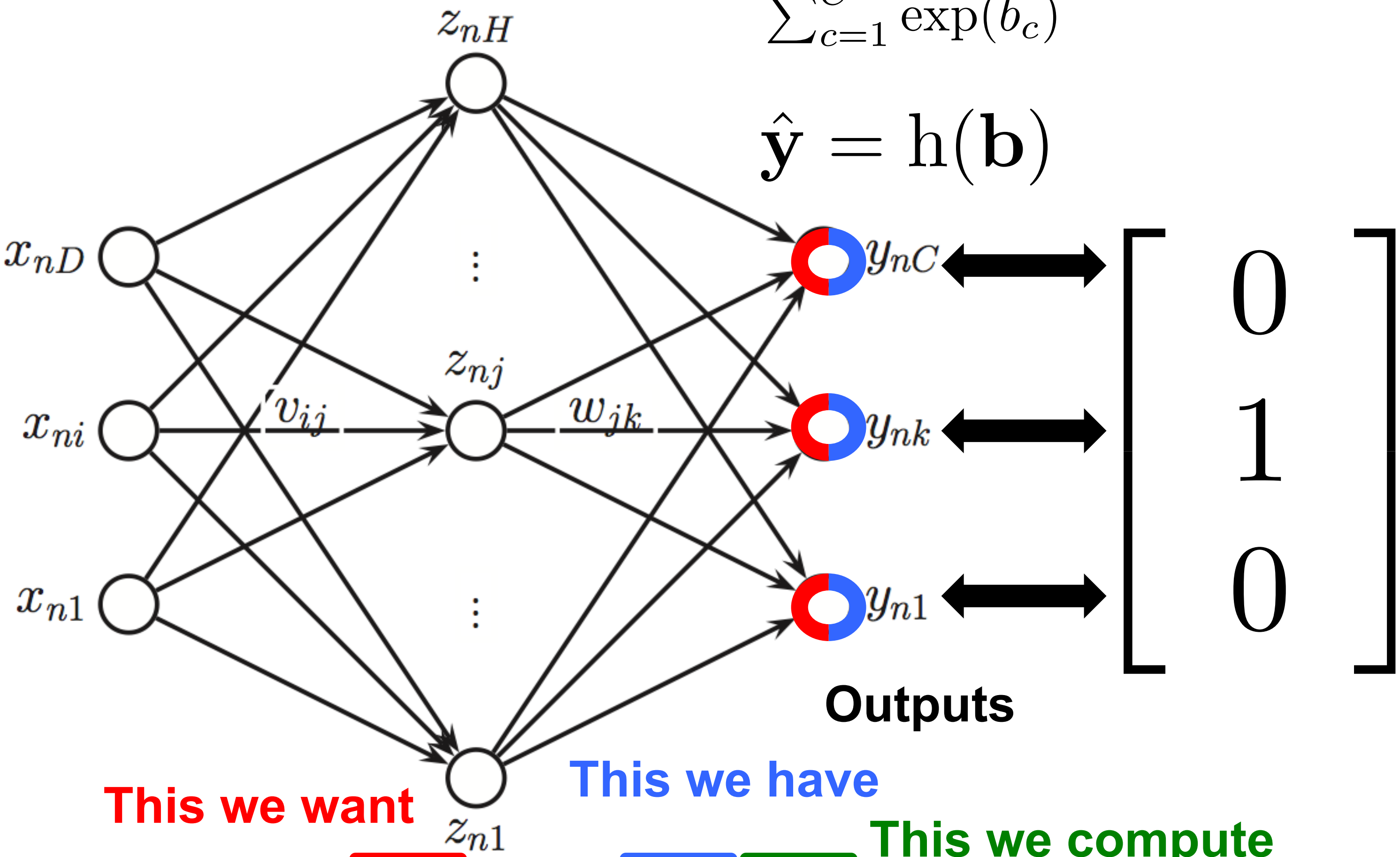
$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c}$$

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

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

This we have

This we compute

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

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

In backward mode?

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

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

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

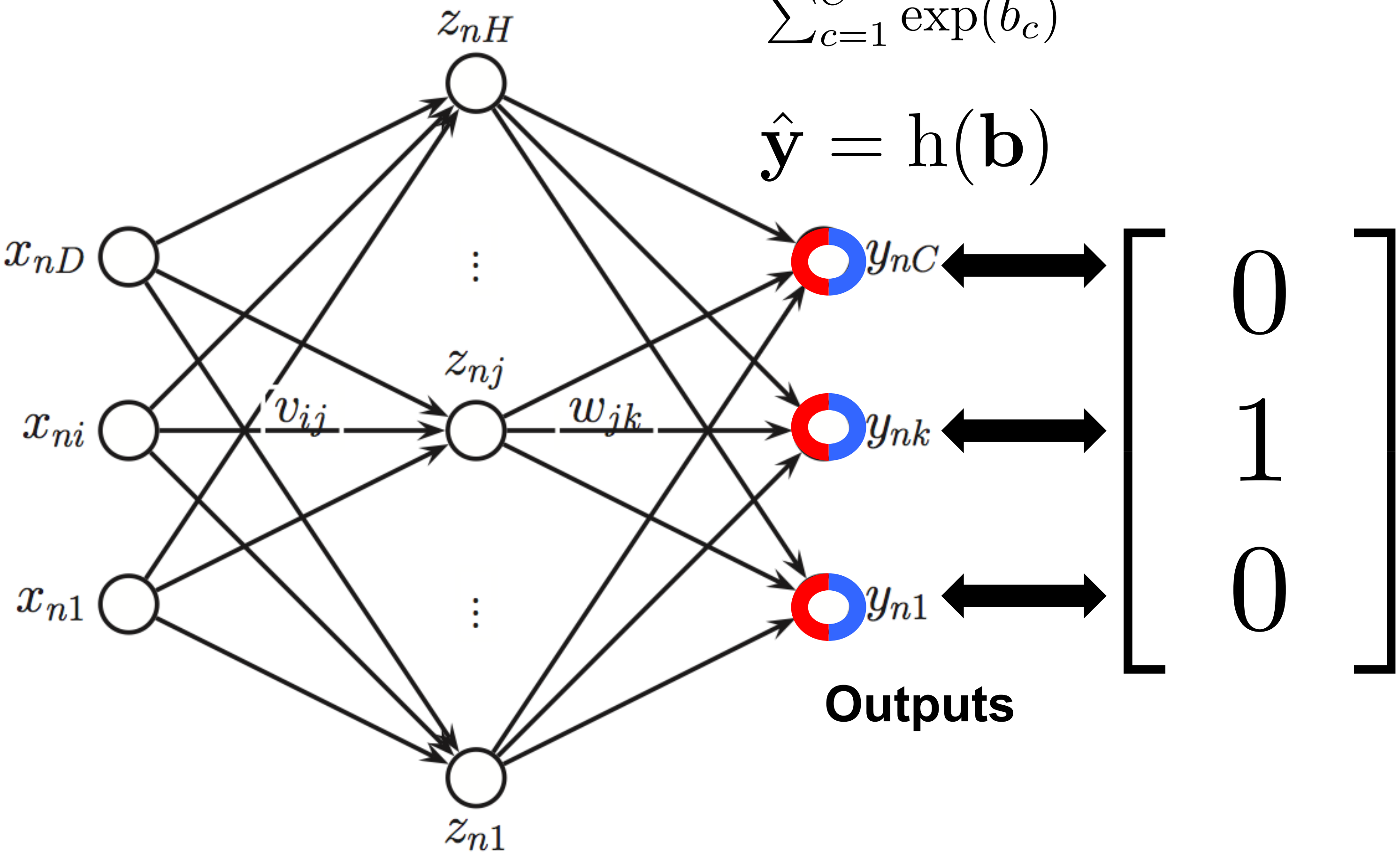
$$= [c = k]\hat{y}_k - \hat{y}_c \hat{y}_k \quad = ([c = k] - \hat{y}_c)\hat{y}_k$$

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

A neural network in backward mode:

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

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

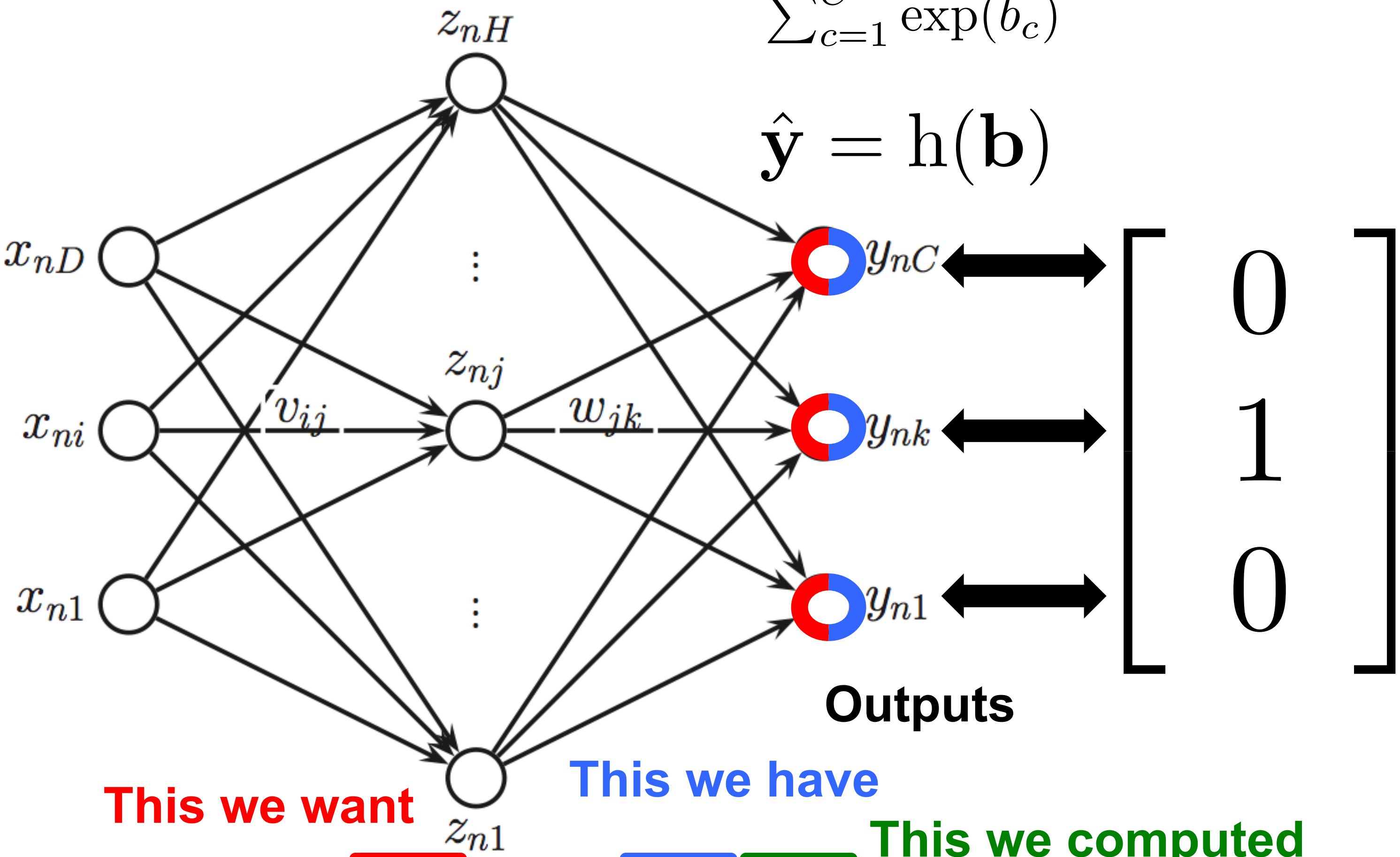


$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c} \qquad \frac{\partial L}{\partial b_k} = \sum_c \frac{\partial L}{\partial \hat{y}_c} \frac{\partial \hat{y}_c}{\partial b_k}$$

A neural network in backward mode:

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

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



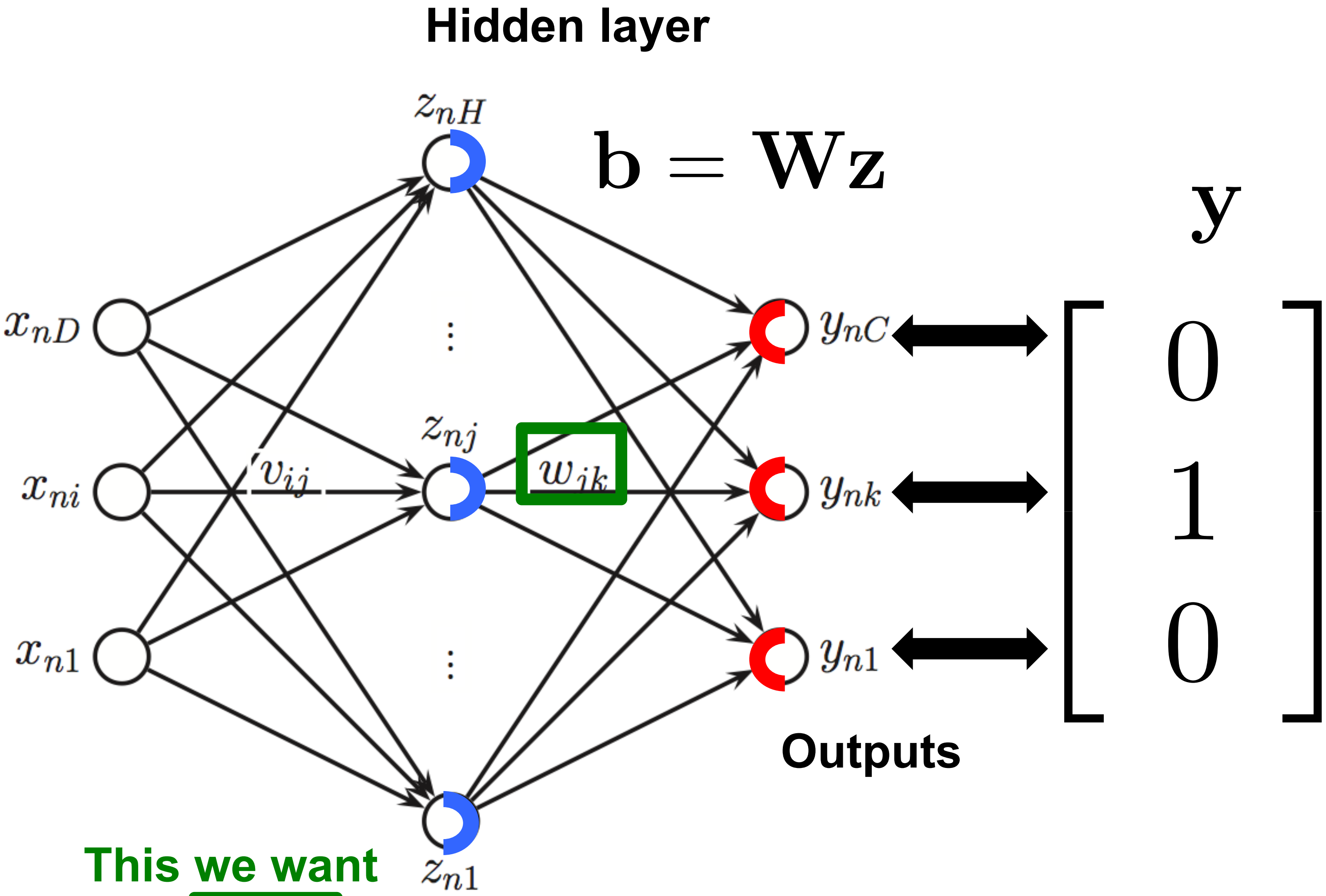
This we want

This we have

This we computed

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

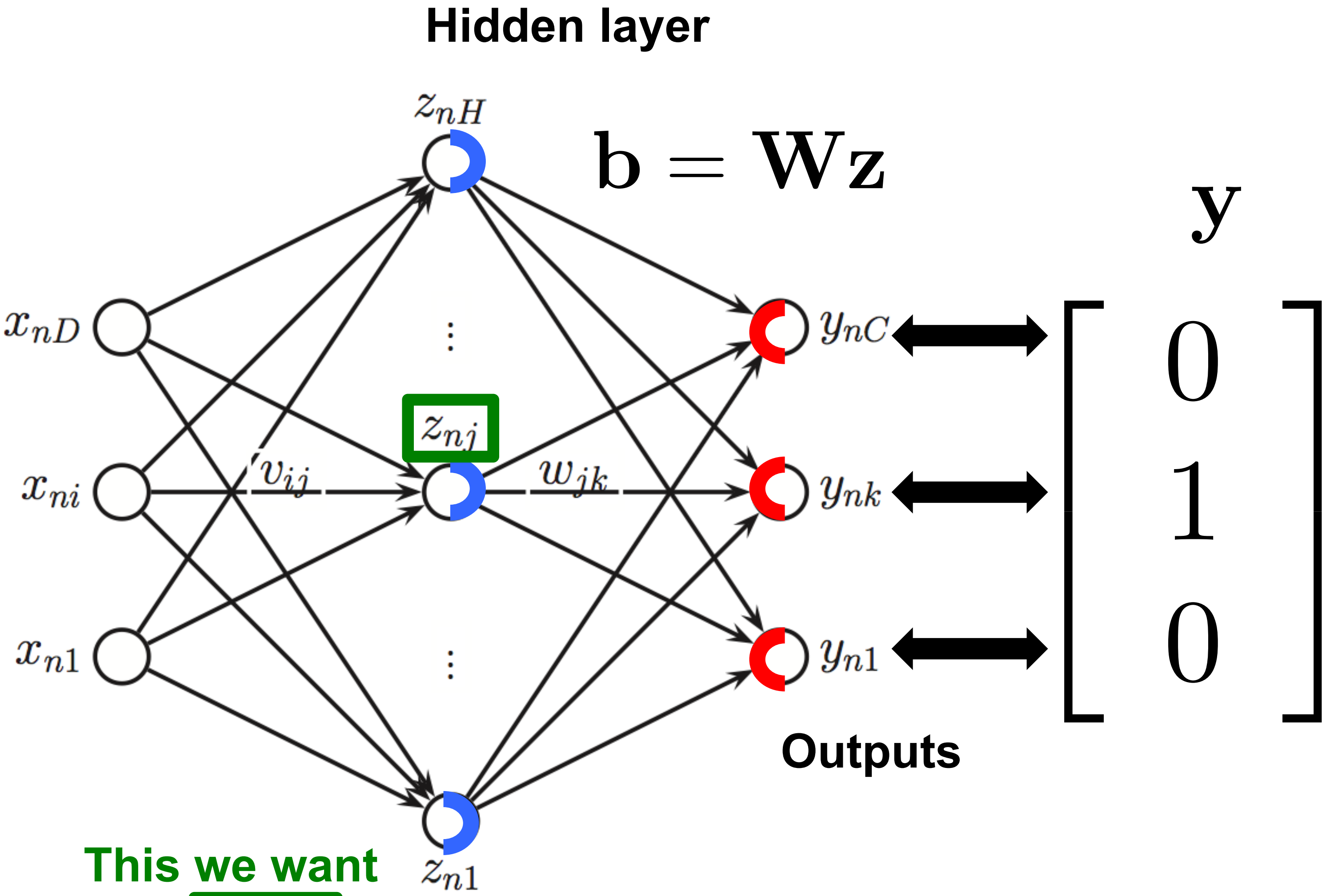
A neural network in backward mode:



This we want

$$\frac{\partial l}{\partial w_{jk}} = \text{?}$$

A neural network in backward mode:

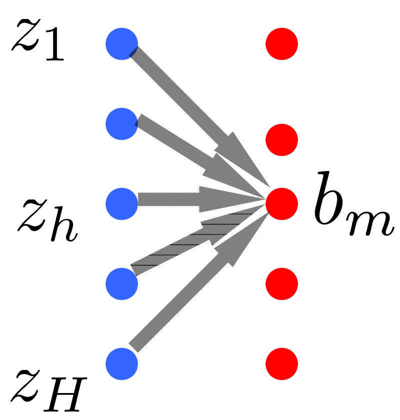


This we want

$$\frac{\partial l}{\partial z_j} = \; ?$$

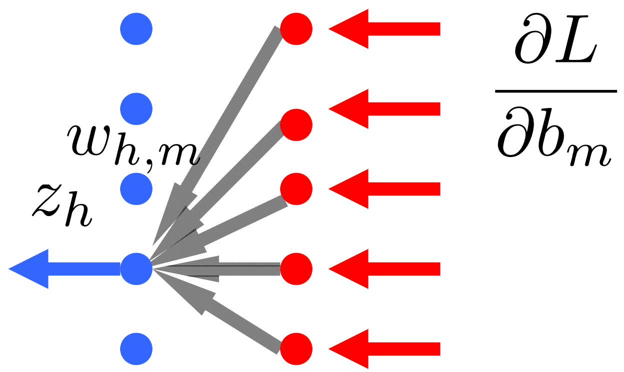
Linear layer in forward mode: all for one

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



Linear layer in backward mode: one from all

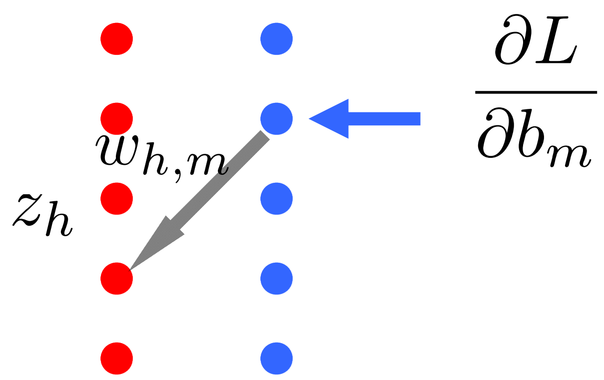
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



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

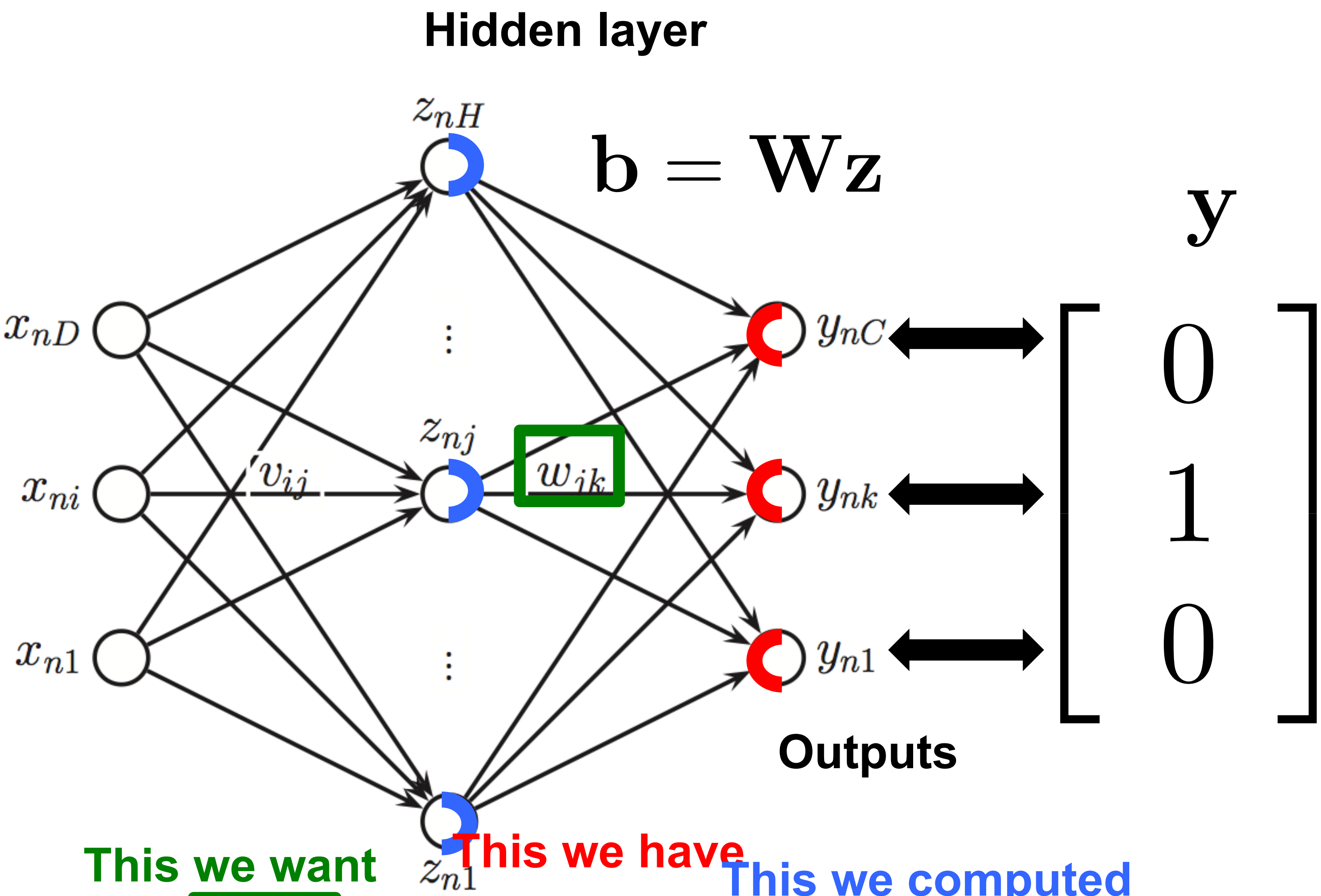
Linear layer parameters in backward: 1-to-1

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



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

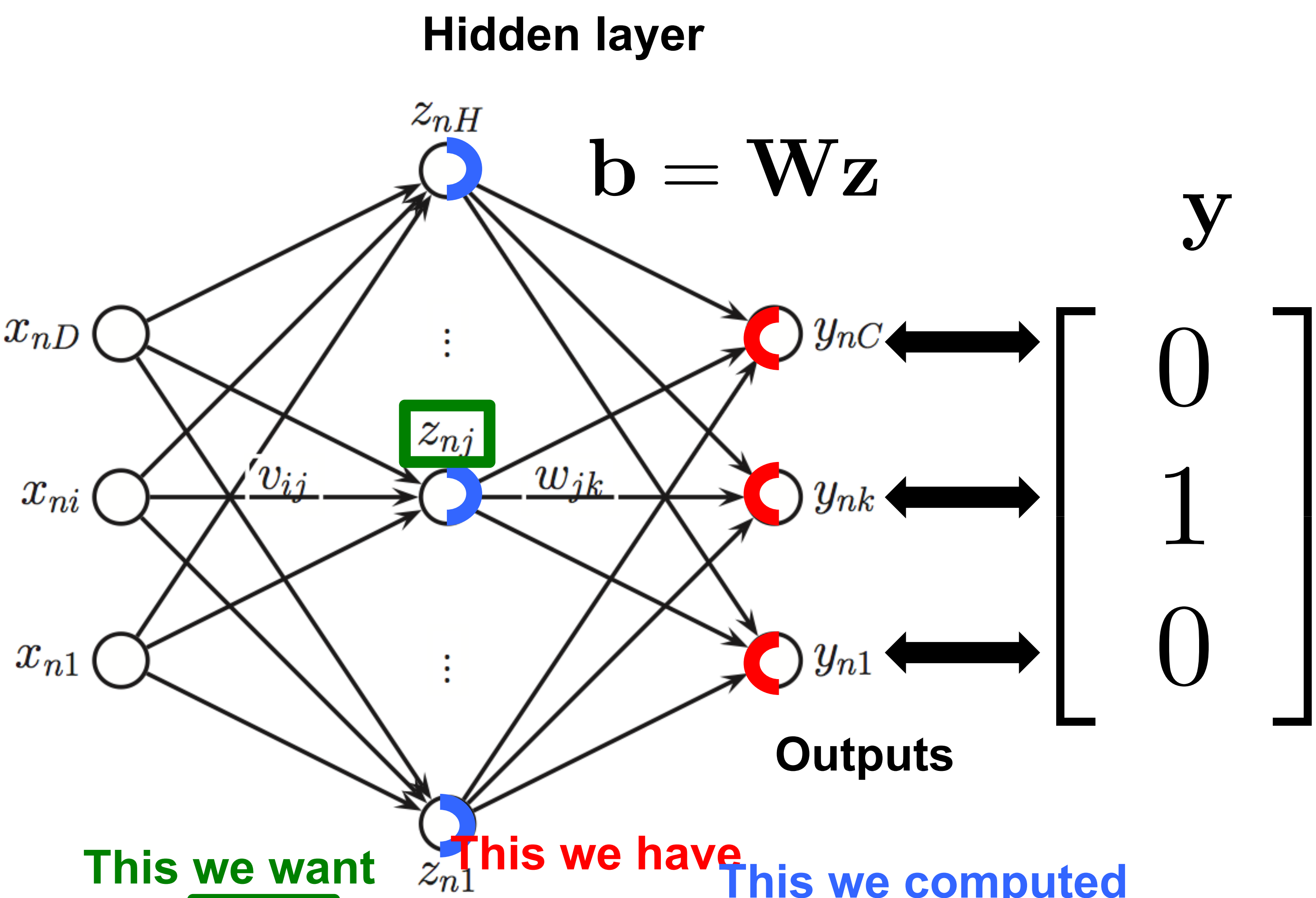
A neural network in backward mode:



This we want — This we have — This we computed

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

A neural network in backward mode:



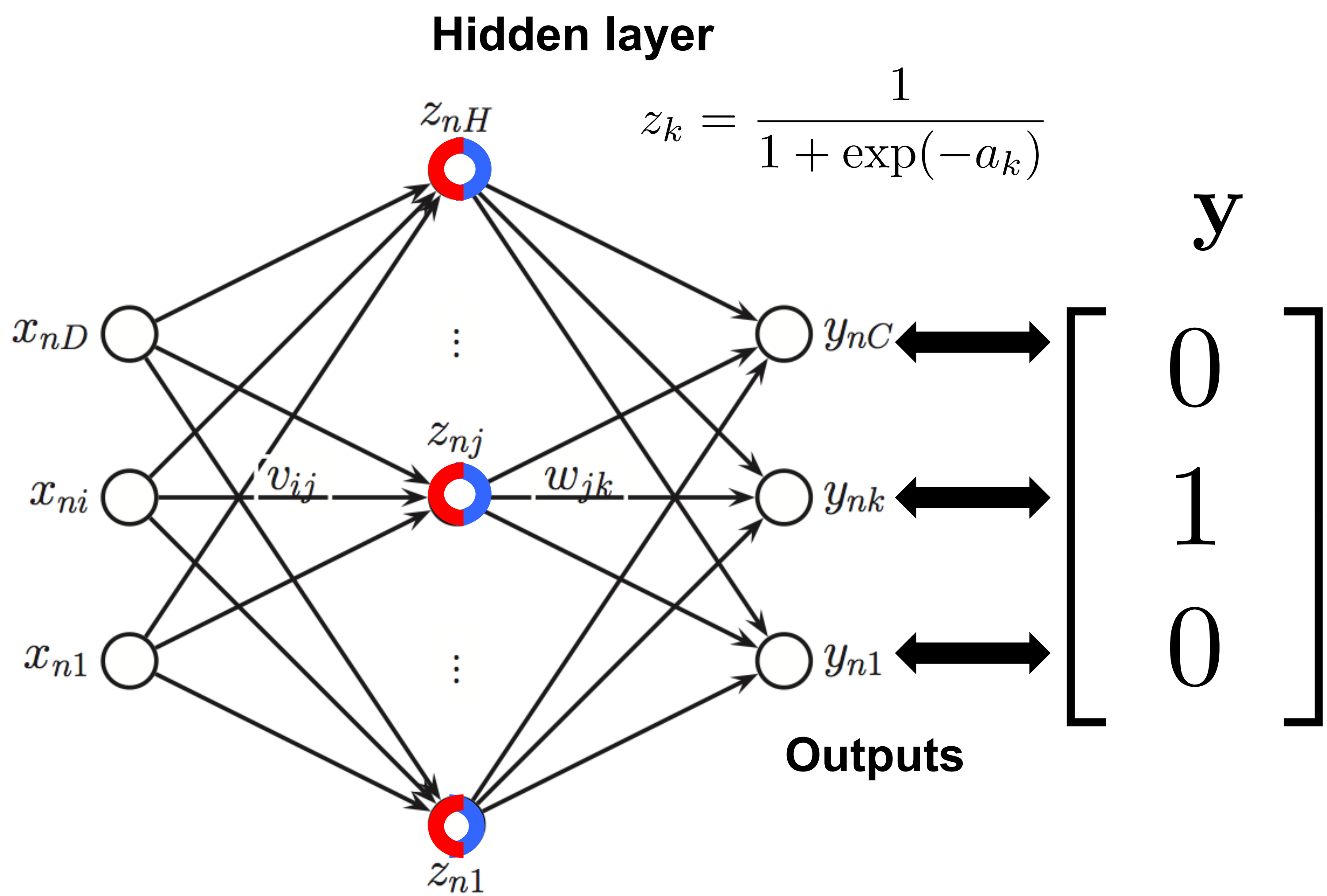
This we want

This we have

This we computed

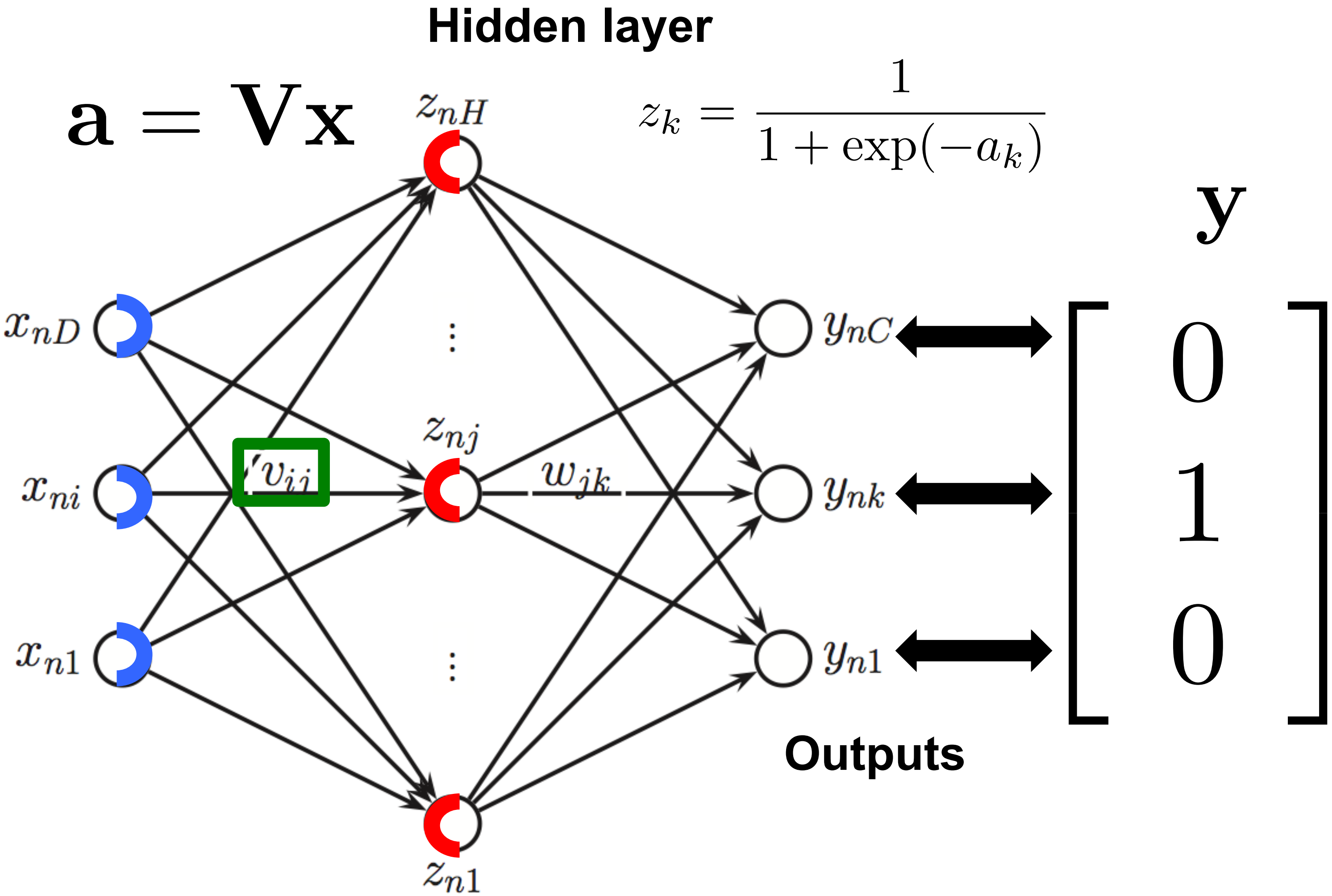
$$\frac{\partial l}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} w_{j,m}$$

A neural network in backward mode:



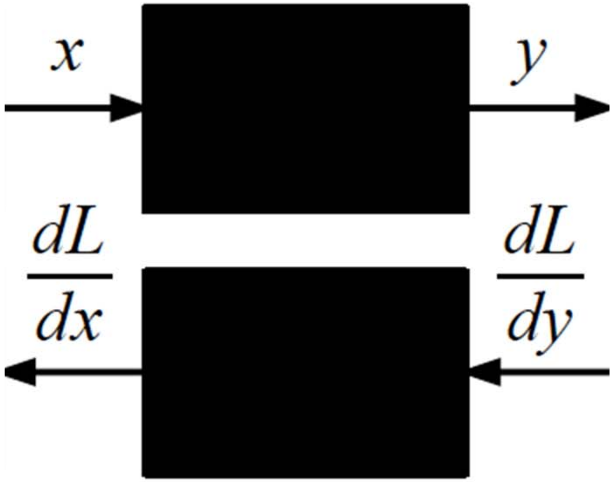
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

A neural network in backward mode:



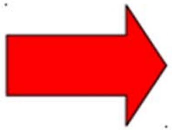
$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k}\frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

Chain Rule



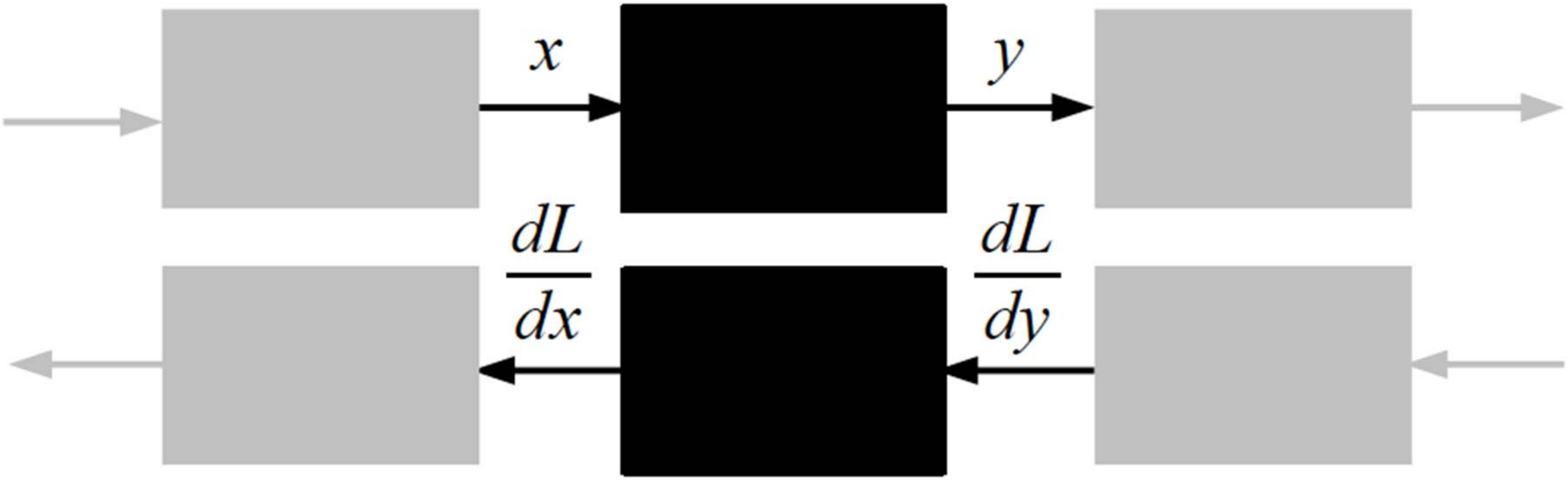
Given $y(x)$ and dL/dy,

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 ?

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

Slide Credit: Marc'Aurelio Ra