# *An Introduction to Statistical Machine Learning - Neural Networks -*

**Samy Bengio**

bengio@idiap.ch

Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP)

CP 592, rue du Simplon 4

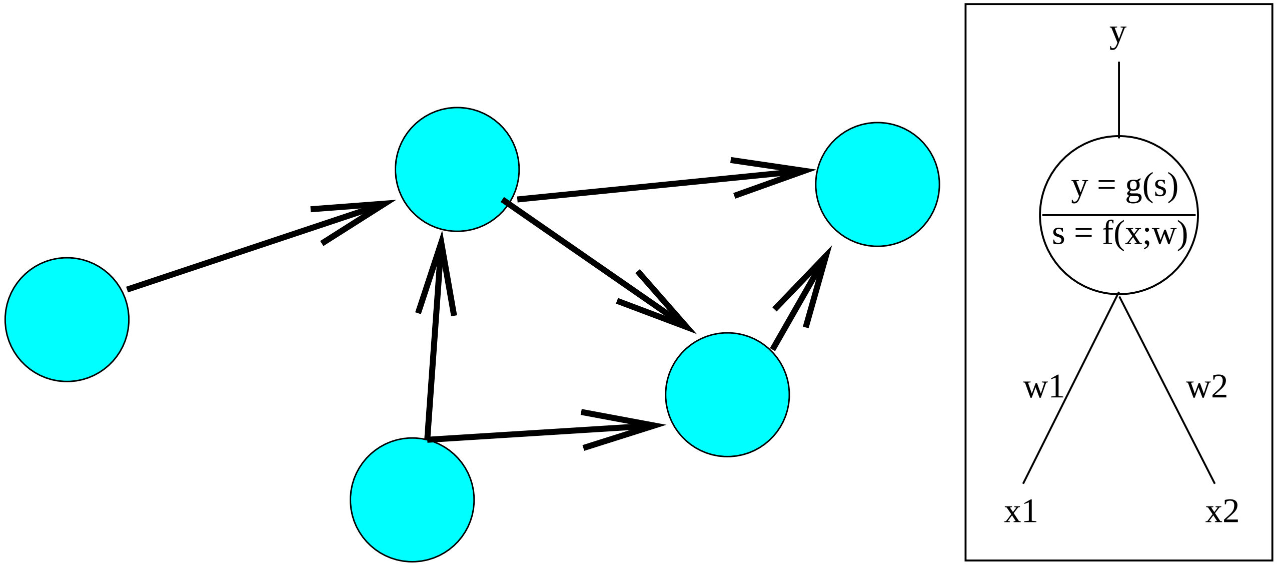
1920 Martigny, Switzerland

http://www.idiap.ch/~bengio

# Artificial Neural Networks and Gradient Descent

1. Artificial Neural Networks
2. Multi Layer Perceptrons
3. Gradient Descent
4. ANN for Classification
5. Tricks of the Trade
6. Other ANN Models

# Artificial Neural Networks

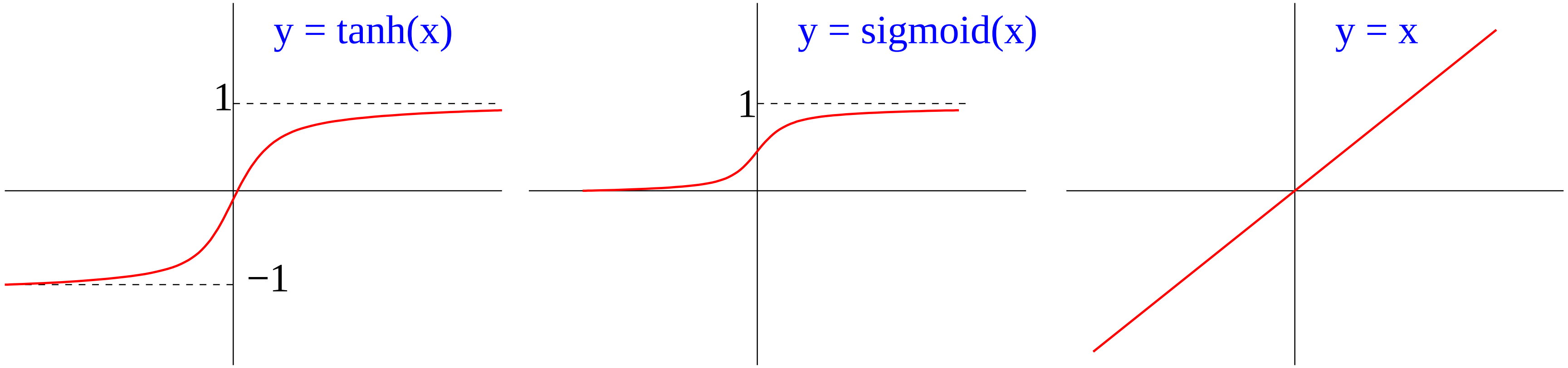



- An ANN is a set of units (neurons) connected to each other
- Each unit may have multiple inputs but have one output
- Each unit performs 2 functions:
  - integration: $s = f(x; \theta)$
  - transfer: $y = g(s)$

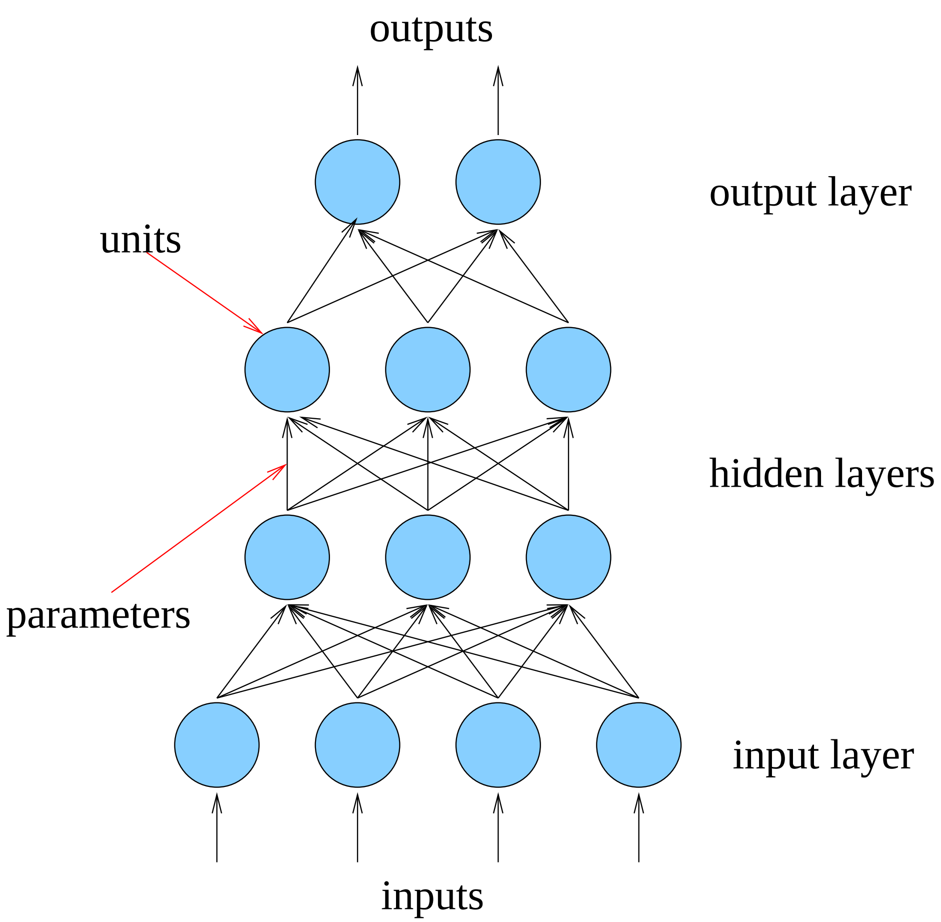
# Artificial Neural Networks: Functions

- Example of integration function: $s = \theta_0 + \sum_i x_i \cdot \theta_i$
- Examples of transfer functions:
  - tanh: $y = \tanh(s)$
  - sigmoid: $y = \dfrac{1}{1 + \exp(-s)}$
- Some units receive inputs from the outside world.
- Some units generate outputs to the outside world.
- The other units are often named hidden.
- Hence, from the outside, an ANN can be viewed as a function.
- There are various forms of ANNs. The most popular is the Multi Layer Perceptron (MLP).

# Transfer Functions (Graphical View)



# Multi Layer Perceptrons (Graphical View)



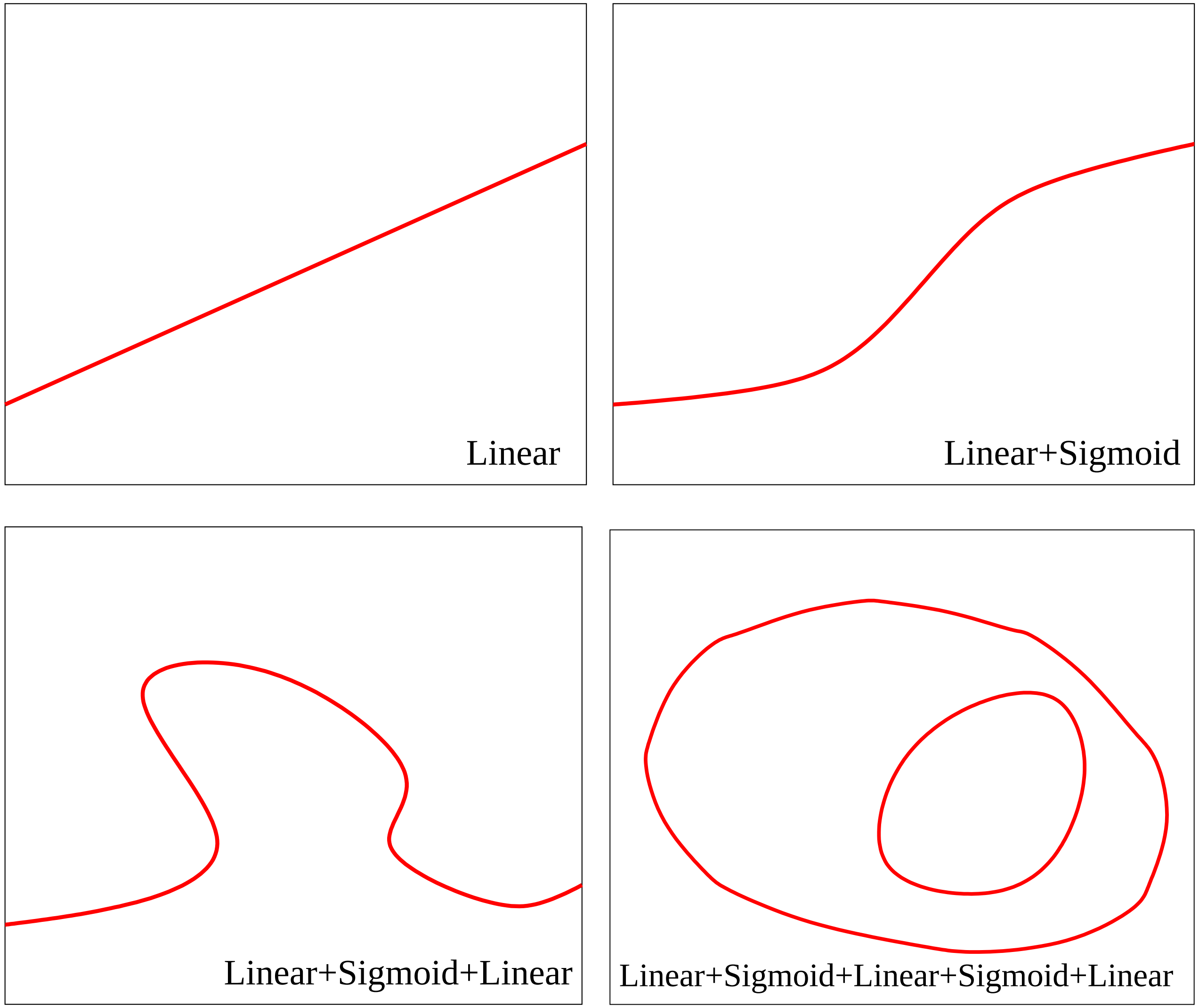
# Multi Layer Perceptrons

- An MLP is a function: $\hat{y} = \mathrm{MLP}(x;\theta)$
- The parameters $\theta = \{w_{i,j}^l, b_i^l \quad : \quad \forall i, j, l\}$
- From now on, let $x_i(p)$ be the $i^{\text{th}}$ value in the $p^{\text{th}}$ example represented by vector $x(p)$ (and when possible, let us drop $p$).
- Each layer $l$ $(1 \leq l \leq M)$ is fully connected to the previous layer
- Integration: $s_i^l = b_i^l + \sum_j y_j^{l-1} \cdot w_{i,j}^l$
- Transfer: $y_i^l = \tanh(s_i^l)$ or $y_i^l = \mathrm{sigmoid}(s_i^l)$ or $y_i^l = s_i^l$
- The output of the zeroth layer contains the inputs $y_i^0 = x_i$
- The output of the last layer $M$ contains the outputs $\hat{y}_i = y_i^M$

# Multi Layer Perceptrons (Characteristics)

- An MLP can approximate any continuous functions
- However, it needs to have at least 1 hidden layer (sometimes easier with 2), and enough units in each layer
- Moreover, we have to find the correct value of the parameters $\theta$
- How can we find these parameters?
- Answer: optimize a given criterion using a gradient method.
- Note: capacity controlled by the number of parameters

# Separability



# Gradient Descent

- Objective: minimize a criterion $C$ over a set of data $D_n$:

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p))$$

where

$$\hat{y}(p) = \text{MLP}(x(p); \theta)$$

- We are searching for the best parameters $\theta$:

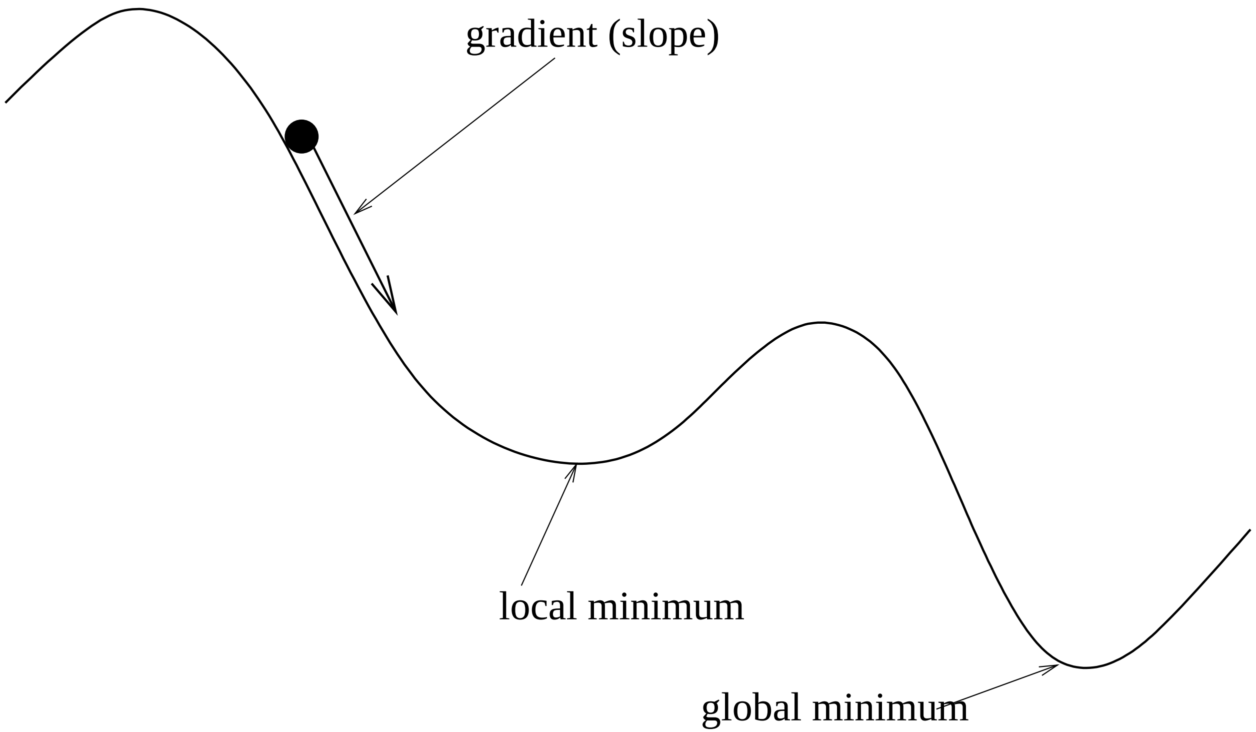
$$\theta^* = \arg\min_{\theta} C(D_n, \theta)$$

- Gradient descent: an iterative procedure where, at each iteration $s$ we modify the parameters $\theta$:

$$\theta^{s+1} = \theta^s - \eta \frac{\partial C(D_n, \theta^s)}{\partial \theta^s}$$

where $\eta$ is the learning rate.

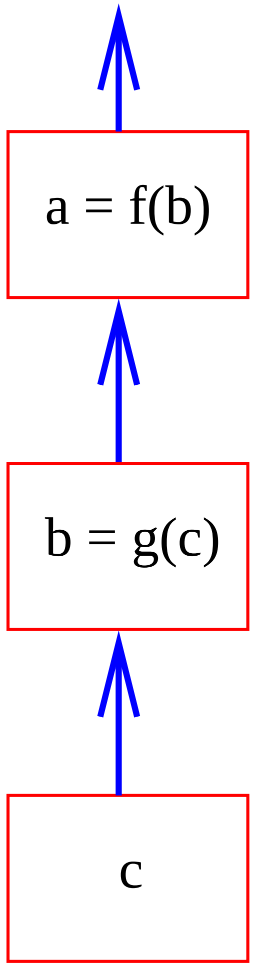
# Gradient Descent (Graphical View)



# Gradient Descent: The Basics

Chain rule:

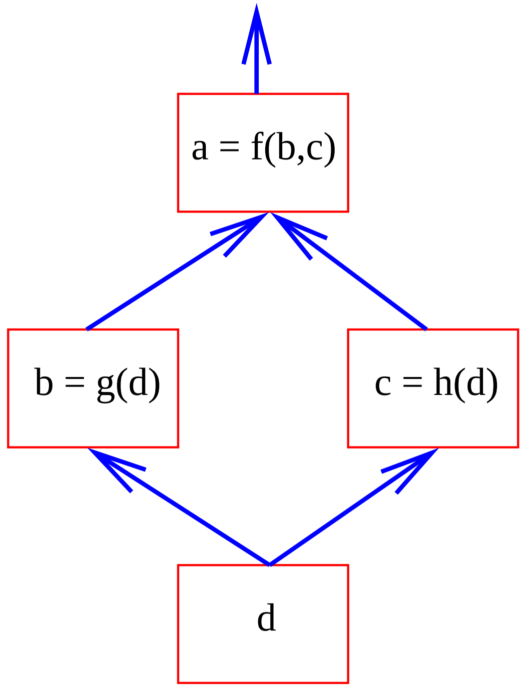
- if $a = f(b)$ and $b = g(c)$
- then $\dfrac{\partial a}{\partial c} = \dfrac{\partial a}{\partial b} \cdot \dfrac{\partial b}{\partial c} = f'(b) \cdot g'(c)$



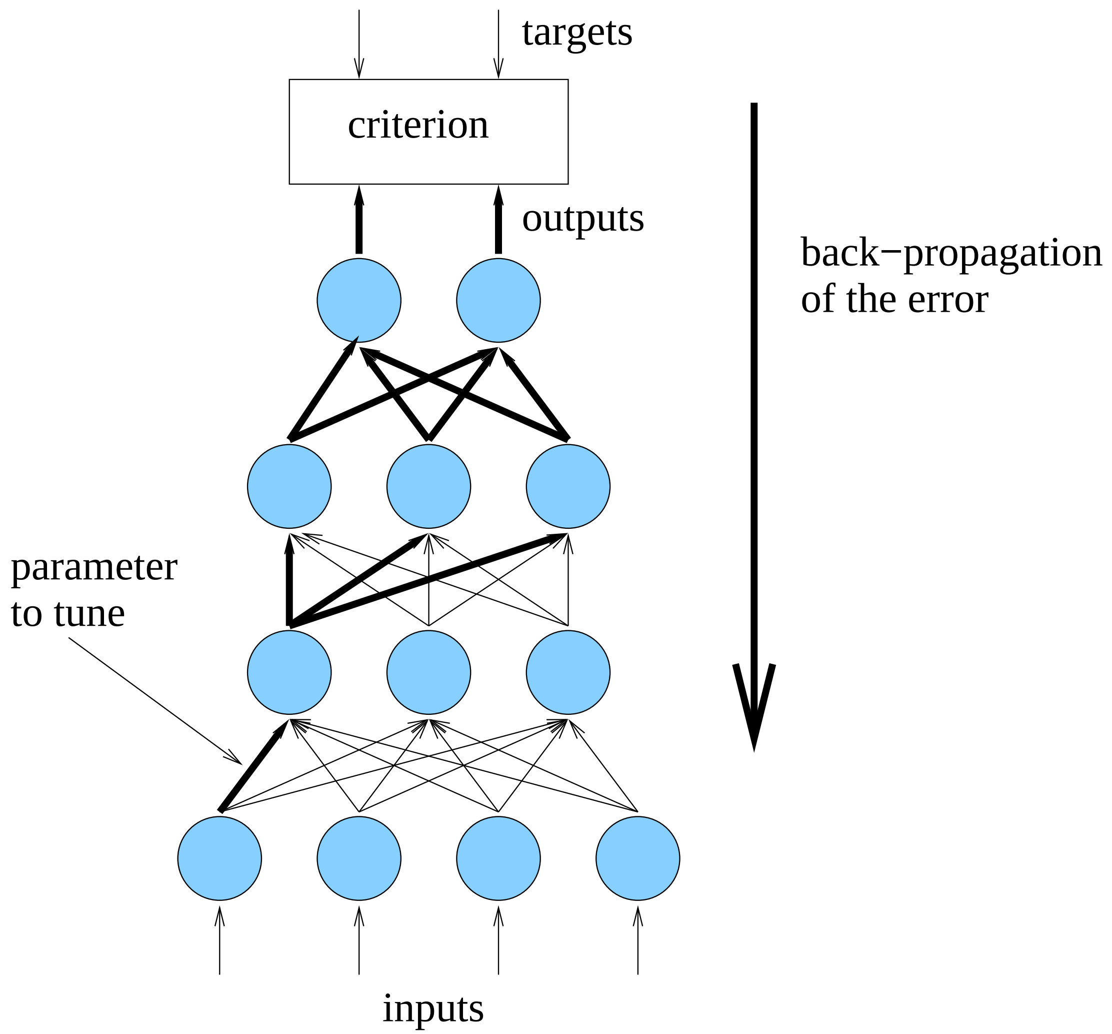
# Gradient Descent: The Basics

Sum rule:

- if $a = f(b, c)$ and $b = g(d)$ and $c = h(d)$
- then $\dfrac{\partial a}{\partial d} = \dfrac{\partial a}{\partial b} \cdot \dfrac{\partial b}{\partial d} + \dfrac{\partial a}{\partial c} \cdot \dfrac{\partial c}{\partial d}$
- $\dfrac{\partial a}{\partial d} = \dfrac{\partial f(b,c)}{\partial b} \cdot g'(d) + \dfrac{\partial f(b,c)}{\partial c} \cdot h'(d)$



# Gradient Descent Basics (Graphical View)



# Gradient Descent: Criterion

- First: we need to pass the gradient through the criterion
- The global criterion $C$ is:

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p))$$

- Example: the mean squared error criterion (MSE):

$$L(y, \hat{y}) = \sum_{i=1}^{d} \frac{1}{2}(y_i - \hat{y}_i)^2$$

- And the derivative with respect to the output $\hat{y}_i$:

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}_i} = \hat{y}_i - y_i$$

# Gradient Descent: Last Layer

- Second: the derivative with respect to the parameters of the last layer $M$

$$\hat{y}_i = y_i^M = \tanh(s_i^M)$$

$$s_i^M = b_i^M + \sum_j y_j^{M-1} \cdot w_{i,j}^M$$

- Hence the derivative with respect to $w_{i,j}^M$ is:

$$\begin{aligned} \frac{\partial \hat{y}_i}{\partial w_{i,j}^M} &= \frac{\partial \hat{y}_i}{\partial s_i^M} \cdot \frac{\partial s_i^M}{\partial w_{i,j}^M} \\ &= (1 - (\hat{y}_i)^2) \cdot y_j^{M-1} \end{aligned}$$

- And the derivative with respect to $b_i^M$ is:

$$\begin{aligned} \frac{\partial \hat{y}_i}{\partial b_i^M} &= \frac{\partial \hat{y}_i}{\partial s_i^M} \cdot \frac{\partial s_i^M}{\partial b_i^M} \\ &= (1 - (\hat{y}_i)^2) \cdot 1 \end{aligned}$$

# Gradient Descent: Other Layers

- Third: the derivative with respect to the output of a hidden layer $y_j^l$

$$\frac{\partial \hat{y}_i}{\partial y_j^l} = \sum_k \frac{\partial \hat{y}_i}{\partial y_k^{l+1}} \cdot \frac{\partial y_k^{l+1}}{\partial y_j^l}$$

where

$$\begin{aligned} \frac{\partial y_k^{l+1}}{\partial y_j^l} &= \frac{\partial y_k^{l+1}}{\partial s_k^{l+1}} \cdot \frac{\partial s_k^{l+1}}{\partial y_j^l} \\ &= (1 - (y_k^{l+1})^2) \cdot w_{k,j}^{l+1} \end{aligned}$$

and

$$\frac{\partial \hat{y}_i}{\partial y_i^M} = 1 \text{ and } \frac{\partial \hat{y}_i}{\partial y_{k \neq i}^M} = 0$$

# Gradient Descent: Other Parameters

- Fourth: the derivative with respect to the parameters of a hidden layer $y_j^l$

$$
\begin{aligned}
\frac{\partial \hat{y}_i}{\partial w_{j,k}^l} &= \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{j,k}^l} \\
&= \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot y_k^{l-1}
\end{aligned}
$$

and

$$
\begin{aligned}
\frac{\partial \hat{y}_i}{\partial b_j^l} &= \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l} \\
&= \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot 1
\end{aligned}
$$

# Gradient Descent: Global Algorithm

1. For each iteration

   (a) Initialize gradients $\frac{\partial C}{\partial \theta_i} = 0$ for each $\theta_i$

   (b) For each example $z(p) = (x(p), y(p))$

      i. Forward phase: compute $\hat{y}(p) = \text{MLP}(x(p), \theta)$

      ii. Compute $\frac{\partial L(y(p), \hat{y}(p))}{\partial \hat{y}(p)}$

      iii. For each layer $l$ from $M$ to 1:

         A. Compute $\frac{\partial \hat{y}(p)}{\partial y_j^l}$

         B. Compute $\frac{\partial y_j^l}{\partial b_j^l}$ and $\frac{\partial y_j^l}{\partial w_{j,k}^l}$

         C. Accumulate gradients:

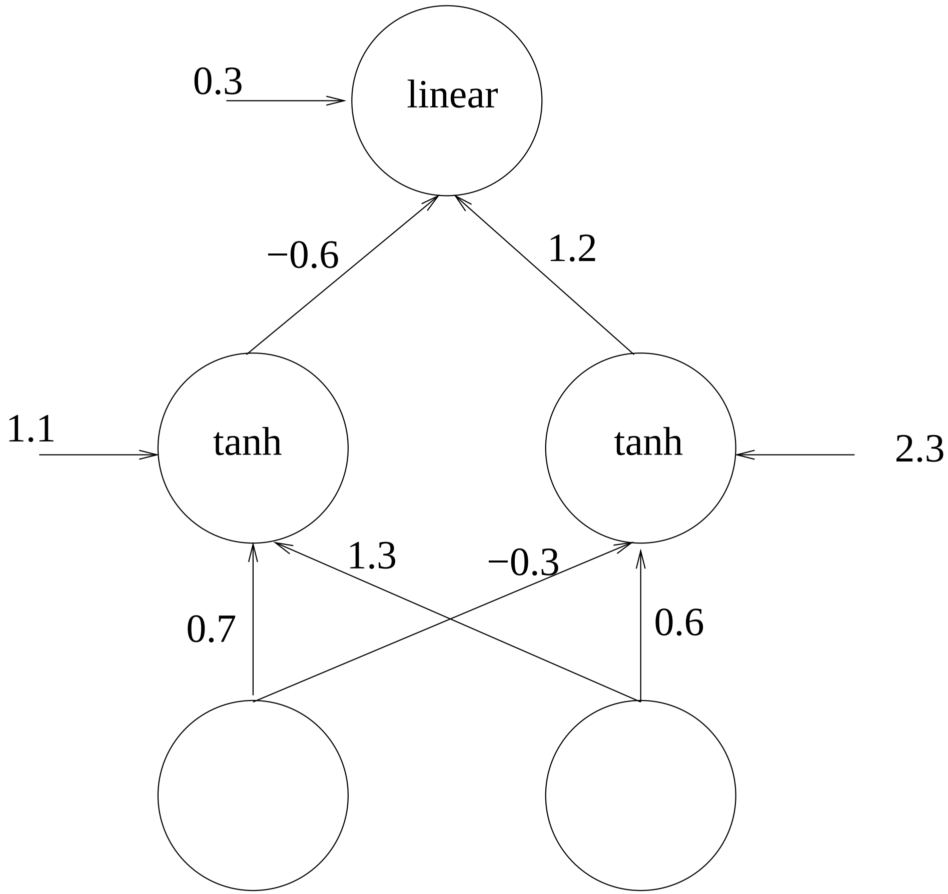
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l}$$

$$\frac{\partial C}{\partial w_{j,k}^l} = \frac{\partial C}{\partial w_{j,k}^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{j,k}^l}$$

   (c) Update the parameters: $\theta_i^{s+1} = \theta_i^s - \eta \cdot \frac{\partial C}{\partial \theta_i^s}$

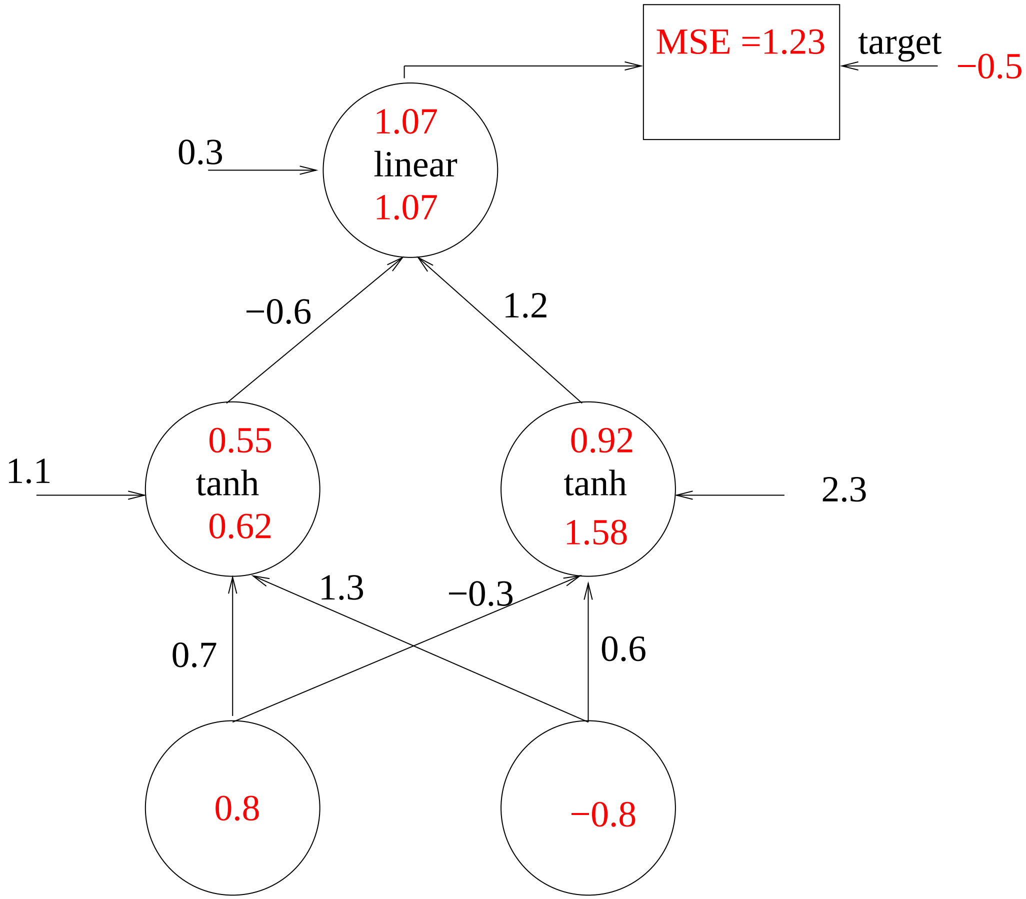
# Gradient Descent: An Example (1)

Let us start with a simple MLP:



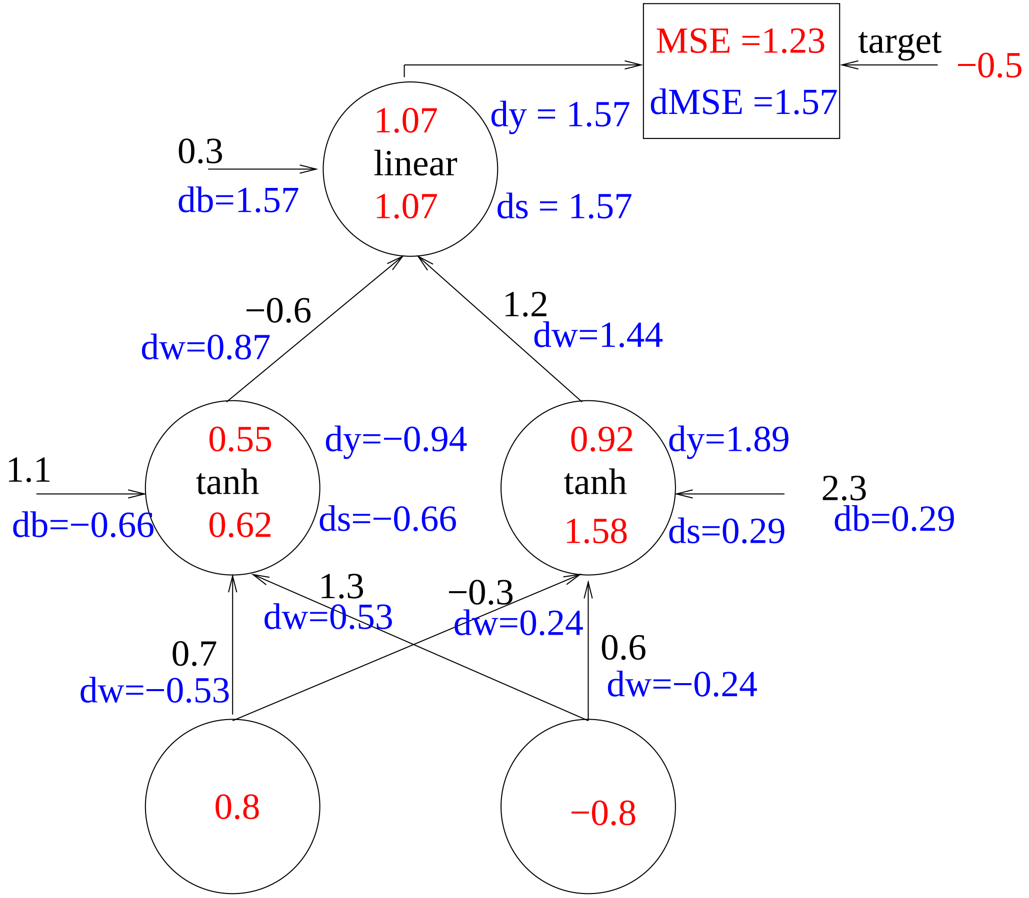
# Gradient Descent: An Example (2)

We forward one example and compute its MSE:



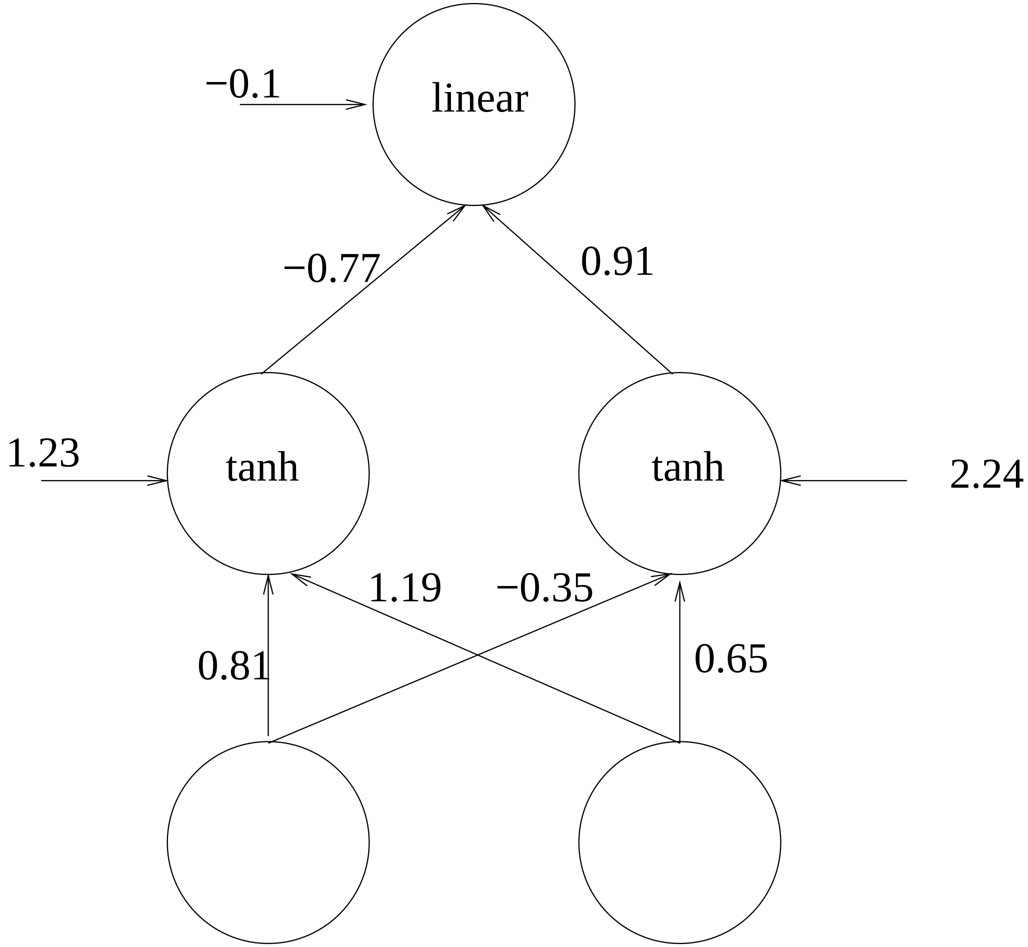
# Gradient Descent: An Example (3)

We backpropagate the gradient everywhere:



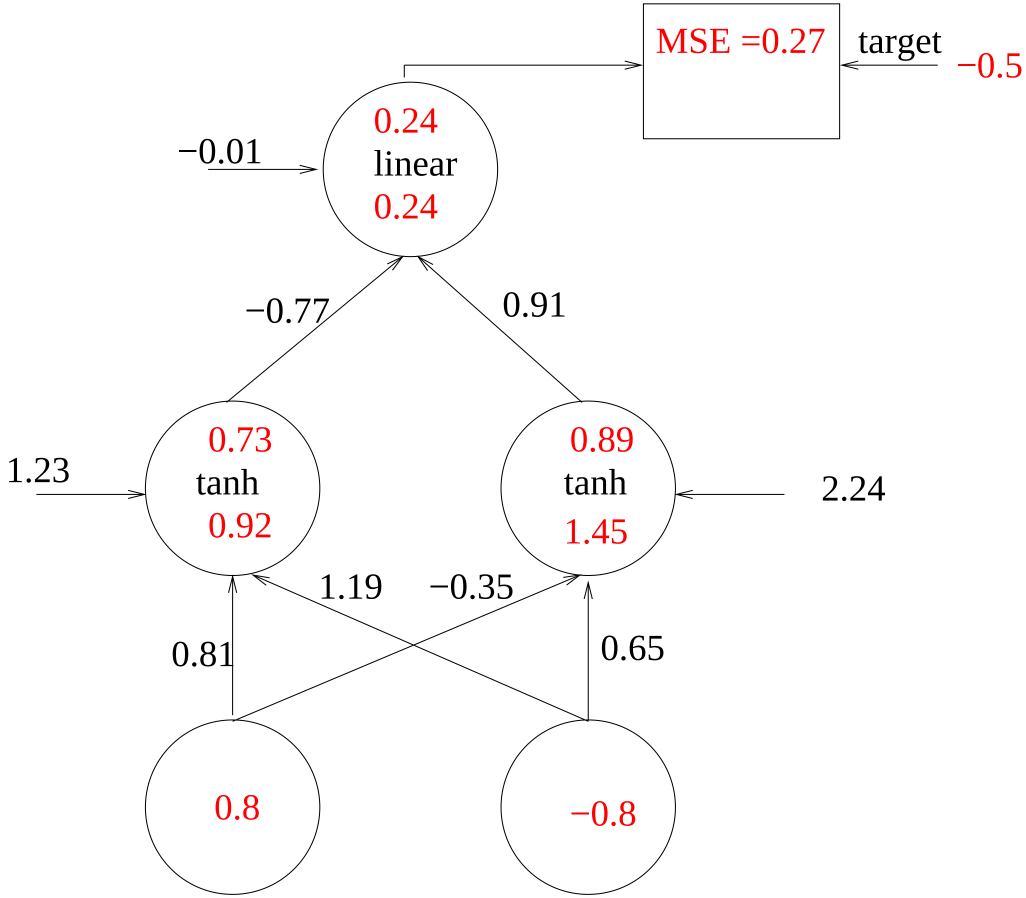
# Gradient Descent: An Example (4)

We modify each parameter with learning rate 0.1:



# Gradient Descent: An Example (5)

We forward again the same example and compute its (smaller) MSE:



# ANN for Binary Classification

- One output with target coded as $\{-1, 1\}$ or $\{0, 1\}$ depending on the last layer output function (linear, sigmoid, tanh, ...)
- For a given output, the associated class corresponds to the nearest target.
- How to obtain class posterior probabilities:
  - use a sigmoid with targets $\{0, 1\}$
  - the output will encode $P(Y = 1|X = x)$
- Note: we do not optimize directly the classification error...

# ANN for Multiclass Classification

- Simplest solution: one-hot encoding
  - One output per class, coded for instance as $(0, \cdots, 1, \cdots, 0)$
  - For a given output, the associated class corresponds to the index of the maximum value in the output vector
  - How to obtain class posterior probabilities:
    - use a softmax: $\hat{y}_i = \dfrac{\exp(s_i)}{\sum_j \exp(s_j)}$
    - each output $i$ will encode $P(Y = i | X = x)$
- Otherwise: each class corresponds to a different binary code
  - For example for a 4-class problem, we could have an 8-dim code for each class
  - For a given output, the associated class corresponds to the nearest code (according to a given distance)
  - Example: Error Correcting Output Codes (ECOC)

# Error Correcting Output Codes

- Let us represent a 4-class problem with 6 bits:

  **class 1:** 1 1 0 0 0 1

  **class 2:** 1 0 0 0 1 0

  **class 3:** 0 1 0 1 0 0

  **class 4:** 0 0 1 0 0 0

- We then create 6 classifiers (or 1 classifier with 6 outputs)
- For example: the first classifier will try to separate classes 1 and 2 from classes 3 and 4
- When a new example comes, we compute the distance between the code obtained by the 6 classifiers and the 4 classes:

  **obtained:** 0 1 1 1 1 0

  **distances:** (let us use Manhatan distance)

  | | |
  |---|---|
  | **to class 1:** 5 | **to class 3:** 2 |
  | **to class 2:** 4 | **to class 4:** 3 |

# Tricks of the Trade

- A good book to make ANNs working:

  G. B. Orr and K. Müller. *Neural Networks: Tricks of the Trade*. 1998. Springer.

- Stochastic Gradient
- Initialization
- Learning Rate and Learning Rate Decay
- Weight Decay

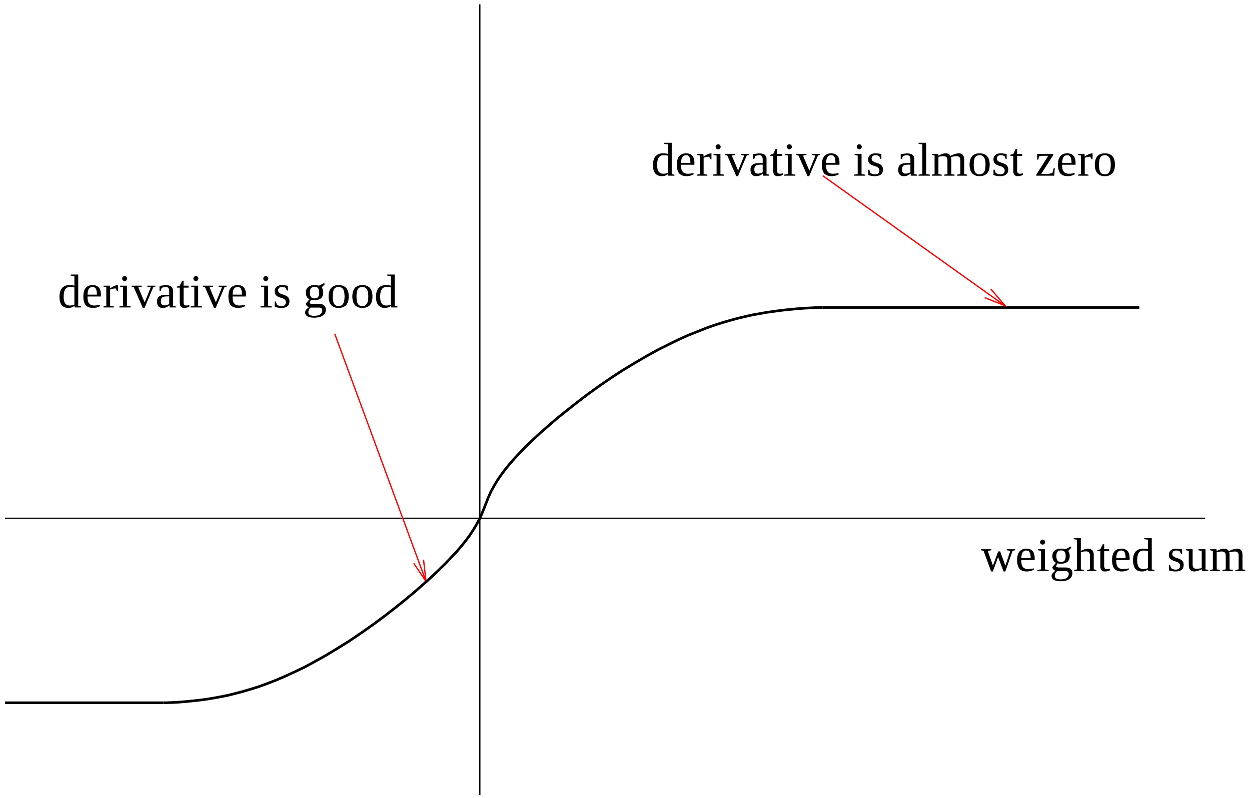
# Stochastic Gradient Descent

- The gradient descent technique is batch:
  - First accumulate the gradient from all examples, then adjust the parameters
  - What if the data set is very big, and contains redundencies?
- Other solution: stochastic gradient descent
  - Adjust the parameters after each example instead
  - Stochastic: we approximate the full gradient with its estimate at each example
  - Nevertheless, convergence proofs exist for such method.
  - Moreover: much faster for large data sets!!!
- Other gradient techniques: second order methods such as conjugate gradient: good for small data sets

# Initialization

- How should we initialize the parameters of an ANN?
- One common problem: saturation

  When the weighted sum is big, the output of the tanh (or sigmoid) saturates, and the gradient tends towards 0



# Initialization

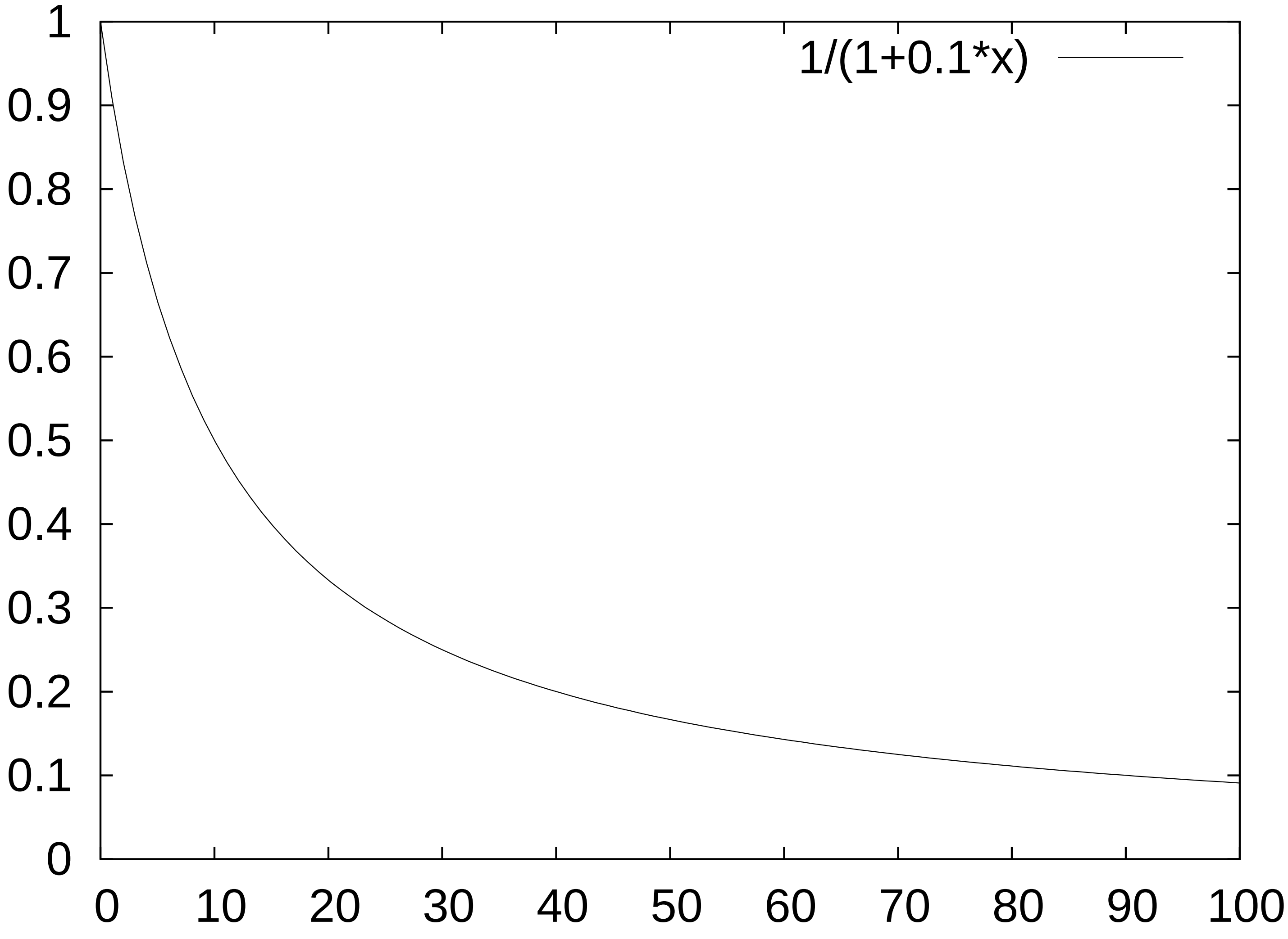
- Hence, we should initialize the parameters such that the average weighted sum is in the linear part of the transfer function:
  - See Leon Bottou's thesis for details
  - input data: normalized with zero mean and unit variance,
  - targets:
    - regression: normalized with zero mean and unit variance,
    - classification:
      - output transfer function is tanh: 0.6 and -0.6
      - output transfer function is sigmoid: 0.8 and 0.2
      - output transfer function is linear: 0.6 and -0.6
  - parameters: uniformly distributed in $\left[\frac{-1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right]$

# Learning Rate and Learning Rate Decay

- How to select the learning rate $\eta$ ?
- If $\eta$ is too big: the optimization diverges
- If $\eta$ is too small: the optimization is very slow and may be stuck into local minima
- One solution: progressive decay
  - initial learning rate $\eta = \eta_0$
  - learning rate decay $\eta_d$
  - At each iteration $s$:

$$\eta(s) = \frac{\eta_0}{(1 + s \cdot \eta_d)}$$

# Learning Rate Decay (Graphical View)



# Weight Decay

- One way to control the capacity: regularization
- For MLPs, when the weights tend to 0, sigmoid or tanh functions are almost linear, hence with low capacity
- Weight decay: penalize solutions with high weights and bias (in amplitude)

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p)) + \frac{\beta}{2} \sum_{j=1}^{|\theta|} \theta_j^2$$

where $\beta$ controls the weight decay.

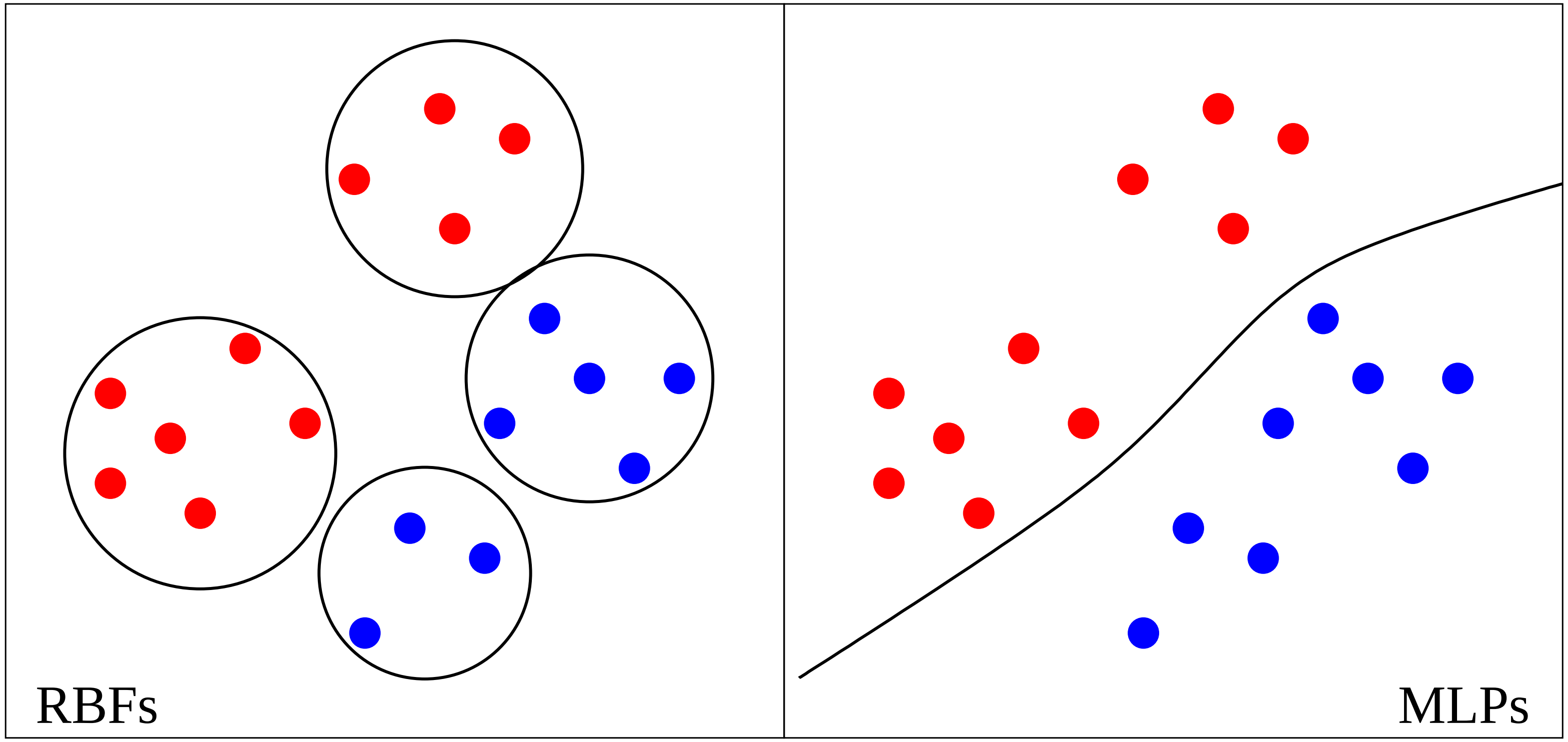
- Easy to implement:

$$\theta_j^{s+1} = \theta_j^s - \sum_{p=1}^{n} \eta \frac{\partial L(y(p), \hat{y}(p))}{\partial \theta_j^s} - \eta \cdot \beta \cdot \theta_j^s$$

# Radial Basis Function (RBF) Models

- Normal MLP but the hidden layer $l$ is encoded as follows:
  - $s_i^l = -\frac{1}{2} \sum_j (\gamma_{i,j}^l)^2 \cdot (y_j^{l-1} - \mu_{i,j}^l)^2$
  - $y_i^l = \exp(s_i^l)$
- The parameters of such layer $l$ are $\theta_l = \{\gamma_{i,j}^l, \mu_{i,j}^l \quad : \quad \forall i, j\}$
- These layers are useful to extract local features (whereas tanh layers extract global features)
- Initialization: use K-Means for instance

# Difference Between RBFs and MLPs



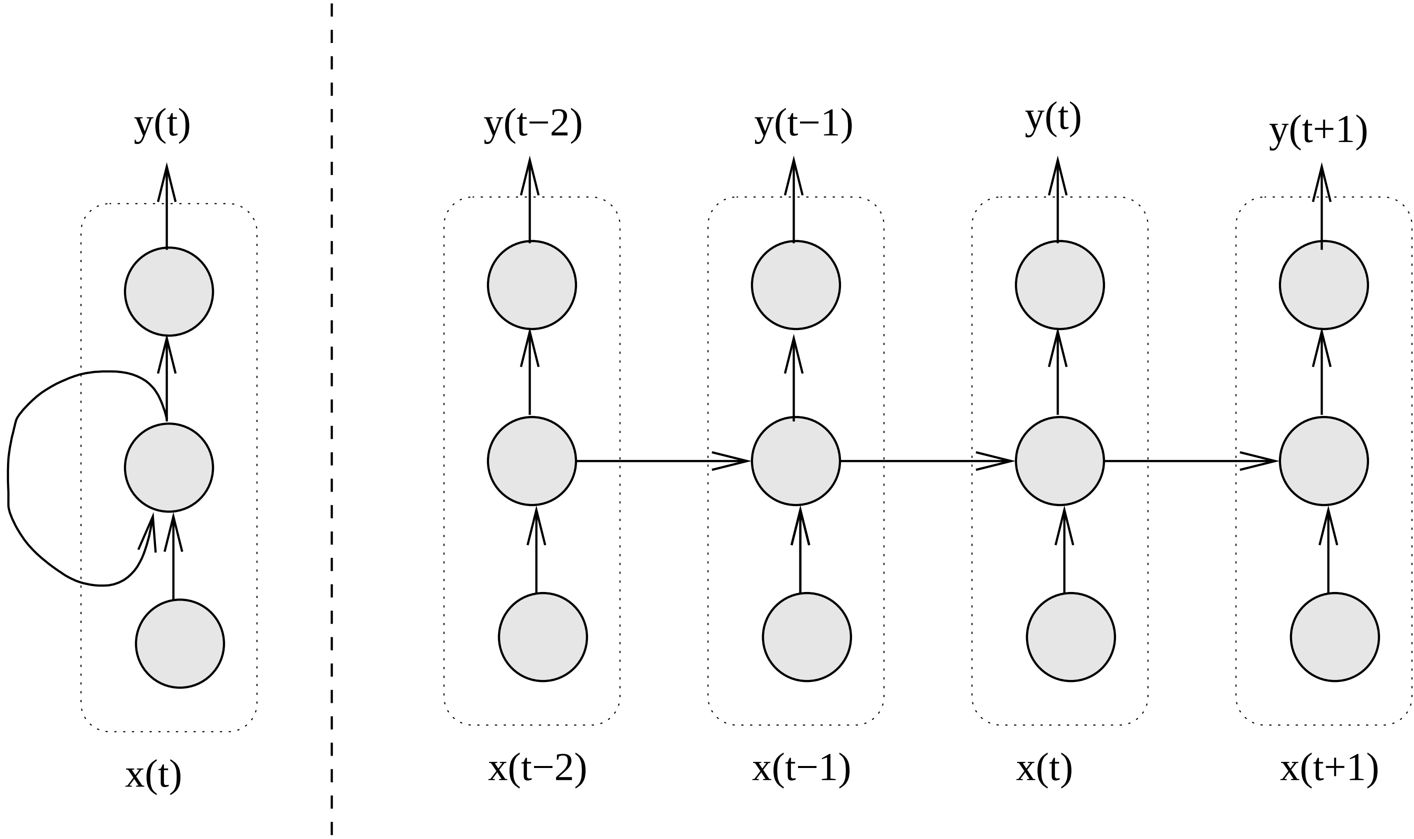
# Recurrent Neural Networks

- Such models admit layers $l$ with integration functions $s_i^l = f(y_j^{l+k})$ where $k \geq 0$, hence loops, or recurrences
- Such layers $l$ encode the notion of a temporal state
- Useful to search for relations in temporal data
- Do not need to specify the exact delay in the relation
- In order to compute the gradient, one must enfold in time all the relations between the data:

$$s_i^l(t) = f(y_j^{l+k}(t-1)) \text{ where } k \geq 0$$

- Hence, need to exhibit the whole time-dependent graph between input sequence and output sequence
- Caveat: it can be shown that the gradient vanishes exponentially fast through time

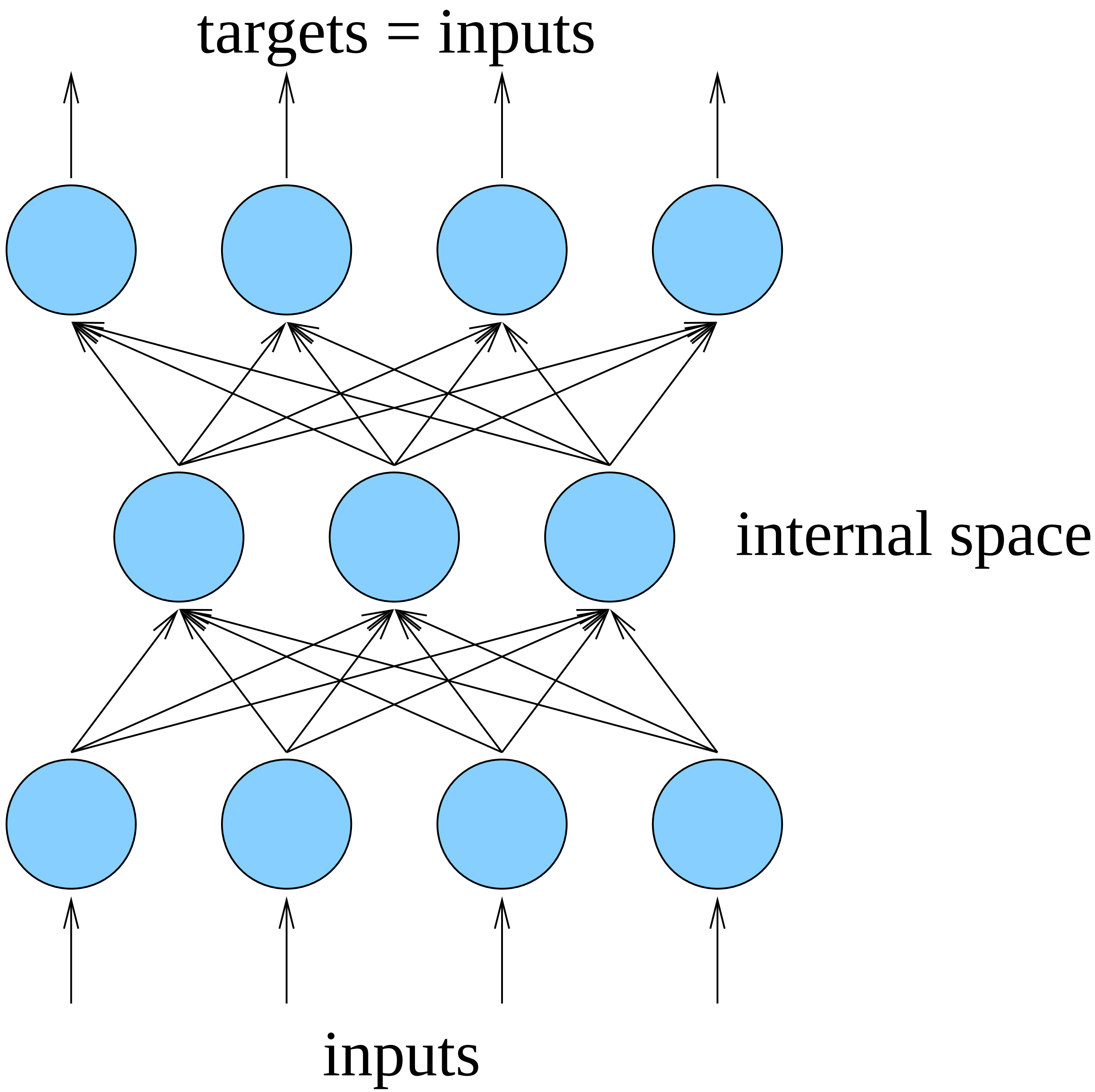
# Recurrent NNs (Graphical View)



# Auto Associative Networks

- Apparent objective: learn to reconstruct the input
- In such models, the target vector is the same as the input vector!
- Real objective: learn an internal representation of the data
- If there is one hidden layer of linear units, then after learning, the model implements a principal component analysis with the first $N$ principal components ($N$ is the number of hidden units).
- If there are non-linearities and more hidden layers, then the system implements a kind of non-linear principal component analysis.

# Auto Associative Nets (Graphical View)



# Mixture of Experts

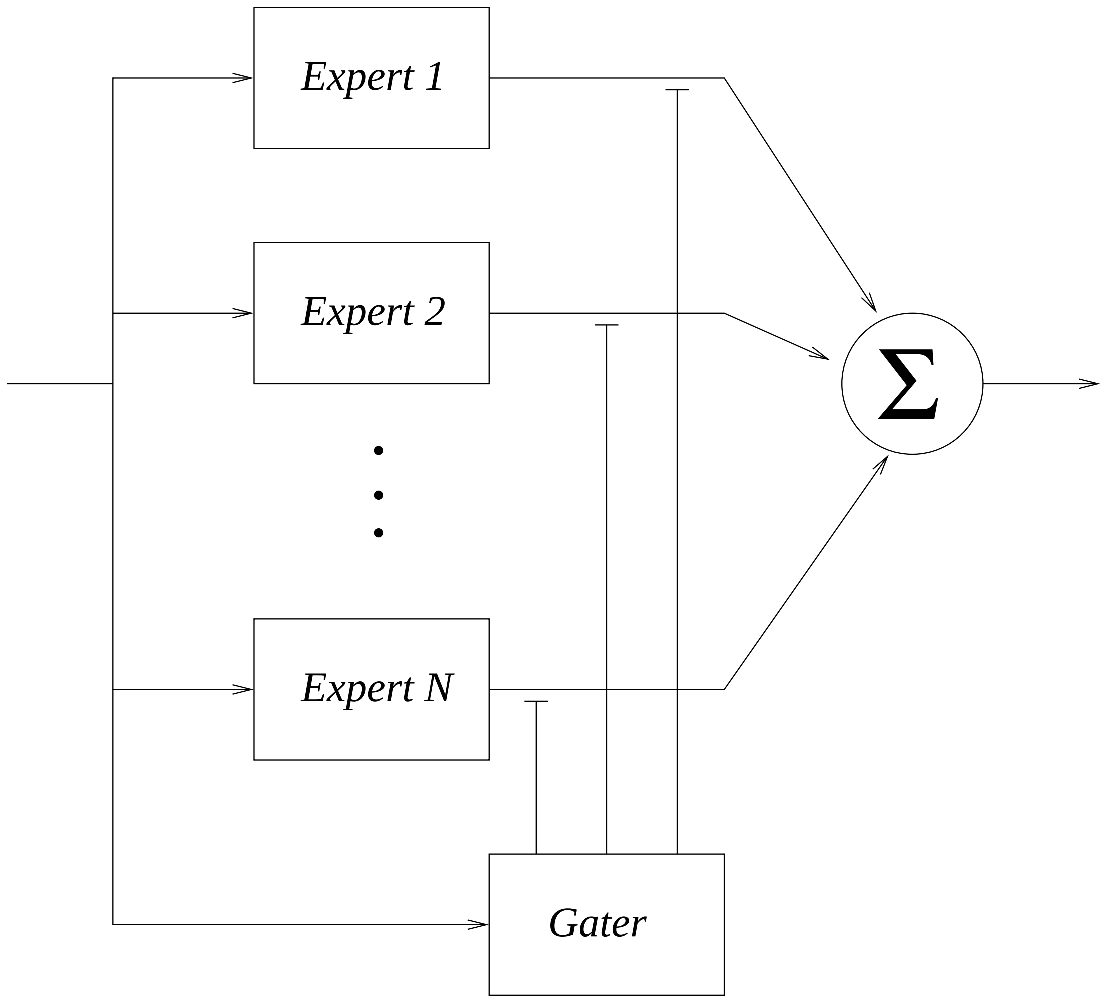
- Let $f_i(x;\theta_{f_i})$ be a differentiable parametric function
- Let there be $N$ such functions $f_i$.
- Let $g(x;\theta_g)$ be a gater: a differentiable function with $N$ positive outputs such that

$$\sum_{i=1}^{N} g(x;\theta_g)[i] = 1$$

- Then a mixture of experts is a function $h(x;\theta)$:

$$h(x;\theta) = \sum_{i=1}^{N} g(x;\theta_g)[i] \cdot f_i(x;\theta_{f_i})$$

# Mixture of Experts - (Graphical View)



# Mixture of Experts - Training

- We can compute the gradient with respect to every parameters:
  - parameters in the expert $f_i$:

$$
\begin{aligned}
\frac{\partial h(x;\theta)}{\partial \theta_{f_i}} &= \frac{\partial h(x;\theta)}{\partial f_i(x;\theta_{f_i})} \cdot \frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}} \\
&= g(x;\theta_g)[i] \cdot \frac{\partial f_i(x;\theta_{f_i})}{\partial \theta_{f_i}}
\end{aligned}
$$

  - parameters in the gater $g$:

$$
\begin{aligned}
\frac{\partial h(x;\theta)}{\partial \theta_g} &= \sum_{i=1}^{N} \frac{\partial h(x;\theta)}{\partial g(x;\theta_g)[i]} \cdot \frac{\partial g(x;\theta_g)[i]}{\partial \theta_g} \\
&= \sum_{i=1}^{N} f_i(x;\theta_{f_i}) \cdot \frac{\partial g(x;\theta_g)[i]}{\partial \theta_g}
\end{aligned}
$$

# Mixture of Experts - Discussion

- The gater implements a soft partition of the input space
  (to be compared with, say, K-Means → hard partition)
- Useful when there might be regimes in the data
- Extension: hierarchical mixture of experts, when the experts are themselves represented as mixtures of experts!
- Special case: when the experts can be trained by EM, the mixture and the hierachical mixture can also be trained by EM.