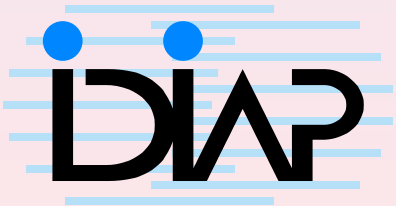
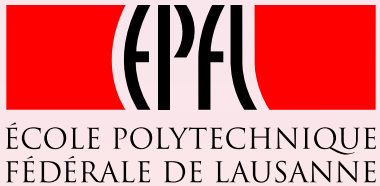
Statistical Machine Learning from Data

Multi-Layer Perceptrons

Samy Bengio

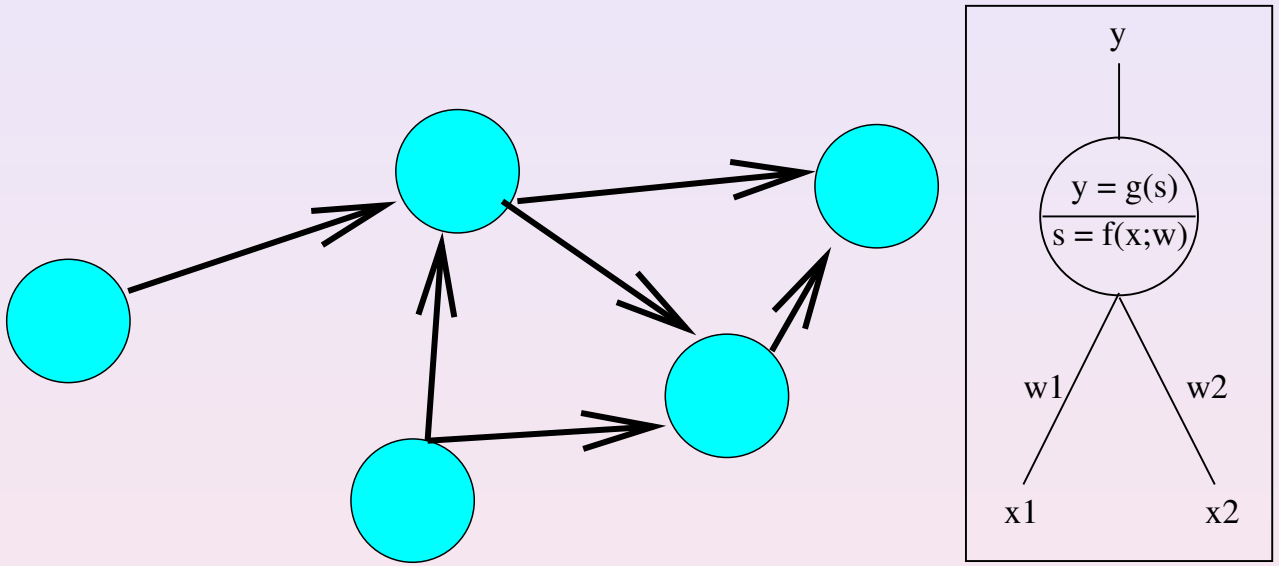
IDIAP Research Institute, Martigny, Switzerland, and
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
bengio@idiap.ch
http://www.idiap.ch/~bengio





January 17, 2006

Artificial Neural Networks

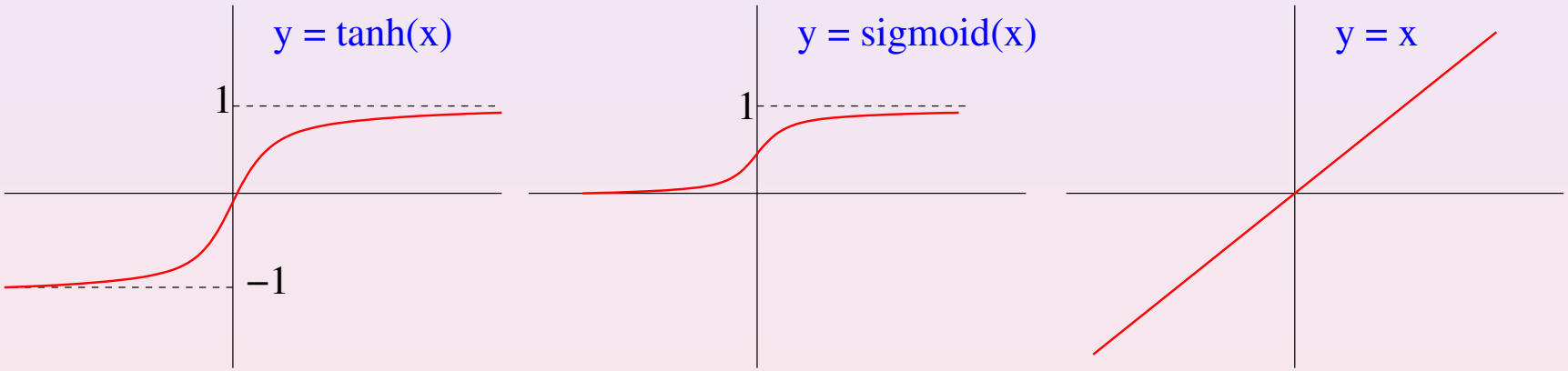


- An ANN is a set of units (neurons) connected to each other
- Each unit may have multiple inputs but have one output
- Each unit performs 2 functions:
  - integration: $s = f(x; \theta)$
  - transfer: $y = g(s)$

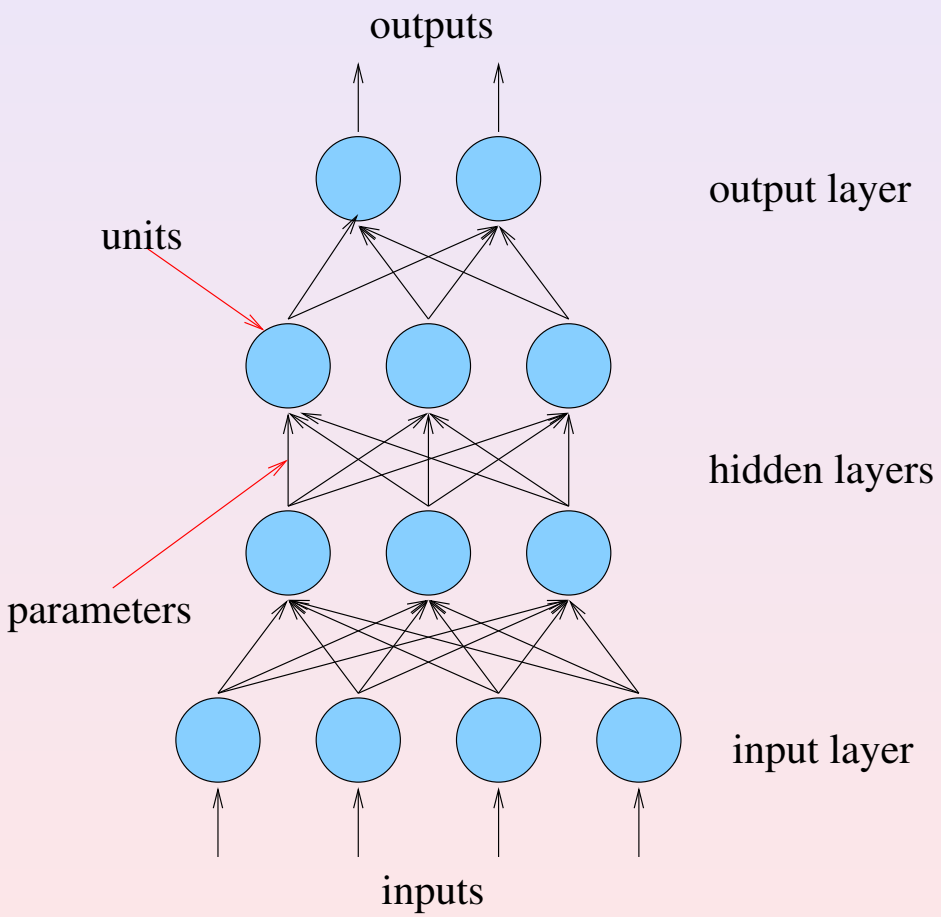
Artificial Neural Networks: Functions

- Example of integration function: $s = \theta_0 + \sum_i x_i \cdot \theta_i$
- Examples of transfer functions:
  - tanh: $y = \tanh(s)$
  - sigmoid: $y = \frac{1}{1 + \exp(-s)}$
- 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)



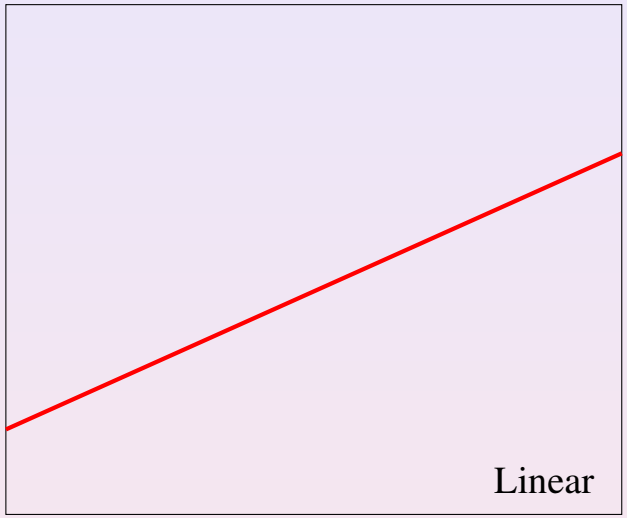
Multi Layer Perceptrons

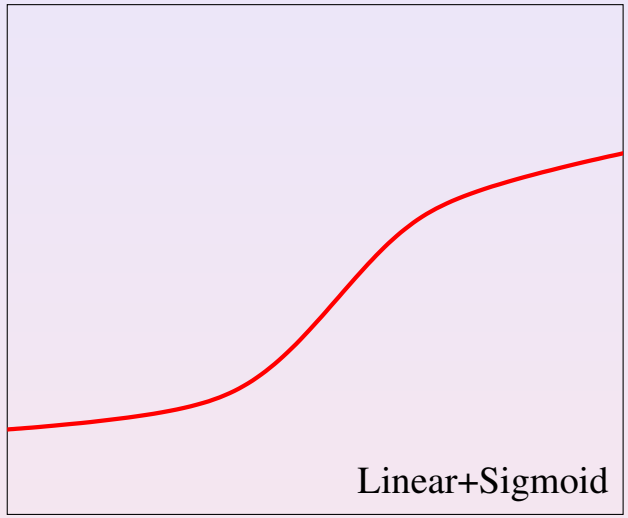
- An MLP is a function: $\hat{y} = \text{MLP}(x; \theta)$
- The parameters $\theta = \{w^l_{i,j}, b^l_i \quad : \quad \forall i, j, l\}$
- From now on, let $x_i(p)$ be the i^{th} value in the p^{th} example represented by vector $x(p)$ (and when possible, let us drop p).
- Each layer l ($1 \leq l \leq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

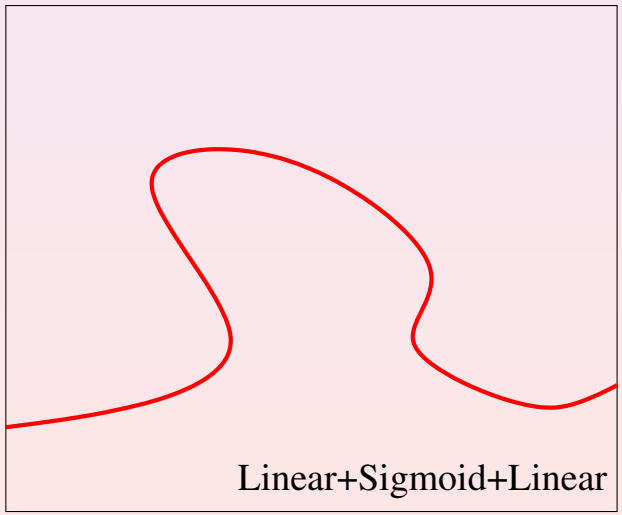
Characteristics of MLPs

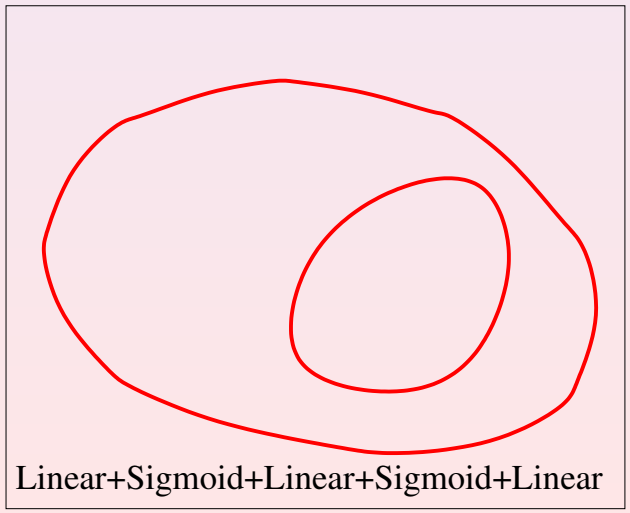
- An MLP can approximate any continuous functions
- However, it needs to have at least 1 hidden layer (sometimes easier with 2), and enough units in each layer
- Moreover, we have to find the correct value of the parameters θ
- This is an NP-complete problem!!!!
- How can we find these parameters?
- 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 θ:

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

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

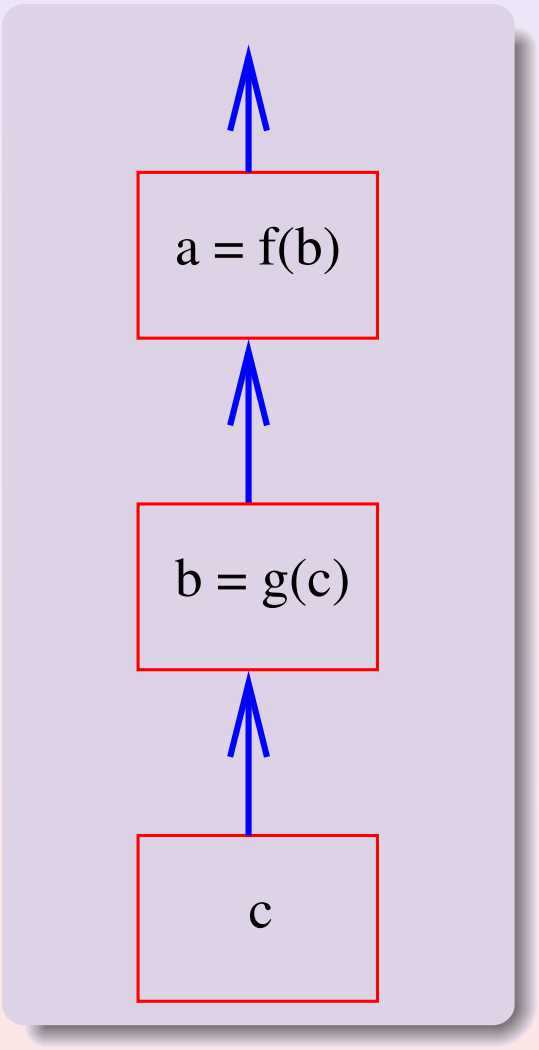
$$\theta^{s+1} = \theta^s - \eta \frac{\partial C(D_n, \theta^s)}{\partial \theta^s}$$

where η is the learning rate. WARNING: local optima.

Gradient Descent: The Basics

Chain Rule

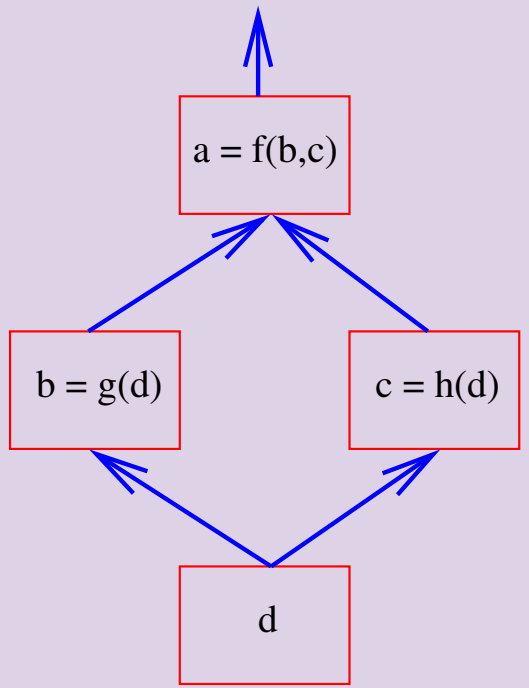
- if $a = f(b)$ and $b = g(c)$
- then $\frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} = f'(b) \cdot g'(c)$



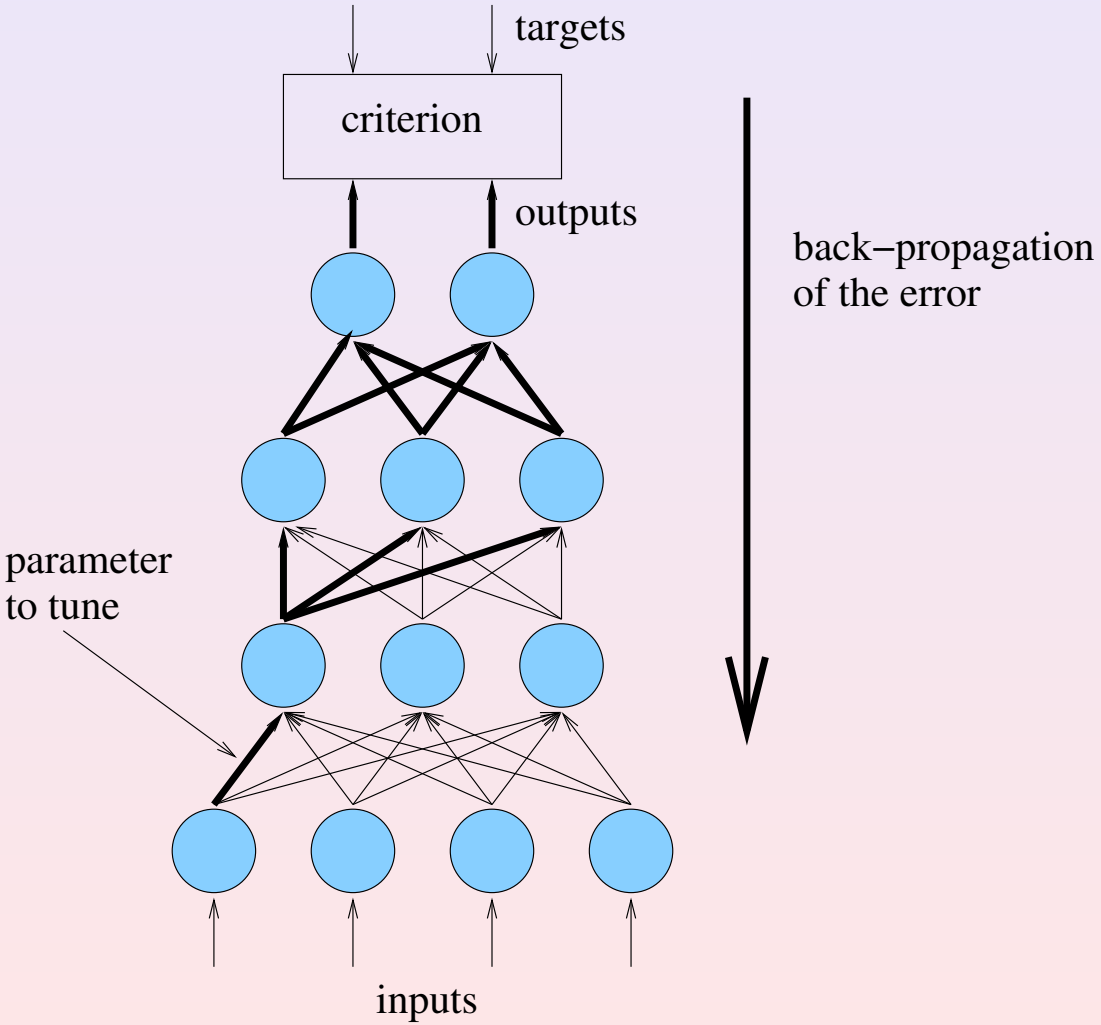
Gradient Descent: The Basics

Sum Rule

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



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: derivative wrt 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:

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

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

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

Gradient Descent: Other Layers

- Third: derivative wrt 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: derivative wrt the parameters of hidden layer y^l_j

$$\begin{aligned}\frac{\partial \hat{y}_i}{\partial w^l_{j,k}} &= \frac{\partial \hat{y}_i}{\partial y^l_j} \cdot \frac{\partial y^l_j}{\partial w^l_{j,k}} \\ &= \frac{\partial \hat{y}_i}{\partial y^l_j} \cdot y^{l-1}_k\end{aligned}$$

and

$$\begin{aligned}\frac{\partial \hat{y}_i}{\partial b^l_j} &= \frac{\partial \hat{y}_i}{\partial y^l_j} \cdot \frac{\partial y^l_j}{\partial b^l_j} \\ &= \frac{\partial \hat{y}_i}{\partial y^l_j} \cdot 1\end{aligned}$$

Gradient Descent: Global Algorithm

For each iteration

1. Initialize gradients $\frac{\partial C}{\partial \theta_i} = 0$ for each θ_i
2. For each example $z(p) = (x(p), y(p))$
   1. Forward phase: compute $\hat{y}(p) = \text{MLP}(x(p), \theta)$
   2. Compute $\frac{\partial L(y(p), \hat{y}(p))}{\partial \hat{y}(p)}$
   3. For each layer l from M to 1:
      1. Compute $\frac{\partial \hat{y}(p)}{\partial y_j^l}$
      2. Compute $\frac{\partial y_j^l}{\partial b_j^l}$ and $\frac{\partial y_j^l}{\partial w_{j,k}^l}$
      3. Accumulate gradients:

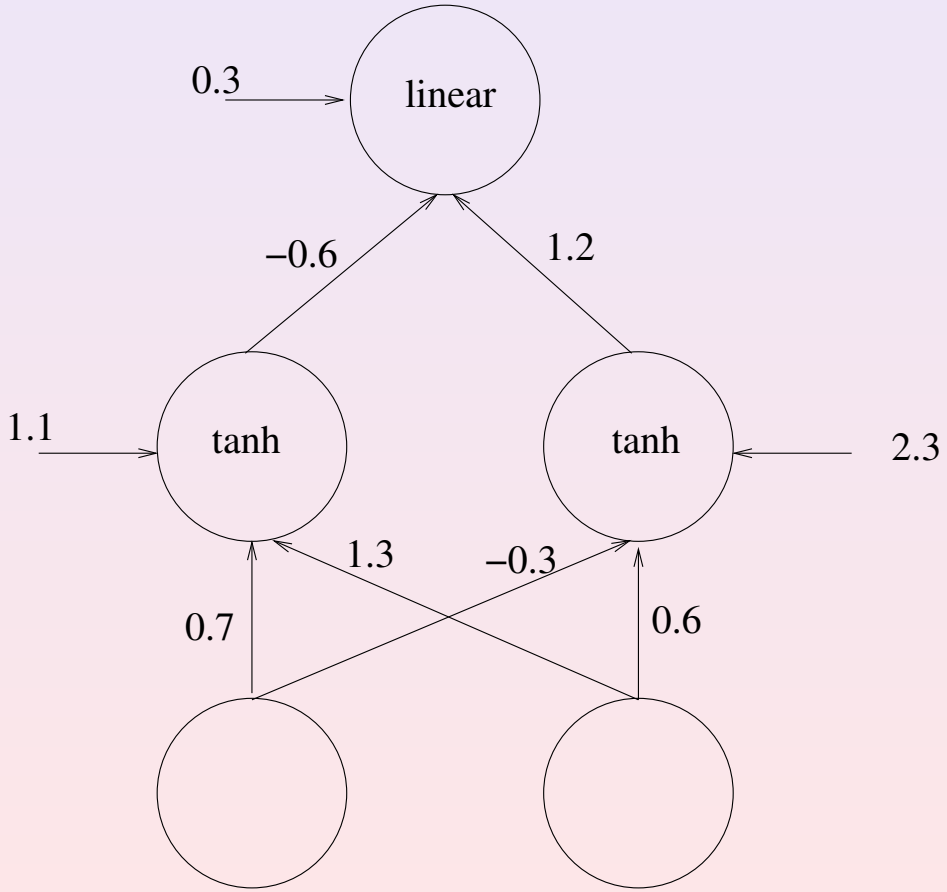
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l}$$

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

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

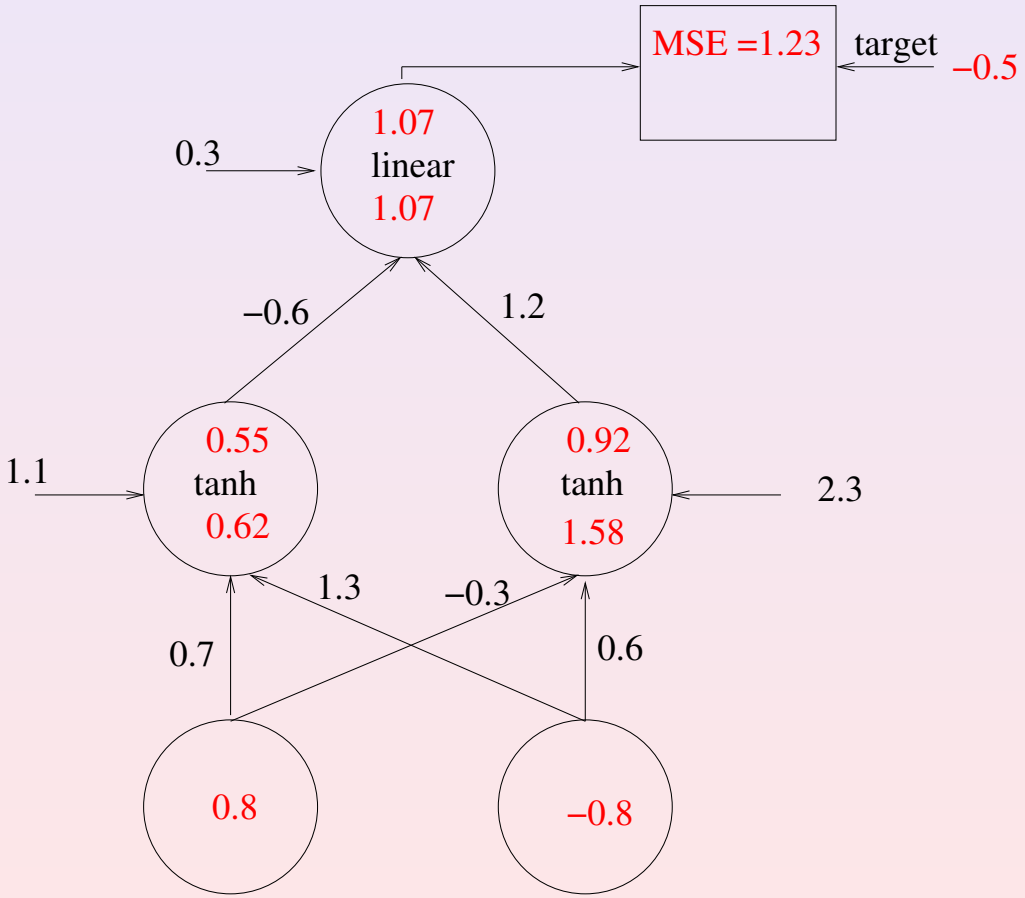
Gradient Descent: An Example (1)

Let us start with a simple MLP:



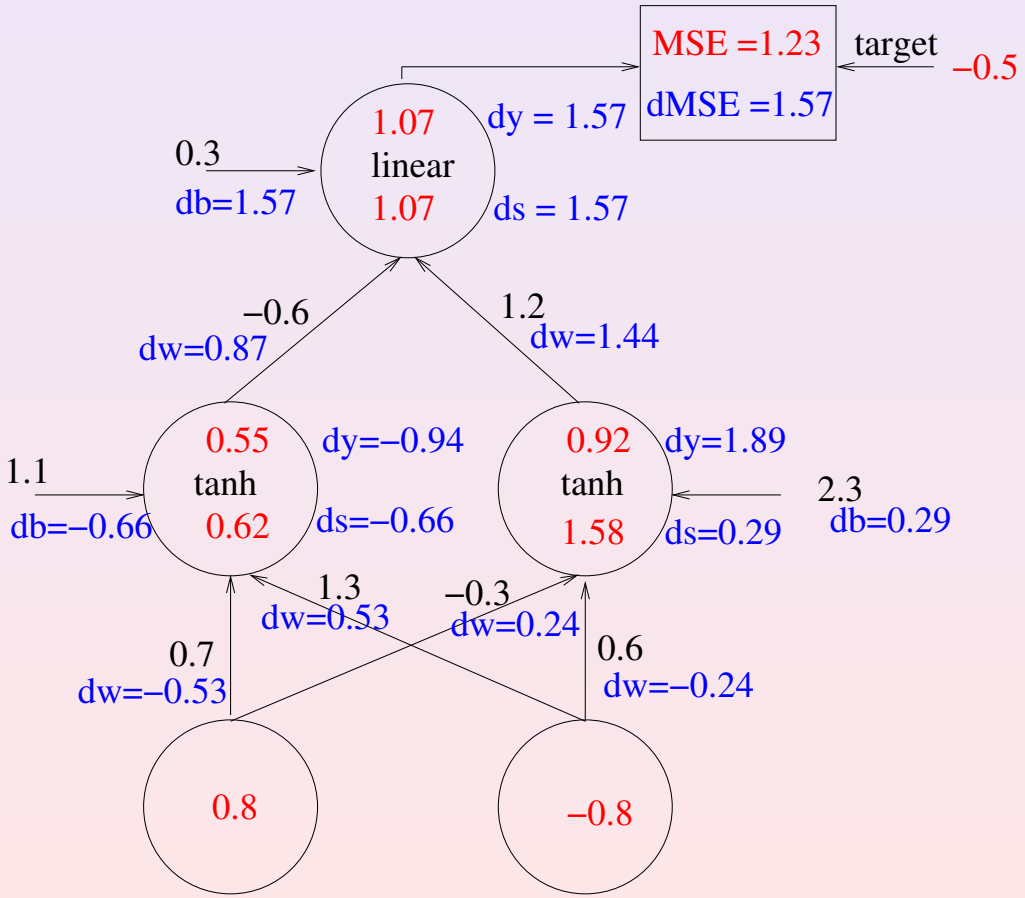
Gradient Descent: An Example (2)

We forward one example and compute its MSE:



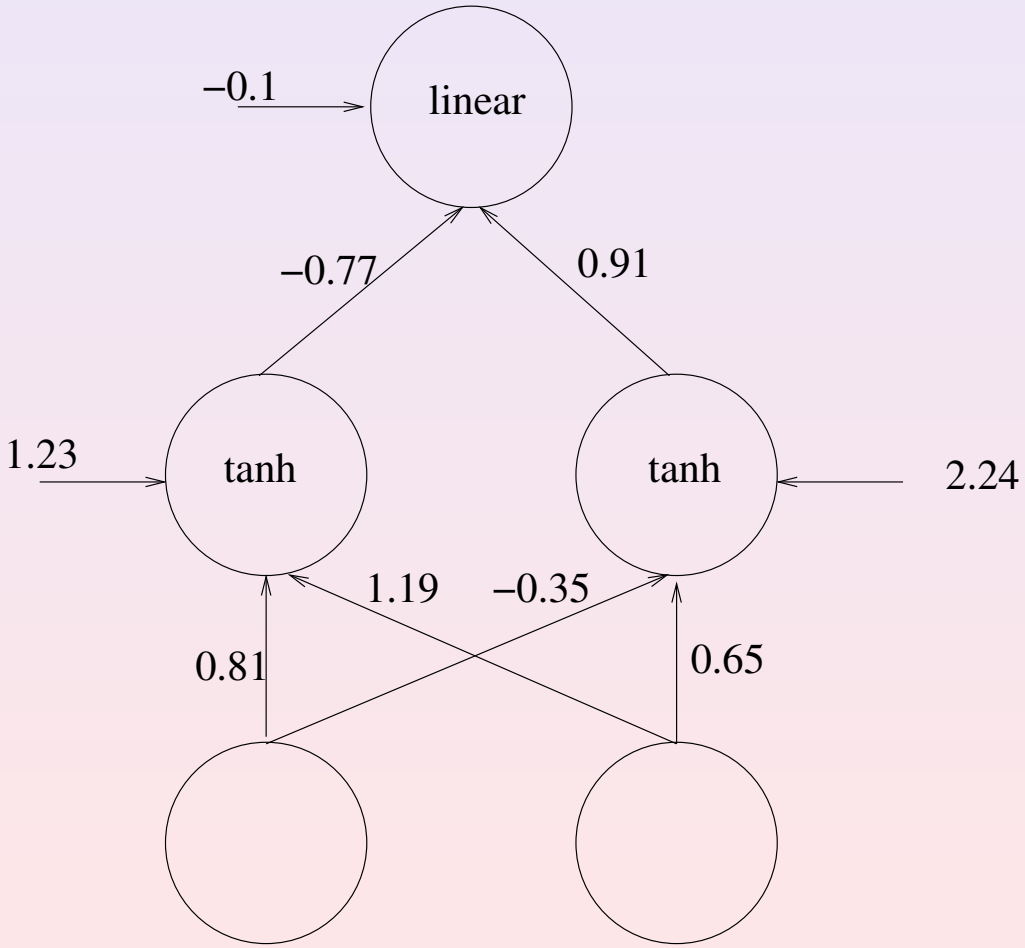
Gradient Descent: An Example (3)

We backpropagate the gradient everywhere:



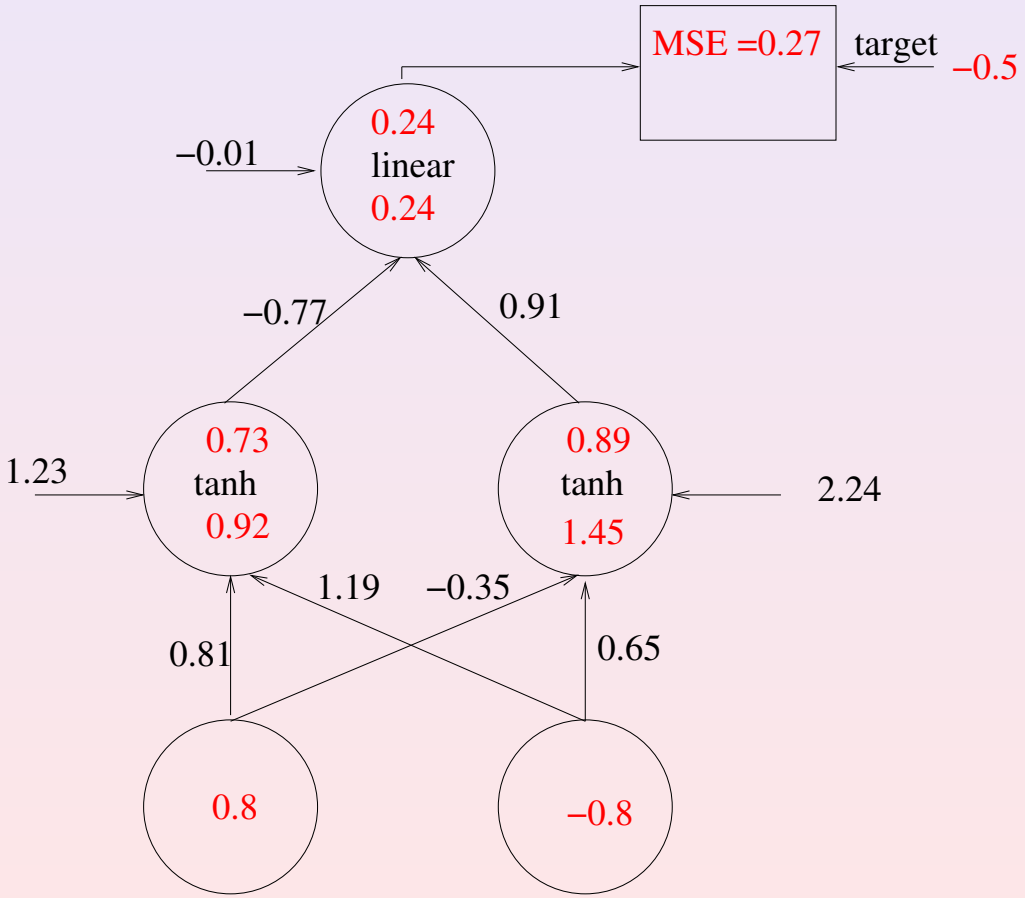
Gradient Descent: An Example (4)

We modify each parameter with learning rate 0.1:



Gradient Descent: An Example (5)

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



MLP are Universal Approximators

- It can be shown that, under reasonable assumptions, one can approximate any smooth function with an MLP with one layer of hidden units.
- First intuition:
  - Let us consider a classification task
  - Let us consider hard transfert functions for hidden units:

$$y = \text{step}(s) = \begin{cases} 1 & \text{if } s > 0 \\ 0 & \text{otherwise} \end{cases}$$

  - Let us consider linear transfert functions for output units:

$$y = s$$

  - First attempt:

$$\hat{y} = c + \sum_{i=1}^{N} v_i \cdot \text{sign}\left(\sum_{j=1}^{M} x_j \cdot w_{i,j} + b_i \right)$$

Illustration: Universal Approximators

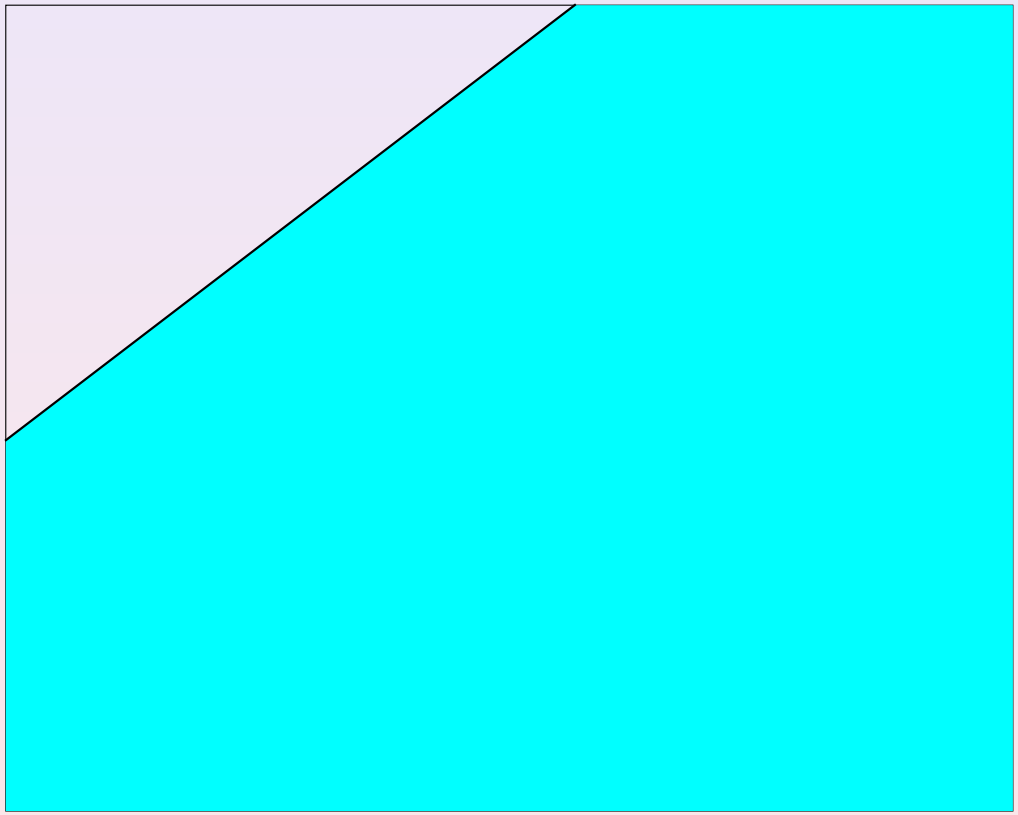


Illustration: Universal Approximators

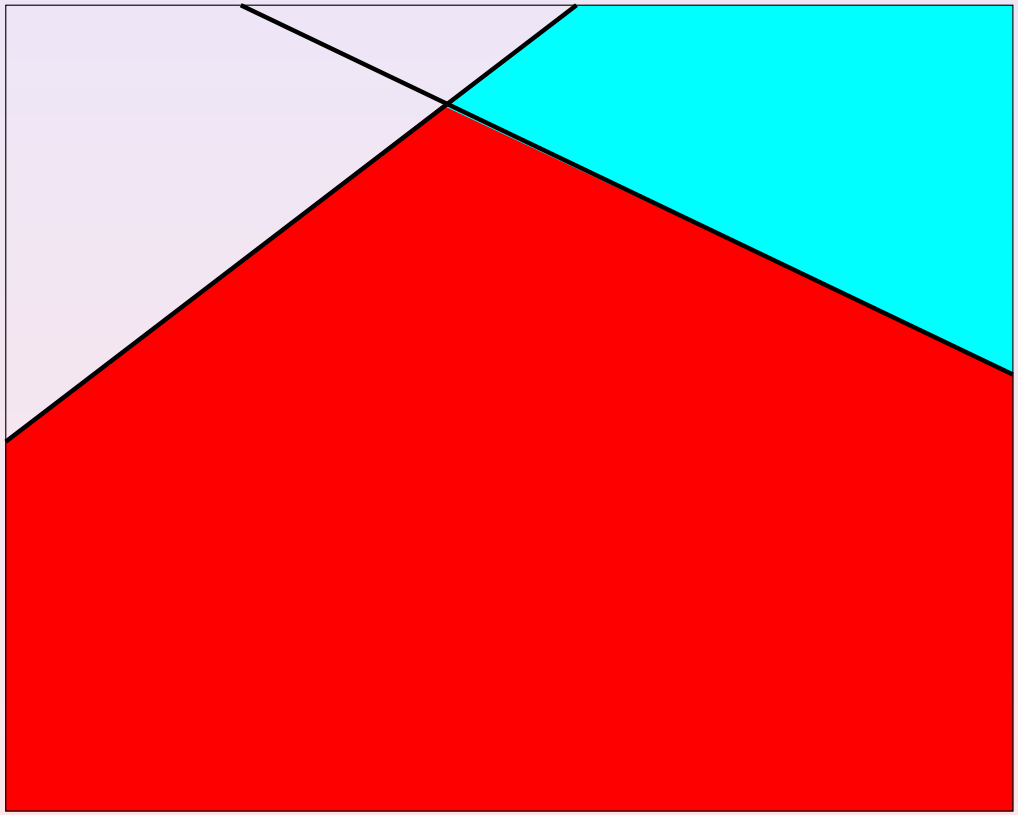


Illustration: Universal Approximators

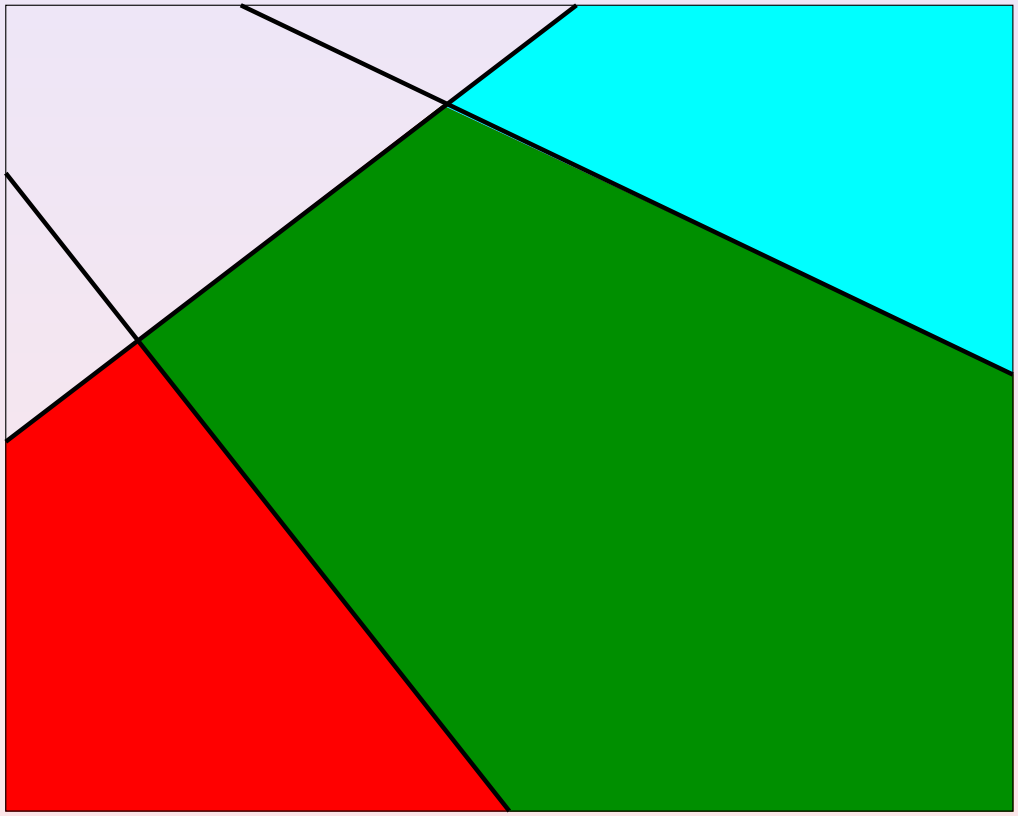


Illustration: Universal Approximators

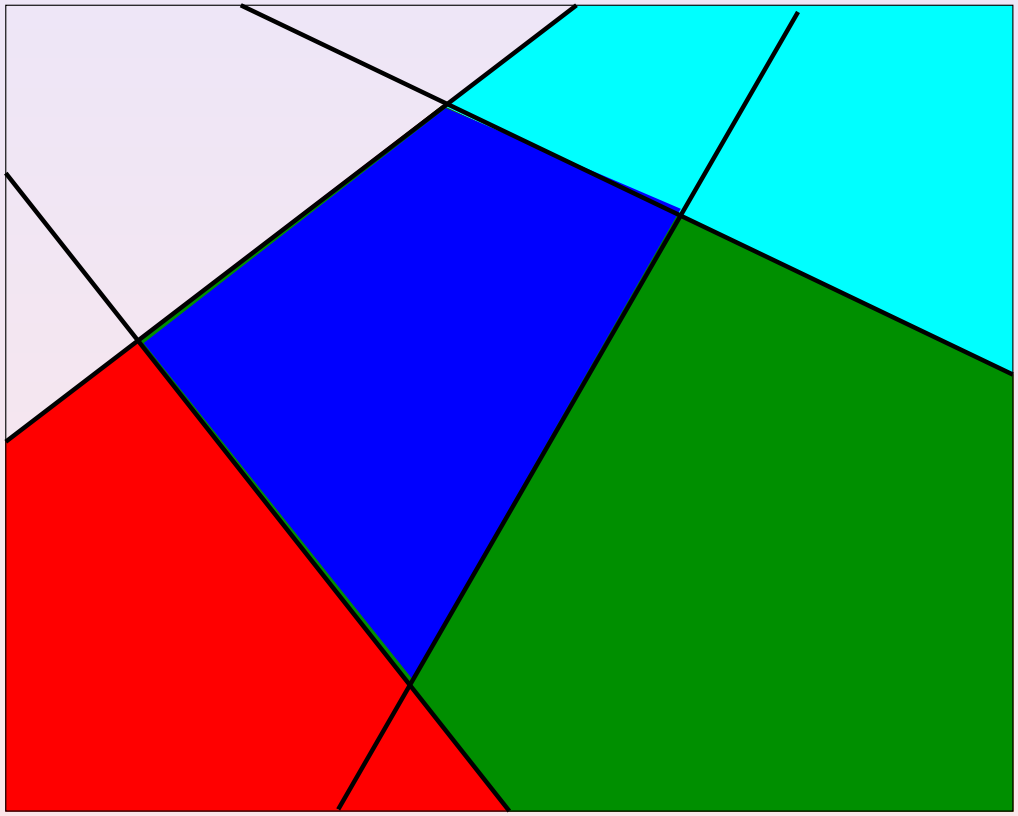
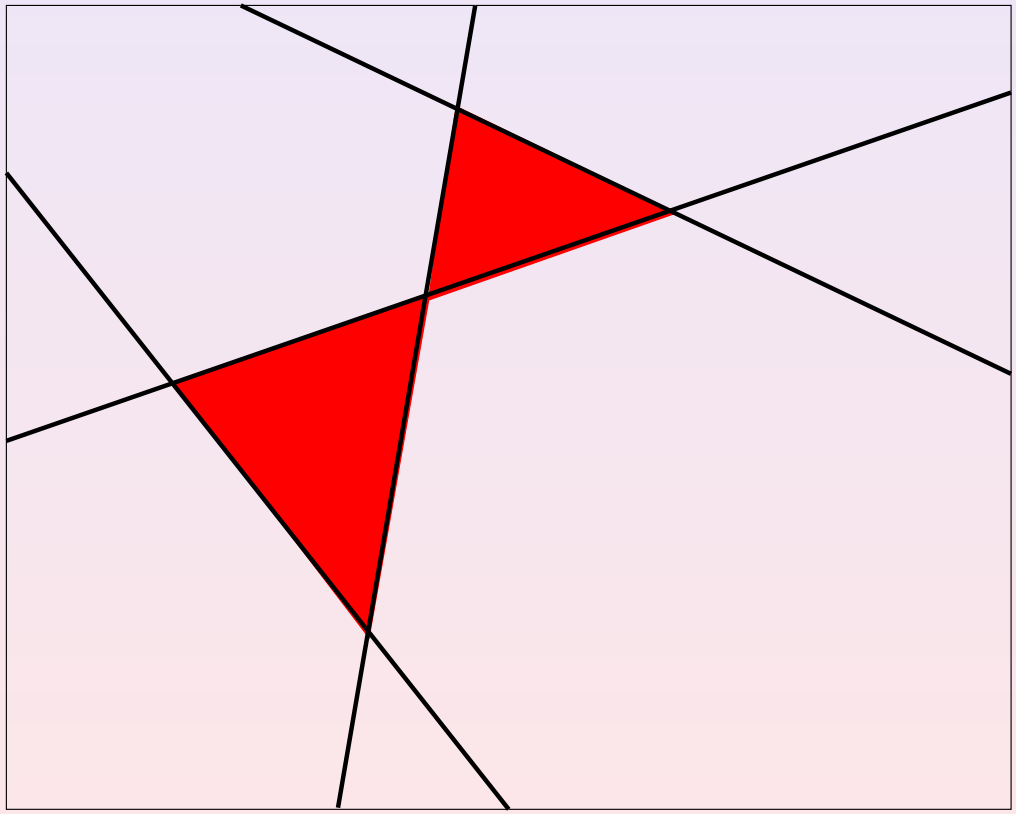


Illustration: Universal Approximators

... but what about that?



Universal Approximation by Cosines

- Let us consider simple functions of two variables $y(x_1, x_2)$
- Fourrier decomposition:

$$y(x_1, x_2) \simeq \sum_s A_s(x_1) \cos(s x_2)$$

where coefficients of A_s are functions of x_1.

- Further Fourrier decomposition:

$$y(x_1, x_2) \simeq \sum_s \sum_l A_{s,l} \cos(l x_1) \cos(s x_2)$$

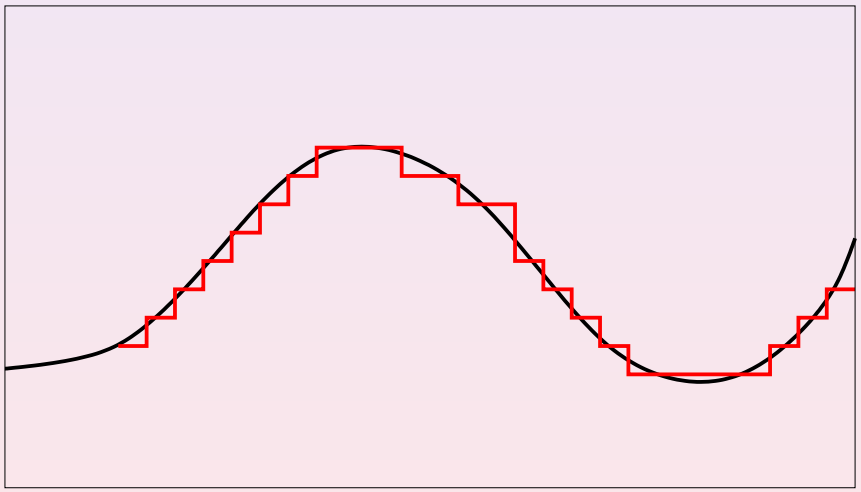
- We know that $\cos(\alpha)\cos(\beta) = \frac{1}{2}\cos(\alpha+\beta) + \frac{1}{2}\cos(\alpha-\beta)$:

$$y(x_1, x_2) \simeq \sum_s \sum_l A_{s,l} \left[\frac{1}{2} \cos(l x_1 + s x_2) + \frac{1}{2} \cos(l x_1 - s x_2) \right]$$

Universal Approximation by Cosines

- The cos function can be approximated with linear combinations of step functions:

$$f(z) = f_0 + \sum_i (f_{i+1} - f_i)\text{step}(z - z_i)$$



- So $y(x_1, x_2)$ can be approximated by a linear combination of step functions whose arguments are linear combinations of x_1 and x_2, and which can be approximated by tanh functions.

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 $\{0, 1\}$
  - if the model is correctly trained (with, for instance MSE criterion)

    $$\Longrightarrow \hat{y}(x) = E[Y|X = x] = 1 \cdot P(Y = 1|X = x) + 0 \cdot P(Y = 0|X = x)$$

  - the output will thus 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: $\hat{y}_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$
    - 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

Error Correcting Output Codes

- Given our 4-class problem represented 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

- 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

What is a Good Error Correcting Output Code

- How to devise a good error correcting output code?
- Maximize the minimum Hamming distance between any pair of code words.
- A good ECOC should satisfy two properties:
  - Row separation. (Hamming distance)
  - Column separation. Column functions should be as uncorrelated as possible with each other.

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.*

Content:

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

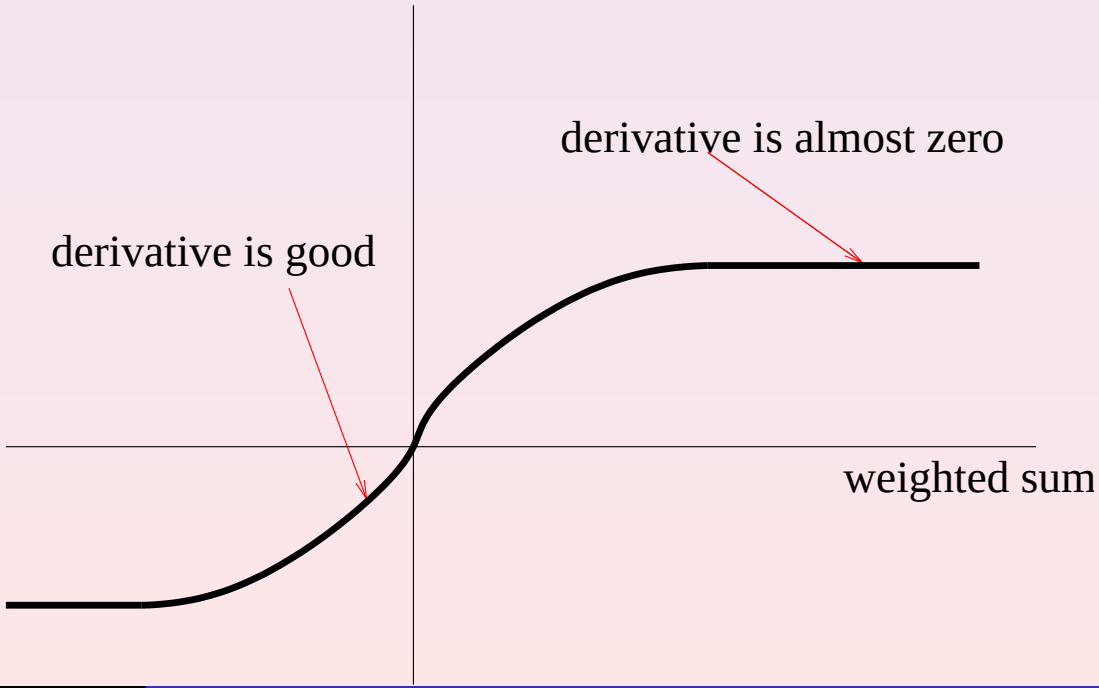
Stochastic Gradient Descent

- 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:

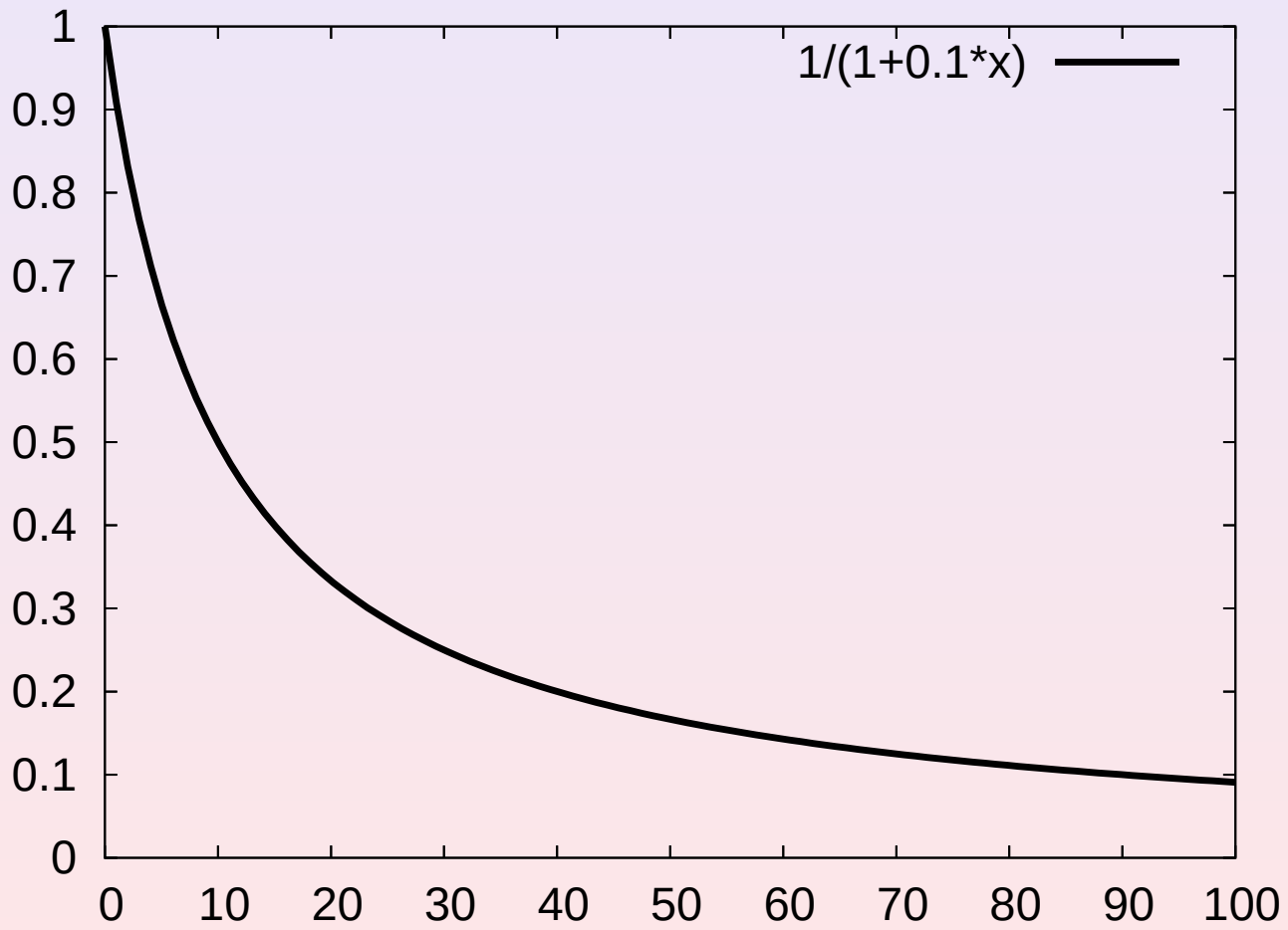
- See Leon Bottou's thesis for details
- input data: normalized with zero mean and unit variance,
- targets:
  - regression: normalized with zero mean and unit variance,
  - classification:
    - output transfer function is tanh: 0.6 and -0.6
    - output transfer function is sigmoid: 0.8 and 0.2
    - 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 η ?
- If η is too big: the optimization diverges
- If η 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 η_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 β controls the weight decay.

- Easy to implement:

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

Examples of Training Criteria

- Mean-squared error, for regression:

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

- Cross-entropy criterion, for classification (targets $\in \{-1, 1\}$):

$$L(y, \hat{y}) = \log(1 + \exp(-y_i \hat{y}_i))$$

- Hard version:

$$L(y, \hat{y}) = \|1 - y_i \hat{y}_i\|_+$$

Examples of Training Criteria