Multi Layer Perceptron:
An Advanced Introduction

Prof Sébastien Marcel
Senior researcher
www.idiap.ch/~marcel

Idiap Research Institute
Martigny, Switzerland
www.idiap.ch

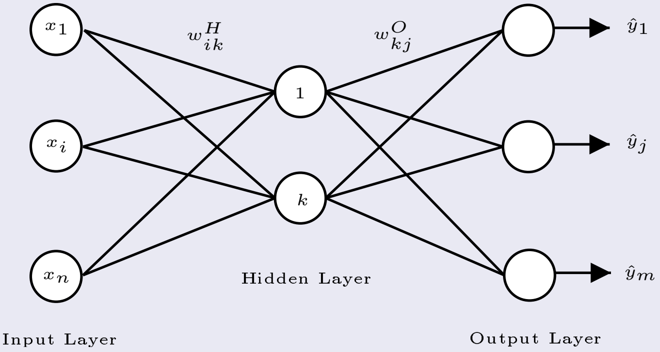
January 25, 2010

Outline

The Multi Layer Perceptron

MLP

It contains 1 input layer, 1 or several hidden layer and 1 output layer:



It can approximate any continuous functions !

The Multi Layer Perceptron

A MLP is a function: $\hat{y} = MLP(x; W)$

W is the set of parameters $\{w_{ij}^l, w_{i0}^l\}\ \forall i, j, l$

For each unit i on layer l of the MLP

- integration: $a_i^l = \sum_j^{H_l} y_j^{l-1} w_{ij}^l + w_{i0}^l$,
- transfer: $y_i^l = f(a_i^l)$ where $f(x) = tanh(x)$ or $\frac{1}{1+exp(-x)}$ or x

Input/Output limit cases

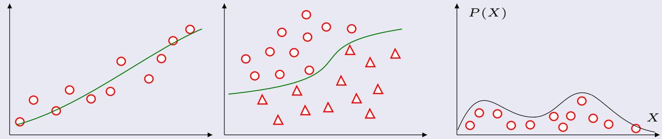
- on the input layer ($l = 0$) $y_i^l = x_i\ \forall i = 1..n$,
- on the output layer ($l = L$) $\hat{y}_i = y_i^L\ \forall i = 1..m$.

The Multi Layer Perceptron

3 forms of data for 3 types for problems

The data $D_P = \{z_1, z_2, ..., z_P\} \in \mathcal{Z}$ is independently and identically distributed and is drawn from an unknown distribution $p(Z)$

- classification: $Z = (X, Y) \in \mathbb{R}^n \times \{-1, 1\}$
- regression: $Z = (X, Y) \in \mathbb{R}^n \times \mathbb{R}^m$
- density estimation: $Z \in \mathbb{R}^n$



Outline

1. The Multi Layer Perceptron
2. Training
   - Cost function and Criterion
   - Gradient Descent
   - Gradient Descent: Example of Calculus
3. More about classification and MLP tricks
4. Conclusion

Outline

Cost function and Criterion

The goal is to minimize a cost function C over the set of data D_P:

$$C(D_P, W) = \sum_{p=1}^{P} L(y(p), \hat{y}(p))$$

- $y(p)$ is the output target vector for example p,
- $\hat{y}$ is the output of the MLP ($\hat{y} = MLP(x; W)$),
- $x(p)$ is the input vector for example p (let's omit p).
- L is a criterion to optimize such as the mean squared error (MSE):

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

Outline

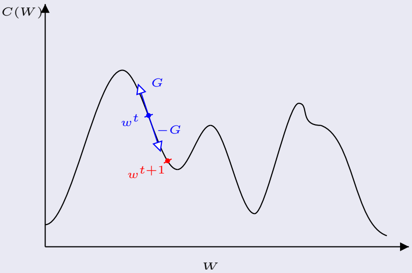
Gradient Descent

Gradient Descent

the gradient descent is an iterative procedure to modify the weights:

$$W^{t+1} = W^t - \eta \frac{\partial C(D, W^t)}{\partial W^t}$$

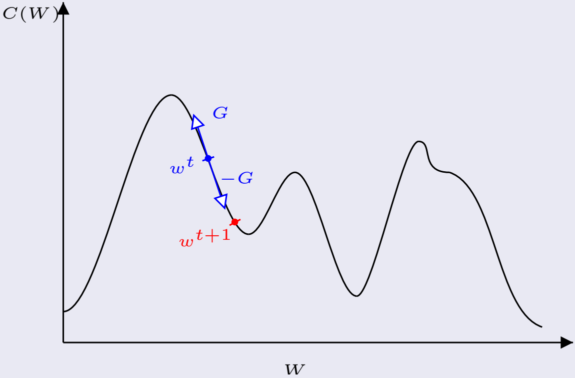
where η is the learning rate (neither too small or too big)



Gradient Descent

Gradient Descent

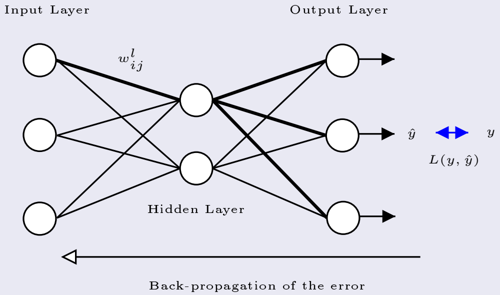
the goal is to "move" w^t in the opposite direction of the gradient to reach the global minimum.



Gradient Descent

Gradient computing and updating

Computing the gradient and updating the weights is performed from the output neurons to the input neurons, in the inverse order of the propagation (Gradient Back-Propagation).

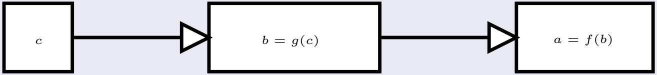


Gradient Descent

the chain rule

let us denote $a = f(b)$ and $b = g(c)$, then

$$\frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} = f'(b) \cdot g'(c) \tag{1}$$



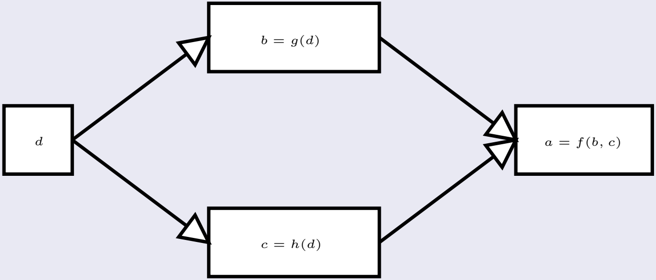
Gradient Descent

the sum rule

let us denote $a = f(b, c)$, $b = g(d)$ and $c = h(d)$, then

$$\frac{\partial a}{\partial d} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial d} + \frac{\partial a}{\partial c} \cdot \frac{\partial c}{\partial d} \quad (2)$$

$$= \frac{\partial f(b, c)}{\partial b} \cdot g'(d) + \frac{\partial f(b, c)}{\partial c} \cdot h'(d) \quad (3)$$



Gradient Descent

cost function derivative $\Leftrightarrow$ criterion derivative:

$$\frac{\partial C(D_P, W)}{\partial W} \Leftrightarrow \frac{\partial C_p(W)}{\partial W}$$

remember that:

$$C(D_P, W) = \sum_{p=1}^{P} L(y(p), \hat{y}(p))$$

$$C_p(W) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{m}(y_i - y_i^L)^2$$

Gradient Descent

computes the derivative of the criterion with respect to weights w_{ij}^l

$$\begin{aligned}
\frac{\partial C_p(W)}{\partial w_{ij}^l} &= \frac{\partial C_p(W)}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial w_{ij}^l} \\
&= \frac{\partial C_p(W)}{\partial a_j^l} \cdot y_i^{l-1} \\
&= \frac{\partial C_p(W)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial a_j^l} \cdot y_i^{l-1} \\
&= \Phi_j^l \cdot f'(a_j^l) \cdot y_i^{l-1} \qquad (4)
\end{aligned}$$

now let's compute Φ_j^l

Gradient Descent

for $l = L$ (output layer):

$$\begin{aligned} \Phi_j^L &= \frac{\partial C_p(W)}{\partial y_j^L} \\ &= \frac{\partial \frac{1}{2} \sum_{i=1}^{m} (y_i - y_i^L)^2}{\partial y_j^L} \\ &= (y_j^L - y_j) \end{aligned} \tag{5}$$

Thus, we compute for each output neuron j, the difference between the output y_j^L and the target y_j (for example p).

Gradient Descent

for $l \neq L$ (hidden layers):

$$\begin{aligned}
\Phi_j^l &= \frac{\partial C_p(W)}{\partial y_j^l} = \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a_k^{l+1}} \cdot \frac{\partial a_k^{l+1}}{\partial y_j^l} \\
&= \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a_k^{l+1}} \cdot \frac{\partial \sum_{i=1}^{H_l} w_{ik}^{l+1} y_i^l}{\partial y_j^l} \\
&= \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a_k^{l+1}} \cdot w_{jk}^{l+1} = \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial y_k^{l+1}} \cdot \frac{\partial y_k^{l+1}}{\partial a_k^{l+1}} \cdot w_{jk}^{l+1} \\
&= \sum_{k=1}^{H_{l+1}} \Phi_k^{l+1} \cdot f'(a_k^{l+1}) \cdot w_{jk}^{l+1} \qquad (6)
\end{aligned}$$

Thus, Φ_j^l can be computed using layer $l+1$.

Gradient Descent

For each weight, the update is done using the following rule:

$$w_{ij,t+1}^{l} = w_{ij,t}^{l} - \eta \cdot \frac{\partial C_p}{\partial w_{ij,t}^{l}} \quad (7)$$

where η is the learning rate, and $\frac{\partial C_p}{\partial w_{ij,t}^{l}}$ is defined by:

$$\frac{\partial C_p}{\partial w_{ij,t}^{l}} = \begin{cases} l = L & : \quad f'(a_j^l) \cdot y_i^{l-1} \cdot (y_j^L - y_j) \\ l \neq L & : \quad f'(a_j^l) \cdot y_i^{l-1} \cdot \left[\sum_{k=1}^{H_{l+1}} \Phi_k^{l+1} \cdot f'(a_k^{l+1}) \cdot w_{jk}^{l+1}\right] \end{cases}$$

Outline

1. The Multi Layer Perceptron

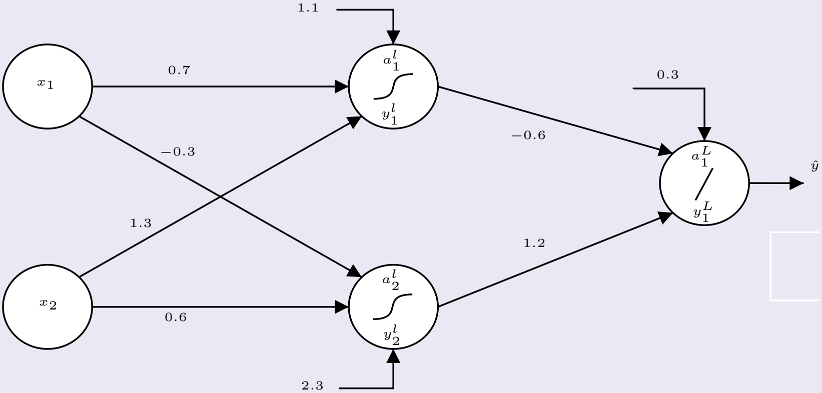
2. Training
   - Cost function and Criterion
   - Gradient Descent
   - Gradient Descent: Example of Calculus

3. More about classification and MLP tricks

4. Conclusion

Gradient Descent: Example

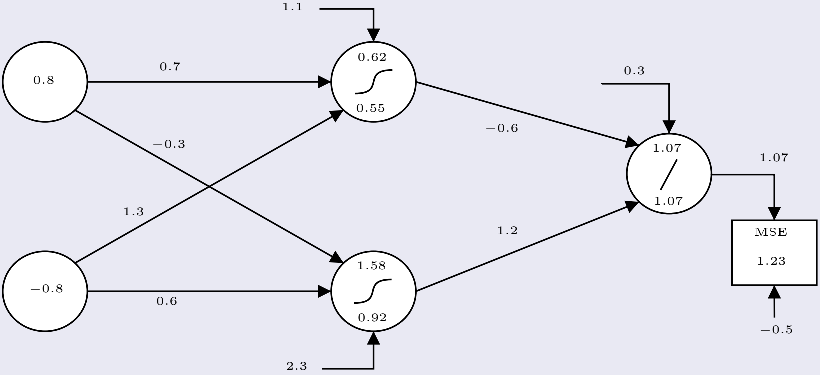
Initial MLP:



Note that: $y^L_1 = a^L_1$ and $y^l_j = tanh(a^l_j)$

Gradient Descent: Example

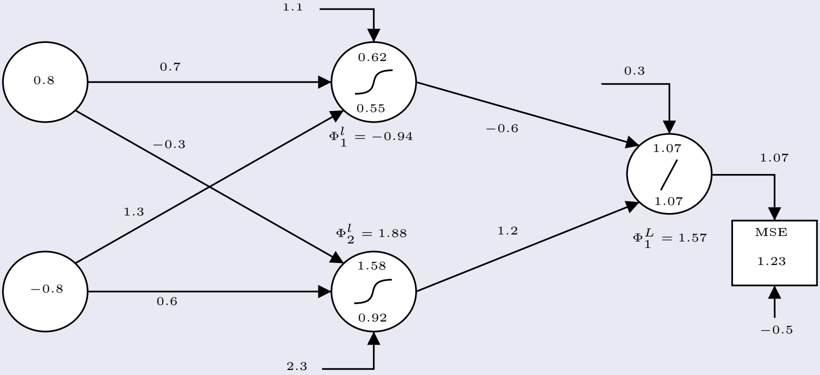
Forward:



Note that: $MSE = \frac{1}{2}\sum_j (y_j - y_j^L)^2$

Gradient Descent: Example

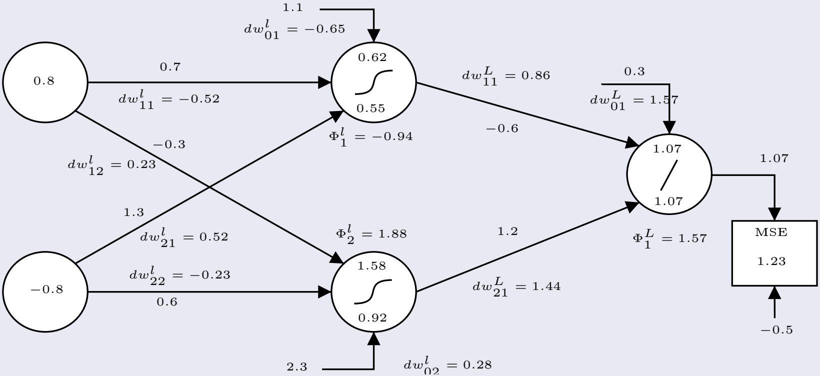
Backward:



Note that: $\Phi_j^L = (y_j^L - y_j)$, and that: $\Phi_j^l = \Phi_1^L \cdot f'(a_1^L) \cdot w_{j1}^L$.

Gradient Descent: Example

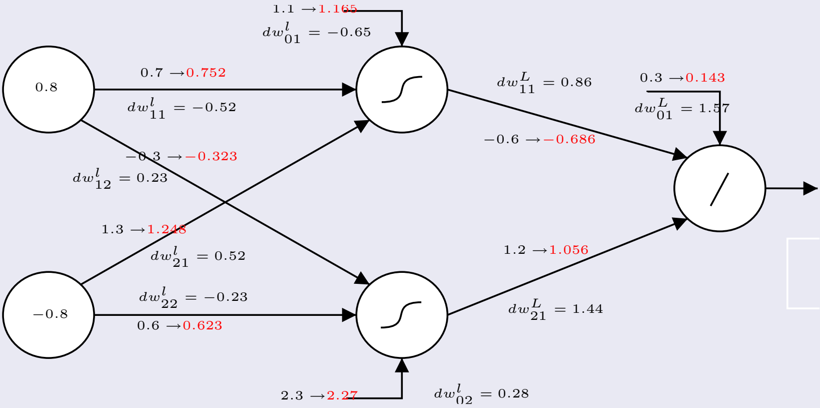
Backward (cont):



Note that: $\frac{\partial C}{\partial w^l_{ij}} = dw^l_{ij} = \Phi^l_j \cdot f'(a^l_j) \cdot y^{l-1}_i$, and that: $y^l_{0j} = a^l_{0j}$, $tanh'(a) = 1 - tanh(a)^2 = 1 - y^2$.

Gradient Descent: Example

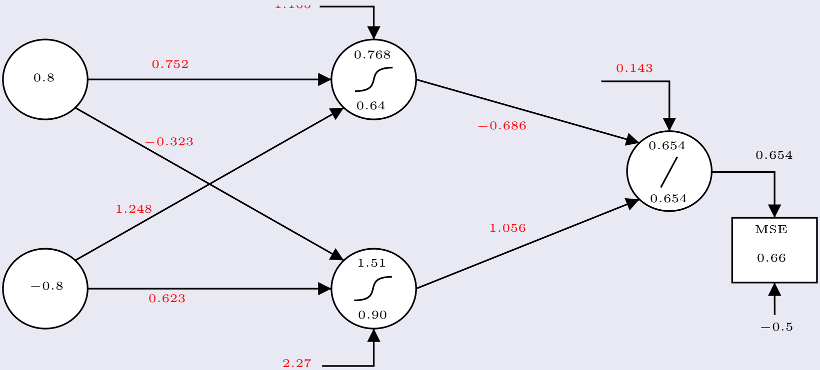
Update:



Note that: $w^l_{ij,t+1} = w^l_{ij,t} - \eta \cdot dw^l_{ij}$ with $\eta = 0.1$ for instance

Gradient Descent: Example

Re-Forward



Gradient Descent: Summary

For each iteration t

- Initialize the gradients $\frac{\partial C_p}{\partial w_{ij,t}^l}$ to 0
- For each example p $(x(p), y(p))$:
  1. Compute $\hat{y}(p) = MLP(x(p); W)$
  2. Compute $f'(a_j^L)$
  3. Compute Φ_j^L using Equation (5)
  4. Compute gradient $\frac{\partial C_p}{\partial w_{ij,t}^L}$ using Equation (4)
  5. Accumulate the above gradient
  6. For each layer l from $L-1$ to 1:
     - Compute $f'(a_j^l)$
     - Compute Φ_j^l using Equation (6)
     - Compute gradient $\frac{\partial C_p}{\partial w_{ij,t}^l}$ using Equation (4)
     - Accumulate the above gradient
- Update weights w_{ij}^l using Equation (7)

Outline

More about Classification

2-class problem

- use 1 output,
- encode the target as $\{+1, -1\}$ or $\{0, 1\}$ depending on the transfer function (linear, tanh, sigmoid),

multi-class problem

- use 1 output per class
- encode the target as $(0, ..., 1, ..., 0)$

MLP Tricks

Stochastic gradient

- use stochastic gradient instead of global (batch) gradient,
- adjust the weights at each example,

Initialization

to avoid the saturation of the transfer function (gradient tends toward 0)

Learning rate

- if too big the optimization diverges,
- if too small the optimization is very slow or is stuck into a local minima

more in the book: Orr, G. B. and Muler, K. "Neural Networks: Tricks of the Trade", Springer, 1998

MLP Tricks: initialization

input data

normalized with zero mean and unit variance,

targets

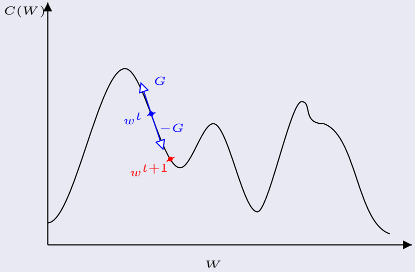
- for regression: normalized with zero mean and unit variance,
- for classification, if output transfer function is:
  - *tanh*(.) targets should be 0.6 and −0.6,
  - *sigmoid*(.) targets should be 0.8 and 0.2,
  - *linear*(.) targets should be 0.6 and −0.6.

weights w_{ij}

uniformly distributed in $\left[\frac{-1}{\sqrt{\text{fan in}_j}}, \frac{1}{\sqrt{\text{fan in}_j}}\right]$ where fan in_j is the number of units preceding unit j.

MLP Tricks: inertia momentum

to avoid to be stucked in a local minima



$$w^l_{ij,t+1} = w^l_{ij,t} - \eta \cdot dw^l_{ij} + \beta \cdot (w^l_{ij,t} - w^l_{ij,t-1})$$

where β is the inertia momentum rate

Outline

Future Lectures

Artificial Neural Networks

- Hopfield auto-associative memory
- Kohonen auto-organizing maps

Distribution Modelling

- Gaussian Mixture Models
- Hidden Markov Models
- Bayesian Networks

More

Support Vector Machines, Boosting, ...

References

Material

- This lecture is at www.idiap.ch/~marcel
- Machine learning Library: http://www.torch.ch

Books

- Bishop, C. "Neural Networks for Pattern Recognition", 1995
- Vapnik, V. "The Nature of Statistical Learning Theory", 1995
- Orr, G. B. and Muler, K. "Neural Networks: Tricks of the Trade", Springer, 1998

Extended Lectures on Machine Learning Algorithms

- Bengio, Y. www.iro.umontreal.ca/~pift6266/A03
- Bengio, S. www.idiap.ch/~bengio/lectures/index.html
- Jordan, M. www.cs.berkeley.edu/~jordan/courses.html