Artificial Neural Networks for Pattern Recognition: application to Face Detection and Recognition

Sébastien Marcel
marcel@idiap.ch

IDIAP Research Institute
Martigny, Switzerland

http://www.idiap.ch
Outline

- Introduction to Statistical Machine Learning
- Artificial Neural Networks
- Application Examples
Outline

- Introduction to Statistical Machine Learning
  - Learning and Learning
  - Capacity and Generalization
  - Regression, Classification and Density Estimation
  - Applications
- Artificial Neural Networks
- Application Examples
Learning and Learning

• Learning by heart:

\[
\begin{align*}
1 + 0 &= 1 & 1 \times 0 &= 0 \\
1 + 1 &= 2 & 1 \times 1 &= 1 \\
1 + 2 &= 3 & 1 \times 2 &= 2 \\
\vdots & \quad \vdots \\
\end{align*}
\]

any computer can do that!

• Learning by heart is not learning:
  – learning is “to gain knowledge or understanding of or skill in by study, instruction, or experience” (Merriam-Webster),
  – the difficulty of learning is to be able to generalize.
Capacity and Generalization

- Capacity: \# of parameters ($\theta$) required to fit the data with a function

\[ y = f(x, \theta) \]

- Generalization: the performance of the above function on unseen data
Regression, Classification and Density Estimation

- there are 3 kinds of problems:

  - Machine Learning Algorithms address the above problems using various tools:
    - Artificial Neural Networks,
    - Support Vector Machines,
    - Gaussian Mixture Models,
    - Hidden Markov Models,
    - and many others ...
Applications

• in Computer Vision:
  – Face detection, face recognition, face orientation estimation,
  – Gesture recognition,
  – Optical character recognition,
  – Handwritten recognition.

• in Speech Processing:
  – Speech recognition,
  – Speaker recognition.

• but also in Finance, Telecoms, Games, Robotic and more ...
Outline

- Introduction to Statistical Machine Learning
- **Artificial Neural Networks**
  - Biological Bases
  - History
  - The Formal Neuron
  - The Perceptron
  - Linearly Separable
  - The problem of XOR
  - The Multi Layer Perceptron
  - Cost function and Criterion
  - Gradient Descent
  - More about classification and MLP tricks
- Application Examples
ANN: Biological Bases

- the brain:
  - $10^{12}$ neurons massively connected,
  - 1 neuron is connected with $10^3$ others in average,

- the neuron:
  - core: where DNA belongs (approx. 30000 genes)
  - dendrite: receives a signal from other neurons (via their axon),
  - axon: propagates the signal to other neurons.

Now, let’s forget about biology and let’s come back to Mathematics!
History of ANN

- Mc Culloch and Pitts (1943): formal neuron inspired from biology
- Hebb (1949): the first training rule
- Rosenblatt (1962): the Perceptron
- Minsky and Papert (1969): limitations of Perceptron
- then, nothing during 16 years, research goes for symbolic AI
The Formal Neuron

- Mc Culloch and Pitts (1943):
  - \(x\) is the input \(\in \mathbb{R}^n\),
  - \(w_1 \ldots w_n\) are the weights,
  - \(w_0\) is the bias,
  - \(a\) is the result of the integration function \(f(x; w) = \sum_{i=1}^{n} w_i x_i + w_0\),
  - \(y\) is the output of the transfer function \(g(a) = tanh(a)\).
The Role of the Bias

- the formal neuron is a linear separator:

\[ x_1w_1 + x_2w_2 = 0 \]

\[ \iff x_2 = -\frac{w_1}{w_2}x_1 \]

- without the bias the linear separation is not always possible:
The Perceptron

- Rosenblatt (1962):
  - a retina: binary input of the perceptron,
  - association cells: “pre-processing”,
  - decision cells: linear units.
The Perceptron

- Rosenblatt (1962):

\[ x_1 \quad w_{11} \quad a_1 \quad \hat{y}_1 \\
\]

\[ x_i \quad w_{ij} \quad a_j \quad \hat{y}_j \\
\]

\[ x_n \quad w_{nj} \quad a_m \quad \hat{y}_m \\
\]

- Training rules:
  - \[ w_{ij}^{t+1} = w_{ij}^t + \alpha (\hat{y}_j - y_j) x_i \] (Rosenblatt)
  - \[ w_{ij}^{t+1} = w_{ij}^t + \alpha (\hat{y}_j - a_j) y_i \] (Widrow-Hoff)

- Limitations of Perceptron: restricted to linearly separable problems
Linearly Separable

- OR and AND: are linearly separable:

  ![Graph of OR and AND separability]

- one solution to AND: $w_1 = 1$, $w_2 = 1$ and $w_0 = 1.5$
- one solution to OR: $w_1 = 1$, $w_2 = 1$ and $w_0 = -0.5$
The problem of XOR

- XOR is not linearly separable:

- impossible to solve $x_1 w_1 + x_2 w_2 + w_0 = 0$, but what about multiple equations $x_1 w_{11} + x_2 w_{21} + w_{01} = 0$, $x_1 w_{12} + x_2 w_{22} + w_{02} = 0$, ...

- Solution: the Multi Layer Perceptron
The Multi Layer Perceptron

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

- It can approximate any continuous functions,

regression, classification and density estimation

- Problem: how to modify the weights?
Outline

• Introduction to Statistical Machine Learning
• Artificial Neural Networks
  – Biological Bases
  – History
  – The Formal Neuron
  – The Perceptron
  – Linearly Separable
  – The problem of XOR
  – The Multi Layer Perceptron
  – Cost function and Criterion
  – Gradient Descent
  – More about classification and MLP tricks
• Application Examples
The Multi Layer Perceptron

- A Multi Layer Perceptron (MLP) is a function: $\hat{y} = MLP(x; W)$,
- $W$ is the set of parameters $\{w^l_{ij}, w^l_{i0}\} \forall i, j, l$
- For each unit $i$ on layer $l$ of the MLP:
  - integration: $a^l_i = \sum_j^{H_l} y^l_{j-1} w^l_{ij} + w^l_{i0}$,
  - transfer: $y^l_i = f(a^l_i)$ where $f(x) = tanh(x)$ or $\frac{1}{1+exp(-x)}$ or $x$
- Input/Output limit cases:
  - on the input layer ($l = 0$) $y^l_i = x_i \forall i = 1..n$, 
  - on the output layer ($l = L$) $\hat{y}_i = y^L_i \forall i = 1..m$.
- the data $D_P = \{z_1, z_2, ..., z_P\} \in Z$ is independently and identically distributed and is drawn from an unknown distribution $p(Z)$,
- 3 forms of data for 3 types for problems:
  - 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: $Z \in \mathbb{R}^n$
Cost function and Criterion

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

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

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

$$MSE(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$
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 \( \eta \) is the learning rate (neither too small or too big)

- the goal is to “move” \( w^t \) in the opposite direction of the gradient to reach the global minimum.
Gradient Descent

- 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).

\[ w_{ij} \]

\[ \hat{y} \leftrightarrow y \]

\[ L(y, \hat{y}) \]

Back-propagation of the error
\begin{itemize}
  \item the chain rule:
    \begin{itemize}
      \item let us denote $a = f(b)$ and $b = g(c)$
      \item then
      \end{itemize}
      \[
      \frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} = f'(b) \cdot g'(c)
      \] (1)
\end{itemize}
\[
\begin{array}{c}
\text{c} \\
\text{---} \\
\text{---} \\
\text{b = g(c)} \\
\text{---} \\
\text{---} \\
\text{a = f(b)} \\
\end{array}
\]
the sum rule:
- let us denote $a = f(b, c)$, $b = g(d)$ and $c = h(d)$,
- then

$$\frac{\partial a}{\partial d} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial d} + \frac{\partial a}{\partial c} \cdot \frac{\partial c}{\partial d}$$

$$= \frac{\partial f(b, c)}{\partial b} \cdot g'(d) + \frac{\partial f(b, c)}{\partial c} \cdot h'(d)$$
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 $w_{ij}^l$.

\[
\frac{\partial C_p(W)}{\partial w_{ij}^l} = \frac{\partial C_p(W)}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial w_{ij}^l} = \frac{\partial C_p(W)}{\partial a_j^l} \cdot y_i^{l-1} = \frac{\partial C_p(W)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial a_j^l} \cdot y_i^{l-1} = \Phi_j^l \cdot f'(a_j^l) \cdot y_i^{l-1}
\]

- now let’s compute $\Phi_j^l$
Gradient Descent

- for $l = L$ (output layer):

$$
\Phi_j^L = \frac{\partial C_p(W)}{\partial y_j^L}
= \frac{\partial \frac{1}{2} \sum_{i=1}^{m} (y_i - y_i^L)^2}{\partial y_j^L}
= (y_j^L - y_j)
$$

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):

\[
\Phi^l_j = \frac{\partial C_p(W)}{\partial y^l_j} = \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a^{l+1}_k} \cdot \frac{\partial a^{l+1}_k}{\partial y^l_j}
\]

\[
= \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a^{l+1}_k} \cdot \partial \sum_{i=1}^{H_l} w^{l+1}_{ik} y^l_i
\]

\[
= \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial a^{l+1}_k} \cdot w^{l+1}_{jk} = \sum_{k=1}^{H_{l+1}} \frac{\partial C_p(W)}{\partial y^{l+1}_k} \cdot \frac{\partial y^{l+1}_k}{\partial a^{l+1}_k} \cdot w^{l+1}_{jk}
\]

\[
= \sum_{k=1}^{H_{l+1}} \Phi^{l+1}_k \cdot f'(a^{l+1}_k) \cdot w^{l+1}_{jk}
\]

(6)

Thus, $\Phi^l_j$ can be computed using layer $l + 1$. 
Gradient Descent

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

\[
w_{ij,t+1}^l = w_{ij,t}^l - \eta \cdot \frac{\partial C_p}{\partial w_{ij,t}^l}
\]

(7)

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

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

- Initial MLP:

- Note that: $y_1^L = a_1^L$ and $y_j^l = \tanh(a_j^l)$
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: $\Phi_j^l = \Phi_1^L \cdot f'(a_1^L) \cdot w_j^L$. 
Gradient Descent: Example

- Backward (cont):

\[ \frac{\partial C}{\partial w_{ij}^l} = dw_{ij}^l = \Phi_j^l \cdot f'(a_j^l) \cdot y_i^{l-1}, \]

- and that: \( y_{0j}^l = a_{0j}^l, \tan h'(a) = 1 - \tan h(a)^2 = 1 - y^2. \)
Gradient Descent: Example

- Update:

\[
\begin{align*}
0.8 & \quad dw_{11}^l = -0.52 \\
& \quad dw_{12}^l = 0.23 \\
& \quad 0.7 \rightarrow 0.752 \\
-0.8 & \quad dw_{21}^l = 0.52 \\
& \quad dw_{22}^l = -0.23 \\
& \quad 0.6 \rightarrow 0.623
\end{align*}
\]

\[
\begin{align*}
1.1 & \rightarrow 1.165 \\
& \quad dw_{01}^l = -0.65 \\
& \quad 1.3 \rightarrow 1.248 \\
-0.6 & \rightarrow -0.686 \\
& \quad 2.3 \rightarrow 2.27 \\
& \quad dw_{02}^l = 0.28 \\
& \quad dw_{11}^L = 0.86 \\
& \quad dw_{01}^L = 1.57 \\
& \quad 1.2 \rightarrow 1.056 \\
& \quad dw_{21}^L = 1.44
\end{align*}
\]

- Note that: \( w_{ij,t+1}^l = w_{ij,t}^l - \eta \cdot dw_{ij}^l \) with \( \eta = 0.1 \) for instance
Gradient Descent: Example

- Re-Forward:
Gradient Descent: Summary

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

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,
    * \( \text{sigmoid}(.) \) targets should be 0.8 and 0.2,
    * \( \text{linear}(.) \) 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 \( \text{fan in}_j \) is the number of units preceding unit \( j \).
MLP Tricks: inertia momentum

• to avoid to be stucked in a local minima:

\[
w_{ij,t+1} = w_{ij,t} - \eta \cdot dw_{ij} + \beta \cdot (w_{ij,t} - w_{ij,t-1})
\]

where \( \beta \) is the inertia momentum rate
Outline

- Introduction to Statistical Machine Learning
- Artificial Neural Networks
- Application Examples
  - Face detection
  - Face recognition
Face Processing using MLP

- the input $x$ of the MLP is a particular representation of the face image
- face representations:
  - Raw pixels:
  - Principal Componant subspace obtained by PCA:
- target coding:
  - Face detection: face ($+1$) vs non-face ($-1$),
  - Face authentication: client ($+1$) vs impostor ($-1$)
Face Processing using MLP

- Raw pixels:
  - let us denote the image $I$ of size $n = w \times h$,
  - then the input of the MLP is $x \in \mathbb{R}^n$,
  - for an image $30 \times 40$, a MLP with 90 hidden units has:

$$\text{(1200 inputs + 1 bias) \times 90 + 90 + 1 bias = 109291}$$

- **Warning !!:** a large number of parameters $\Rightarrow$ a large number of examples which is not always possible
- **Solution:** reduce the dimensionality $m \ll n$ using Principal Component Analysis (PCA)
Face Processing using MLP

- Principal Component subspace obtained by PCA:
  - \( u = Wx \) where \( u \in \mathbb{R}^m \) and \( W \) is a \( m \times n \) matrix,
    \[
    \mu = \frac{1}{P} \sum_{i=1}^{P} x_i
    \]
    \[
    \Sigma = \frac{1}{P} \sum_{i=1}^{P} (x_i - \mu)(x_i - \mu)^T
    \]
  - compute the \( m \) eigenvectors \( e_1...e_m \) corresponding to the \( m \) largest non-zero eigenvalues \( (\Sigma - \alpha_i I)e_i = 0, i = 1..m \),
  - \( W = [e_1...e_m]^T \),
  - the input of the MLP for a given face \( x \) becomes \( u = Wx \)
Face Processing using MLP

- Select a threshold to take the final decision:
  - False Rejection ($FRR$): when the system rejects a face,
  - False Acceptance Rate ($FAR$): when the system accepts a non-face,
  - the decision threshold $\Theta$ chosen on a evaluation data set.
Future Lectures

- Artificial Neural Networks:
  - Hopfield auto-associative memory
  - Kohonen auto-organizing maps
- Gaussian Mixture Models
- Hidden Markov Models
- Support Vector Machines and links with MLP
References

- This lecture is at http://www.idiap.ch/~marcel
- Machine learning Library: http://www.torch.ch
- Books:
  - Bishop, C. “Neural Networks for Pattern Recognition”, 1995
- Extended Lectures on Machine Learning Algorithms:
  - Bengio, Y. http://www.iro.umontreal.ca/~pift6266/A03