Introduction to Neural Networks

Vincent Barra

LIMOS, UMR CNRS 6158, Blaise Pascal University, Clermont-Ferrand, FRANCE

December 19, 2015

What are ANN ?

Experiment

Pigeons are art experts (Watanabe et al., 1995).

- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist (e.g. Van Gogh)

Results

- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists
- Pigeons do not simply memorise the pictures

  → They can extract and recognise patterns (the 'style')
  → They generalise from the already seen to make predictions

What are ANN ?

Experiment

Pigeons are art experts (Watanabe et al., 1995).

- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist (e.g. Van Gogh)

Results

- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists
- Pigeons do not simply memorise the pictures

  → They can extract and recognise patterns (the 'style')
  → They generalise from the already seen to make predictions

NEURAL NETWORKS

This is what neural networks (biological and artificial) are good at (unlike conventional computer)

Connectionism

→ computer modeling approach to computation that is loosely based upon the architecture of the brain.

Many different models, but all include:

1. Multiple, individual nodes or units that operate at the same time (in parallel)
2. A network that connects the nodes together
3. Information is stored in a distributed fashion among the links that connect the nodes
4. Learning can occur with gradual changes in connection strength

NEURAL NETWORKS

This is what neural networks (biological and artificial) are good at (unlike conventional computer)

Connectionism

→ computer modeling approach to computation that is loosely based upon the architecture of the brain.

Many different models, but all include:

1. Multiple, individual nodes or units that operate at the same time (in parallel)
2. A network that connects the nodes together
3. Information is stored in a distributed fashion among the links that connect the nodes
4. Learning can occur with gradual changes in connection strength

- History traces back to the 50's but became popular in the 80's
- Peaked in the 90's. Today:
  - → Hundreds of variants
  - → Less a model of the actual brain than a useful tool, but still some debate
- Numerous applications
  - → Handwriting, face, speech recognition
  - → Vehicles that drive themselves
  - → Models of reading, sentence production, dreaming
- Debate:Can human consciousness or cognitive abilities be explained by a connectionist model or does it require the manipulation of symbols?

Brain vs. ANN

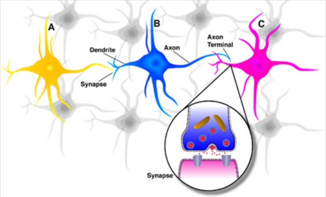
- $\approx$ 200 billion neurons, $\approx$ 32 trillion synapses
- Element size: 10^{-6} m
- Energy use: 25W
- Processing speed: 100 Hz
- Highly parallel, Distributed
- Fault Tolerant
- Learns: Yes
- Intelligent/Conscious: Usually !

- 1 billion bytes RAM but trillions of bytes on disk
- Element size: 10^{-9} m
- Energy use $\approx$ 60W (CPU)
- Processing speed: 10^9 Hz
- Serial (or paralell), Centralized
- Generally not Fault Tolerant
- Learns: some
- Intelligent/Conscious: no !

IDEA

Make the computer more robust, intelligent, and learn,...

MODEL



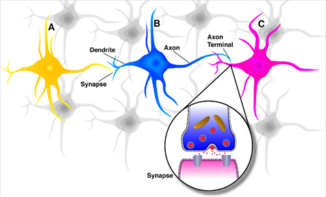
Although heterogeneous, at a low level the brain is composed of neurons

- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately summed
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)

Idea

Make the computer more robust, intelligent, and learn,...

Model



Although heterogeneous, at a low level the brain is composed of neurons

- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately summed
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)

Brains learn

- Altering strength between neurons
- Creating/deleting connections

Hebbs Postulate (Hebbian Learning)

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Long Term Potentiation (LTP)

- Cellular basis for learning and memory
- LTP is the long-lasting strengthening of the connection between two nerve cells in response to stimulation
- Discovered in many regions of the cortex

.

BRAINS LEARN

- Altering strength between neurons
- Creating/deleting connections

HEBBS POSTULATE (HEBBIAN LEARNING)

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

LONG TERM POTENTIATION (LTP)

- Cellular basis for learning and memory
- LTP is the long-lasting strengthening of the connection between two nerve cells in response to stimulation
- Discovered in many regions of the cortex

BRAINS LEARN

- Altering strength between neurons
- Creating/deleting connections

HEBBS POSTULATE (HEBBIAN LEARNING)

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

LONG TERM POTENTIATION (LTP)

- Cellular basis for learning and memory
- LTP is the long-lasting strengthening of the connection between two nerve cells in response to stimulation
- Discovered in many regions of the cortex

.

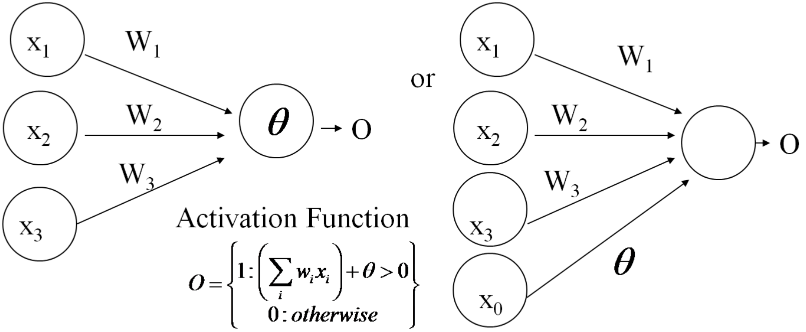
Perceptron - Introduction

Seminal paper

F Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychological review, 65:386-408, 1958

Definition

- Initial proposal of connectionist networks
- Essentially a linear discriminant composed of nodes, weights



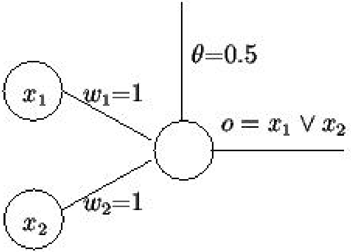
$$O = \begin{cases} 1 : \left(\sum_i w_i x_i\right) + \theta > 0 \\ 0 : otherwise \end{cases}$$

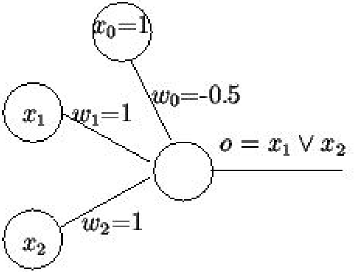
Intuitive training from a training set $E = \{(x^s, c^s), 1 \leq i \leq l\}$.
Assumptions: outputs are binary $\Rightarrow$ Correction of error

random initialization of w_i
repeat
  Pick up an example $(x^s, c^s) \in E$
  Compute the output o^s of the perceptron when presenting x^s
  for $i \in \{0 \cdots n\}$ **do**
    $w_i \leftarrow w_i + (c^s - o^s)x_i^s$
  end
until *test*;

EXAMPLE - BINARY OR

Step	w_0	w_1	w_2	Input	$\sum_{i=0}^{2} w_i x_i$	o	c	w_0	w_1	w_2
Init								0	1	-1
1	0	1	-1	100	0	0	0	0	1	-1
2	0	1	-1	101	-1	0	1	1	1	0
3	1	1	0	110	2	1	1	1	1	0
4	1	1	0	111	2	1	1	1	1	0
5	1	1	0	100	1	1	0	0	1	0
6	0	1	0	101	0	0	1	1	1	1
7	1	1	1	110	2	1	1	1	1	1
8	1	1	1	111	3	1	1	1	1	1
9	1	1	1	100	1	1	0	0	1	1
10	0	1	1	101	1	1	1	0	1	1





Properties

- Essentially a linear discriminant
- Perceptron theorem: If a linear discriminant exists that can separate the classes without error, the training procedure is guaranteed to find that line or plane.

But...

- Data is generally non linearly separable (in the original space) (example XOR).
- One solution: Minimize a classification error $\Rightarrow$ outputs become real (no thershold θ)
- Another solution: construct multiple layers of perceptrons to get around this problem.
- Another solution: transform the inputs using an appropriate mapping (e.g. kernel trick)

PROPERTIES

- Essentially a linear discriminant
- Perceptron theorem: If a linear discriminant exists that can separate the classes without error, the training procedure is guaranteed to find that line or plane.

BUT...

- Data is generally non linearly separable (in the original space) (example XOR).
- One solution: Minimize a classification error $\Rightarrow$ outputs become real (no thershold θ)
- Another solution: construct multiple layers of perceptrons to get around this problem.
- Another solution: transform the inputs using an appropriate mapping (e.g. kernel trick)

Input: training set $E = \{(x^s, c^s), 1 \leq i \leq l\}$.
random initialization of w_i
repeat
  for $i \in \{1 \cdots n\}$ **do**
    $\Delta w_i \leftarrow 0$
  end
  for *every sample* $(x^s, c^s) \in E$ **do**
    Compute o^s
    for $i \in \{1 \cdots n\}$ **do**
      $\Delta w_i \leftarrow \Delta w_i + \epsilon(c^s - o^s)x_i^s$
    end
  end
  for $i \in \{1 \cdots n\}$ **do**
    $w_i \leftarrow w_i + \Delta w_i$
  end
until *test*;

Variant: Widrow Hoff Rule (Adaline)

random initialization of w_i
repeat
    Pick up an sample $(x^s, c^s) \in E$
    Compute o^s
    for $i \in \{1 \cdots n\}$ **do**
        $w_i \leftarrow w_i + \epsilon(c^s - o^s)x_i^s$
    end
until *test*;

Last words on Perceptron

- Oscillation problems
- Gradient descent or Adaline will converge to some minimum even if the classes are not linearly separable, unlike the earlier perceptron training method

And so

- Another solution: construct multiple layers of perceptrons to get around this problem
- Another solution: transform the inputs using an appropriate mapping (e.g. kernel trick)

Variant: Widrow Hoff Rule (Adaline)

random initialization of w_i
repeat
  Pick up an sample $(x^s, c^s) \in E$
  Compute o^s
  for $i \in \{1 \cdots n\}$ **do**
    $w_i \leftarrow w_i + \epsilon(c^s - o^s)x_i^s$
  end
until *test*;

Last words on Perceptron

- Oscillation problems
- Gradient descent or Adaline will converge to some minimum even if the classes are not linearly separable, unlike the earlier perceptron training method

And so

- Another solution: construct multiple layers of perceptrons to get around this problem
- Another solution: transform the inputs using an appropriate mapping (e.g. kernel trick)

Variant: Widrow Hoff Rule (Adaline)

random initialization of w_i
repeat
  Pick up an sample $(x^s, c^s) \in E$
  Compute o^s
  for $i \in \{1 \cdots n\}$ **do**
    $w_i \leftarrow w_i + \epsilon(c^s - o^s)x_i^s$
  end
until *test*;

Last words on Perceptron

- Oscillation problems
- Gradient descent or Adaline will converge to some minimum even if the classes are not linearly separable, unlike the earlier perceptron training method

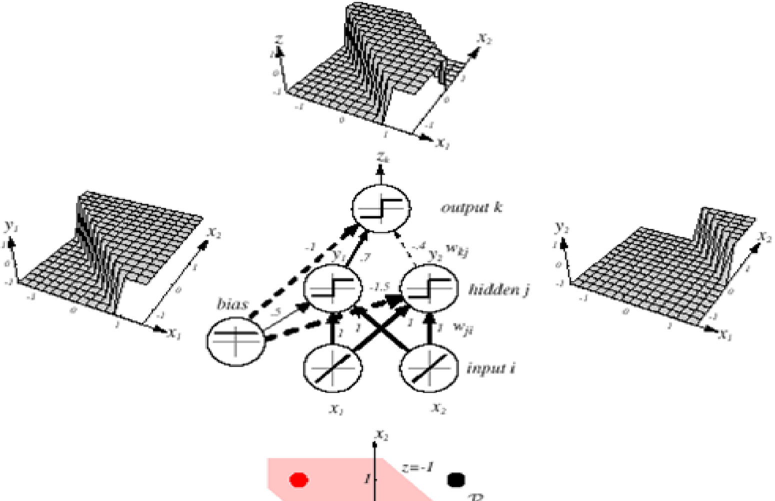
And so

- Another solution: construct multiple layers of perceptrons to get around this problem
- Another solution: transform the inputs using an appropriate mapping (e.g. kernel trick)

MLP - DEFINITION

- Multilayer networks: to bypass the linear classification problem.
- Typically fully connected, feedforward networks.
- A three-layer neural network consists of an input layer, a hidden layer and an output layer interconnected by modifiable weights represented by links between layers

Example: XOR



- A single bias unit is connected to each unit other than the input units
- Net activation:

$$net_j = \sum_{i=0}^{n} x_i w_{ji} = w_{j.}^T x$$

, where the subscript i indexes units in the input layer, j in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit j.

- Each hidden unit emits an output that is a nonlinear function of its activation: $y_j = f(net_j) \Rightarrow f$:activation function
- Each output unit similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=0}^{n_H} y_j w_{kj} = w_{k.}^T y$$

where the subscript k indexes units in the ouput layer and n_H denotes the number of hidden units

- An output unit computes the nonlinear function of its net, emitting $z_k = f(net_k)$
- In the case of c outputs (classes), network $\approx c$ discriminants functions $z_k = g_k(x)$ and classify the input x according to the largest discriminant function $g_k(x), 1 \leq k \leq c$

General Feedforward Operation in case of c output units

$$g_k(x) = z_k = f\left(\sum_{j=0}^{n_H} f\left(\sum_{i=0}^{n} x_i w_{ji}\right)\right)$$

- Hidden units enable us to express more complicated nonlinear functions and thus extend the classification
- The activation function does not have to be a sign function, it is often required to be continuous and differentiable
- the activation in the output layer can be different from the activation function in the hidden layer or have different activation for each individual unit
- assumption: all activation functions are identical

Backpropagation Algorithm

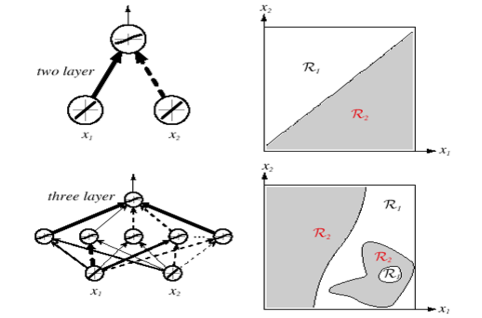


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- Any function from input to output can be implemented as a three-layer neural network
- results of greater theoretical interest than practical, since the construction of such a network requires the nonlinear functions and the weight values which are unknown!

Credit assignment problem

- Goal: to set the interconnexion weights based on the training patterns and the desired outputs
- In a three-layer network, it is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights
- The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights, this is known as The credit assignment problem

Modes of operation

1. feedforward: presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)
2. supervised learning: presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

Credit assignment problem

- Goal: to set the interconnexion weights based on the training patterns and the desired outputs
- In a three-layer network, it is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights
- The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights, this is known as The credit assignment problem

Modes of operation

1. feedforward: presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)
2. supervised learning: presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

Let t_k be the k-th target (or desired) output and z_k be the k-th computed output, $1 \leq k \leq c$ and w represents all the weights of the network.

The training error is

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t_k - z_k\|^2$$

The backpropagation learning rule is based on gradient descent The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$

where η is the learning rate($\approx$ relative size of the change in weights)
$w(m+1) = w(m) + \Delta w(m)$ m:m-th pattern presented

- Error on the hidden-to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

δ_k:sensitivity of unit k

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k)f'(net_k)$$

Since $net_k = w_k^T y$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

and so the weight update (or learning rule) for the hidden-to-output weights is:

$$\Delta w_{kj} = \eta(t_k - z_k)f'(net_k)y_j$$

Backpropagation Algorithm - Learning

- Error on the input-to-hidden units

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

But

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}
\end{aligned}$$

Similarly as in the preceding case, we define the sensitivity for a hidden unit

$$\delta_j = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

(The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the hidden-to-output weights w_{kj}; all multipled by $f(net_j)$ and so the weight update (or learning rule) for the input-to-hidden weights is:

Input: $E, n_H, \eta, thres, m = 0$
random initialization of w_{ji}
repeat
$m \leftarrow m + 1$
choose randomly x^m
$\forall i, j \quad w_{ji} \leftarrow w_{ji} + \eta \delta_j x_i$
$\forall j, k \quad w_{kj} \leftarrow w_{kj} + \eta \delta_k y_j$
until $\|\nabla J(w)\| < thres$;
return W

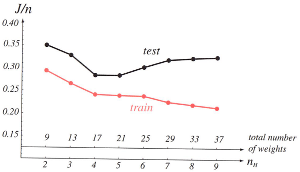
ACTIVATION FUNCTION

- $\rightarrow$ continuous, smooth
- $\rightarrow$ nonlinear (!)
- $\rightarrow$ saturation ($\exists$ max and min value) $\rightarrow$ weights and activation bounded
- $\rightarrow$ monotonicity (not essential)
- $\rightarrow$ linearity for a small value of net

$\Rightarrow$ Sigmod functions

$$f(net) = a.th(b.net) = a\frac{e^{b.net} - e^{-b.net}}{e^{b.net} + e^{-b.net}}$$

NUMBER OF HIDDEN UNITS / HIDDEN LAYERS



1. n_H: governs the expressive power $\Rightarrow$ complexity of the decision boundary. Possible pruning (OBS, OBD).
2. n_H too large $\Rightarrow$ overfitting ; n_H too small$\Rightarrow$ too few expression power
3. number of layers: in theory, 3 layers are sufficient, but for some application, 4 layers are useful (affine transformation invariance)

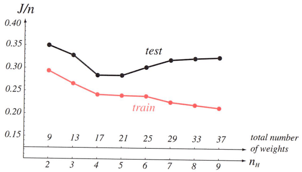
ACTIVATION FUNCTION

- $\rightarrow$ continuous, smooth
- $\rightarrow$ nonlinear (!)
- $\rightarrow$ saturation ($\exists$ max and min value) $\rightarrow$ weights and activation bounded
- $\rightarrow$ monotonicity (not essential)
- $\rightarrow$ linearity for a small value of net

$\Rightarrow$ Sigmod functions

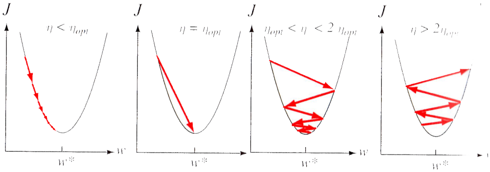
$$f(net) = a.th(b.net) = a\frac{e^{b.net} - e^{-b.net}}{e^{b.net} + e^{-b.net}}$$

NUMBER OF HIDDEN UNITS / HIDDEN LAYERS



1. n_H: governs the expressive power $\Rightarrow$ complexity of the decision boundary. Possible pruning (OBS, OBD).
2. n_H too large $\Rightarrow$ overfitting ; n_H too small$\Rightarrow$ too few expression power
3. number of layers: in theory, 3 layers are sufficient, but for some application, 4 layers are useful (affine transformation invariance)

Practical Issues



Weights

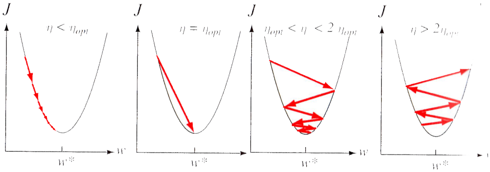
- → $W_{init} = 0 \Rightarrow W_n = 0, \forall n$
- → optimal learning rate: $\left(\frac{\partial J^2}{\partial w^2}\right)^{-1}$
- → weight decay: avoid overfitting: $W_{new} = W_{old}(1 - \epsilon)$
- → momentum: the network learns more quickly when plateaus in the error surface exist: $W_{n+} = W_n + (1 - \alpha)\Delta W_{bp} + \alpha \Delta W_{n-1}$

Training strategy

1. stochastic
2. batch
3. online

Each as pros and cons

Practical Issues



Weights

- $\rightarrow$ $W_{init} = 0 \Rightarrow W_n = 0, \forall n$
- $\rightarrow$ optimal learning rate: $\left(\frac{\partial J^2}{\partial w^2}\right)^{-1}$
- $\rightarrow$ weight decay: avoid overfitting: $W_{new} = W_{old}(1 - \epsilon)$
- $\rightarrow$ momentum: the network learns more quickly when plateaus in the error surface exist: $W_{n+} = W_n + (1-\alpha)\Delta W_{bp} + \alpha \Delta W_{n-1}$

Training strategy

1. stochastic
2. batch
3. online

Each as pros and cons

Stopping criterion

- The algorithm terminates when the change in the criterion function $J(w)$ is smaller than some preset value
- There are other stopping criteria that lead to better performance than this one
- So far, we have considered the error on a single pattern, but we want to consider an error defined over the entirety of patterns in the training set
- The total training error is the sum over the errors of n individual patterns

$$J = \sum_{p=1}^{n} J_p \quad (1)$$

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set
- However, given a large number of such individual updates, the total error of equation (1) decreases

STOPPING CRITERION

- The algorithm terminates when the change in the criterion function $J(w)$ is smaller than some preset value
- There are other stopping criteria that lead to better performance than this one
- So far, we have considered the error on a single pattern, but we want to consider an error defined over the entirety of patterns in the training set
- The total training error is the sum over the errors of n individual patterns

$$J = \sum_{p=1}^{n} J_p \quad (1)$$

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set
- However, given a large number of such individual updates, the total error of equation (1) decreases

HOW TO ?

- Before training starts, the error on the training set is high; through the learning process, the error becomes smaller
- The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network
- The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase
- A validation set is used in order to decide when to stop training ; we do not want to overfit the network and decrease the power of the classifier generalization

RULE

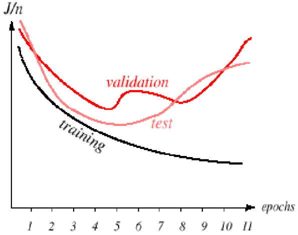
we stop training at a minimum of the error on the validation set

HOW TO ?

- Before training starts, the error on the training set is high; through the learning process, the error becomes smaller
- The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network
- The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase
- A validation set is used in order to decide when to stop training ; we do not want to overfit the network and decrease the power of the classifier generalization

RULE

we stop training at a minimum of the error on the validation set



Plot of the average error per pattern $\left(\frac{1}{n}\sum_{p=1}^{n} J_p\right)$. The validation error or generalization error per pattern are virtually highet than the training error. In some protocols, training is stopped at the first minimum of the validation set.

REFERENCE

A Kolmogorov, On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, Doklady Akademia Nauk SSSR, 114:953-956, 1957

RESULTS

1. Boolean functions: Any boolean function can be represented by a two-layer network with sufficient hidden units.
2. Continuous functions: Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network (Sigmoid functions can act as a set of basis functions for composing more complex functions, like sine waves in Fourier analysis). . More precisely, any continuous function $g(x)$ defined on the unit hypercube of $\mathbb{R}^n$, $n \geq 2$ can be represented in the form

$$g(x) = \sum_{j=1}^{2n+1} \eta_j \left(\sum_{i=1}^{d} \psi_{ij}(x_i) \right)$$

   → each of $2n+1$ hidden units takes as input a sum of d nonlinear functions, one for each input feature x_i
   → each hidden unit emits a nonlinear function η of ites total input
   → the output unit merely emits the sum of the contributions of the hidden units
3. Arbitrary function: Any function can be approximated to arbitrary accuracy by a three-layer network

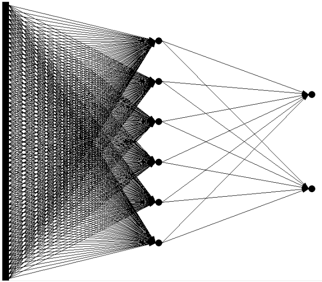
BUT...

Kolmogorov's theorem tells very little about how to find the nonlinear functions based on data; this is the central problem in network-based pattern recognition

EXPERIMENT

- Task: Learn to discriminate between two different voices (David and Steve) saying 'Hello'
- Format: frequency distribution (60 bins), Analogy: cochlea

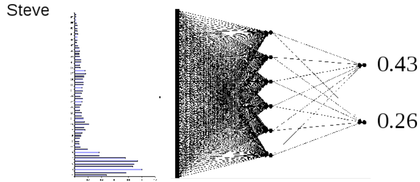
NETWORK ARCHITECTURE

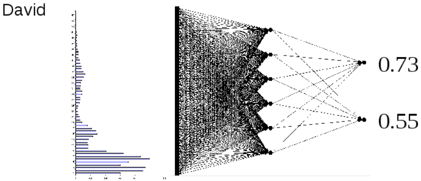


Feed Forward Network

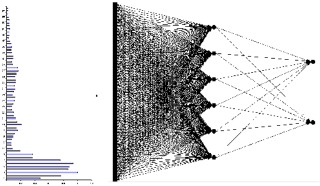
- 60 inputs (number of bins)
- 6 hidden nodes
- 2 outputs (0-1 for Steve and David)

Training





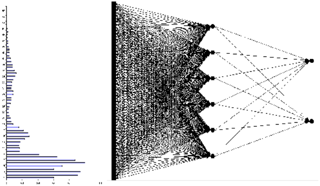
Steve



0.43 – 0 = 0.43

0.26 – 1 = 0.74

David

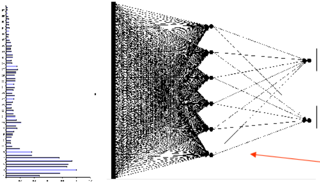


0.73 – 1 = 0.27

0.55 – 0 = 0.55

Training

Steve

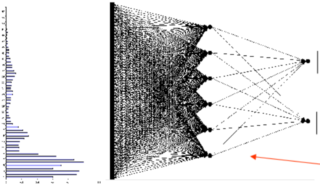


$|0.43 - 0| = 0.43$

$|0.26 - 1| = 0.74$

1.17

David



$|0.73 - 1| = 0.27$

$|0.55 - 0| = 0.55$

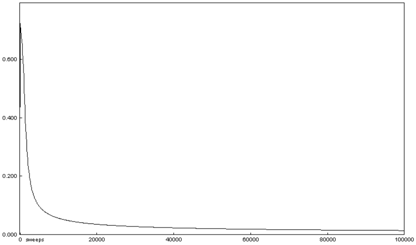
0.82

Presenting the training set

Repeat

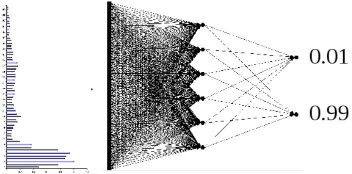
1. Present data
2. Calculate error
3. Backpropagate error
4. Adjust weights

Repeat process multiple times

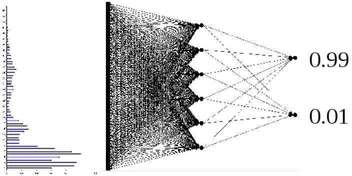


Presenting unknown data

Steve



David



SUPERVISED/UNSUPERVISED LEARNING

- We just discussed a form of supervised learning: A 'teacher' tells the network what the correct output is, based on the input until the network learns the target concept.
- We can also train networks when there is no teacher. The network learns a prototype based on the distribution of patterns in the training data. Such networks allow us to:

  → Discover underlying structure of the data
  → Encode or compress the data
  → Transform the data

EXAMPLES: HOPFIELD NETWORKS / SOM

A Hopfield network is a type of content-addressable memory

→ Non-linear system with attractor points that represent concepts
→ Given a fuzzy input the system converges to the nearest attractor

- Possibility to have 'spurious' attractors that is a blend of multiple stored patterns
- Also possible to have chaotic patterns that never converge

A SOM is a data visualization technique based on self-organizing neural networks

SUPERVISED/UNSUPERVISED LEARNING

- We just discussed a form of supervised learning: A ‘teacher’ tells the network what the correct output is, based on the input until the network learns the target concept.
- We can also train networks when there is no teacher. The network learns a prototype based on the distribution of patterns in the training data. Such networks allow us to:
  - $\rightarrow$ Discover underlying structure of the data
  - $\rightarrow$ Encode or compress the data
  - $\rightarrow$ Transform the data

EXAMPLES: HOPFIELD NETWORKS / SOM

A Hopfield network is a type of content-addressable memory

- $\rightarrow$ Non-linear system with attractor points that represent concepts
- $\rightarrow$ Given a fuzzy input the system converges to the nearest attractor
  - Possibility to have ‘spurious’ attractors that is a blend of multiple stored patterns
  - Also possible to have chaotic patterns that never converge

A SOM is a data visualization technique based on self-organizing neural networks

DEFINITION

- Recurrent; Every unit is connected to every other unit
- Weights connecting units are symmetrical: $w_{ij} = w_{ji}$
- If the weighted sum of the inputs exceeds a threshold, its output is 1 otherwise its output is -1
- Units update themselves asynchronously as their inputs change

HOPFIELD MEMORIES

Setting the weights:

- A pattern is a setting of on or off for each unit
- Given a set of Q patterns to store
  - $\rightarrow$ For every weight connecting units i and j

$$w_{ij} = \sum_{p=1}^{Q} x_i^p x_j^p$$

  - $\rightarrow$ $\approx$ Hebbian rule which makes the weight strength proportional to the product of the firing rates of the two interconnected units

Definition

- Recurrent; Every unit is connected to every other unit
- Weights connecting units are symmetrical: $w_{ij} = w_{ji}$
- If the weighted sum of the inputs exceeds a threshold, its output is 1 otherwise its output is -1
- Units update themselves asynchronously as their inputs change

Hopfield Memories

Setting the weights:

- A pattern is a setting of on or off for each unit
- Given a set of Q patterns to store

$\rightarrow$ For every weight connecting units i and j

$$w_{ij} = \sum_{p=1}^{Q} x_i^p x_j^p$$

$\rightarrow$ $\approx$ Hebbian rule which makes the weight strength proportional to the product of the firing rates of the two interconnected units

Seminal paper

T Kohonen, Self Organizing formation of topologically correct feature maps, Biological Cybernetics, 43:59,69, 1982

Definition

- Self Organizing maps: data visualization technique (a.k.a. Kohonen Networks, Competitive Learning, Winner-Take-All Learning)
- Generally reduces the dimensions of data through the use of self-organizing neural networks
- Useful technique to make sense of large data sets
- closely related to MDS

Seminal paper

T Kohonen, Self Organizing formation of topologically correct feature maps, Biological Cybernetics, 43:59,69, 1982

Definition

- Self Organizing maps: data visualization technique (a.k.a. Kohonen Networks, Competitive Learning, Winner-Take-All Learning)
- Generally reduces the dimensions of data through the use of self-organizing neural networks
- Useful technique to make sense of large data sets
- closely related to MDS

DEFINITION

Two layer network: Input units, output units, each input unit i is connected to each output unit $j \rightarrow w_{ij}$

ALGORITHM

random initialization of w_{ij}
repeat
Pick up an example $(x^i, c^i) \in E$
Assign input unit values according to the values in the current example
Find the 'winner', i.e. the output unit that most closely matches the input units, using some distance metric, e.g.
for $j \in \{0 \cdots m\}$ **do**

$$\hat{j} = ArgMax_j \left(\sum_{i=1}^{n} (w_{ij} - x_i)^2 \right)$$

end
Modify weights on the winner to more closely match the input

$$\Delta W^{t+1} = c(X_i^t - W^t)$$

$c > 0$: usually decreases as the learning proceeds
until;

DEFINITION

Two layer network: Input units, output units, each input unit i is connected to each output unit $j \rightarrow w_{ij}$

ALGORITHM

random initialization of w_{ij}
repeat
  Pick up an example $(x^i, c^i) \in E$
  Assign input unit values according to the values in the current example
  Find the 'winner', i.e. the output unit that most closely matches the input units, using some distance metric, e.g.
  for $j \in \{0 \cdots m\}$ **do**

$$\hat{j} = ArgMax_j \left(\sum_{i=1}^{n} (w_{ij} - x_i)^2 \right)$$

  end
  Modify weights on the winner to more closely match the input

$$\Delta W^{t+1} = c(X_i^t - W^t)$$

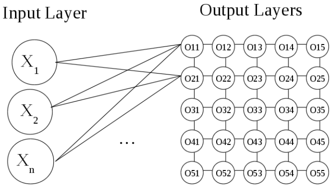
  $c > 0$: usually decreases as the learning proceeds
until;

RESULTS OF THE BASIC ALGORITHM

- → Initially, some output nodes will randomly be a little closer to some particular type of input
- → These nodes become 'winners' and the weights move them even closer to the inputs
- → Over time nodes in the output become representative prototypes for examples in the input

There is no supervised training here
Classification: Given new input, the class is the output node that is the winner

TYPICAL USAGE: 2D FEATURE MAP



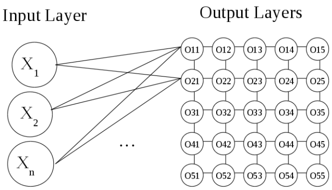
Output nodes form a 2D map organized in a grid-like fashion and weights in a neighborhood around the winner

Results of the basic algorithm

- → Initially, some output nodes will randomly be a little closer to some particular type of input
- → These nodes become 'winners' and the weights move them even closer to the inputs
- → Over time nodes in the output become representative prototypes for examples in the input

There is no supervised training here
Classification: Given new input, the class is the output node that is the winner

Typical Usage: 2D Feature Map



Output nodes form a 2D map organized in a grid-like fashion and weights in a neighborhood around the winner

MODIFIED ALGORITHM

random initialization of w_{ij}
repeat

Pick up an example $(x^i, c^i) \in E$
Assign input unit values according to the values in the current example
Find the 'winner', i.e. the output unit that most closely matches the input units, using some distance metric, e.g.
for $j \in \{0 \cdots m\}$ **do**

$$\hat{j} = ArgMax_j \left(\sum_{i=1}^{n} (w_{ij} - x_i)^2 \right)$$

end
Modify weights on the winner to more closely match the input

$$\Delta W^{t+1} = c(X_i^t - W^t)$$

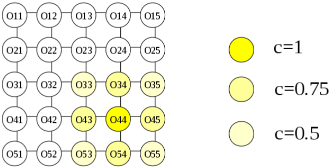
Modify weights in a neighborhood around the winner so the neighbors on the 2D map also become closer to the input

until;

Over time this will tend to cluster similar items closer on the map

UPDATING THE NEIGHBORHOOD

Output Layers



$$\Delta W^{t+1} = c(X_i^t - W^t)$$

- O_{44} is the winner
- Color indicates scaling to update neighbors
- Consider if O_{42} is winner for some other input; 'fight' over claiming O_{43}, O_{33}, O_{53}

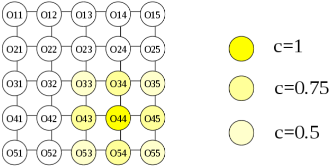
SELECTING THE NEIGHBORHOOD

- Typically, a 'Sombrero Function' or Gaussian function is used
- Neighborhood size usually decreases over time to allow initial 'jockeying for position' and then 'fine-tuning' as algorithm proceeds

Modified algorithm

Updating the Neighborhood

Output Layers



$$\Delta W^{t+1} = c(X_i^t - W^t)$$

- O_{44} is the winner
- Color indicates scaling to update neighbors
- Consider if O_{42} is winner for some other input; ‘fight’ over claiming O_{43}, O_{33}, O_{53}

Selecting the Neighborhood

- Typically, a ‘Sombrero Function’ or Gaussian function is used
- Neighborhood size usually decreases over time to allow initial ‘jockeying for position’ and then ‘fine-tuning’ as algorithm proceeds

Example: Poverty Map

http://www.cis.hut.fi/research/som-research/worldmap.html