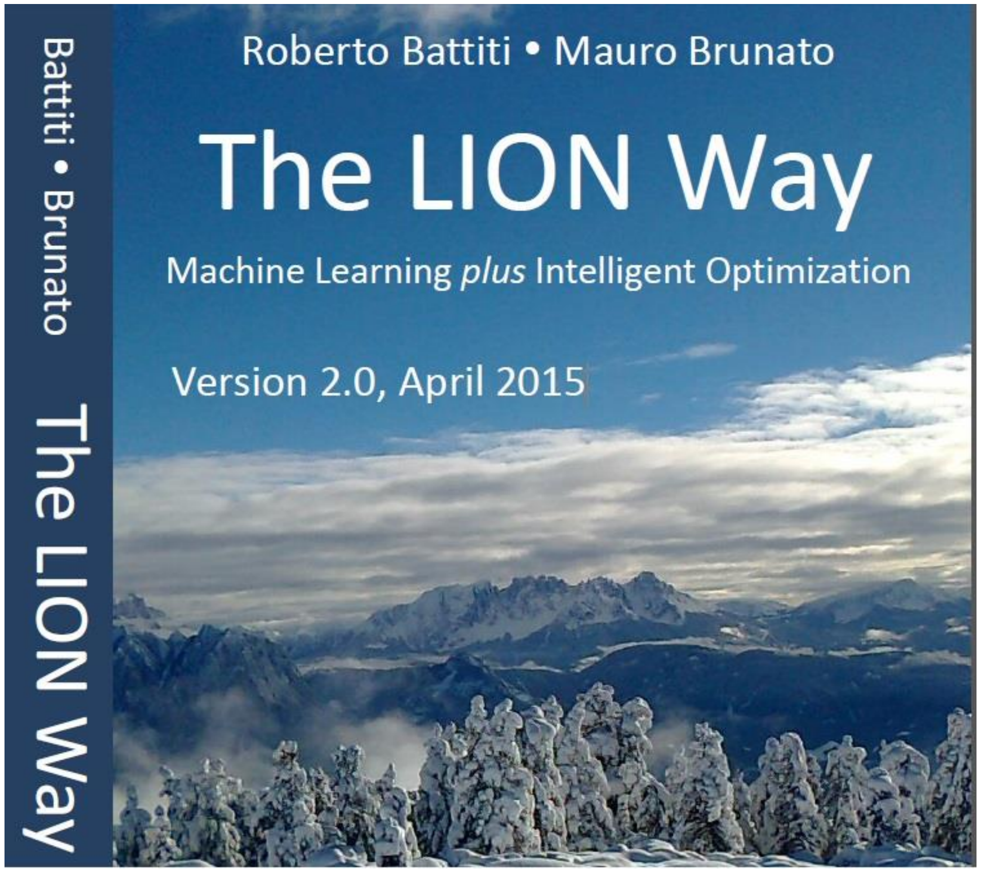
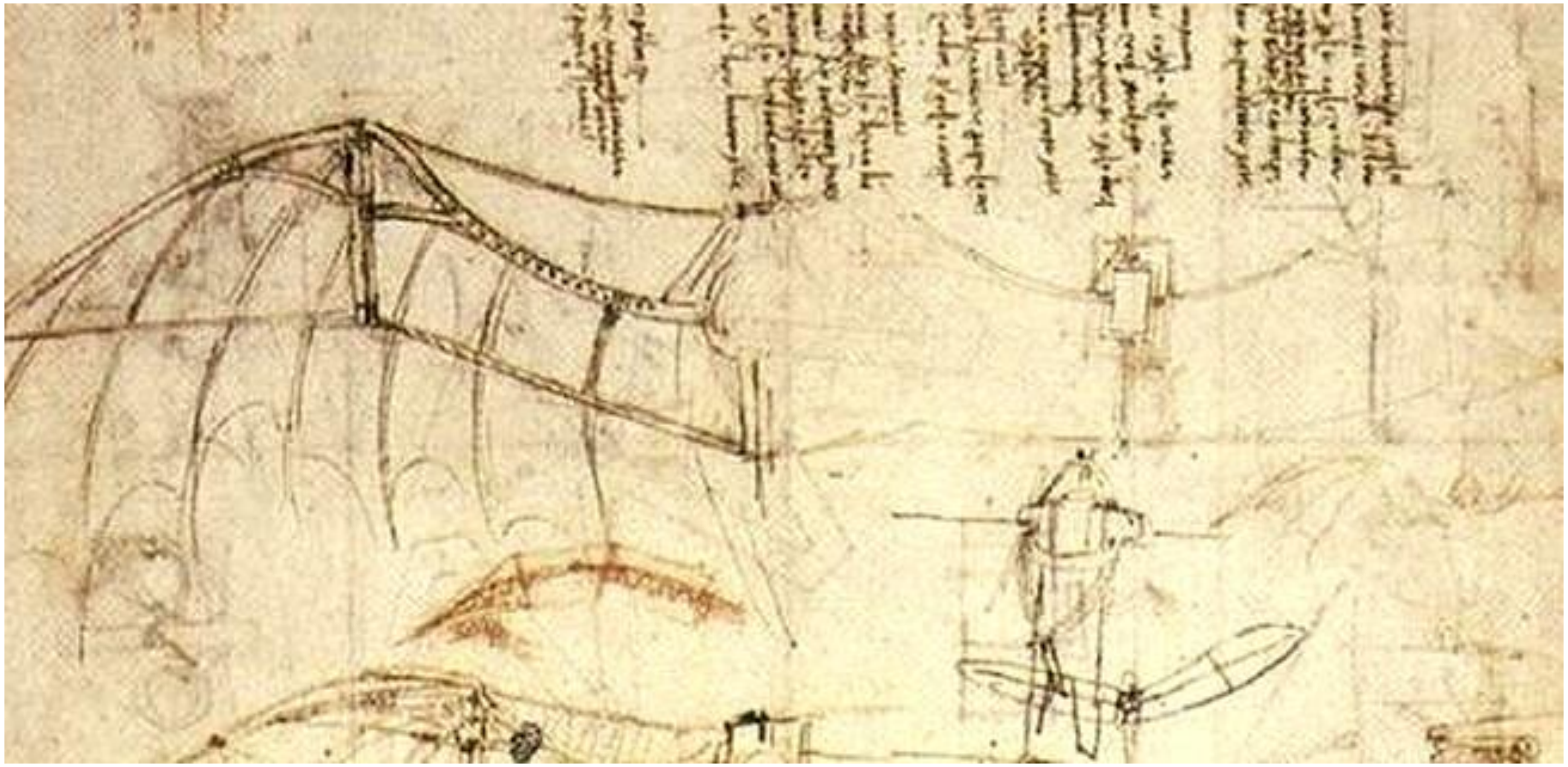

Battiti • Brunato
The LION Way
Roberto Battiti • Mauro Brunato
The LION Way
Machine Learning *plus* Intelligent Optimization
Version 2.0, April 2015

ROBERTO BATTITI, MAURO BRUNATO.
The LION Way: Machine Learning plus *Intelligent Optimization*.
LIONlab, University of Trento, Italy,
Apr 2015

http://intelligent-optimization.org/LIONbook

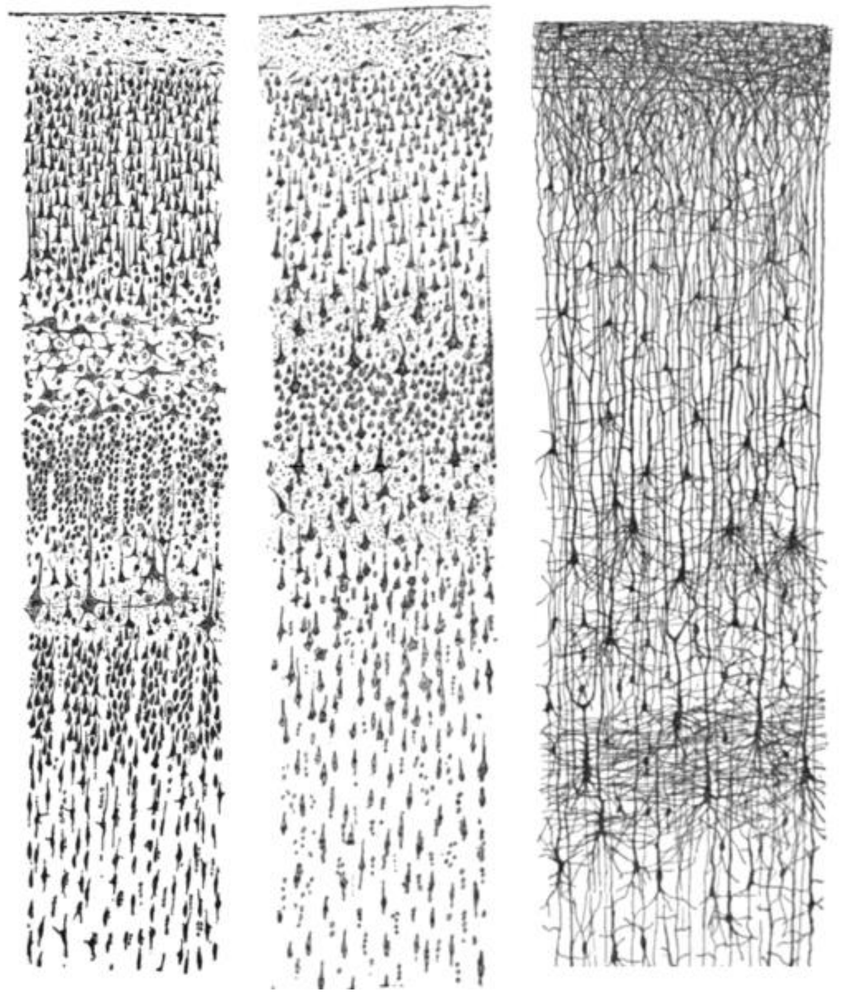
Chap. 9 Neural networks, shallow and deep

Quegli che pigliavano per altore altro che la natura, maestra de' maestri, s'affaticavano invano.
(Leonardo Da Vinci)



The biological metaphor

- Our neural system is composed of 100 billion computing units (neurons) and 10^{15} connections (synapses).
- How can a system composed of many simple interconnected units give rise to highly complex activities?
- Emergence: complex systems arise out of a multiplicity of relatively simple *interacting* units.



Drawings of cortical lamination by Santiago Ramon y Cajal, each showing a vertical cross-section, with the surface of the **cortex** at the top. The different stains show the cell bodies of neurons and the dendrites and axons of a random subset of neurons.

Symbolic vs. sub-symbolic paradigm

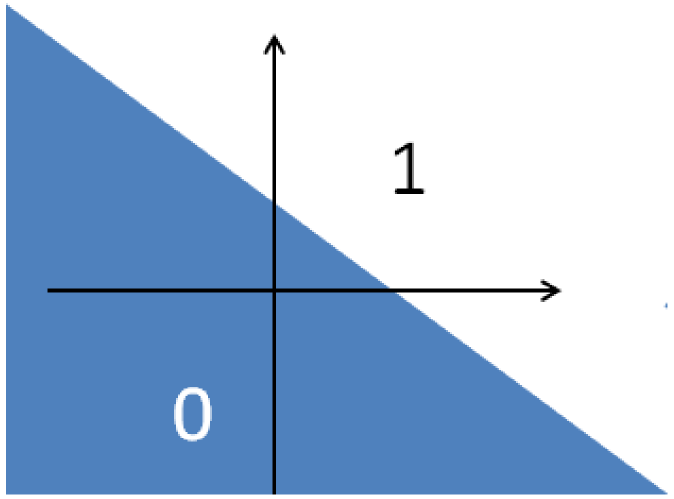
- ”Standard” sequential computers operate in cycles, fetching items from memory, applying mathematical operations and writing results back to memory.
- The *intelligence* of biological brains is different, it lies in the interconnection strengths, learning occurs by modifying connections (**dynamical systems**)
- Neural networks **do not separate memory and processing** but operate via the flow of signals through the network connections.

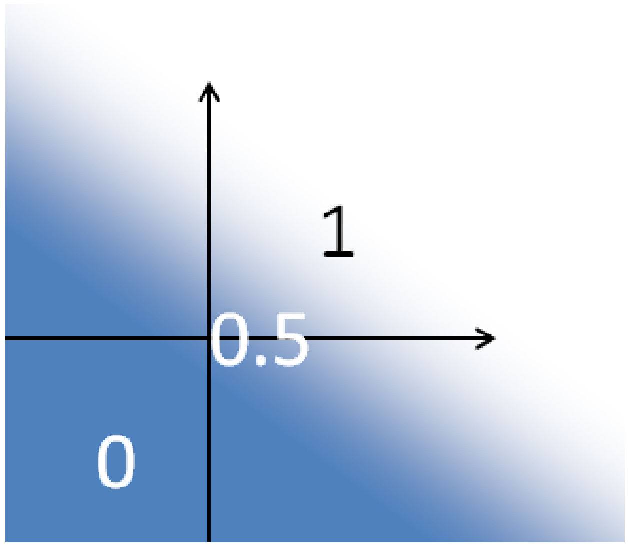
Artificial Neural Networks

- A neuron is modeled as a simple computing unit, a scalar product **w x** ("pattern matching") followed by a sigmoidal ("logistic") function.
- The complexity comes from having more interconnected layers of neurons involved in a complex action (if linear layers are cascaded, the system *remains* linear)
- The "squashing" functions is essential to introduce nonlinearities in the system

Multilayer Perceptrons (MLP)

- By applying a sigmoidal transfer function to the unlimited output of a linear model, one obtains an output in [0,1] which can be interpreted as a probability
- For classification, hyperplane boundaries become "*fuzzy*"



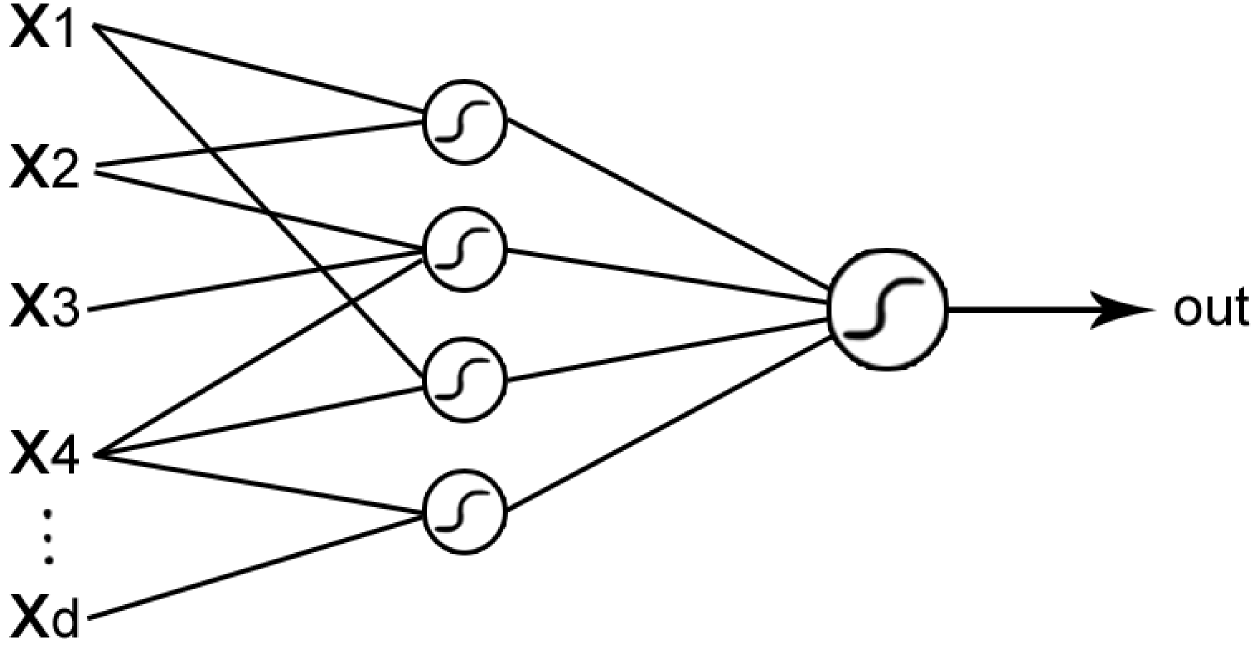


Multilayer Perceptrons (MLP)(2)

- Composing linear transformations → still linear functions.
- Composing linear functions with nonlinear *sigmoids* one can approximate all smooth functions. One hidden layer is sufficient.
- The first linear transformation provides a first ”hidden layer” of outputs, the second transformation produces the visible outputs from the hidden layer.

MLP architecture

- MLPs are composed of a large number of interconnected units working in parallel and organized in **layers** with a **feedforward** information flow.



MLP architecture (2)

- The signals flow sequentially from the input to the output layer.
- For each layer, each unit does the following:

1. scalar product between a vector of weights and the vector of outputs of the previous layer;
2. nonlinear function to each result to produce the input for the next layer

- A popular transfer function is the sigmoidal (or "logistic") function

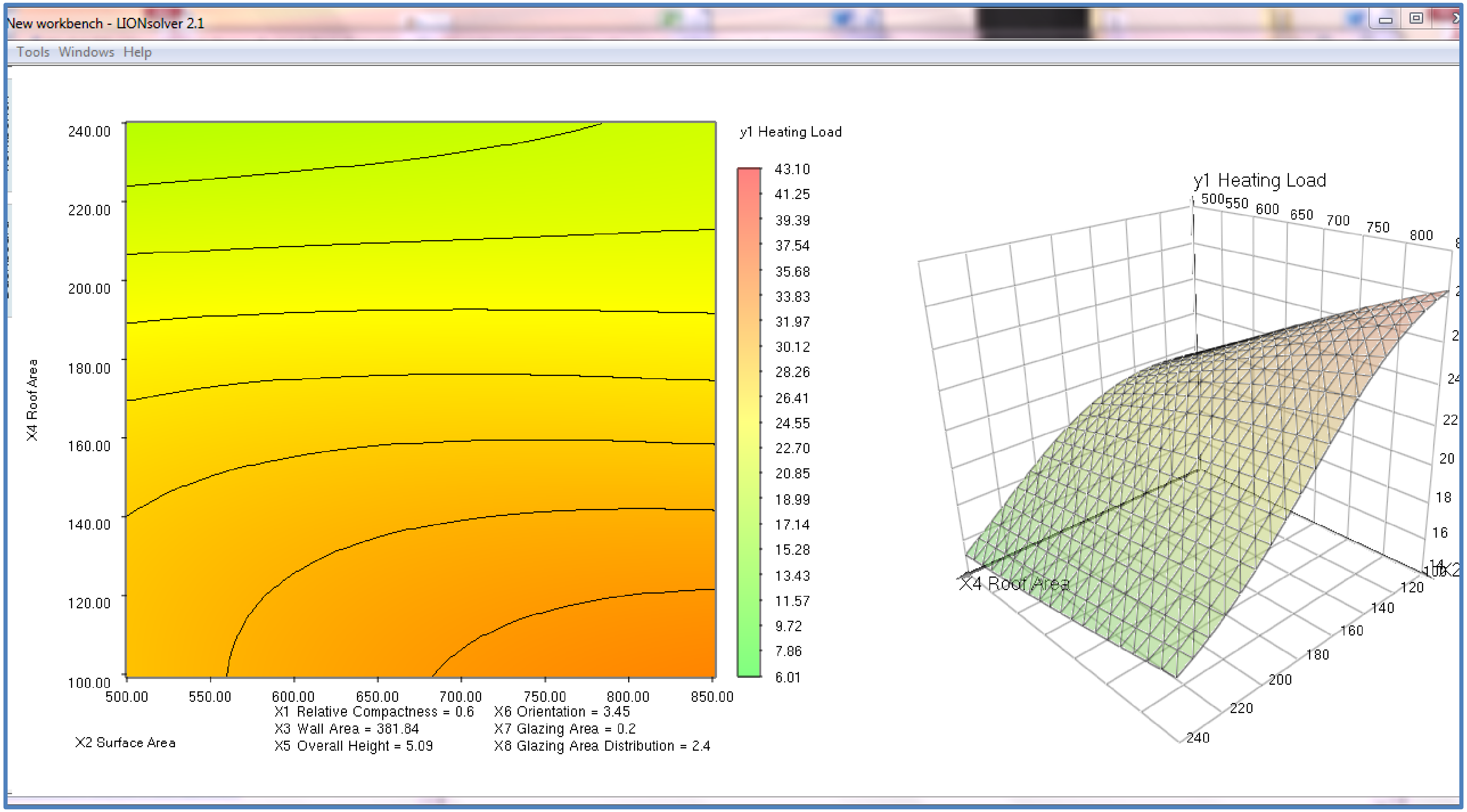
$$f(x) = \frac{1}{1+e^{-x}}.$$

MLPs are universal approximators

- What is the flexibility of the MLP architecture to represent input-output mappings?

- An MLP with one hidden layer and a sufficient number of hidden nodes can approximate any smooth function to any desired accuracy.

- MLPs are very flexible models: they can approximate any smooth input-output transformation.



X1 Relative Compactness = 0.6
X3 Wall Area = 381.84
X5 Overall Height = 5.09
X6 Orientation = 3.45
X7 Glazing Area = 0.2
X8 Glazing Area Distribution = 2.4

Analyzing a neural network output with LIONoso Sweeper. The output value, the energy consumed to heat a house in winter, is shown as a function of input parameters. Color coded output (left), surface plot (right). Nonlinearities are visible.

Error Backpropagation

- How do we **learn** optimal MLPs from examples?

1. take a ”guiding” function to be optimized (e.g., sum-of-squared errors on the training examples)

1. Use gradient descent with respect to the weights to find the better and better weights

1. Stop the descent when results on a validation set are best (if over-learning, generalization can worsen). Learning is not an end, but a *means* for generalizing.

Backpropagation(2)

- One needs derivatives, a simple analysis exercise.

- MLP is a composition of squash functions and scalar products.
- Derivatives can be calculated by using the chain rule for derivatives of composite functions.
- Complexity is O(*number of weights*).
- Formulas are similar to those used for the forward pass, but going in contrary direction, hence the term **error backpropagation**.
- After the network is trained, calculating the output from the inputs requires a number of simple operations proportional to the number of weights.

Batch backpropagation

- Given an MLP, define its sum-of-squared-differences energy as:

$$E(w) = \frac{1}{2}\sum_{p=1}^{P} E_p = \frac{1}{2}\sum_{p=1}^{P}(t_p - o_p(w))^2$$

1. Let the initial weights be randomly distributed
2. Calculate the gradient $\overset{\text{Partial derivatives}}{g_k = \nabla E(w_k)}$
3. The weights at the next iteration k + 1 are updated as follows $w_{k+1} = w_k - \epsilon\, g_k.$

Bold-Driver Backpropagation

- How do we select the learning rate ε ? If small, the learning time increases, if big, the energy oscillates wildly.

1. If successive steps reduce E(w), ε increases exponentially
2. If E(w) increases, ε decreases rapidly

$$\epsilon(t) = \begin{cases} \rho\, \epsilon(t-1), & \text{if } E(w(t)) < E(w(t-1)); \\ \sigma^l\, \epsilon(t-1), & \text{if } E(w(t)) > E(w(t-1)) \end{cases}$$

ρ is close to one (1.1) in order to avoid frequent ”accidents” σ is chosen to provide a rapid reduction(0.5), and l is the minimum integer such that the reduced rate succeeds in diminishing the energy

On-line or stochastic backpropagation

- E is a sum of many terms, one for each pattern p
- The gradient is a sum of the corresponding partial gradients $\nabla E_p(w_k)$

- In “batch” gradient descent, *first* the contributions $\nabla E_p(w_k)$ are summed, *then* the small step is taken

- Stochastic BP: randomly choose a pattern p and take a small step along a single negative $\nabla E_p(w_k)$ *immediately* after calculating it.

On-line or stochastic backpropagation (2)

- Stochastic on-line backpropagation update:

$$w_{k+1} = w_k - \epsilon \nabla E_p(w_k),$$

- where the pattern p is chosen randomly from the training set at each iteration and ε is the learning rate
- Pros: using partial gradients is faster
- Cons: less guarantee of convergence

Advanced optimization for MLP training

- Higher-order derivatives can be used to enhance the search.
- Conjugate gradient and secant methods update an approximation of the Hessian by using only gradient information.
- They are useful for problems with few weights (approx < 100) and requiring high precision in the output value.

Deep neural networks

- Some classes of input-output mappings are easier to build if more hidden layers are considered.
- **The dream**: feed examples to an MLP with many hidden layers and have the MLP automatically develop internal representations (encoded in the activation patterns of the hidden-layers).

Practical obstacles to deep MLPs

- Partial derivatives w.r.t. the weights of the first layers tend to be very small, leading to numerical estimation problems.
- As a result, it can happen that the internal representations developed by the first layers will not differ too much from being randomly generated, and leaving only the topmost levels to do some ”useful” work.
- A very large number of parameters (such as in deep MLP) can lead to overtraining

Applications of deep learning

Lately, deep learning has lead to superior classification results in challenging areas

The main scheme is as follows:

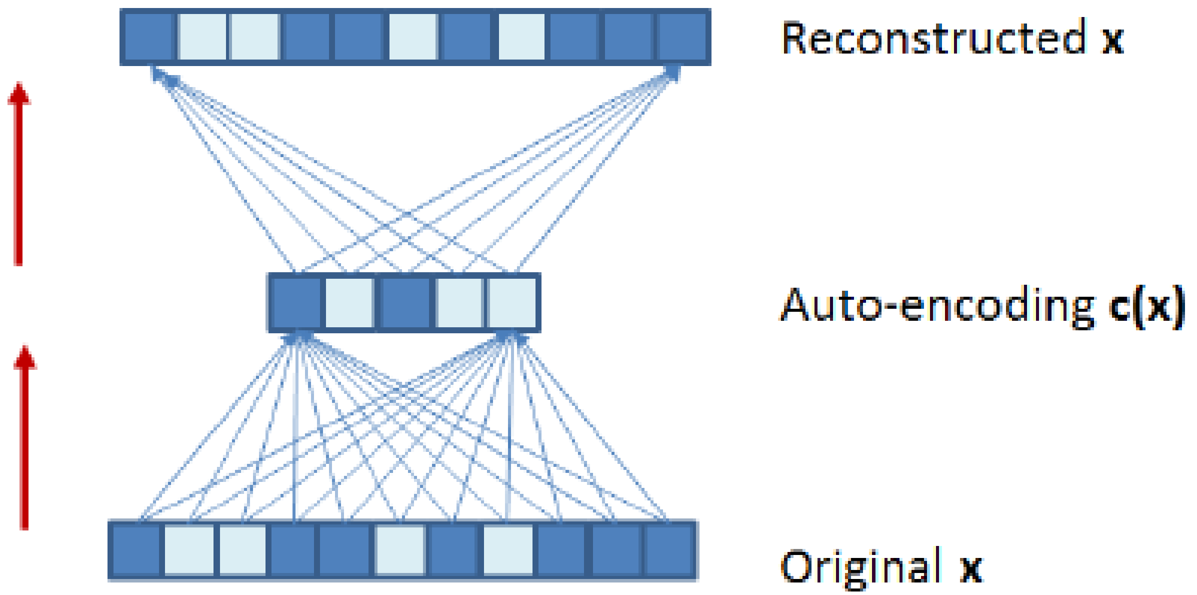
1. use unsupervised learning from many **unlabeled** examples to prepare the deep network in an initial state;

2. use back propagation only for the final tuning with the set of labeled examples

Auto-encoders

- Auto-encoders build internal representations in an unsupervised manner
- One builds a network with a hidden layer and demands that the output simply **reproduces** the input
- Squeezing: in the hidden layer the input is compressed into an encoding **c(x)** with less variables than the original one
- **c(x)** will be forced to discover regularities in the input patterns

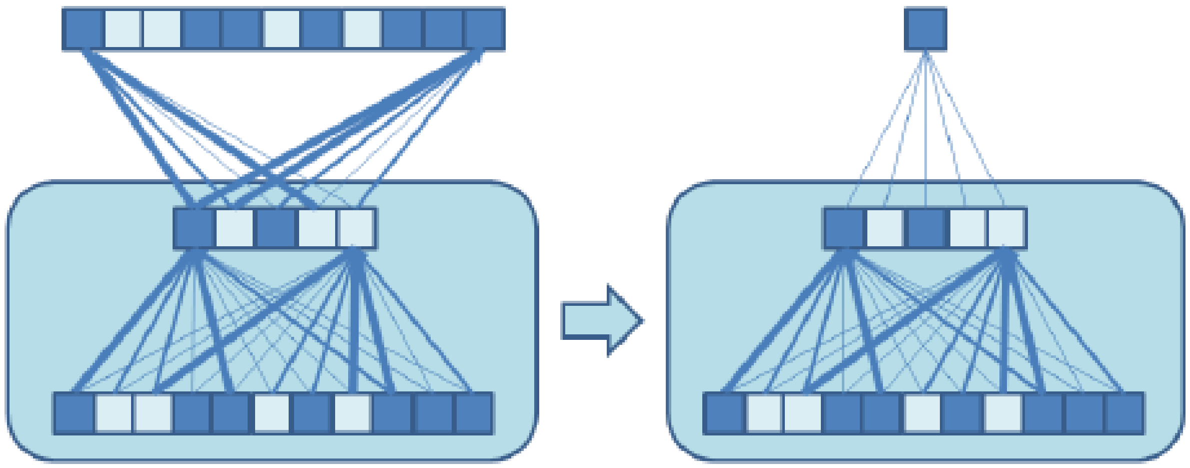
Auto-encoders (2)



Auto-encoders (3)

- The auto-encoder can be trained by backpropagation
- Classification labels are not necessary
- After the auto-encoder is built, the hidden layer (weights and hidden units) is transplanted to a second network with an additional layer, intended for classification
- This new network will be trained on the **labelled** examples to realize a classifier.

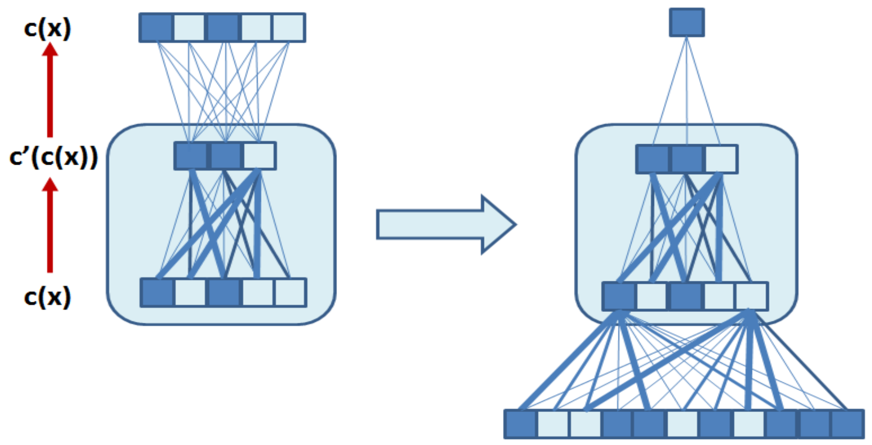
Auto-encoders (4)



Auto-encoders (5)

- A chain of subsequent hidden layers by iterating and composing subsequent encodings c'(c(x))

  - At each iteration, the auto-encoder derives a more compressed internal representation



- Appropriate numbers for the number of layers and the optimal number of units can be obtained pragmatically by cross-validation.

Advanced training methods

Denoising auto-encoders

- Add to each pattern x a random noise and ask the auto-encoding network to reconstruct the original noise-free pattern x.
- This encourages the system to extract even stronger and more significant regularities from the input patterns

Advanced training methods(2)

Random dropout

- In stochastic backpropagation training, each hidden unit is randomly omitted from the network with probability 0.5.
- Each unit cannot rely on the presence of the other hidden units and is encouraged to identify useful information, independently of the other units.

Advanced training methods(3)

Curriculum learning

Training examples are presented to the network by starting from the easiest cases and then gradually proceeding to the more complex ones.

Gist

- Multi-layer perceptron neural networks (**MLPs**) are a flexible (non-parametric) modeling architecture composed of layers of sigmoidal units interconnected in a feedforward manner only between adjacent layers.
- Training from labeled examples can occur via variations of **gradient descent (error backpropagation).**
- Although gradient descent is a weak optimization method, it yields successful practical results.

Gist(2)

- Deep neural networks composed of many layers are becoming effective
- Learning schemes for deep MLP consist of:

1. an unsupervised preparatory phase
2. a final tuning phase using the scarce labeled examples.

Gist(3)

- To improve generalization, the use of controlled amounts of noise during training is effective
- Increasing the effort during training pays dividends in terms of improved generalization