Deep Learning

UGUR HALICI

Dept. EEE, NSNT, Middle East Technical University

11.03.2015

Deep Learning

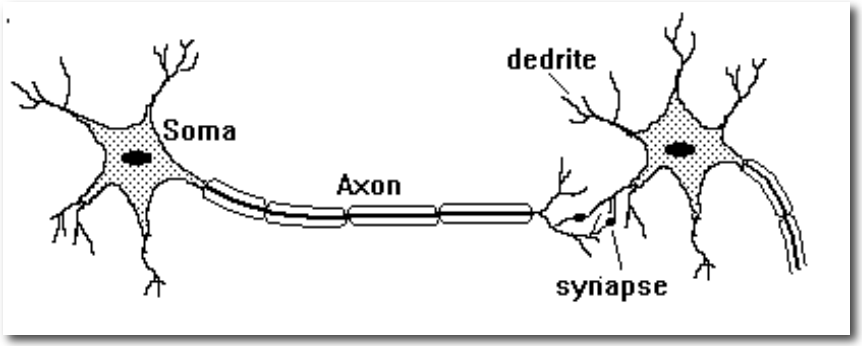
neuro sci tech

Outline

- Biological-artificial neuron
- Network structures
- Multilayer Perceptron
  - Backpropagation algorithm
- Deep Networks
  - Convolutional Neural Networks (CNN)
  - Stacked Autoencoders
  - Deep Boltzmann Machine

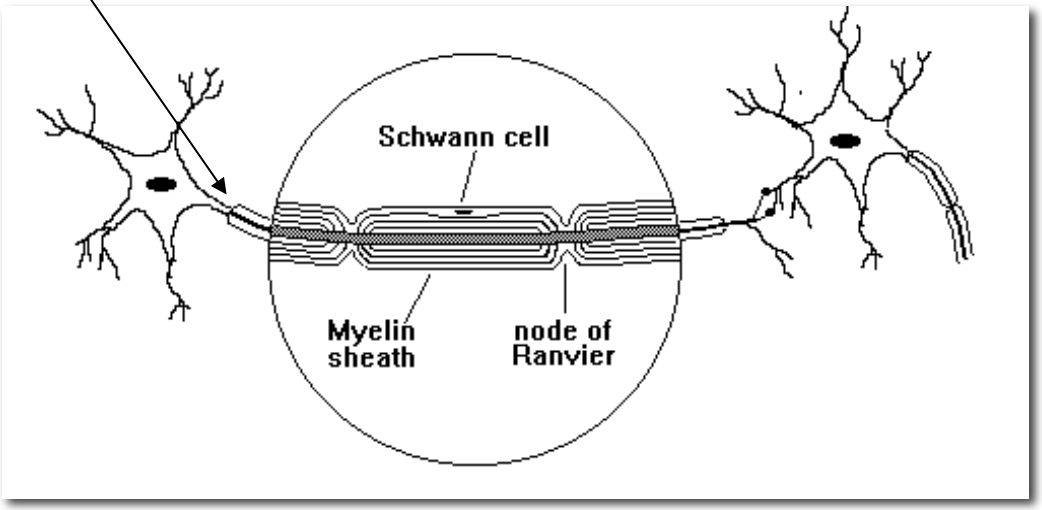
Biological Neuron

- It is estaimated that the human central nervous system is comprised of about $1{,}3x10^{10}$ neurons and that about $1x10^{10}$ of them takes place in the brain
- A neuron has a roughly spherical cell body called **soma**
- The extensions around the cell body like bushy tree are called **dendrites**, which are responsible from receiving the incoming signals generated by other neurons.



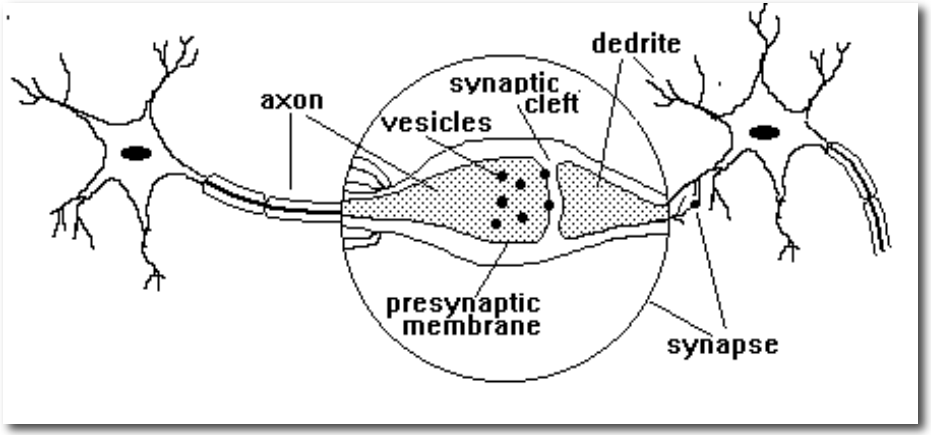
Biological Neuron

- The signals generated in soma are transmitted to other neurons through an extension on the cell body called **axon**.
- An axon, having a length varying from a fraction of a millimeter to a meter in human body, prolongs from the cell body at the point called **axon hillock**.



Biological Neuron

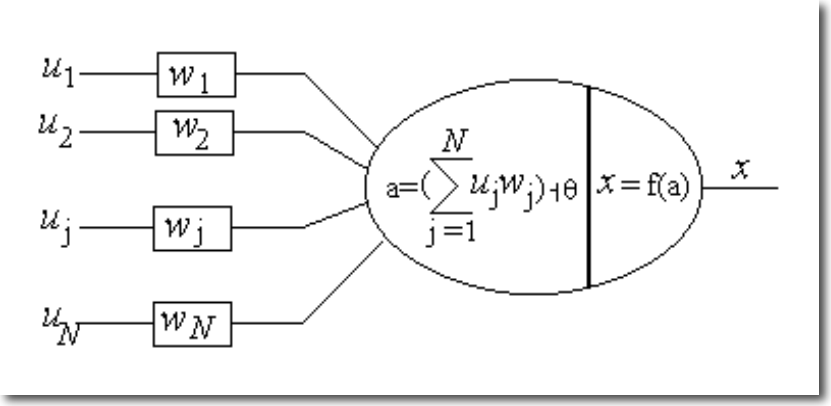
- At the other end, the axon is separated into several branches, at the very end of which the axon enlarges and forms terminal buttons.
- Terminal buttons are placed in special structures called th synapses which are the junctions transmitting signals from one neuron to another.
- A neuron typically drive 10^3 to 10^4 synaptic junctions



Biological Neuron

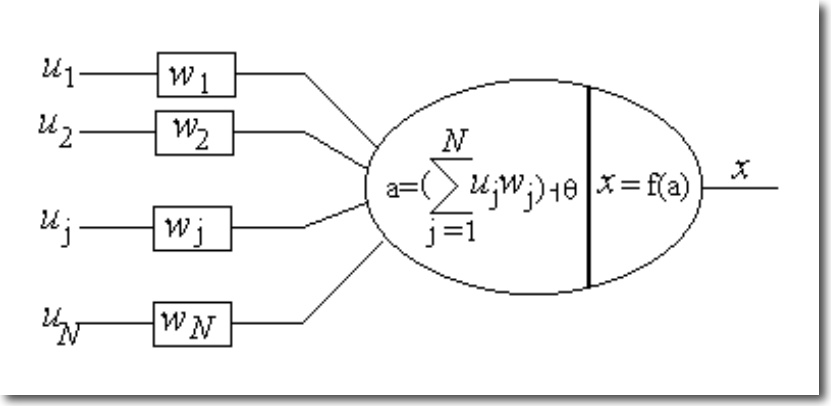
- The transmission of a signal from one neuron to another through synapses is a complex chemical process in which specific transmitter substances are released from the sending side of the junction.
- The effect is to raise or lower the electrical potential inside the body of the receiving cell, depending on the type and strength of the synapses.
- If this potential reaches a threshold, the neuron fires and send pulses through the axon.
- It is this characteristic that the artificial neuron model proposed by McCulloch and Pitts (1943) attempt reproduce.

Artificial Neuron Model : Perceptron



- The neuron model shown in Figure is called perceptron and it is widely used in artificial neural networks with some minor modifications on it.
- It has N input, denoted as u_1, u_2, ...u_N.

Artificial Neuron Model : Perceptron



- Each line connecting these inputs to the neuron is assigned a weight, which are denoted as w_1, w_2, .., w_N respectively.
- Weights in the artificial model correspond to the synaptic connections in biological neurons.
- θ represent threshold in artificial neuron.
- The inputs and the weights are real values.

Artificial Neuron Model: Neuron Activation

The activation is given by the formula:

$$a = (\sum_{j=1}^{N} w_j u_j) + \theta$$

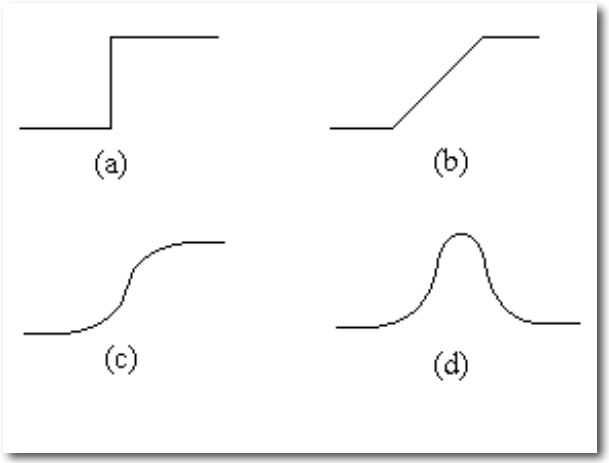
- A negative value for a weight indicates an inhibitory connection while a positive value indicates an excitatory one.
- Although in biological neurons, θ has a negative value, it may be assigned a positive value in artificial neuron models. If θ is positive, it is usually referred as bias. For its mathematical convenience, (+) sign is used in the activation formula.

Artificial Neuron Model: Neuron Output

- The output value of the neuron is a function of its activation in an analogy to the firing frequency of the biological neurons:

$$x = f(a)$$

- Originally the neuron output function f(a) in McCulloch Pitts model proposed as threshold function, however linear, ramp and sigmoid functions are also widely used output functions.



a) threshold function
b) ramp function,
c) sigmoid function,
d) Gaussian function

Artificial Neuron Model: Neuron Output

- Linear:

$$f(a) = \kappa a$$

- Threshold:

$$f(a) = \begin{cases} 0 & a \le 0 \\ 1 & 0 < a \end{cases}$$

- Ramp:

$$f(a) = \begin{cases} 0 & a \le 0 \\ a/\kappa & 0 < a \le \kappa \\ 1 & \kappa < a \end{cases}$$

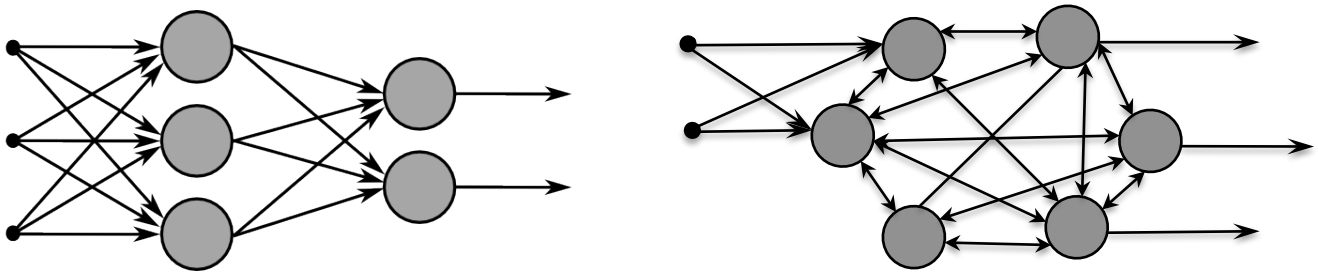
- Sigmoid:

$$f(a) = \frac{1}{1+e^{-\kappa a}}$$

Note: $\tanh(a) = 2 * \text{sigmoid}(a) - 1$

Network Architectures

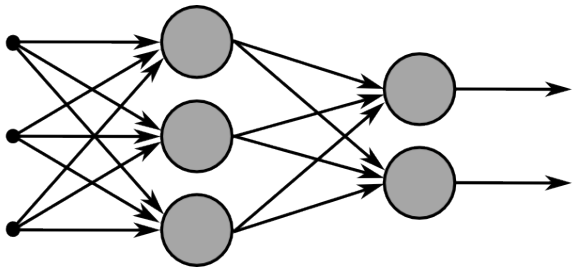
- Neural computing is an alternative to programmed computing, which is a mathematical model inspired by biological models.
- This computing system is made up of a number of artificial neurons and a huge number of interconnections between them.
- According to the structure of the connections, we identify different classes of network architectures.



a) layered feedforward neural network
b) nonlayered recurrent neural network
(there arae also backward connections)

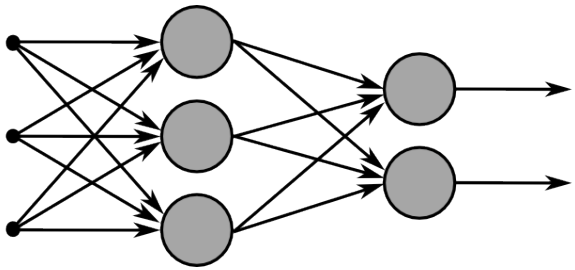
Feedforward Neural Networks

- In feedforward neural networks, the neurons are organized in the form of layers.
- The neurons in a layer get input from the previous layer and feed their output to the next layer.
- In this kind of networks connections to the neurons in the same or previous layers are not permitted.



Feedforward Neural Networks

- The last layer of neurons is called the **output** layer and the layers between the input and output layers are called the **hidden** layers.
- The **input** layer is made up of special input neurons, transmitting only the applied external input to their outputs.



Feedforward Neural Networks

- In a network, if there is only the layer of input nodes and a single layer of neurons constituting the output layer then they are called single layer network.
- If there are one or more hidden layers, such networks are called **multilayer networks** (or **multilayer perceptron**)

Backpropagation Algorithm

- The backpropagation algorithm looks for the minimum of the **error function** in weight space using the method of gradient descent.
- Since this method requires computation of the gradient of the error function at each iteration step, the error function should be differentiable.
- Error function may be chosen as **mean square error** between the actual and desired outputs for a set of samples. (other error functions are available in the literature depending on the structure of the network)

Backpropagation Algorithm

- The combination of weights which minimizes the error function is considered to be a solution of the learning problem.
- The network is trained by using a set of samples together with their desired output, called **training set**.
- Usually there is another set called **test set**, again consists of samples together with desired outputs, for measuring performance

The Backpropagation Algorithm

Step 0. Initialize weights: to small random values;

Step 1. Apply a sample: apply to the input a sample vector $\mathbf{u}^k$ having desired output vector $\mathbf{y}^k$;

Step 2. Forward Phase:

Starting from the first hidden layer and propagating towards the output layer:

2.1. Calculate the activation values for the units at layer *L* as:

2.1.1. If *L*-1 is the input layer

$$a_{h_L}^k = \sum_{j=0}^{N} w_{jh_L} u_j^k$$

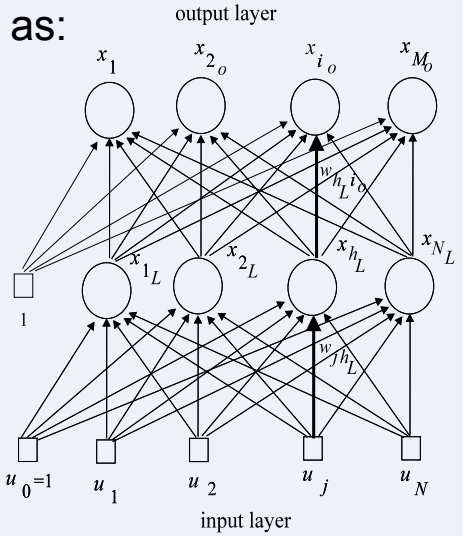
2.1.2. If *L*-1 is a hidden layer

$$a_{h_L}^k = \sum_{j_{L-1}=0}^{N_{L-1}} w_{j_{(L-1)}h_L} x_{j_{(L-1)}}^k$$

2.2. Calculate the output values for the units at layer *L* as:

$$x_{h_L}^k = f_L(a_{h_L}^k)$$

in which use index i_o instead of h_L if it is an output layer



The Backpropagation Algorithm

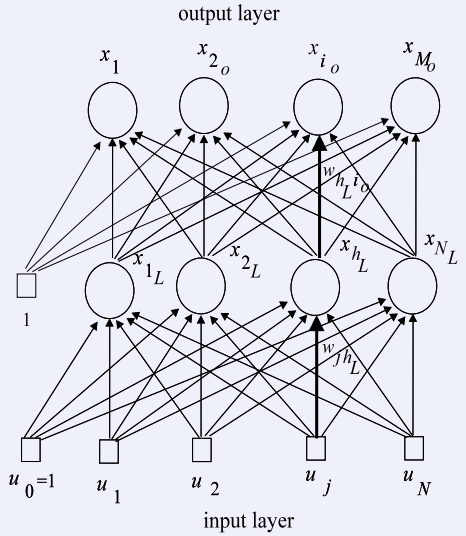
Step 4. Output errors: Calculate the error terms at the output layer as:

$$\delta_{i_o}^k = (y_{i_o}^k - x_{i_o}^k) f_o'(a_{i_o}^k)$$

Step 5. Backward Phase Propagate error backward to the input layer through each layer L using the error term

$$\delta_{h_L}^k = f_L'(a_{h_L}^k) \sum_{i_{L+1}=1}^{N_{L+1}} \delta_{i_{(L+1)}}^k w_{h_L i_{(L+1)}}^k$$

in which, use i_o instead of $i_{(L+1)}$ if L+1 is an output layer;



The Backpropagation Algorithm

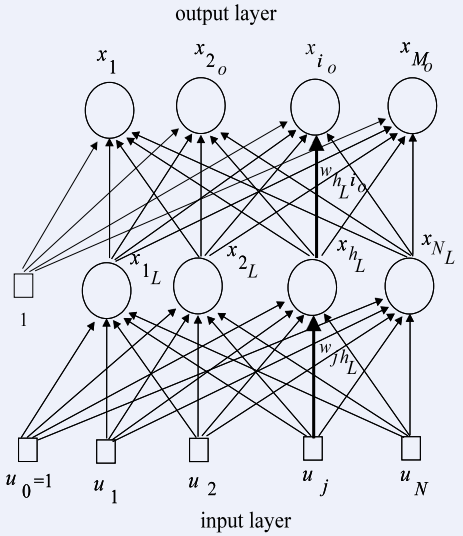
Step 6. Weight update: Update weights according to the formula

$$w_{j_{(L-1)}h_L}(t+1) = w_{j_{(L-1)}h_L}(t) + \eta\delta^{k}_{h_L} x^{k}_{j_{(L-1)}}$$

Step7. Repeat steps 1-6 until the stop criterion is satisfied, which may be chosen as the mean of the total error

$$< e^k >=< \tfrac{1}{2}\sum_{i_o=1}^{M}(y^k_{i_o} - x^k_{i_o})^2 >$$

is sufficiently small.



Deep Learning: Motivation

- Shallow models (SVMs, one-hidden-layer NNets, boosting, etc…) are unlikely candidates for learning high-level abstractions needed for AI
- Supervised training of many-layered NNets is a difficult optimization problem since there are too many weights to be adjusted but labeled samples are limited.
- There is a huge collection of unlabeled data, it is hard to label them.
- Unsupervised learning could do “local-learning” (each module tries its best to model what it sees)

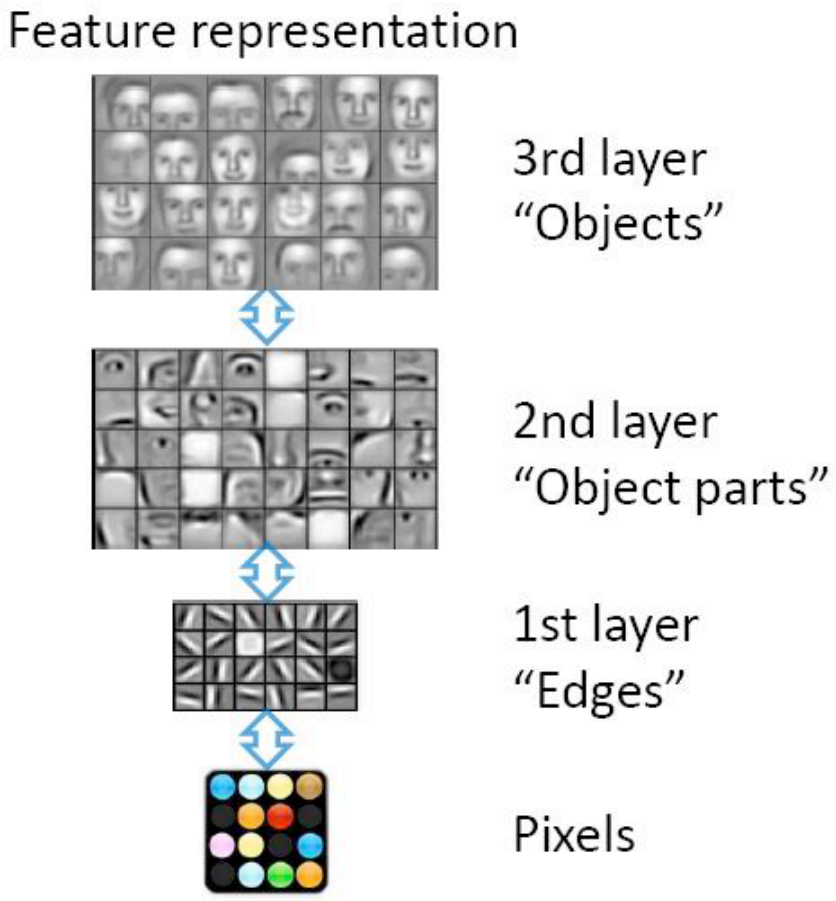
Deep Learning: Overview

- Deep learning is usually best when input space is locally structured spatially or temporally: images, language, etc. vs arbitrary input features
- In current models, layers often learn in an unsupervised mode and discover general features of the input space
- Then final layer features are fed into supervised layer(s)
  - And entire network is often subsequently tuned using supervised training of the entire net, using the initial weightings learned in the unsupervised phase
- Could also do fully supervised versions, etc. (early BP attempts)

Deep Learning

Deep Learning: Overview

- Multiple layers work to build an improved feature space
  - First layer learns 1st order features (e.g. edges…)
  - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
  - further layers learn more complex features



Deep Learning

Convolutional Neural Networks (CNN)

- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks
  - Similarities to Neo-Cognitron

Common Characteristics:

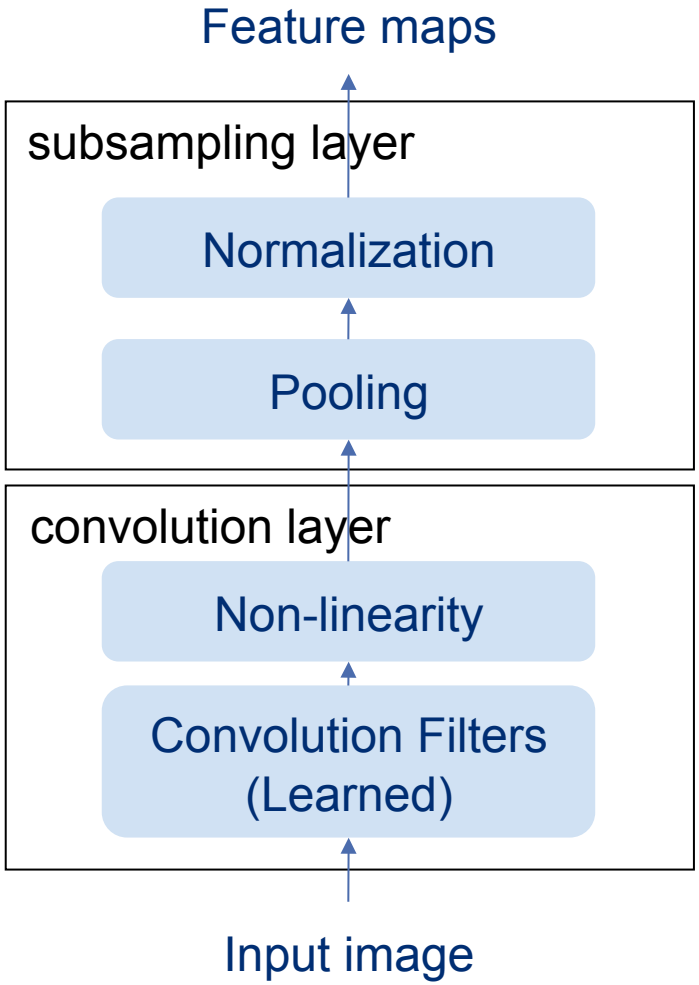
- A special kind of **multi-layer** neural networks.
- Implicitly **extract relevant features**.
- A **feed-forward** network that can extract topological properties from an image.
- Like almost every other neural networks CNNs are **trained** with a version of the **back-propagation** algorithm.

Deep Learning

Convolutional Neural Networks

ConvNet (Y. LeCun and Y. Bengio 1995)

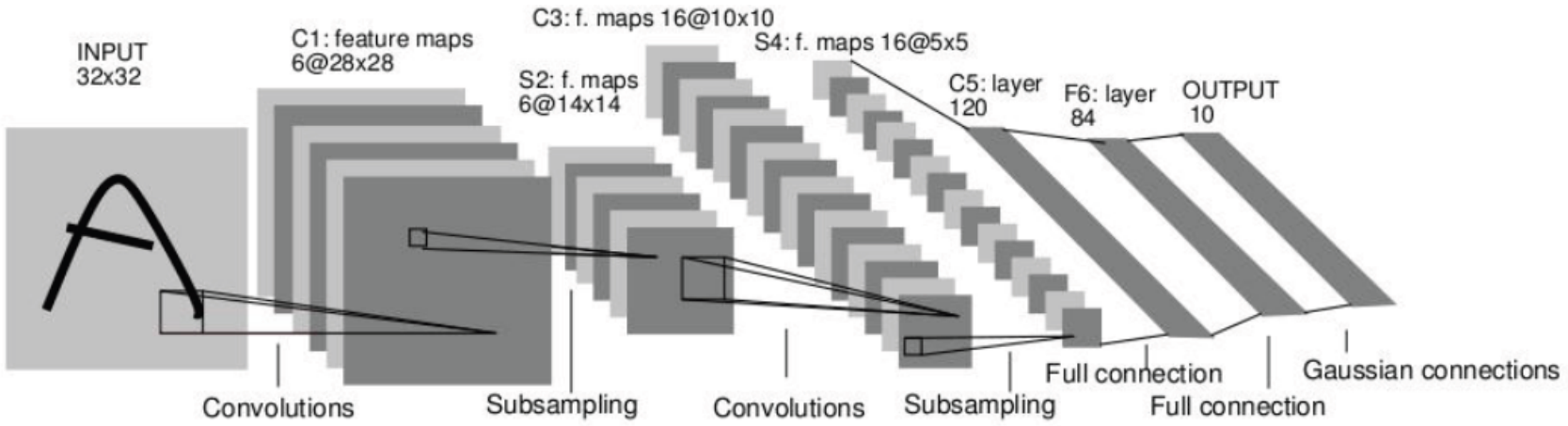
- Neural network with specialized connectivity structure
- Feed-forward:
  - Convolve input (apply filter)
  - Non-linearity (rectified linear)
  - Pooling (local average or max)
- Train convolutional filters by back-propagating classification error



Deep Learning

Convolutional Neural Networks

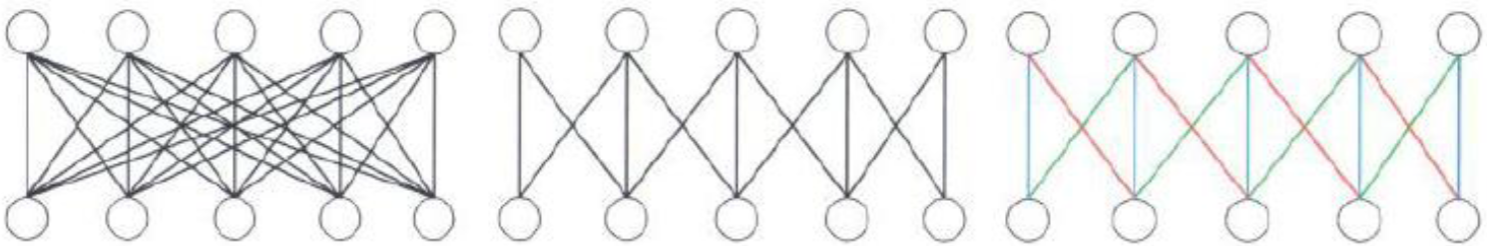
Convolution and Subsampling layers repeated many times



neuro sci tech

Convolutional Neural Networks

Connectivity & weight sharing depends on layer

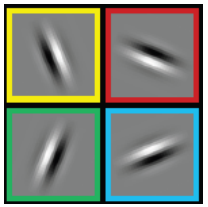


fully connected layer	locally connected layer	convolution layer
All different weights	All different weights	Shared weights

Convolution layer has much smaller number of parameters by local connection and weight sharing

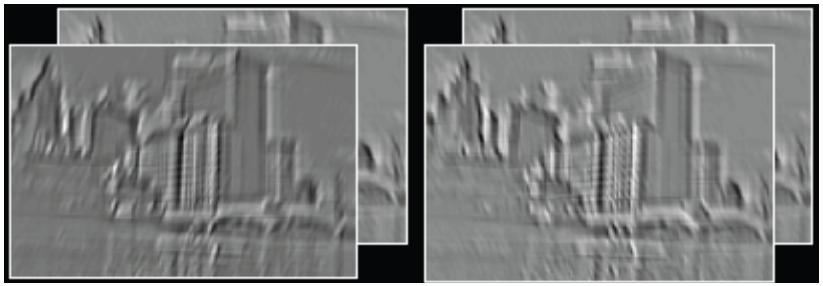
Convolution layer: filters

- Detect the same feature at different positions in the input image

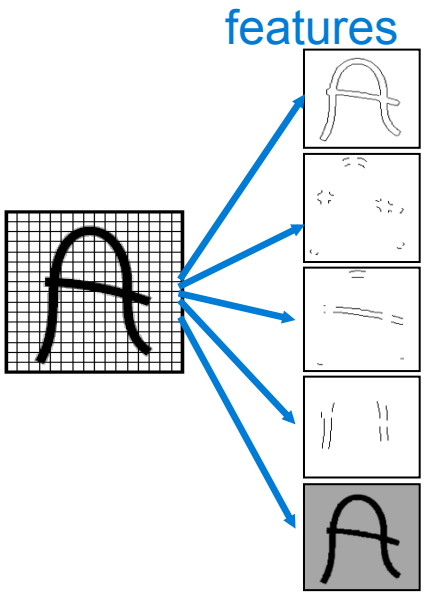


Filter (kernel)

Input

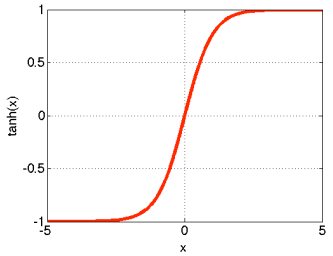


Feature map

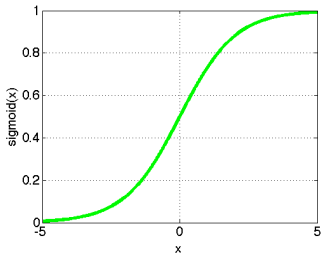


Nonlinearity

Tanh

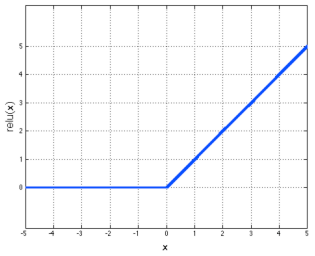


Sigmoid: 1/(1+exp(-x))



Rectified linear (ReLU) : max(0,x)

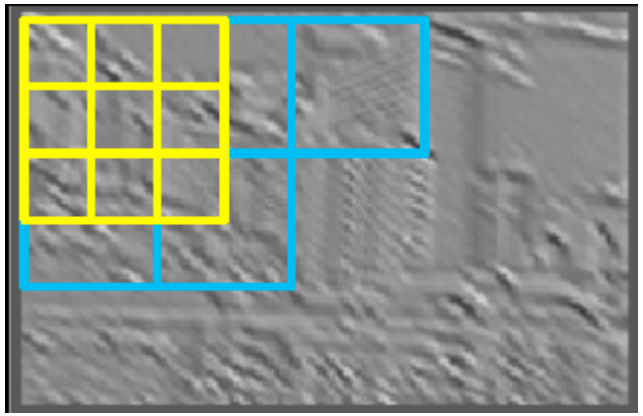
- Simplifies backprop
- Makes learning faster
- Make feature sparse

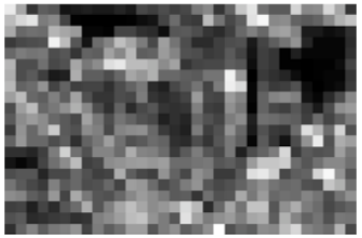


⟶ Preferred option

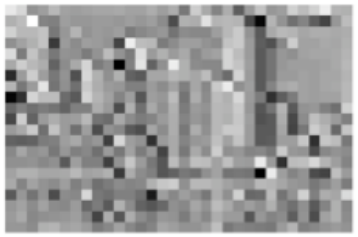
Sub-sampling layer

Spatial Pooling: Average or Max





Max



Average

Role of Pooling

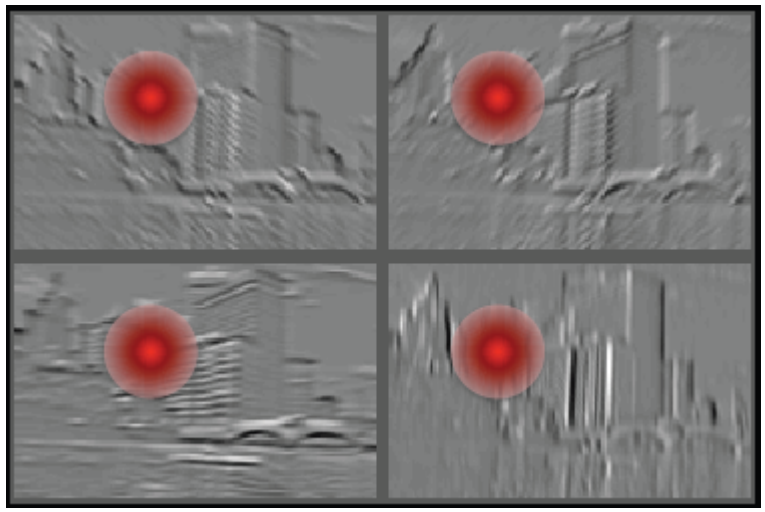
- Invariance to small transformations
- reduce the effect of noises and shift or distortion

Convolutional Neural Networks

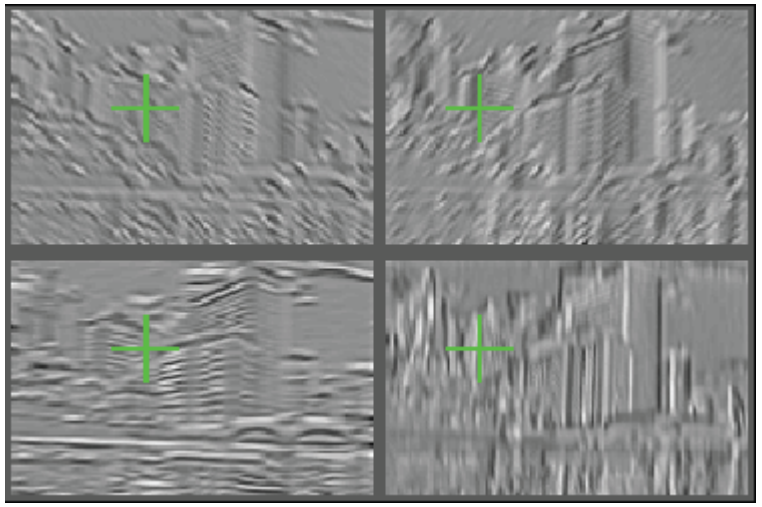
Normalization

Contrast normalization (between/across feature map)

- Equalizes the features map



Feature maps
before contrast normalization

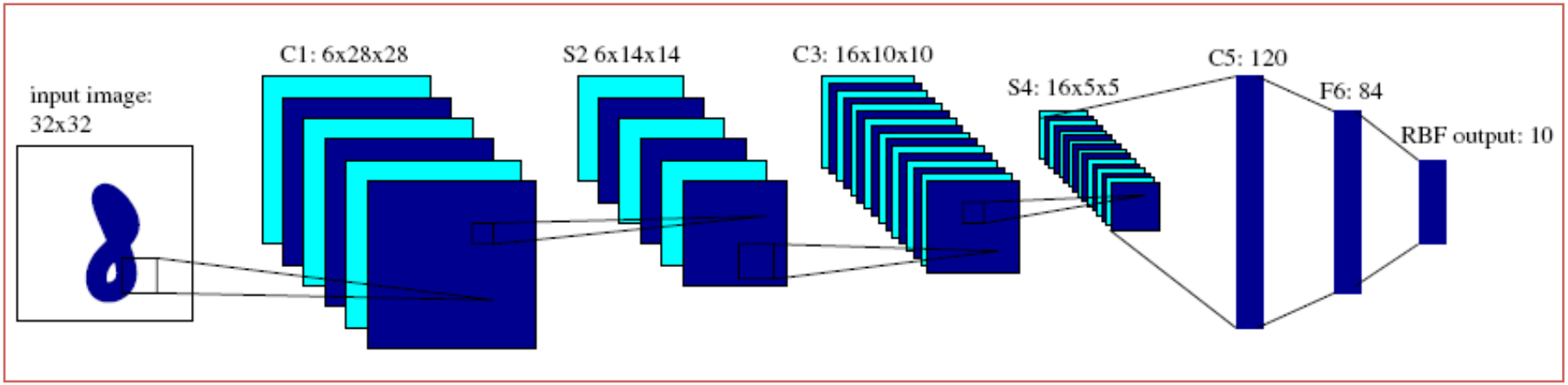


Feature maps
after contrast normalization

Deep Learning

Convolutional Neural Networks

L5Net : Convolution Network by LeCun for Handwritten Digit Recognition



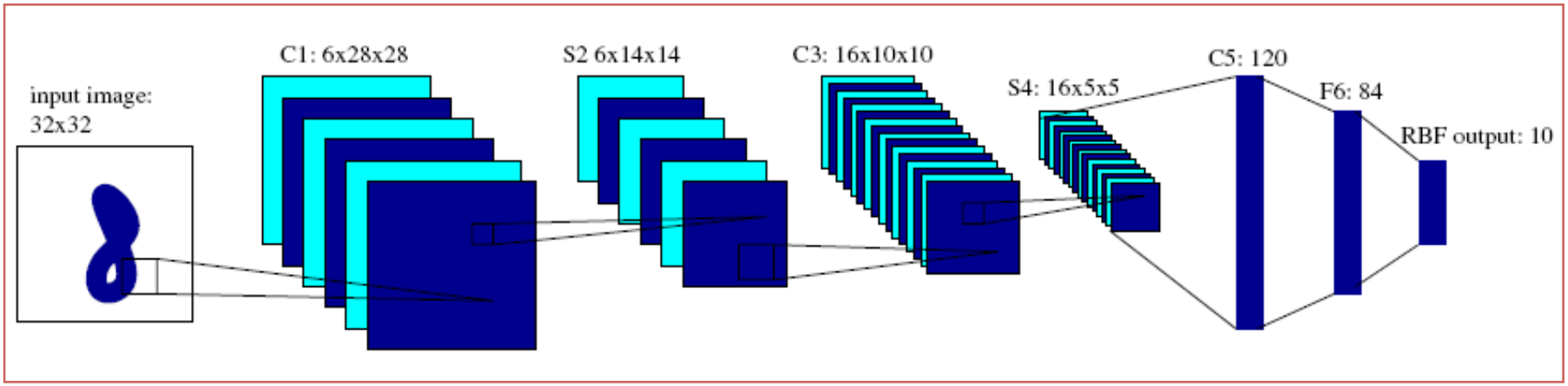
- C1,C3,C5 : Convolutional layer. (5 × 5 Convolution matrix.)
- S2 , S4 : Subsampling layer. (by factor 2)
- F6 : Fully connected layer.

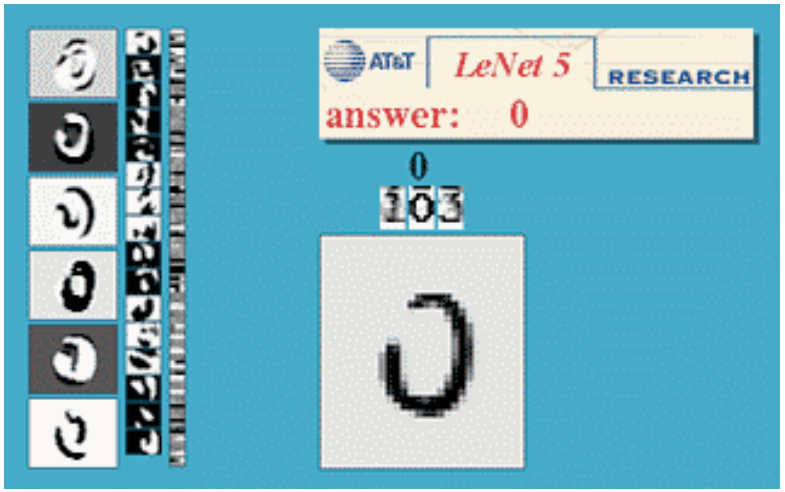
About 187,000 connection.

About 14,000 trainable weight.

Convolutional Neural Networks

L5Net : Convolution Network by LeCun for Handwritten Digit Recognition

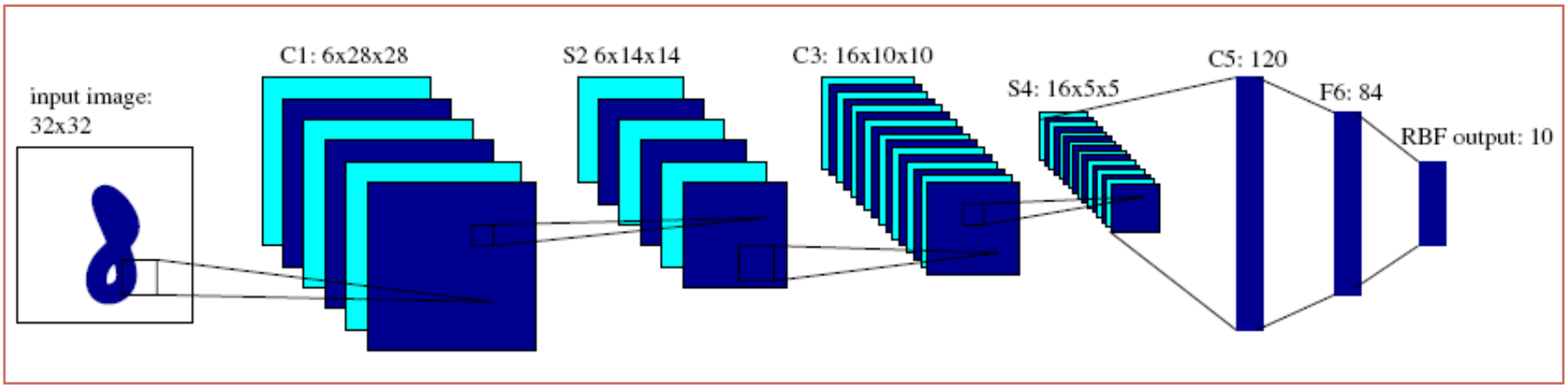


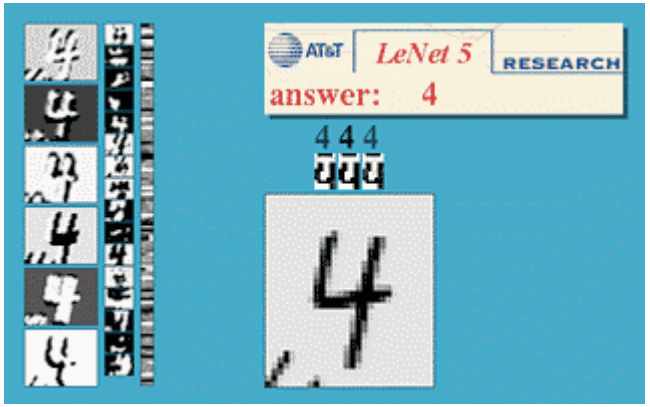


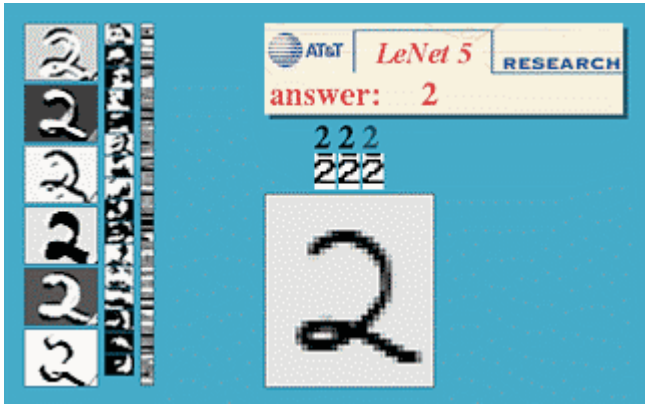
Deep Learning

Convolutional Neural Networks

L5Net : Convolution Network by LeCun for Handwritten Digit Recognition





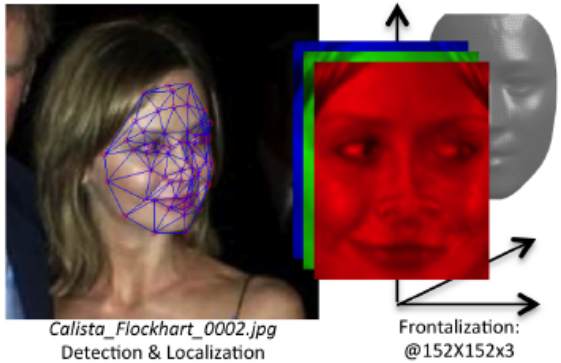


Deep Learning

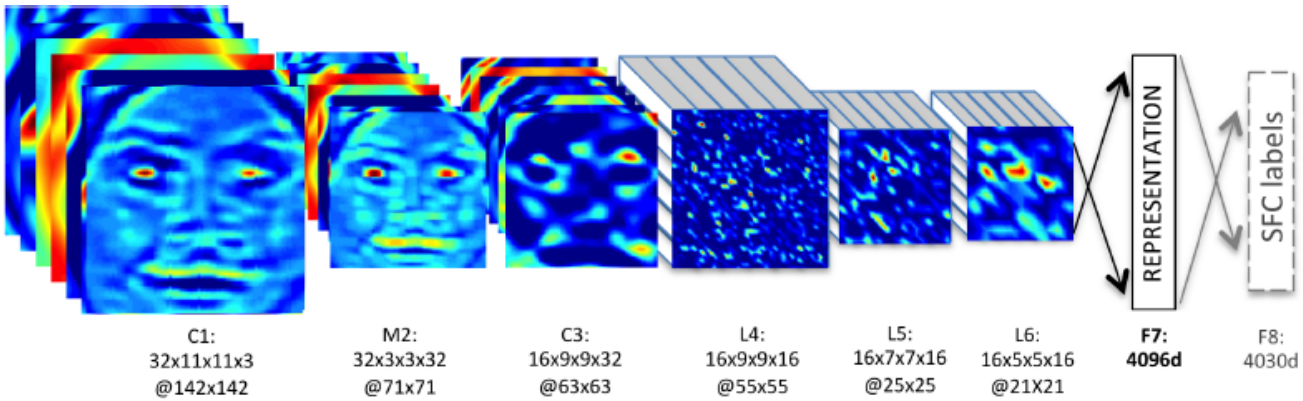
Convolutional Neural Networks

Deep Face: Taigman vd, CVPR 2014

Face Alignment



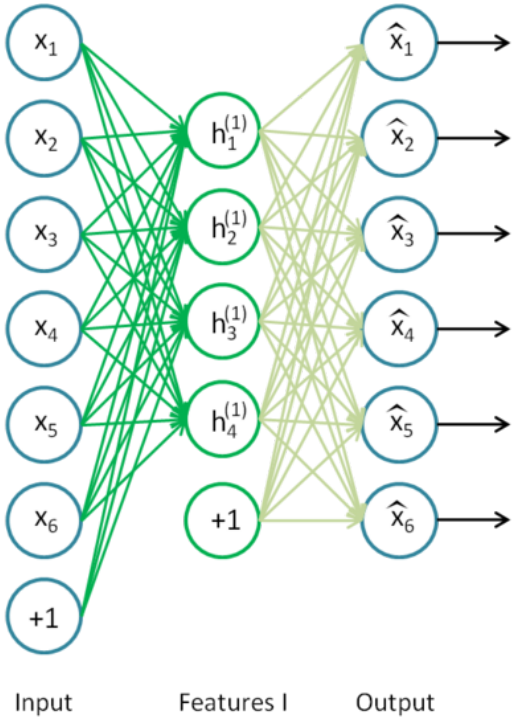
Representation(CNN)

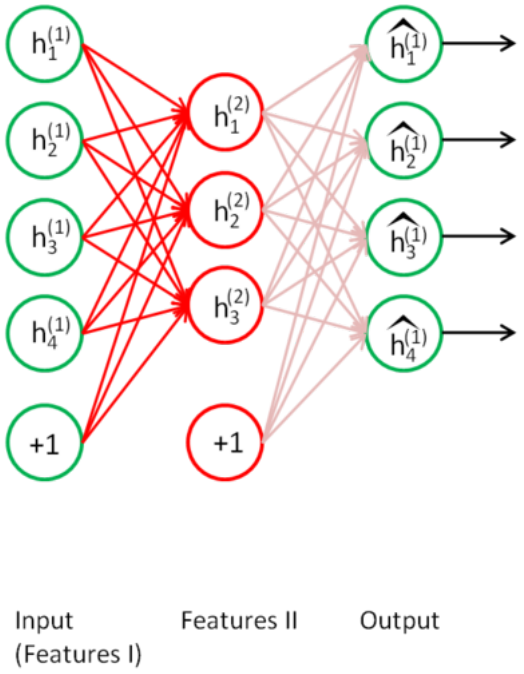


Deep Learning

Stacked Auto-Encoders

- Bengio (2007) – After Deep Belief Networks (Hinton, 2006)
- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training (for example Backpropogation)
- Drop the decode output layer each time

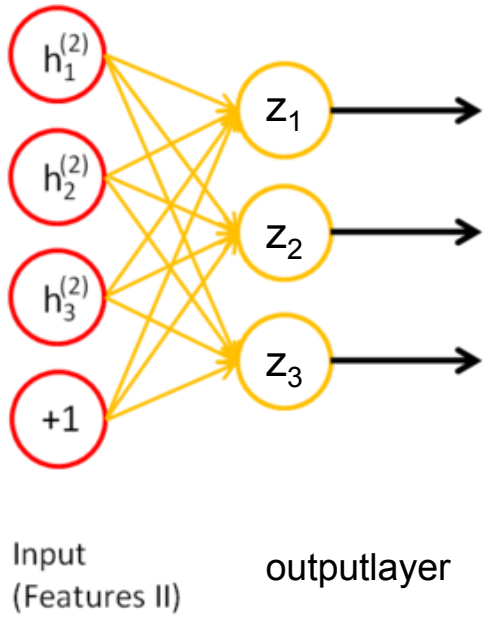




Deep Learning

Stacked Auto-Encoders

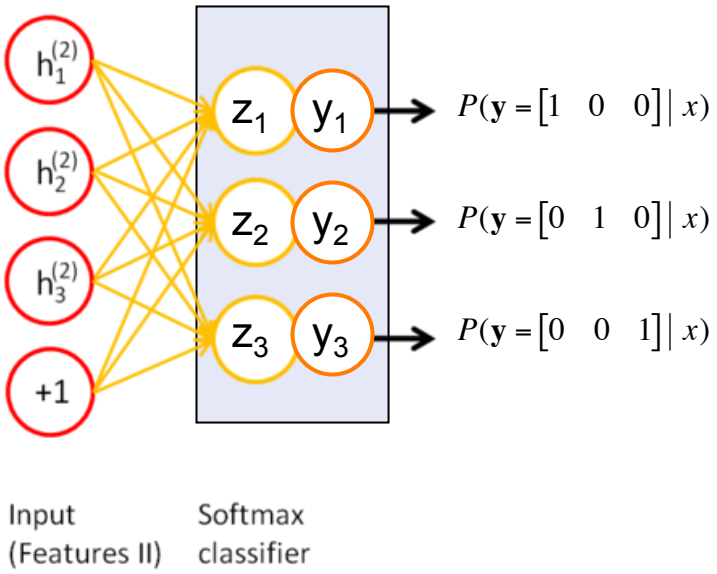
- Do supervised training on the last layer using final features



Deep Learning

Stacked Auto-Encoders

- Using Softmax classifier at the output layer results in better performance



$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

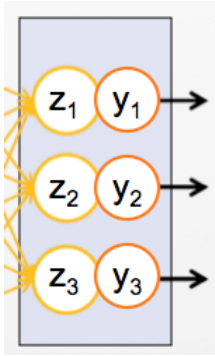
$$\frac{\partial y_i}{\partial z_i} = y_i\,(1 - y_i)$$

Stacked Auto-Encoders

To use softmax in Backpropagation learning

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

The output units use a non-local non-linearity:



$$\frac{\partial y_i}{\partial z_i} = y_i\,(1 - y_i)$$

$$E = -\sum_j t_j \ln y_j$$

target value

The natural cost function is the negative log probability of the right answer

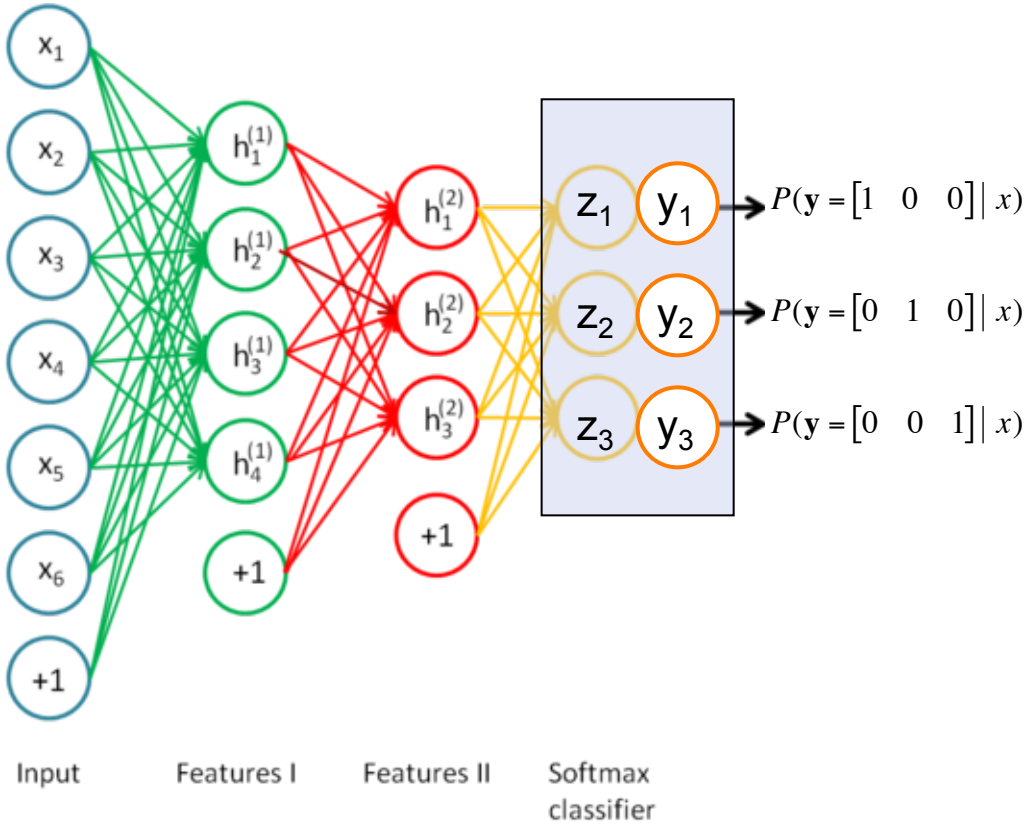
$$\frac{\partial E}{\partial z_i} = \sum_j \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Use gradient of E for updating the weights in the previous layers in Backpropagation

Deep Learning

Stacked Auto-Encoders

- Then do supervised training on the entire network to fine-tune all weights



Stacked Auto-Encoders with Sparse Encoders

- Auto encoders will often do a dimensionality reduction
  - PCA-like or non-linear dimensionality reduction
- This leads to a "dense" representation which is nice in terms of use of resources (i.e. number of nodes)
  - All features typically have non-zero values for any input and the combination of values contains the compressed information

Stacked Auto-Encoders with Sparse Encoders

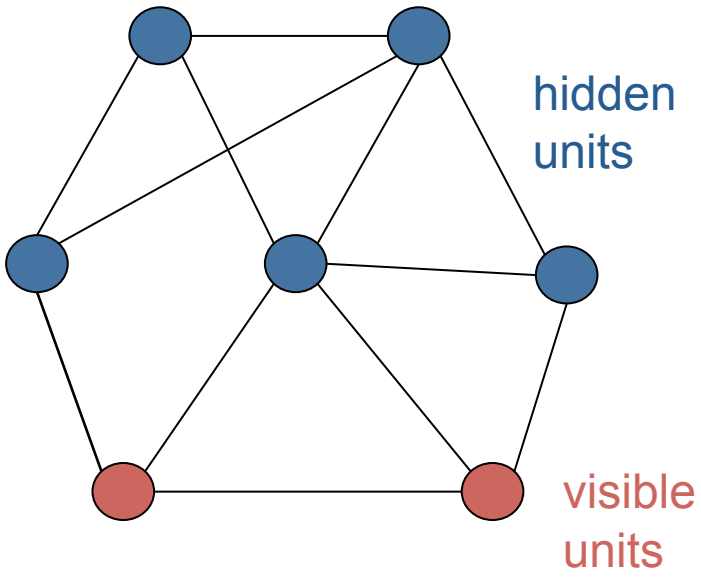
- However, this distributed representation can often make it more difficult for successive layers to pick out the salient features
- A **sparse** representation uses more features where at any given time a significant number of the features will have a 0 value
  - This leads to more localist variable length encodings where a particular node (or small group of nodes) with value 1 signifies the presence of a feature (small set of bases)
- This is easier for subsequent layers to use for learning
- For sparse encoding:
  - Use more hidden nodes in the encoder
  - Use regularization techniques which encourage sparseness (e.g. a significant portion of nodes have 0 output for any given input)

Stacked Auto-Encoders: Denoising

- De-noising Auto-Encoder
  - Stochastically corrupt training instance each time, but still train auto-encoder to decode the uncorrupted instance, forcing it to learn conditional dependencies within the instance
  - Better empirical results, handles missing values well

Deep Boltzman Machine : Boltzman Machine (BM)

- Boltzman machine is a recurrent neural network having stochastic neurons and its dynamic is defined in terms of energies of joint configurations of the visible and hidden units



Deep Boltzman Machine: Stochastic Neuron

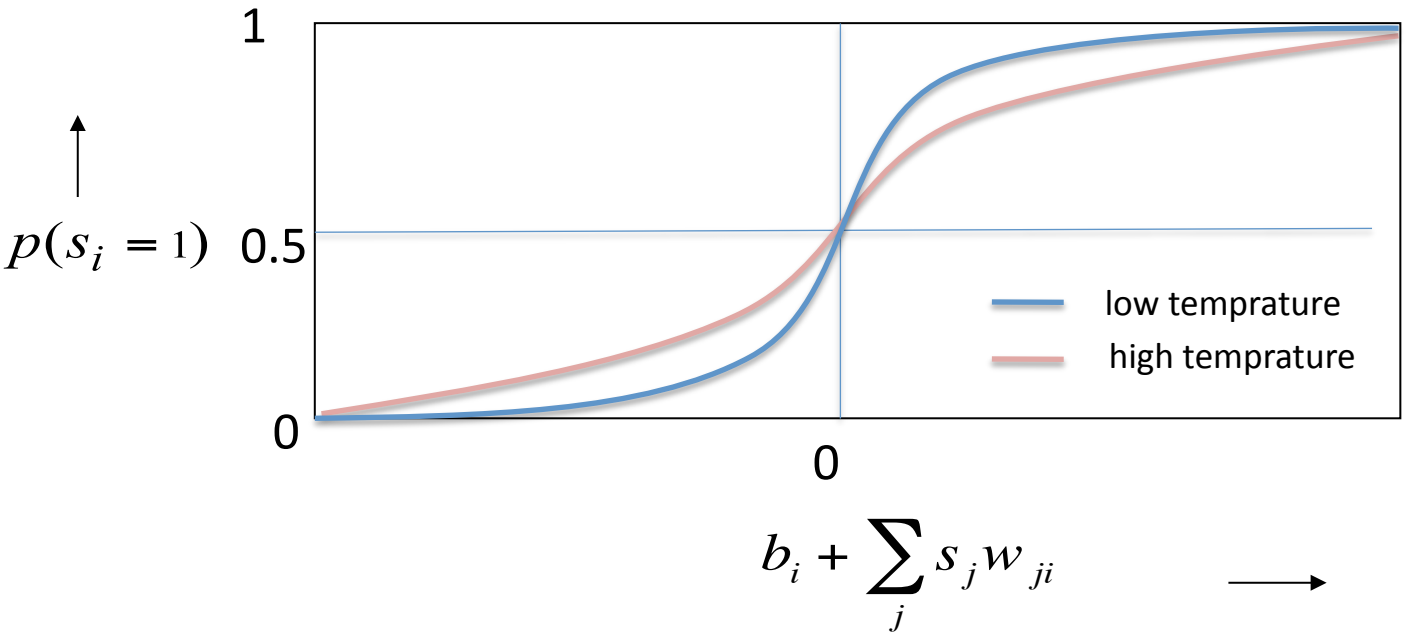
Stochastic neuron have a state of 1 or 0 which is a stochastic function of the neuron's bias, b, and the input it receives from other neurons.

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})/T} = \frac{1}{1 + \exp(-\Delta E_i / T)}$$

where T is a control parameter having analogy in temperature.

Deep Boltzman Machine : Stochastic Neuron

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})/T}$$



Deep Boltzman Machine : BM State Update

In a Boltzmann machine, a trial for a state transition is a two-step process.

1. Given a state of visible and hidden units (v,h), first a unit j is selected as a candidate to change state. The selection probability usually has uniform distribution over the units.
2. Then the chosen unit will have a state of 1 or 0 acccording to formula given before

Deep Boltzman Machine : BM Energy

binary state of unit i in joint configuration v,h

$$E(v,h) = -\sum_{i \in units} s_i^{vh} b_i - \sum_{i<j} s_i^{vh} s_j^{vh} w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

bias of unit i

indexes every non-identical pair of i and j once

weight between units i and j

Stacked Deep Boltzman Machine : BM Energy

- The probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations.

  (*T*: temprature, it drops in $p(v,h)$ equation when it is set as $T=1$)

$$p(v,h) = \frac{e^{-E(v,h)/T}}{\sum_{u,g} e^{-E(u,g)/T}}$$

partition function

- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

$$p(v) = \frac{\sum_{h} e^{-E(v,h)/T}}{\sum_{u,g} e^{-E(u,g)/T}}$$

Deep Boltzman Machine : BM Learning

- Everything that one weight needs to know about the other weights and the data in order to do maximum likelihood learning is contained in the difference of two correlations.

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \left\langle s_i s_j \right\rangle_{\mathbf{v}} - \left\langle s_i s_j \right\rangle_{free} .$$

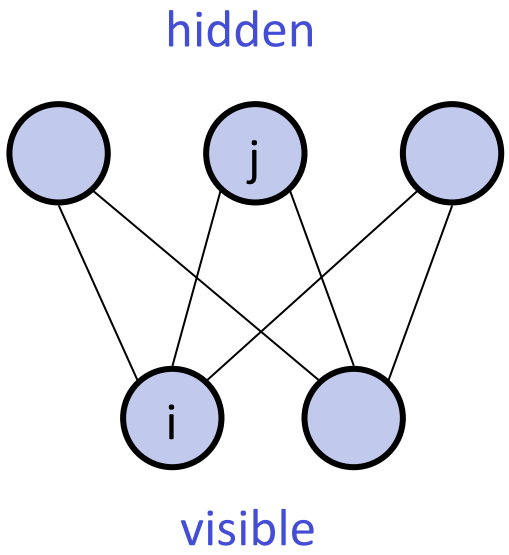
Derivative of log probability of one training vector

Expected value of product of states at thermal equilibrium when the training vector is clamped on the visible units

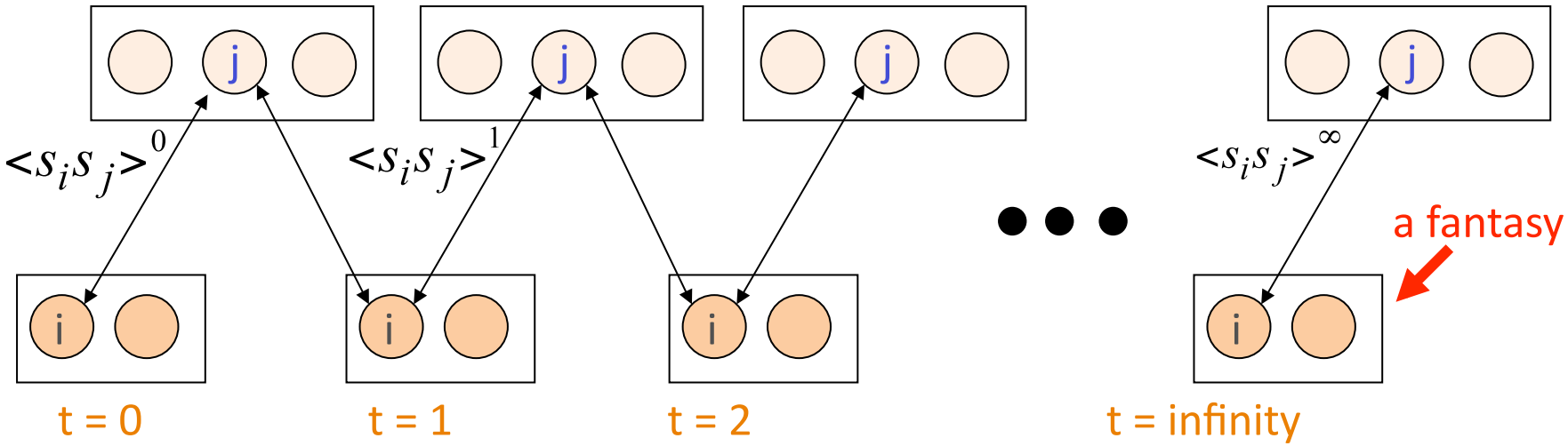
Expected value of product of states at thermal equilibrium when nothing is clamped

Deep Boltzman Machine : Restricted BM (RBM)

- In restricted Boltzman machine the connectivity is restricted to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states. It only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of $\langle s_i s_j \rangle$ to be used in training



Deep Boltzman Machine : Single RBM Learning



Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

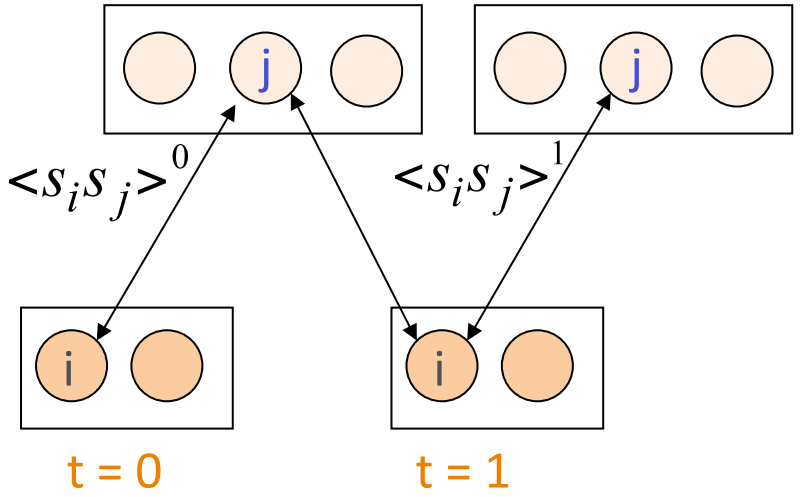
$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^\infty \right)$$

Deep Boltzman Machine : Single RBM Learning

Contrastive divergence learning:

A quick way to train (learn) an RBM

- Start with a training vector on the visible units.
- Update all the hidden units in parallel
- Update the all the visible units in parallel to get a "reconstruction".
- Update the hidden units again.

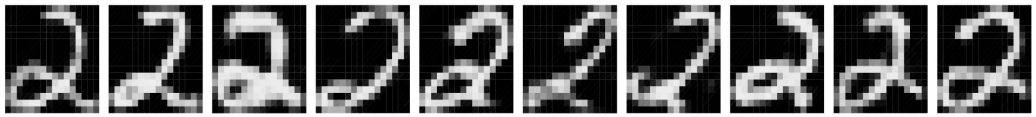


This is not following the gradient of the log likelihood. But it works well.

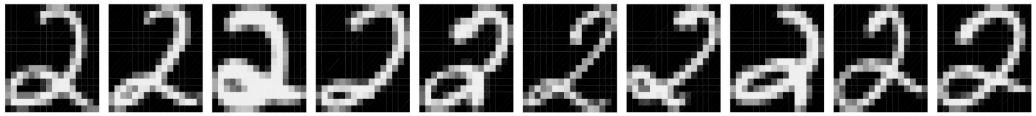
$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^1 \right)$$

Deep Boltzman Machine : Single RBM Learning

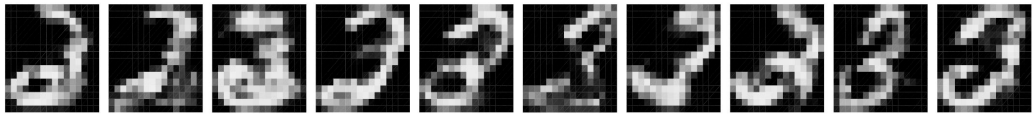
Using an RBM to learn a model of a digit class



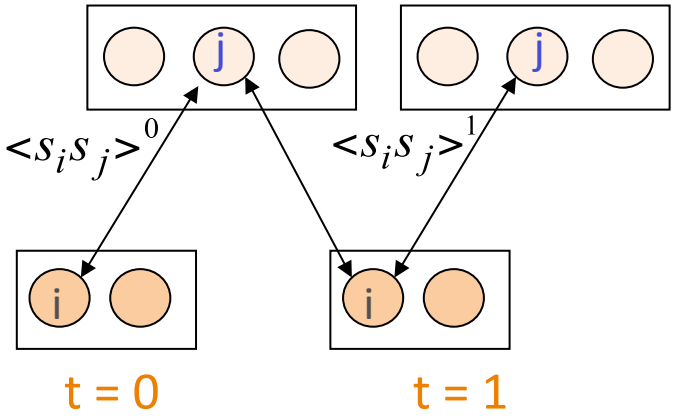
Reconstructions by model trained on 2's



Data



Reconstructions by model trained on 3's



100 hidden units (features)

256 visible units (pixels)

Deep Boltzman Machine : Pre-training DBM

Training a deep network by stacking RBMs

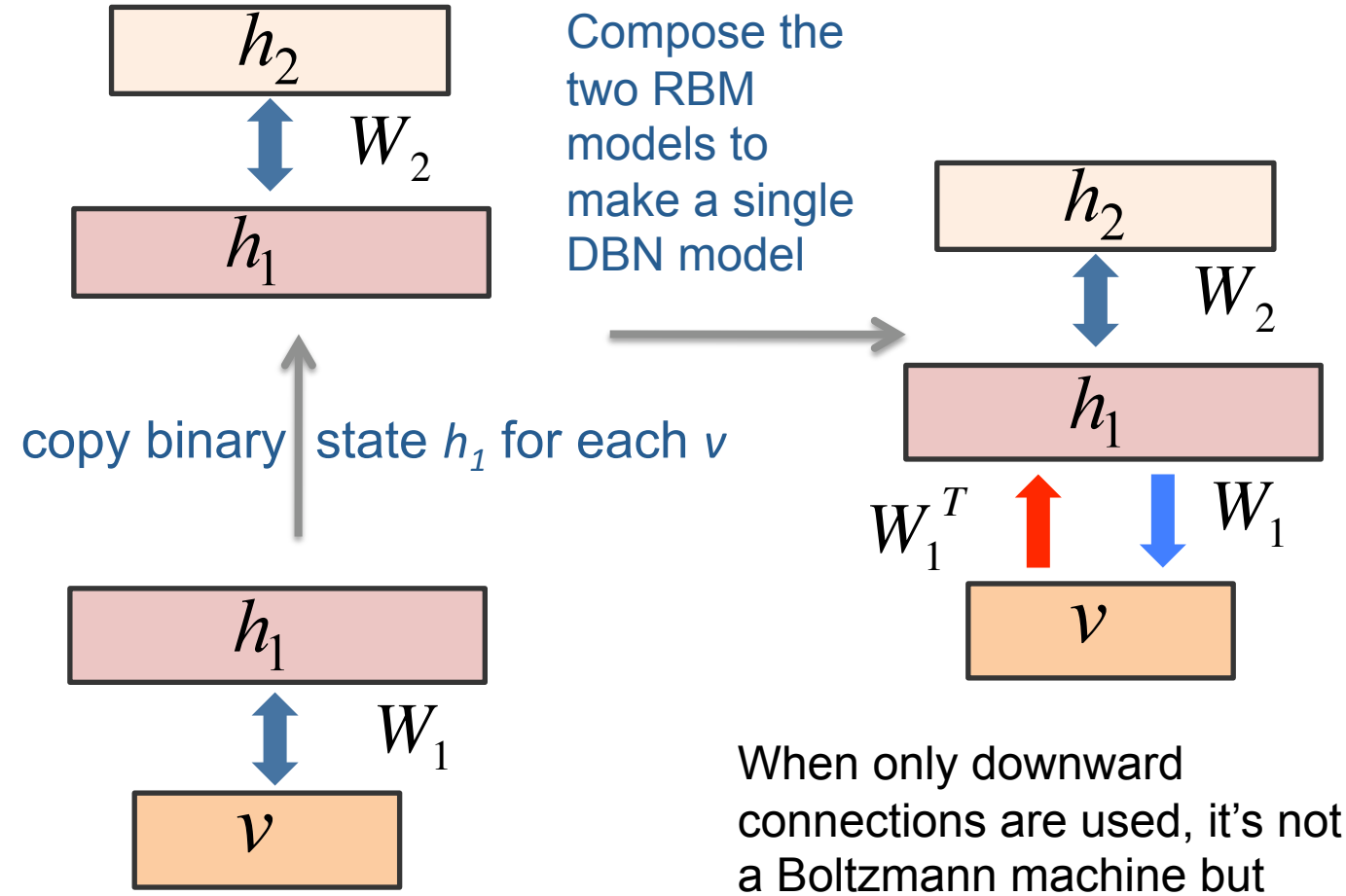
- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features in a second hidden layer.
- Then do it again for the next layers

Deep Boltzman Machine : Pre-training DBM

Combining RBMs to make a Deep BM

Then train this RBM

Train this RBM first



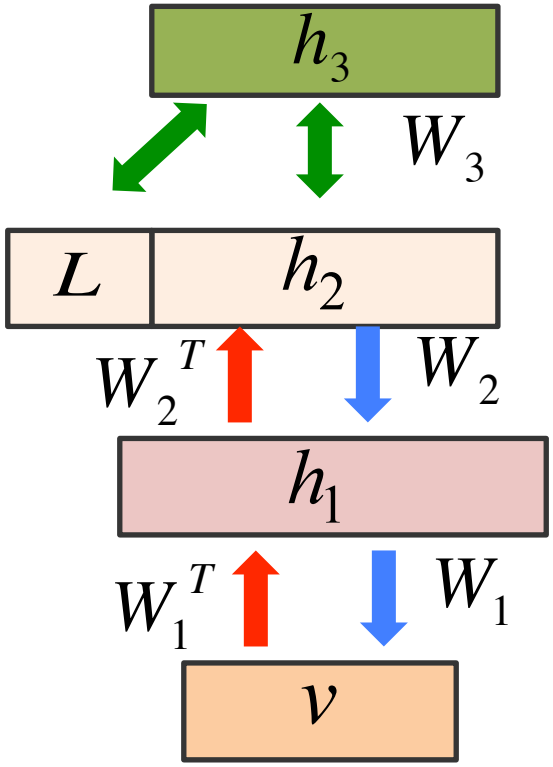
When only downward connections are used, it's not a Boltzmann machine but Deep Belief Network

Deep Learning

Deep Boltzman Machine : Pre-training DBM

In order to use DBM for recognition (discrimination):

- in top RBM use also label units: L
  - use K label nodes iff there are K classes
  - a label vector is obtained by setting the node in L corresponding to the class of the data to “1”, while all the others are “0”
- train the top RBM together with label units
- The model learns a joint density for labels and images.
- Apply pattern at bottom layer and then observe Label at the top layer



3 layer DBM

Deep Boltzman Machine : Pre-training DBM

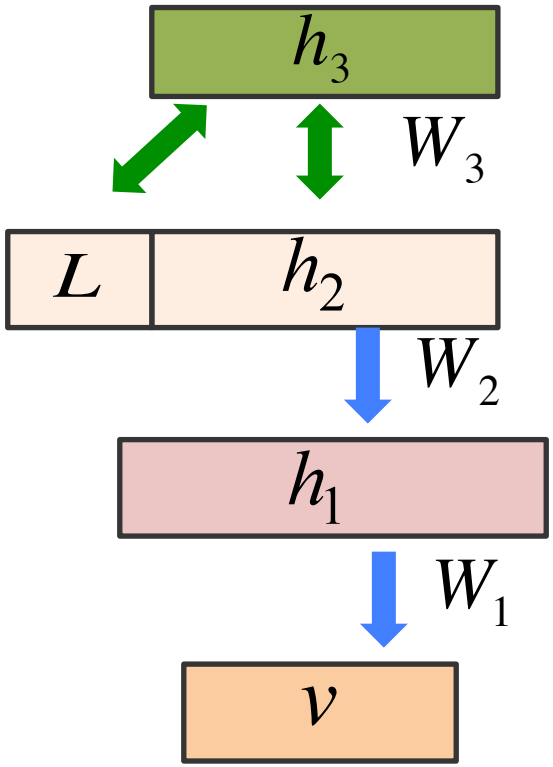
To generate data:

Use generation connections:

W3: recurrent; W1, W2 : feedforward

1. Clamp L to a label vector and then get an equilibrium sample from the top level RBM by performing alternating Gibbs sampling for a long time (using green connections)
2. Perform a top-down pass to get states for all the other layers (using blue connections).

- The lower level bottom-up (red) connections are not part of the generative model.



3 layer DBN

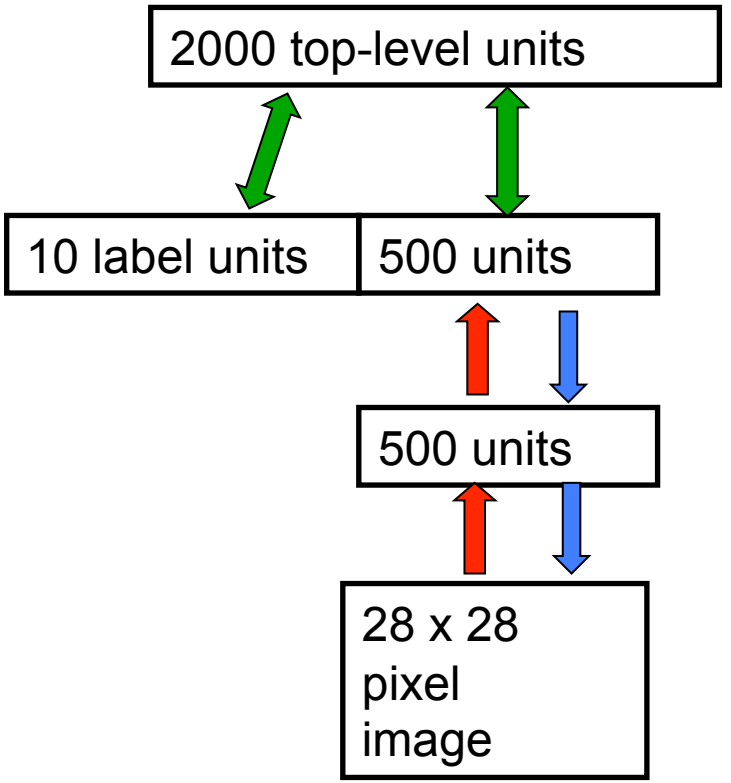
Deep Learning

A neural network model of digit recognition & modeling

- hand written digits as 28x28 images for digits 0,1,2..9
- 10 classes so 10 label units

see movies at :

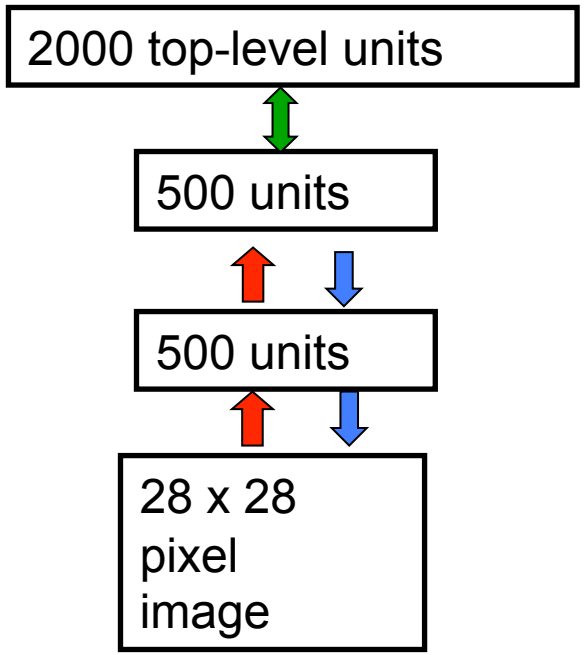
http://www.cs.toronto.edu/~hinton/digits.html



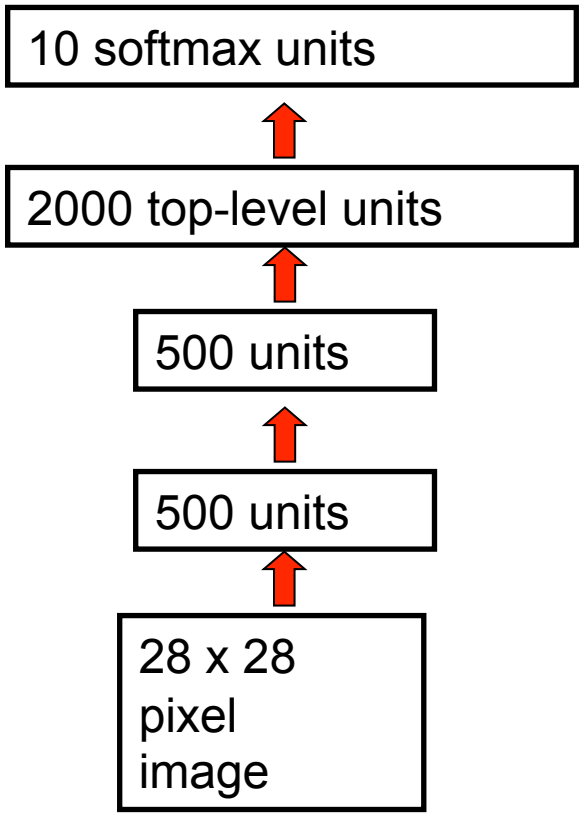
Deep Learning

A neural network model of digit recognition : fine tuning

- First learn one layer at a time by stacking RBMs: Treat this as **"pre-training"** that finds a good initial set of weights which can then be fine-tuned by a local search procedure.



- Then add a 10-way softmax at the top and do backpropagation



Deep Learning

More Application examples on Deep Neural Networks

Speech Recognition Breakthrough for the Spoken, Translated Word

https://www.youtube.com/watch?v=Nu-nlQqFCKg

Published on Nov 8, 2012 Microsoft Chief Research Officer Rick Rashid demonstrates a speech recognition breakthrough via machine translation that converts his spoken English words into computer-generated Chinese language. The breakthrough is patterned after deep neural networks and significantly reduces errors in spoken as well

Deep Learning

More Application examples

- June 2012, Google demonstrated one of the largest neural networks yet, with more than a billion connections. A team led by Stanford computer science professor Andrew Ng and Google Fellow Jeff Dean showed the system images from 10 million randomly selected YouTube videos. One simulated neuron in the software model fixated on images of cats. Others focused on human faces, yellow flowers, and other objects. By using the power of deep learning, the system identified these discrete objects even though no humans had ever defined or labeled them.

https://www.youtube.com/watch?v=-rIb_MEiylw

Deep Learning

neuro sci tech

Closing Remarks

"The basic idea—that software can simulate the neocortex's large array of neurons in an artificial "neural network"—is decades old, and it has led to as many disappointments as breakthroughs. But because of improvements in mathematical formulas and increasingly powerful computers, computer scientists can now model many more layers of virtual neurons than ever before.

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart."

Robert D. Hof on April 23, 2013, MIT Technology review

neuro sci tech

Credits

Deep learning network:

http://deeplearning.net/

Coursera Open Course: Neural Networks by Geofrey Hinton

https://www.coursera.org/course/neuralnets

METU EE543: Neurocomputers, Lecture Notes by Ugur Halıcı

http://www.eee.metu.edu.tr/~halici/courses/543LectureNotes/543index.html

Deep learning tutorial at Stanford

http://ufldl.stanford.edu/tutorial/