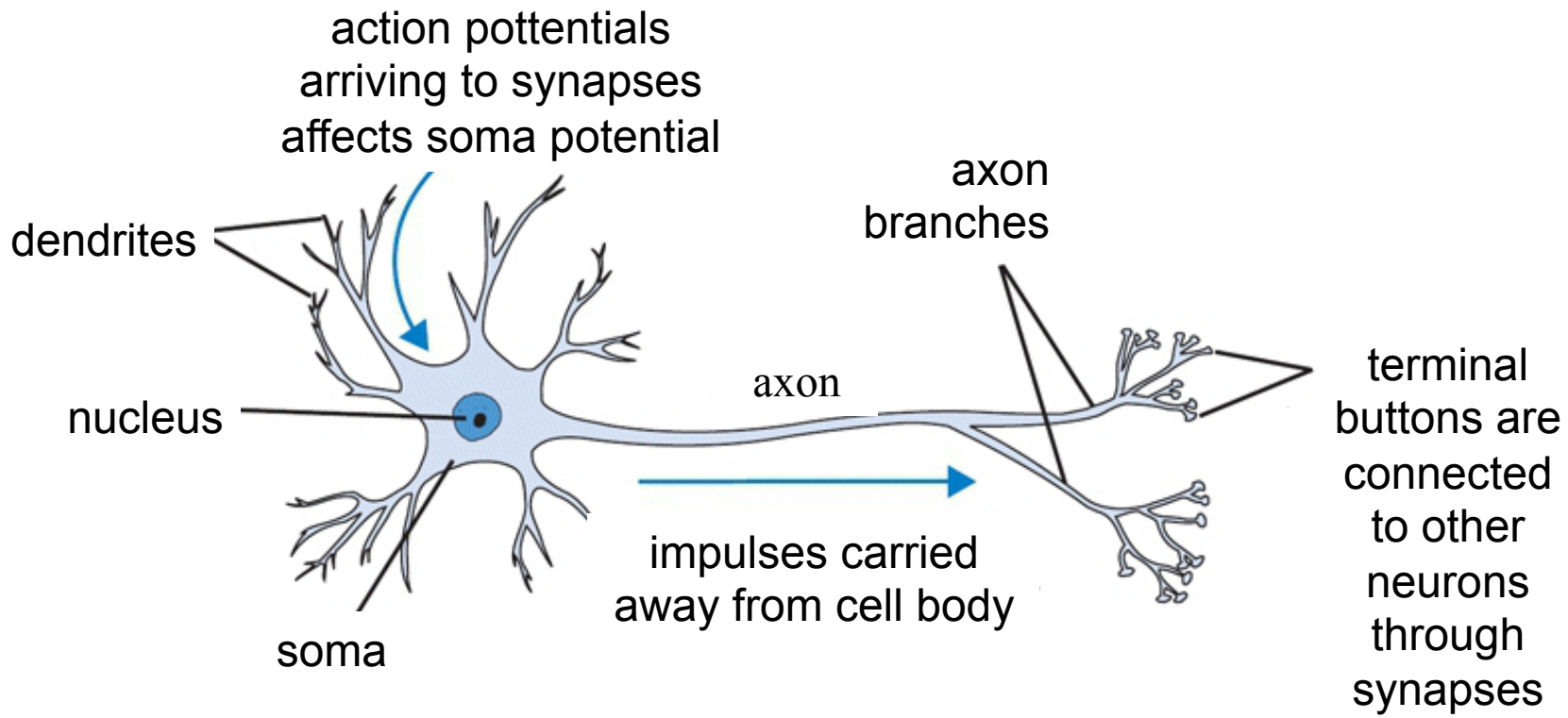
NEUROCOMPUTERS AND DEEP LEARNING

ANN ee543

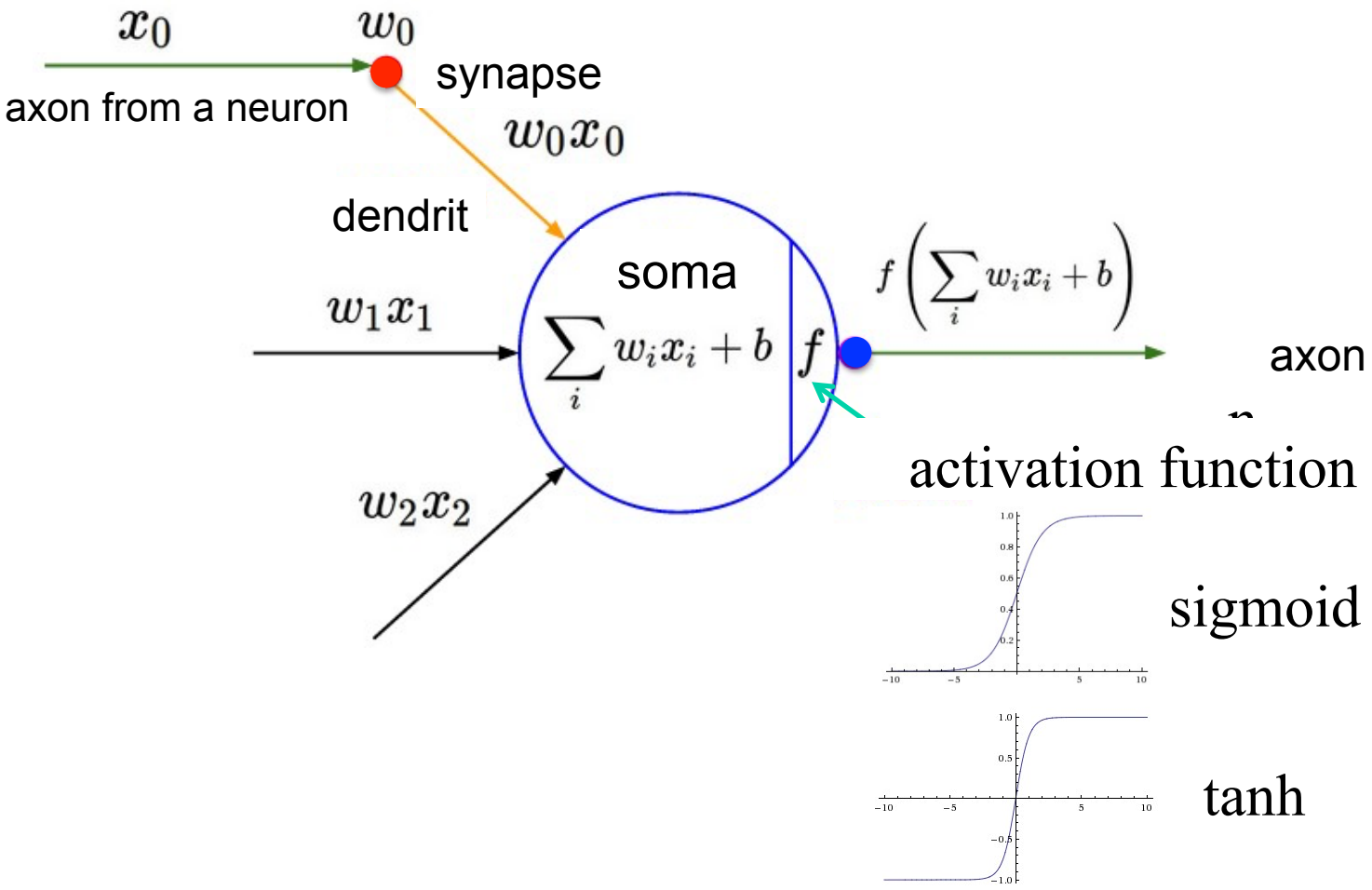
CHAPTER V

Deep Learning

Biological Neuron

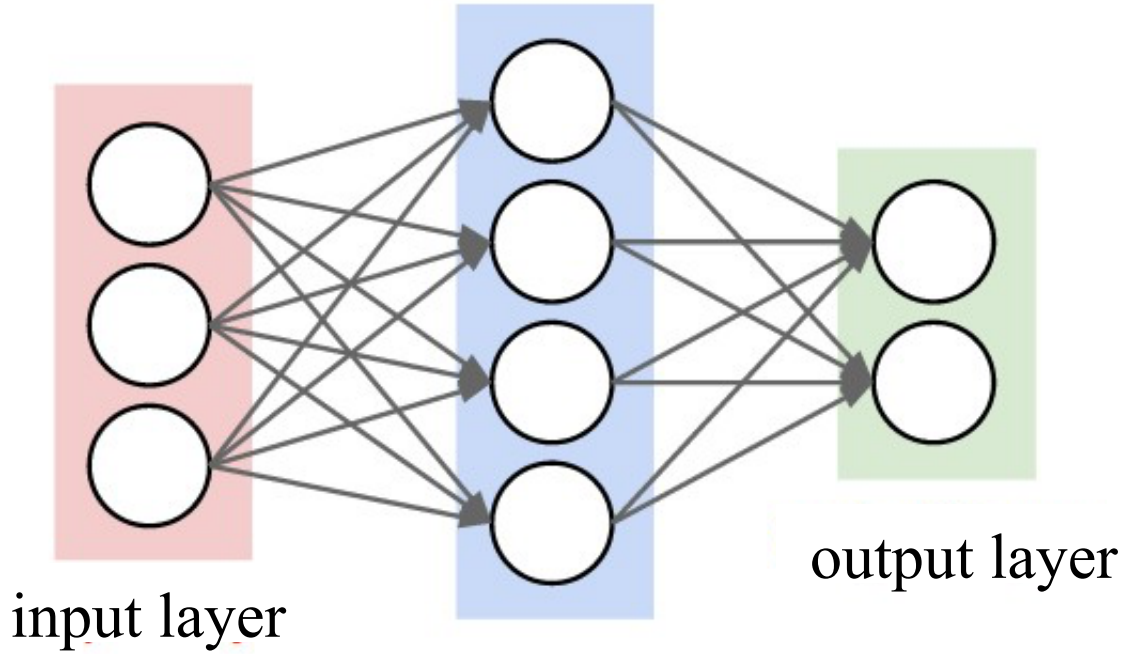


Artificial Neuron (Mc Culloch Pitts)



Fully Connected Multi Layer Network*

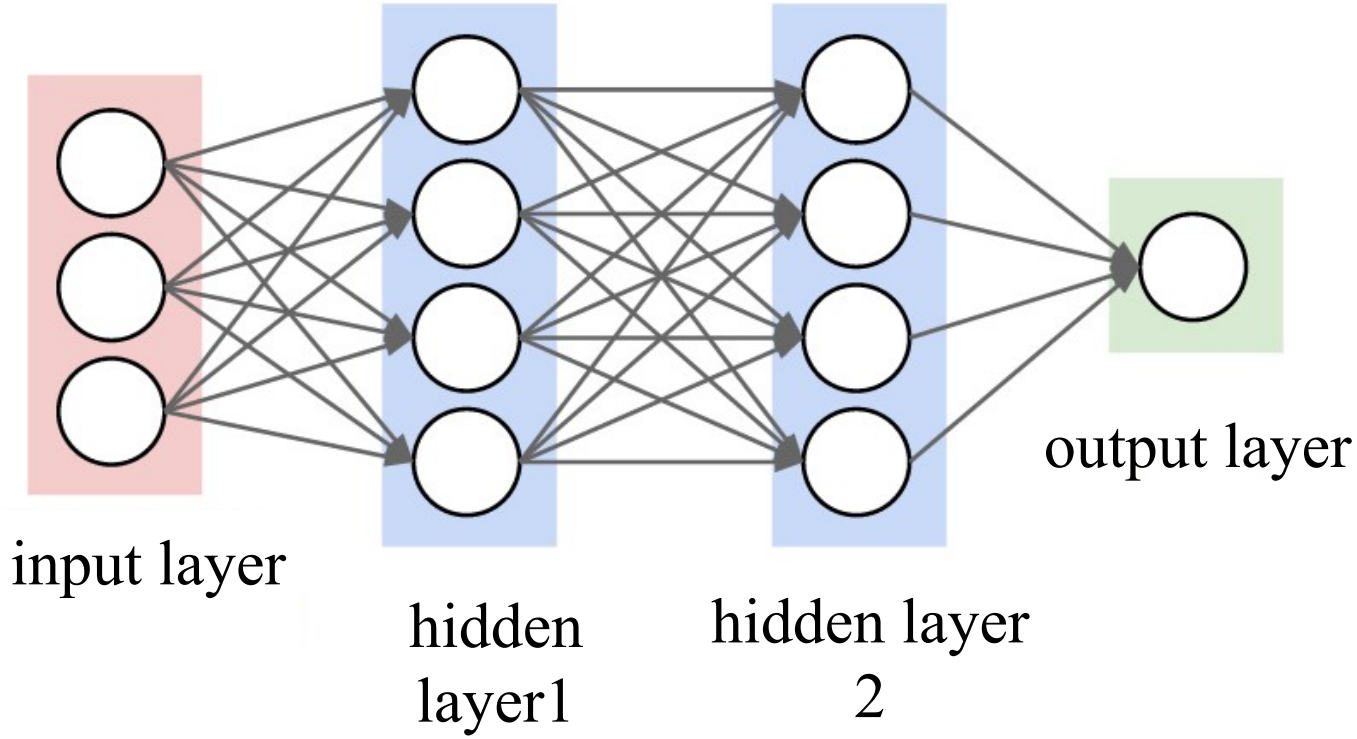
Example:



(*) also called Multilayer Perceptron (MLP)

Fully Connected Multi Layer Network

EXAMPLE:



MLP: Learning

Loss Function: L

- The loss function for training MLP is defined such that its value increases as the the error (i.e. difference between the desired output and the actual output of the network) increases.
- This function takes into account, the difference at each output neuron for each sample in the training set.
- Learning process is the update of the weights (i.e. parameters) in order to minimize the value of the loss function.

MLP: Learning

Gradient descent learning:

Gradient shows the the direction in which the loss function increases the most.

To decrease it the opposite direction of the gradient is considered.

The weight values are updated proportional to the gradient of the loss function with respect to weight.

$$\Delta w_{ij} = -\eta \frac{\partial L}{\partial w_{ij}}$$

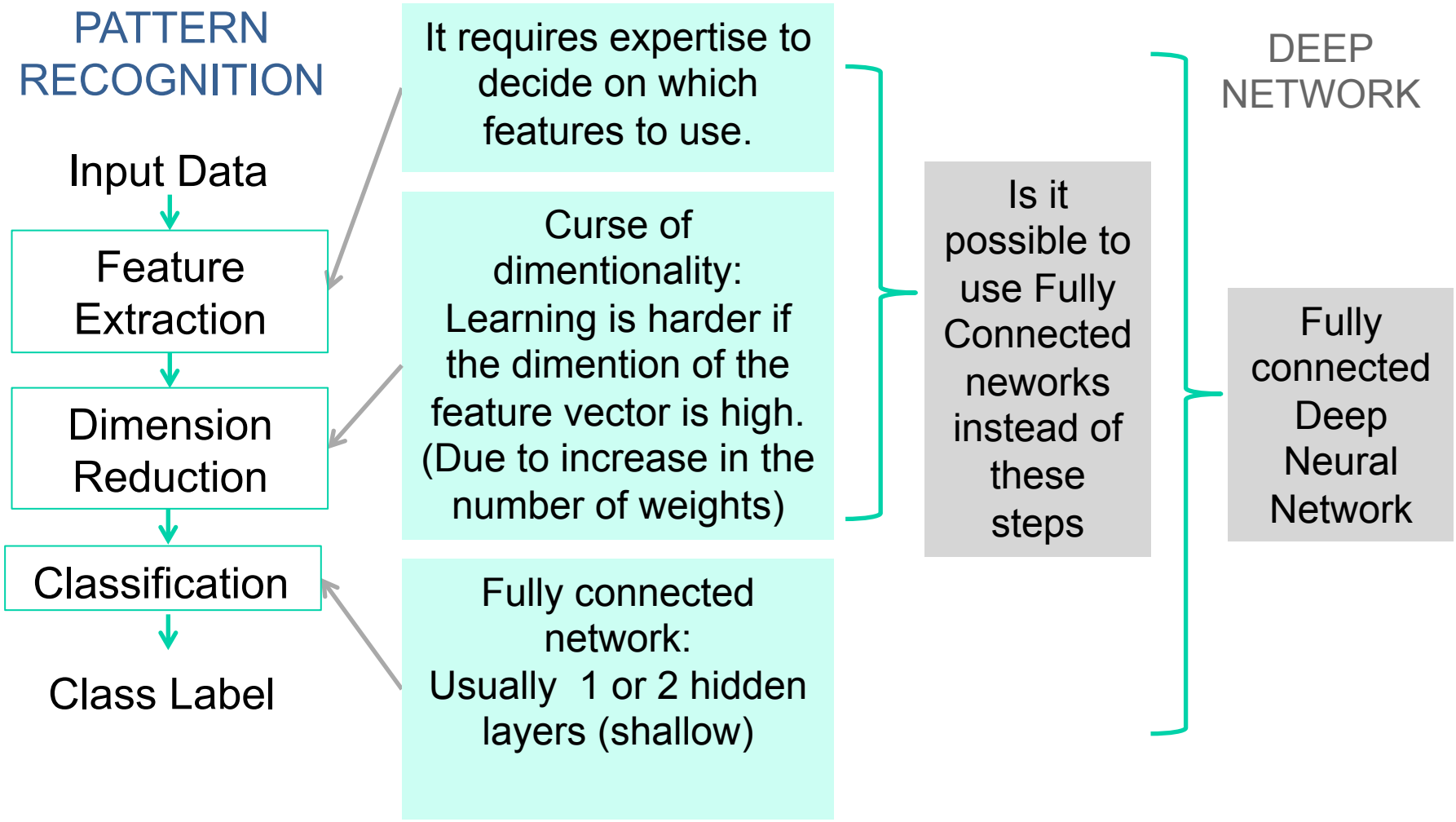
MLP: Learning

Backpropagation:

Backpropagation is the training algorithm based on gradient descent.

It helps to calculate gradient component for each connection weight iteratively based on the gradients calculated on the higher layers.

Deep Learning: Motivation



Deep Learning

PROBLEMS with Fully Connected Deep Network (Deep MLP):

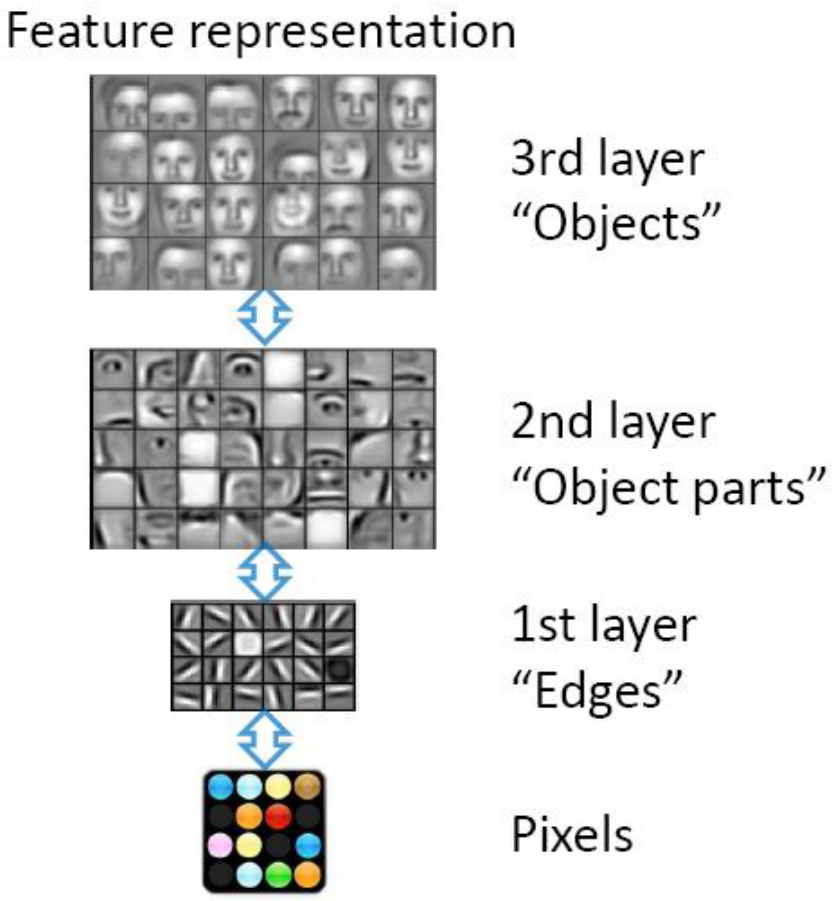
- Training takes long time due to huge number of parameters,
  - Is it possible to reduce the number of parameters?
  - Solution : Convolutional Neural Networks(CNN) :
    local connections and weight sharing (filters)
- Needs too many labeled samples
  - Is it possible to apply unsupervised training ?
  - Solution : Auto Encoder, Restricted Boltzman Machine (RBM)
- Partial derivatives at lower layers during back propagation disappear
  - Is it possible to rain layers one by one?
  - Solution: Stacked Auto Encoder, Deep RBM

Motivation

- Supervised training of deep models (e.g. many-layered NNets) is difficult (optimization problem)
- Shallow models (SVMs, one-hidden-layer NNets, boosting, etc…) are unlikely candidates for learning high-level abstractions needed for AI
- Unsupervised learning could do “local-learning” (each module tries its best to model what it sees)
- Inference (+ learning) is intractable in directed graphical models with many hidden variables

Deep Learning Overview

- Train networks with many layers (vs. shallow nets with just a couple of layers)
- 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, e.g.combinations of edges, etc.)
  - further layers learn more complex features
- Usually best when input space is locally structured – spatial or temporal: images, language, etc. vs arbitrary input features



Deep Learning Overview

- In current models layers often learn in an unsupervised mode and discover general features of the input space – serving multiple tasks related to the unsupervised instances (image recognition, etc.)
- 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)

Convolutional Neural Networks (CNN)

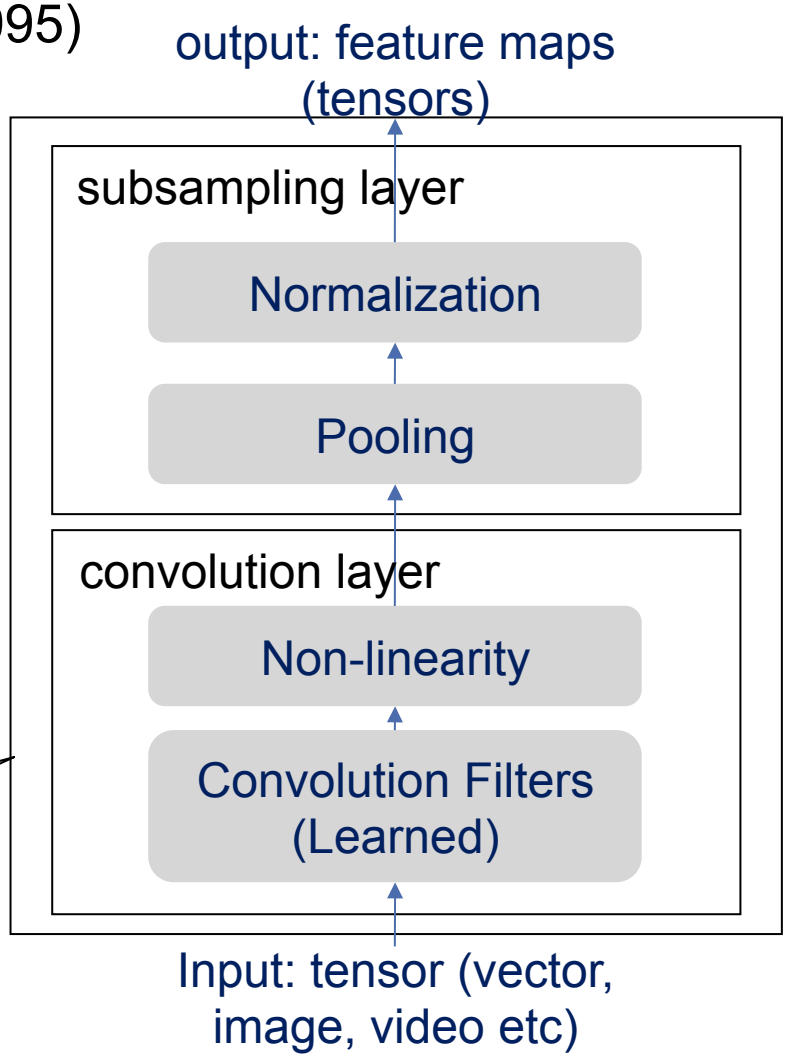
- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks
  - Similarities to Neo-Cognitron

Common Characteristics

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

Convolutional Neural Networks

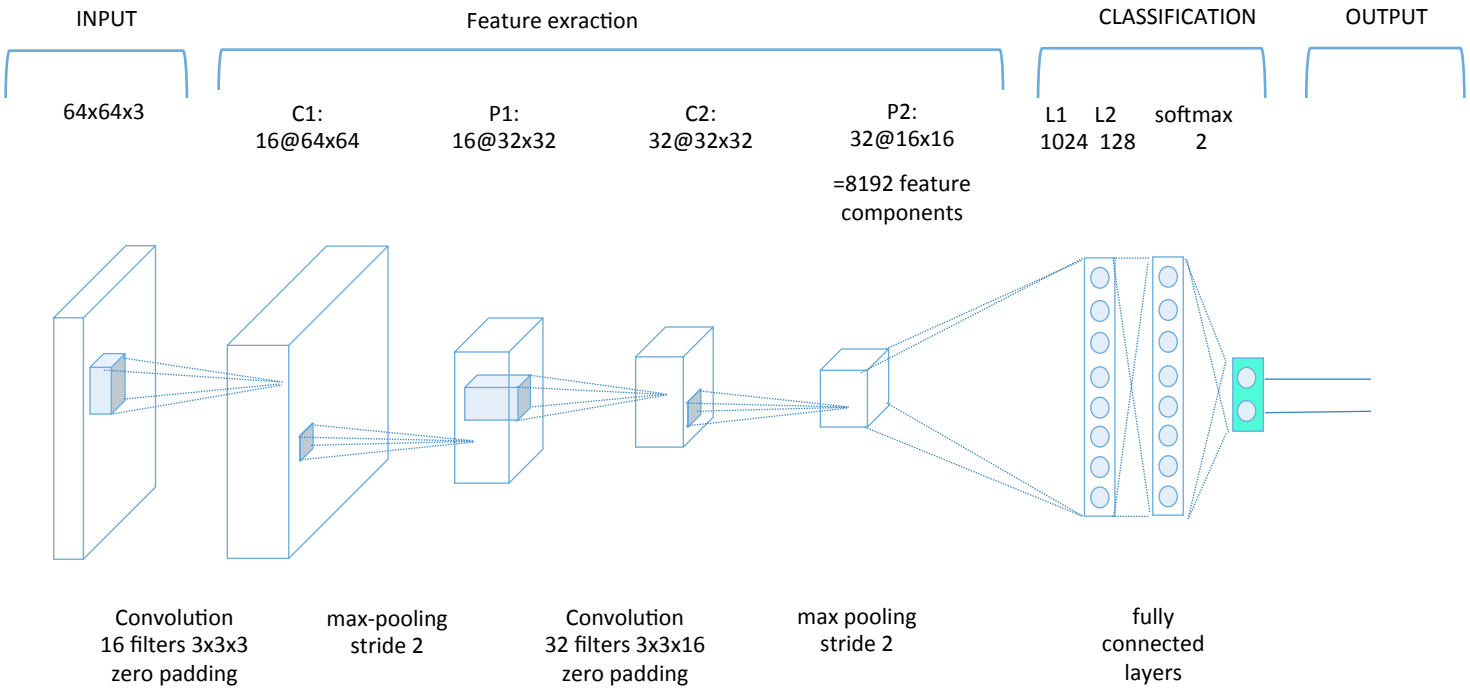
- ConvNet (Y. LeCun and Y. Bengio 1995)
- Neural network with specialized connectivity structure
- Feed-forward:
  - Convolve input (apply filter)
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Train convolutional filters by back-propagating error



in complex layer terminology all these layers is called a single convolutional layer

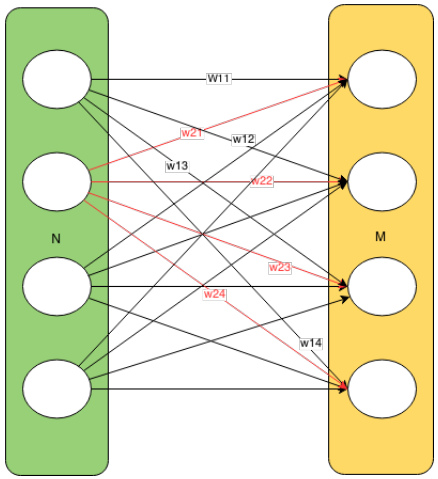
Convolutional Neural Networks (CNN)

Convolution and pooling layers may be repeated many times
(pooling may be omitted for some layers,
for classificitation softmax may be used at the last layer)
EXAMPLE:

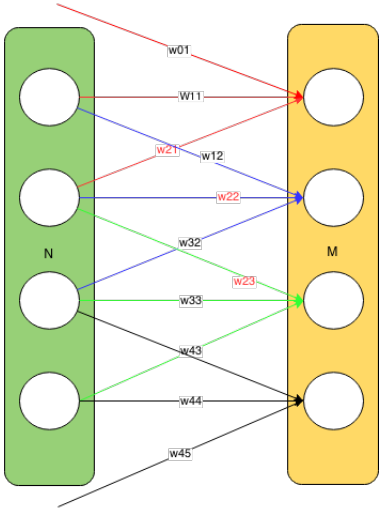


Convolutional Neural Networks

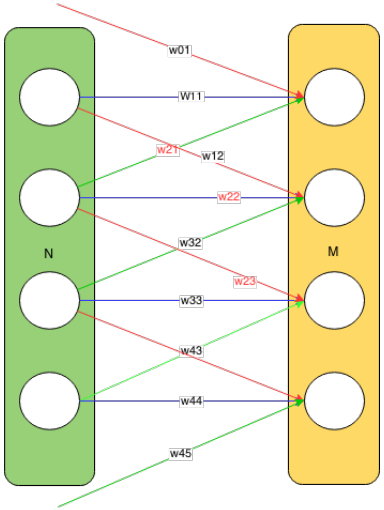
Connectivity & weight sharing depends on layer



fully connected:
all different weights



locally connected:
local connections
with different weight



locally connected & shared
weights: weights having the
same color have same value

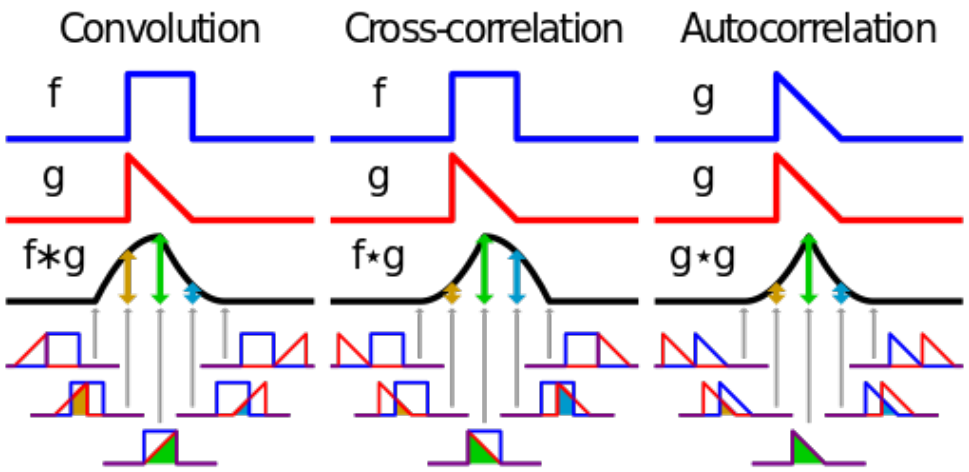
Convolution layer uses filters (i.e: locally connected and shared weight structure) resulting in a much smaller number of parameters (i.e weight values) due to local connection and weight sharing in filters

Locally connected shared weights results in **translation equivariance**,
ie if you shift input m units, output shifts accordingly

Convolutional Neural Networks

Convolution (1D):

$$s(t) = (x * w)(t)$$



continious:

$$s(t) = \int x(a)w(t-a)da$$

discrete:

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Convolutional Neural Networks

Convolution layer: filters

convolution (2D):

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n]$$

due to commutativity of convolution it is equivalently:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n]$$

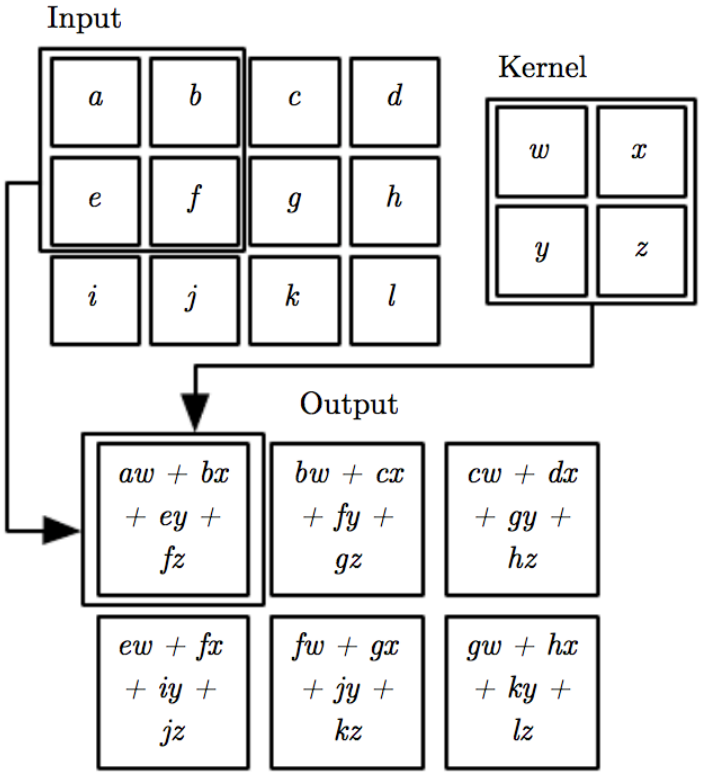
cross correlation (i.e. without kernel flipping)

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n]$$

Convolutional Neural Networks

Convolution layer: filters

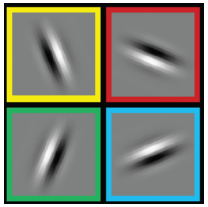
- An example of 2-D convolution without kernel-flipping.
- In this case we restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts.



Convolutional Neural Networks

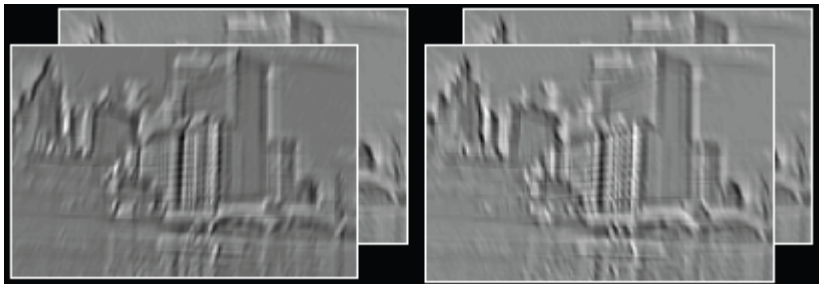
Convolution layer: filters

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

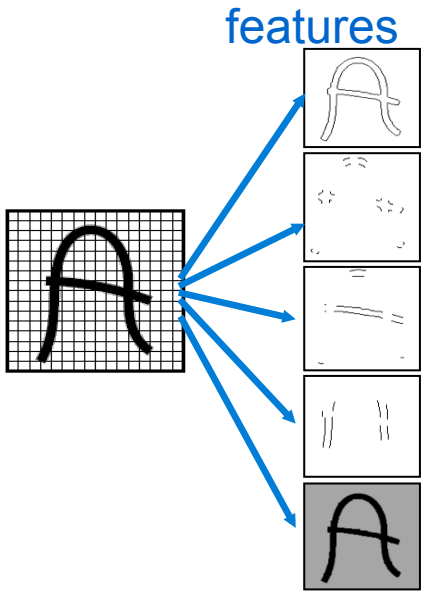


Filter
(kernel)

Input

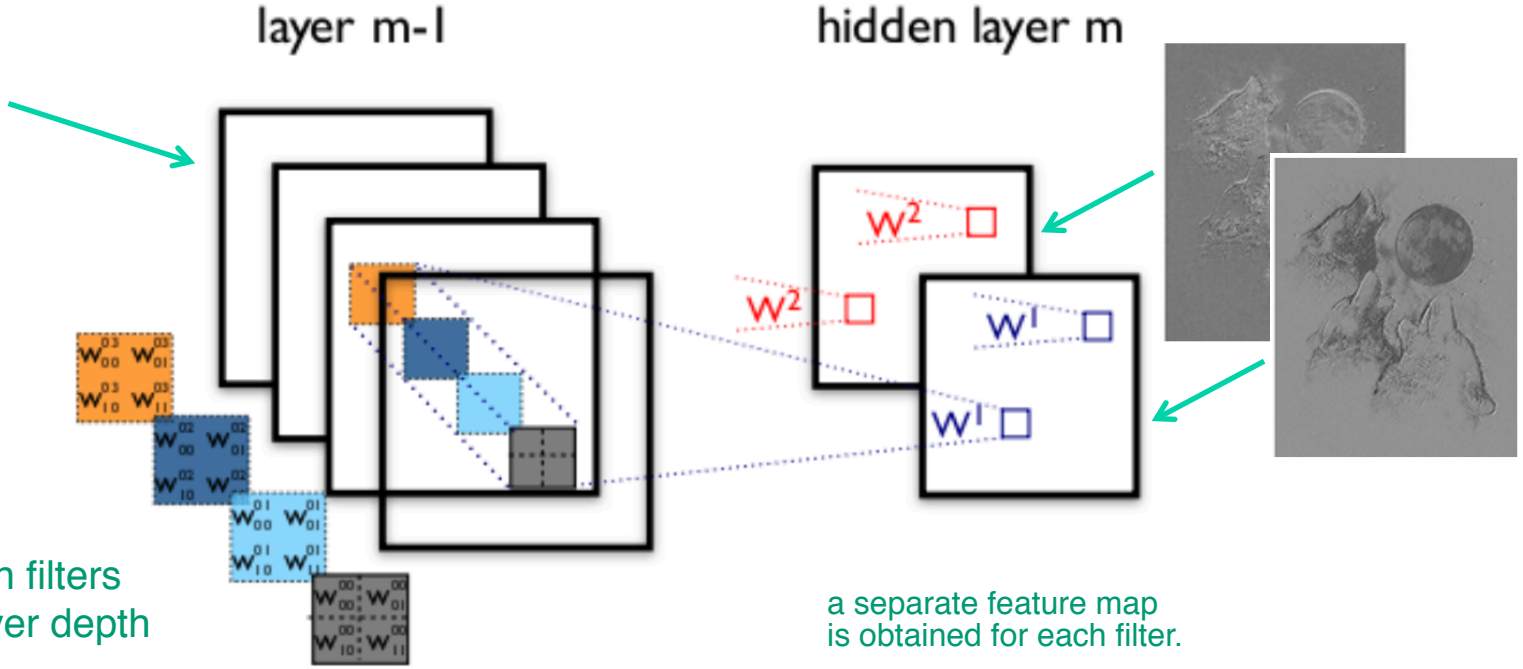


Feature map



Convolutional Neural Networks

CMYK image has 4 layers, so depth=4



convolution filters should cover depth 2x2x4

depth

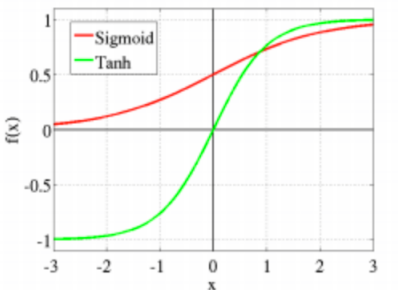
here convolutution is applied in 2D

a separate feature map is obtained for each filter.

Then in the next layer, i.e. m+1, depth is adjusted considering the number of feature maps in the layer m

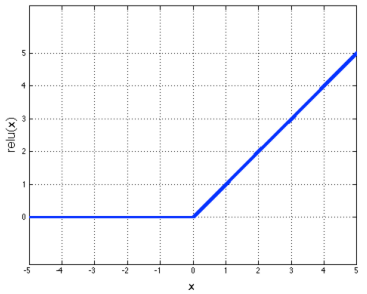
Convolutional Neural Networks

Nonlinearity:



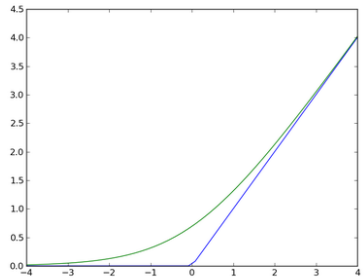
Sigmoid: f(x)=1/(1+exp(-x))

Tanh: f(x)=2/(1+exp(-x))-1=



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

- Simplifies backprop
- Makes learning faster
- Preferred option



Softplus: $f(x)=\ln(1+e^x)$

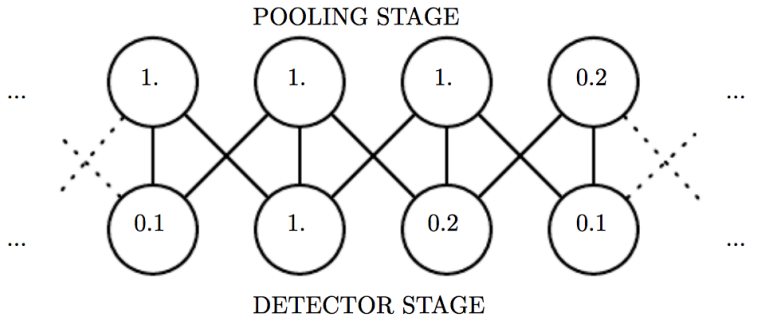
- approximates ReLU,
- differentiable

Convolutional Neural Networks

filter + nonlinearity = detector stage

Pooling:

- max pooling
- average pooling

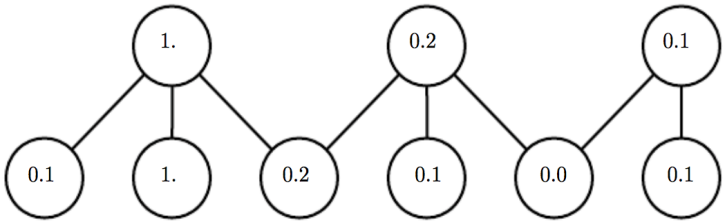


pool width=3, stride=1

- pooling wih down sampling

stride=k, k>1

reduces representation size by a factor k



pool width=3, stride=2

Convolutional Neural Networks

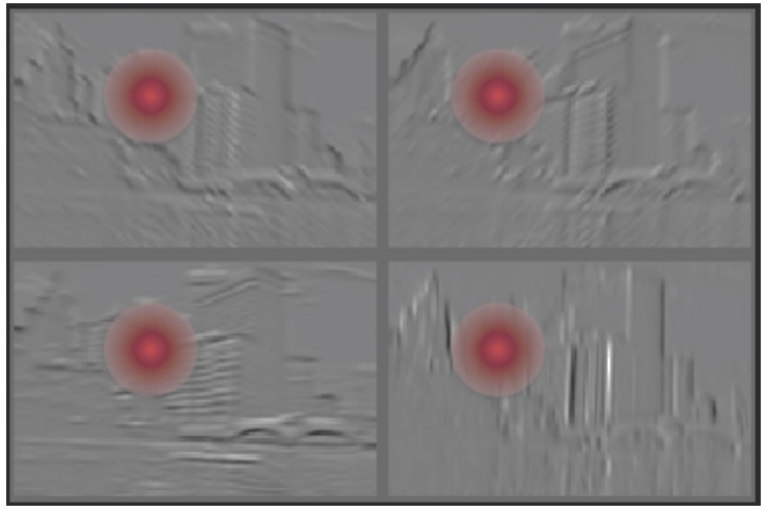
Normalization (optional)

It is used in earlier CNNs,

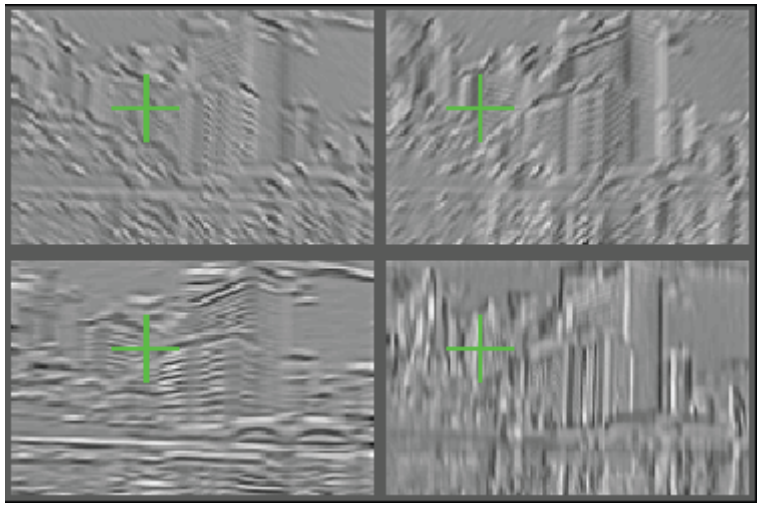
however in recent CNNs normalization is omitted

Contrast normalization (between/across feature map)

- Equalizes the features map

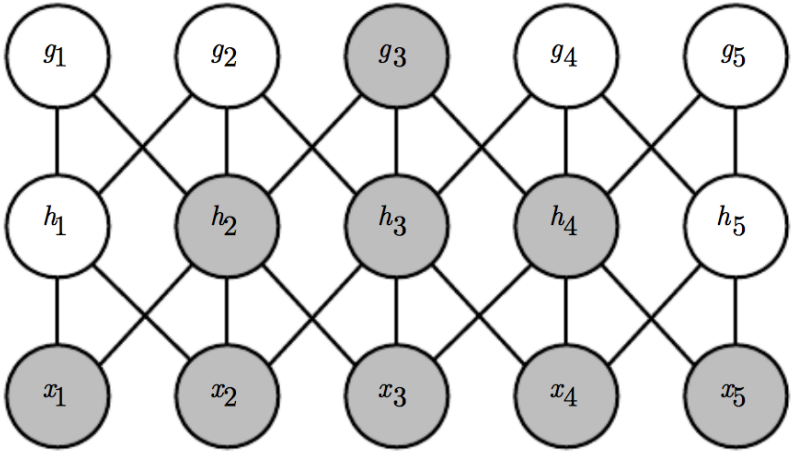


Feature maps
before contrast normalization



Feature maps
after contrast normalization

Convolutional Neural Networks



The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of he unit in the shallow layers.

Convolutional Neural Networks

weight update for shared weights:

Let $w_{(i+k,j+k)}$, k=0,1, M, be shared weights,

then to constrain

$w_{(i+k1,j+k1)}=w_{(i+k2,j+k2)}$, $k_1,k_2=0,\ldots,M-1$,

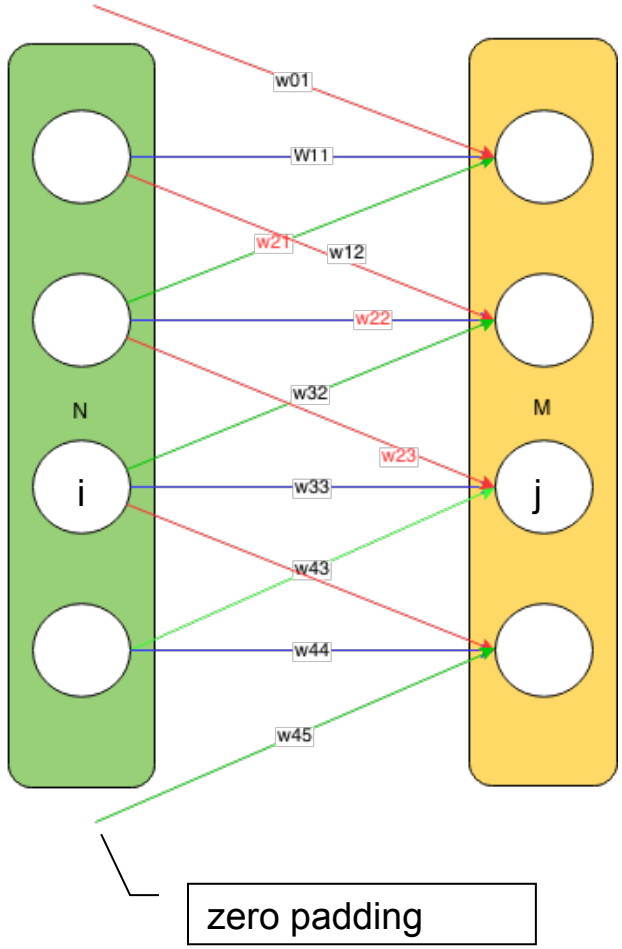
we need

$\Delta w_{(i+k1,j+k1)}=\Delta w_{(i+k2,j+k2)}$ in training,

for this purpose use

$$\Delta w_{(i+k1,j+k1)}=\Delta w_{(i+k2,j+k2)} = \frac{1}{m}\sum_k \frac{\partial e}{\partial w_{(i+k,j+k)}}$$

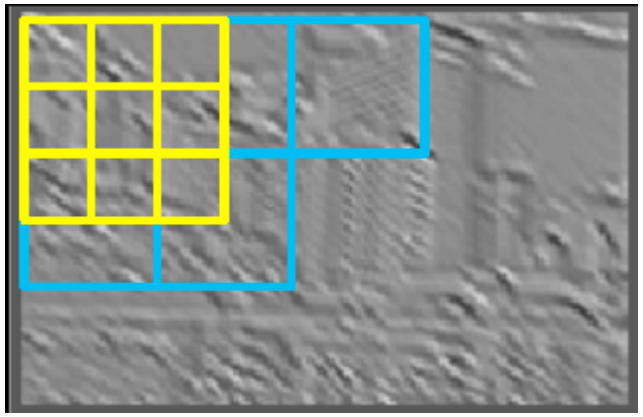
(above formulation is for 1D filters with stride s=1,
the indices should be adjusted accordingly
for higher dimentional filters such as 2D, 3D filters
or for strides s>1)

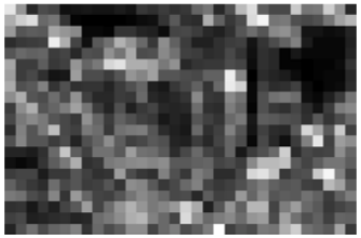


Convolutional Neural Networks

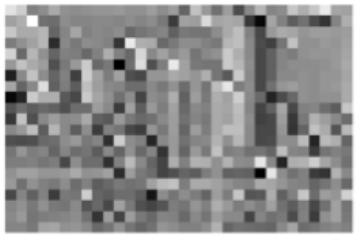
Sub-sampling layer

Spatial Pooling: Average or Max





Max pooling



Average pooling

Role of Pooling

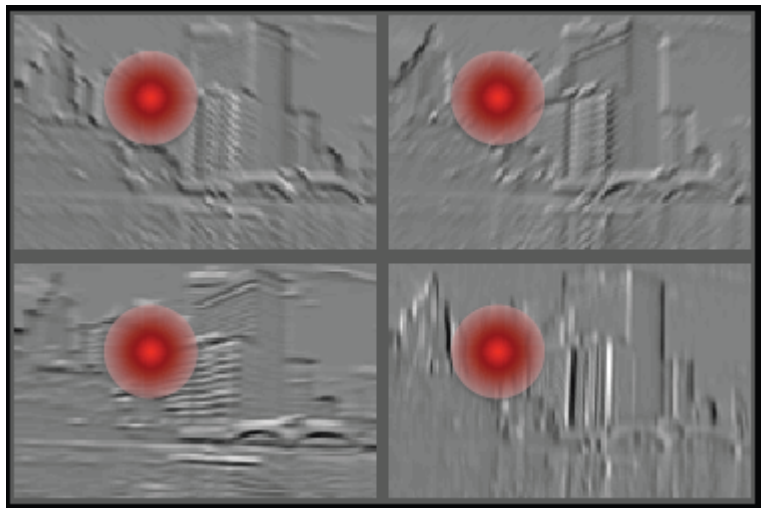
- Invariance to small transformations
- reduce the effect of noises and shift or distortion

Convolutional Neural Networks

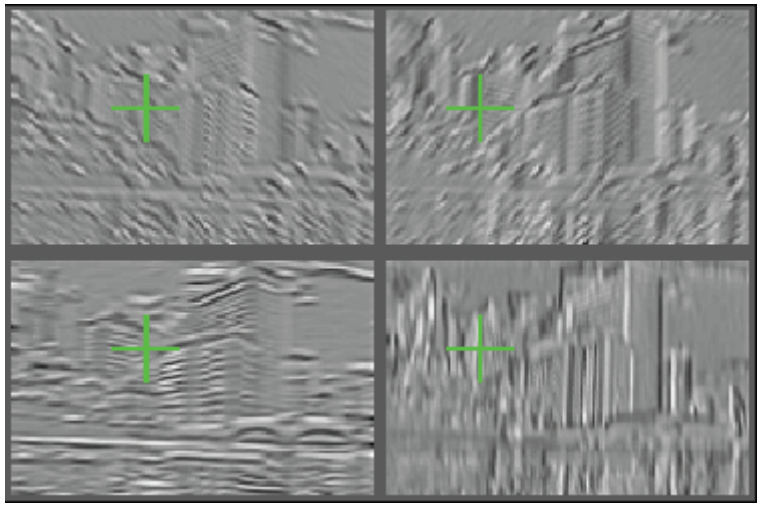
Normalization (optional)

Contrast normalization (between/across feature map)

- Equalizes the features map



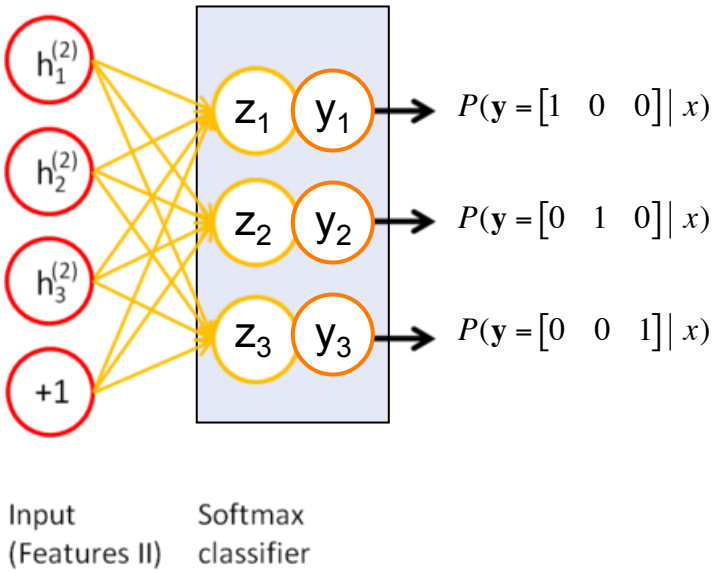
Feature maps
before contrast normalization



Feature maps
after contrast normalization

Convolutional Neural Networks

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



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

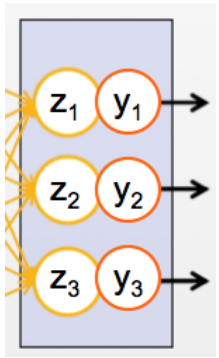
$$\frac{\partial y_i}{\partial z_i} = y_i\,(1 - y_i)$$

Convolutional Neural Networks

To use softmax in Backpropagation learning

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

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



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

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

target value

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

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

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

Convolutional Neural Networks

Remarks:

- CNN may be used either for classfication or regrression similar to MLP. For classificitation, using softmax at the last layer increases the performance
- Instead of using a convolutional layer with large sized filters, use many layer with small filters (for example instead of a single convolutional layer having 7x7xD, if 4 convolutional layers with 3x3xD are used, then the number of parameters to be adjusted will be less although the receptive fields of the last layer is still 7x7
- Zero padding may be used in order to feature map to have the same size as the previous layer after filtering (i.e. at image borders, add imaginarily rows and colums having value 0, before applying the filters)

Convolutional Neural Networks

Remarks:

- Due to memory limitations, instead the whole traing set, use mini-batch selected randomly from training set, then change the minibatch used after updating the weights
- Weight initialization is important (Xavier initialiser may be used)

  Glorot, X., Bengio, Y.: 'Understanding the difficulty of training deep feedforward neural networks', Aistats, 2010, 9, pp. 249–256

- Instead classical gradient descent, Adam optimizer may be used

  Kingma, D., Ba, J.: 'Adam: a method for stochastic optimization'. Available at https://arxiv.org/abs/1412.6980, accessed 29 January 2017

Convolutional Neural Networks

Remarks:

- Over-fit occurs if dataset is not large enough !

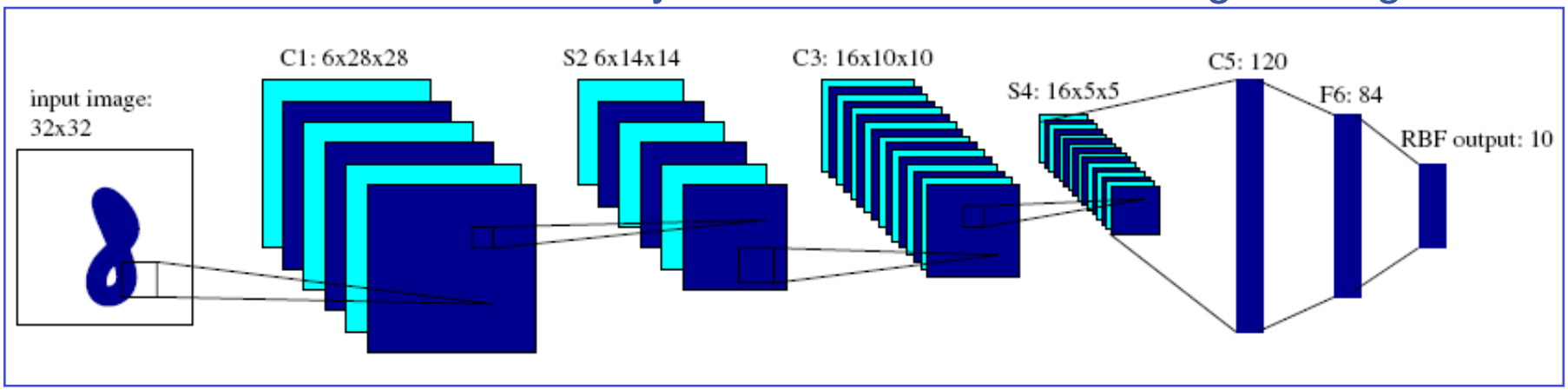
In order to increase **generalisation ability** you may reduce the depth of the network in order to decrease the number of parameters, but there are some other solutions:

- **Data augmentation:** in order to increase training set, produce artificial samples (for example: use also rotated versions, or mirror symetries of the samples at hand)
- **Drop-out:** At each iteration, arbitrarily select some neurons and set their outputs to 0,
- **Transfer learning:** use a CNN trained on a large dataset as the initial weights (omit the fully connected part while transfering weights)

(for example use a CNN trained on ImageNet dataset that contains roughly 1.2 million images of 1000 different classes including those of animals, plants, foods, and instruments)

Convolutional Neural Networks

L5Net : Convolution Network by LeCun for Handwritten Digit Recognition



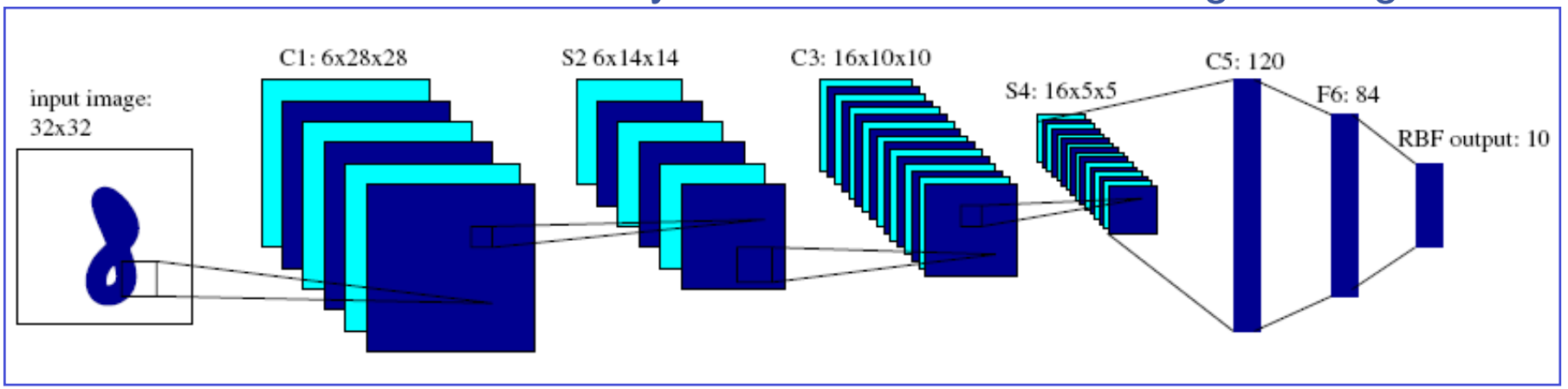
- C1,C3,C5 : Convolutional layer. (5 × 5 Convolution matrix.)
- S2 , S4 : Subsampling layer. (by factor 2)
- F6 : Fully connected layer.

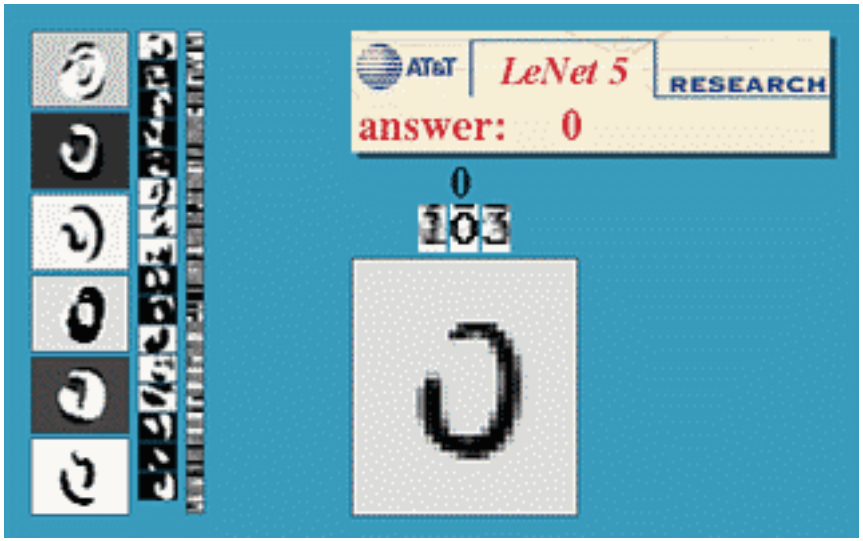
About 187,000 connection.

About 14,000 trainable weight.

Convolutional Neural Networks

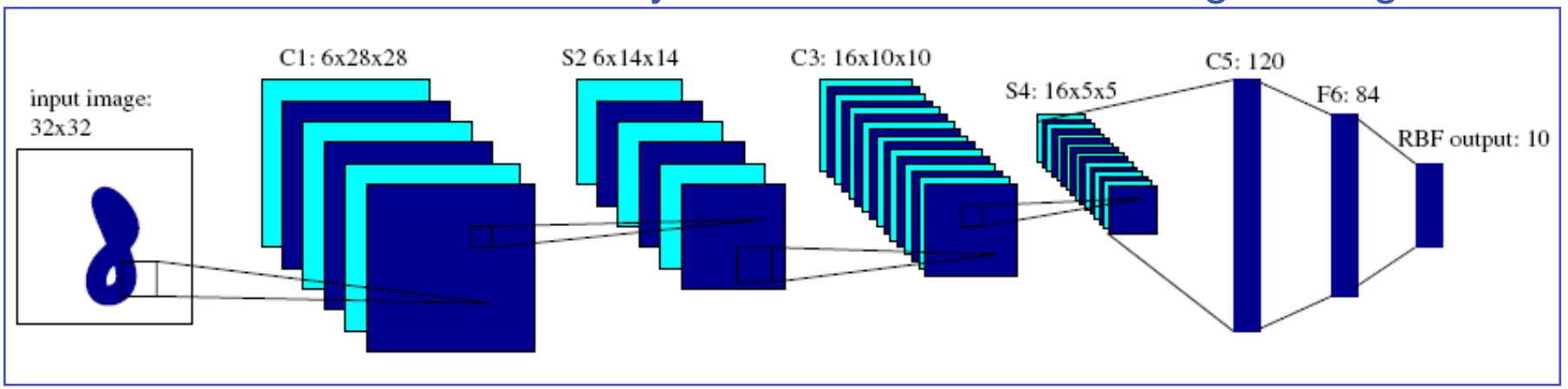
L5Net : Convolution Network by LeCun for Handwritten Digit Recognition

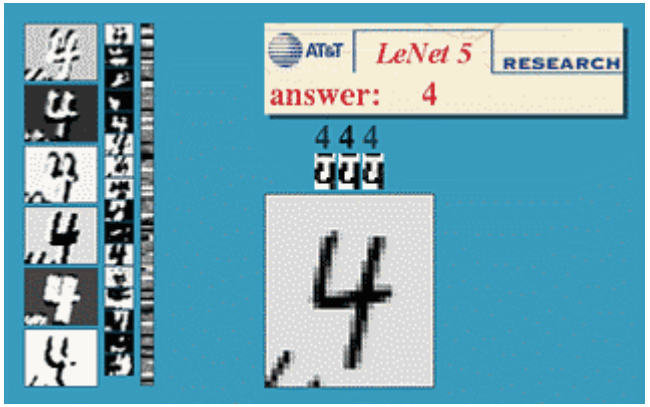


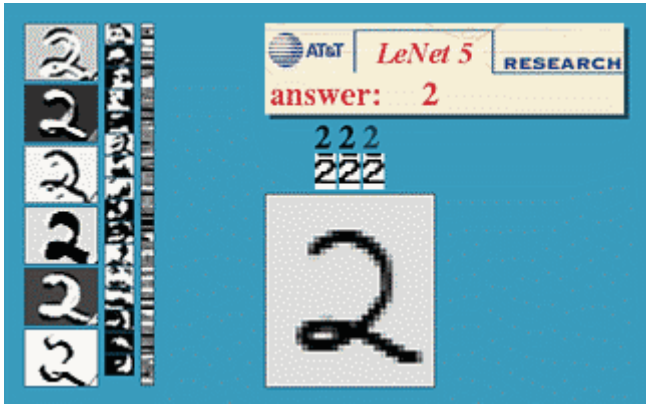


Convolutional Neural Networks

L5Net : Convolution Network by LeCun for Handwritten Digit Recognition



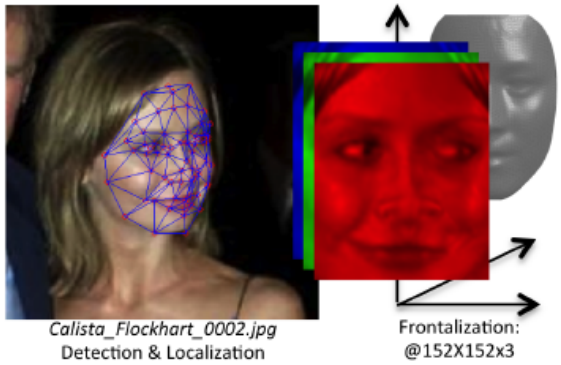




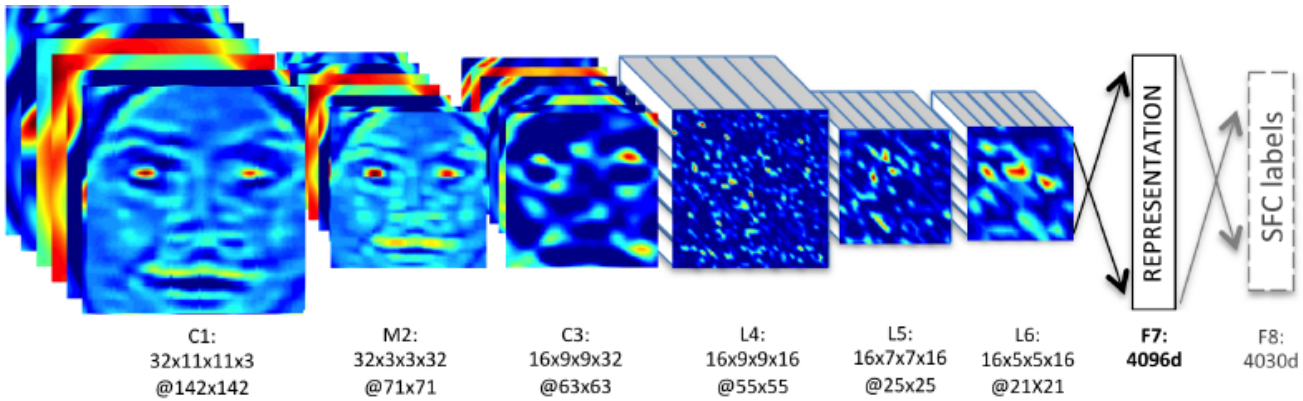
Convolutional Neural Networks

Deep Face: Taigman vd, CVPR 2014

Face Alignment

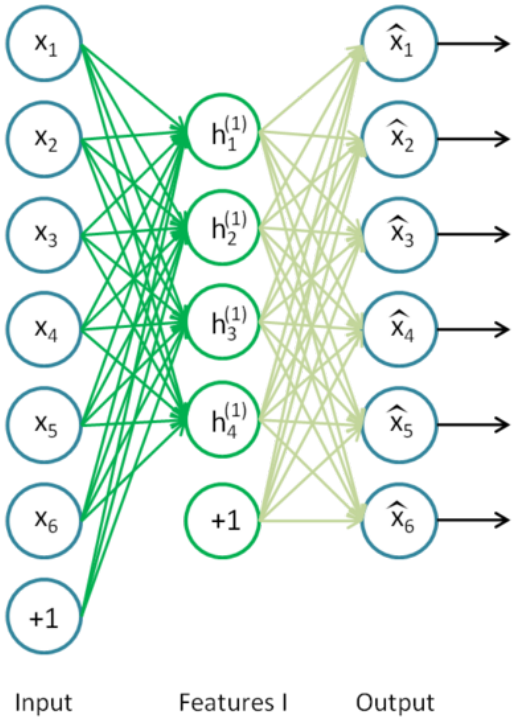


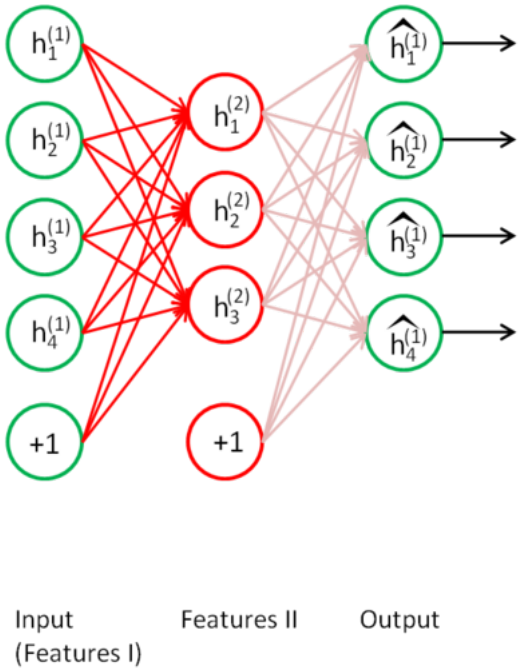
Representation(CNN)



Stacked Auto-Encoders

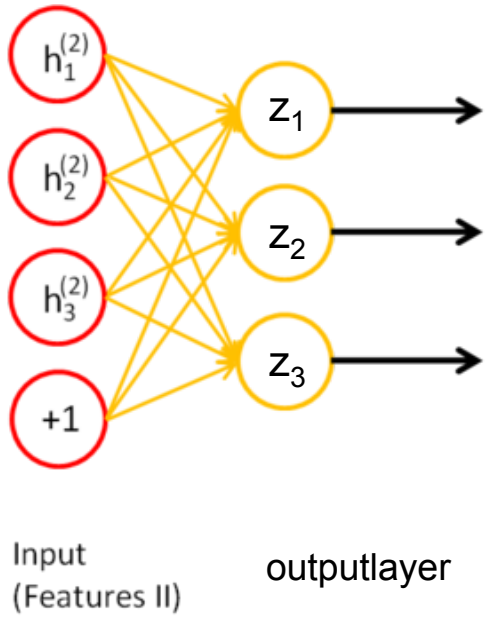
- Bengio (2007) – After Deep Belief Networks (Hinton, 2006)
- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode output layer each time





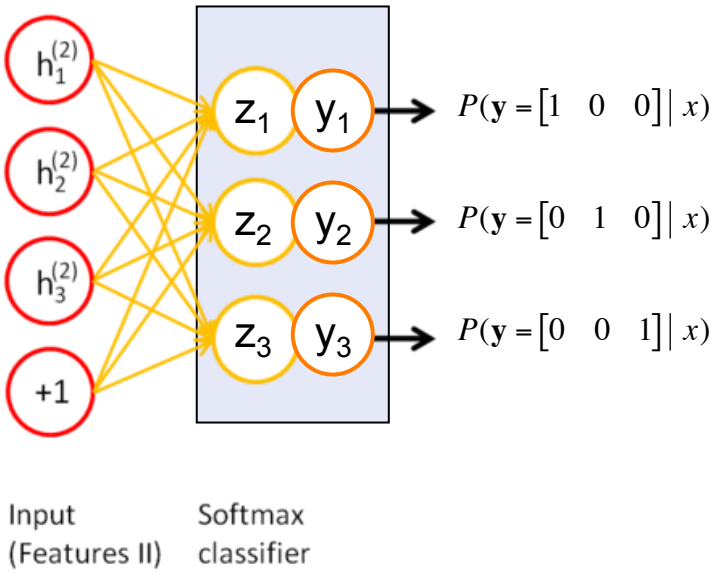
Stacked Auto-Encoders

- Do supervised training on the last layer using final features



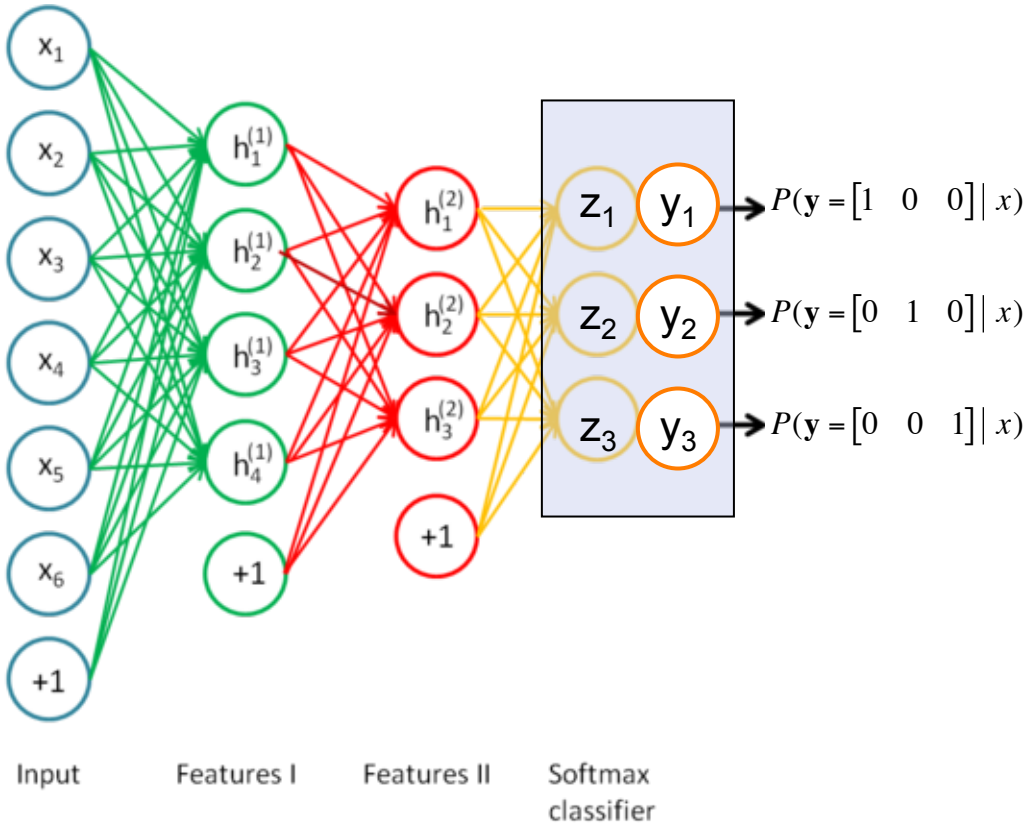
Stacked Auto-Encoders

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



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 to 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