Réseaux de Neurones Profonds, Apprentissage de Représentations

Thierry Artières

ECM, LIF-AMU

December 14, 2017

Outline

History

Key dates

- 1980s : Back-propagation [Rumelhart and Hinton]
- 1990s : Convolutional Networks [LeCun and al.]
- 1990s: Long Short Term Memory networks [Hochreiter and Schmidhuber]
- 2006 : Paper on Deep Learning in Nature [Hinton and al.]
- 2012 : Imagenet Challenge Win [Krizhevsky, Sutskever, and Hinton]
- 2013 : First edition of ICLR
- 2013 : Memory networks [Weston and al.]
- 2014 : Adversarial Networks [Goodfelow and al.]
- 2014 : Google Net [Szegedy and al.]
- 2015 : Residual Networks [He et al.]

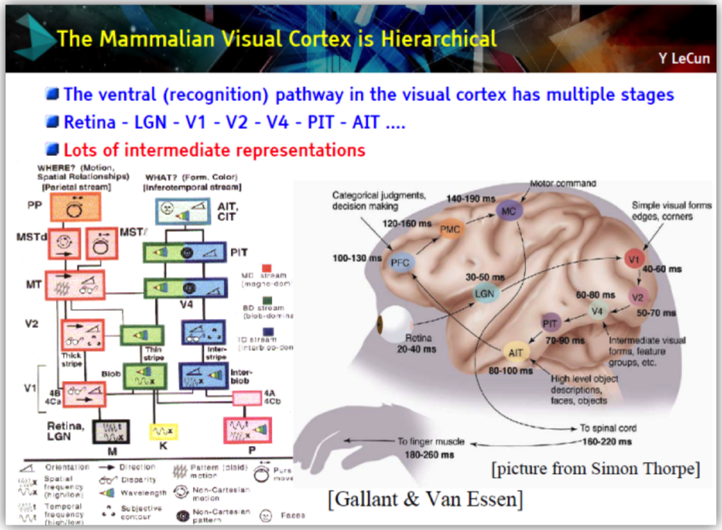
Deep Learning today

Spectaculary breakthroughs - fast industrial transfer

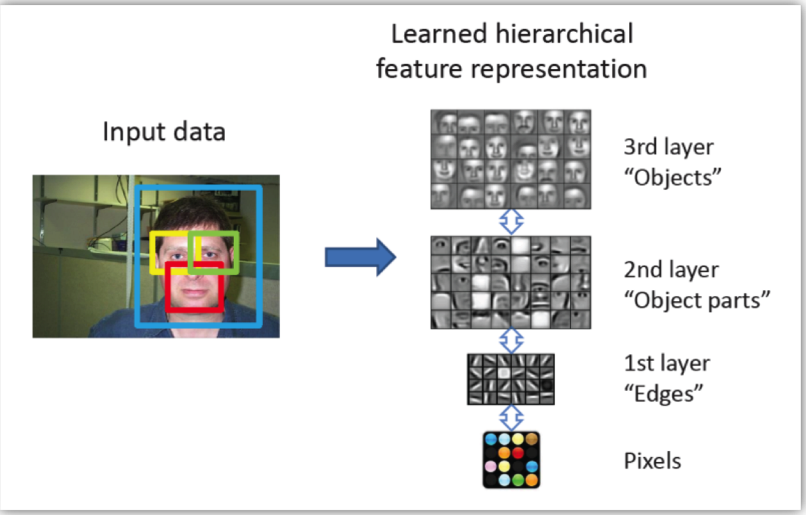
- Images, Videos, Audio, Speech, Texts
- Successful setting
    - Structured data (temporal, spatial...)
    - Huge volumes of datas
    - Huge models (millions of parameters)

	VGGNet	DeepVideo	GNMT
Used For	Identifying Image Category	Identifying Video Category	Translation
Input	Image	Video	English Text
Output	1000 Categories	47 Categories	French Text
Parameters	140M	~100M	380M
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words
Dataset	ILSVRC-2012	Sports-1M	WMT'14

The Graal


The Mammalian Visual Cortex is Hierarchical
Y LeCun
The ventral (recognition) pathway in the visual cortex has multiple stages
Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
Lots of intermediate representations
WHERE? (Motion, Spatial Relationships) [Parietal stream]
WHAT? (Form, Color) [Inferotemporal stream]
PP
MSTd
MSTl
MT
V2
V1
AIT, CIT
PIT
V4
Thick stripe
Thin stripe
Inter-stripe
Blob
Inter-blob
4B
4Ca
4A
4Cb
Retina, LGN
M
K
P
Orientation
Direction
Pattern (plaid) motion
Spatial frequency (high/low)
Disparity
Wavelength
Non-Cartesian motion
Temporal frequency (high/low)
Subjective contour
Non-Cartesian pattern
Faces
Categorical judgments, decision making
Motor command
Simple visual forms edges, corners
140-190 ms
120-160 ms
100-130 ms
PFC
PMC
MC
V1
40-60 ms
30-50 ms
LGN
60-80 ms
V2
V4
50-70 ms
PIT
Retina 20-40 ms
70-90 ms
AIT
80-100 ms
Intermediate visual forms, feature groups, etc.
High level object descriptions, faces, objects
To spinal cord
160-220 ms
To finger muscle
180-260 ms
[picture from Simon Thorpe]
[Gallant & Van Essen]

The key: features

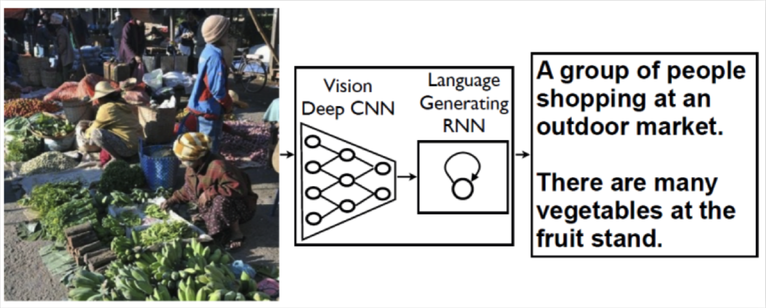


[Honglak et al.]

Annotating real visual scenes

[Farabetr et al., 2012]

Automatic captioning



[Honglak et al., 2014]

Outline

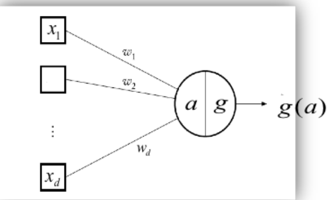
A single Neuron

One Neuron

- Elementary computation

$$\text{activation} = w^T.x = \sum_j w_j x_j + w_0$$

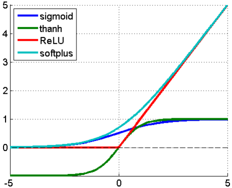
$$\text{output} = g(a(x))$$



Non linearity : g

- Sigmoide, Hyperbolic tangent, Gaussian
- Rectified Linear Unit (RELU)

$$\begin{aligned} f(x) &= 0 \text{ if } x \leq 0 \\ &= x \text{ otherwise} \end{aligned}$$



Multi Layer Perceptron (MLP)

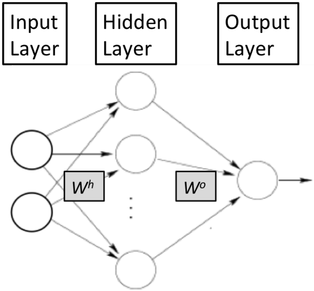
Structure

- Organization in successive layers
  - Input layer
  - Hidden layers
  - Output layer

Function implemented by a MLP

$$g(W^o.g(W^h x))$$

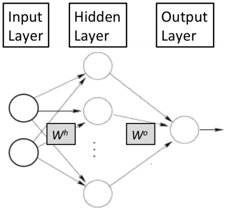
- Inference: Forward propagation from input to output layer



MLP : Forward propagation

Forward propagation of activities, for an input example x

- Fill the input layer with x: $h_0 = x$
- Iterate from the first hidden layer to the last one
  - $h^l = W^l \times h^{l-1}$
  - $h^l = g(h^l)$



MLP Usage for Regression

Notation

- y_{ij} : ideal output of the j^{th} neuron of the output layer when input is example number i
- o_{ij} : real output of the j^{th} neuron of the output layer when input is example number i
- N : number of samples
- O number of outputs of the model = size of the output layer

Training

- Criterion:
  - Mean Squared Error $\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{O}\|y_{ij}-o_{ij}\|^2$

Inference

- Forward propagation from the input layer to the output layer
- Output: $(o_{ij})_{j=1..O}$

MLP Usage for Classification

Training

- One-hot encoding of outputs: As many outputs as there are classes
- MSE criterion as for Regression problems
- Cross Entropy criterion
  - transformation of outputs s_{ij} in a probability distribution
    - Softmax : $p_{ij} = \frac{exp^{-o_{ij}}}{\sum_{k=1}^{O} exp^{-o_{ik}}}$
    - New ouputs of the model : p_{ij} = output of the j^{th} neuron of the output layer when input is example number i
  - Criterion:
    - Cross-entropy $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{O} y_{ij} log(p_{ij})$

Training

- Forward propagation from the input layer to the output layer
- Decision based on the maximum value amongst output cells $c = argmax_{j=1..O} p_{ij}$

Learning a MLP

Learning as an optimization problem

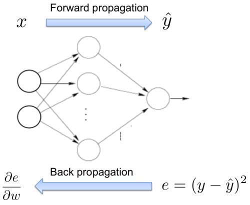
- Objective function of parameters set w for a given training set T

$$C(w) = F(w) + R(w) = \sum_{(x,y)\in T} L_w(x,y,w) + ||w||^2$$

- Gradient descent optimization: $w = w - \epsilon \frac{\partial C(w)}{\partial w}$

Backpropagation

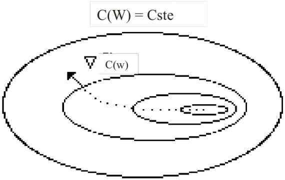
- Use chain rule for computing derivative of the loss with respect to all weights in the NN



Gradient Descent Optimization

Gradient Descent Optimization

- Initialize Weights (Randomly)
- Iterate (till convergence)
  - Restimate $\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}}|_{\mathbf{w}_t}$



Gradient Descent: Tuning the Learning rate

Two classes Classification problem

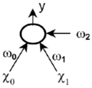
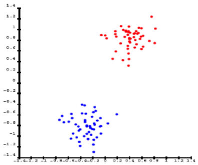


Fig. 9. Simple linear network.



Weight trajectory for two different gradient step settings.

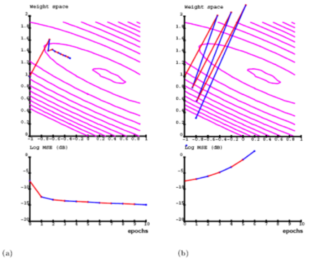


Fig. 11. Weight trajectory and error curve during learning for (a) $\eta = 1.5$ and (b) $\eta = 2.5$.

Images from [LeCun et al.]

Gradient Descent: Tuning the Learning rate

Effect of learning rate setting

- Assuming the gradient direction is good, there is an optima value fir the learning rate
- Using a smaller value slows the convergence and may prevent from converging
- Using a bigger value makes convergence chaotic and may cause divergence

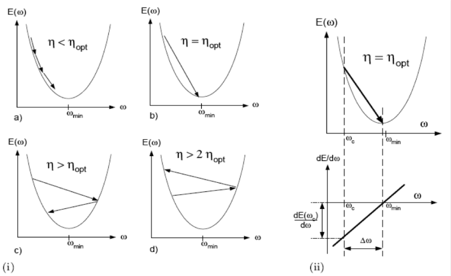


Fig. 6. Gradient descent for different learning rates.

Images from [LeCun et al.]

Gradient Descent: Stochastic, Batch and mini batchs

Objective : Minimize $C(\mathbf{w}) = \sum_{i=1..N} L_w(i)$ with $L_w(i) = L_w(x^i, y^i, w)$

Batch vs Stochastic vs Minibatchs

- Batch gradient descent
  - Use $\nabla C(\mathbf{w})$
  - Every iteration all samples are used to compute the gradient direction and amplitude
- Stochastic gradient
  - Use $\nabla L_w(i)$
  - Every iteration one sample (randomly chosen) is used to compute the gradient direction and amplitude
  - Introduce randomization in the process.
  - Minimize $C(w)$ by minimizing parts of it sucessively
  - Allows faster convergence, avoiding local minima etc
- Minibatch
  - Use $\nabla \sum_{\text{few } j} L_w(j)$
  - Every iteration a batch of samples (randomly chosen) is used to compute the gradient direction and amplitude
  - Introduce randomization in the process.
  - Optimize the GPU computation ability

Gradient Computation: Chain rule

Gradient of a function

$$z = 2 \times f(x + 3 \times y) + 6 \times g(5 \times x) \times h(y)$$

$$\Rightarrow \frac{\partial z}{\partial x}|_{x,y} = 2 \times f'(x + 3 \times y) + 30 \times g'(5 \times x) \times h(y)$$

Equivalent computation with the Chain rule

Set $a(x) = f(x + 3 \times y)$ and $b(x, y) = g(5 \times x)$

$\Rightarrow z = 2 \times a(x) + 6 \times b(x) \times h(y)$

$$\Rightarrow \frac{\partial z}{\partial x}|_{x,y} = \frac{\partial z}{\partial a}|_{x,y} \times \frac{\partial a}{\partial x}|_{x,y} + \frac{\partial z}{\partial b}|_{x,y} \times \frac{\partial b}{\partial x}|_{x,y}$$

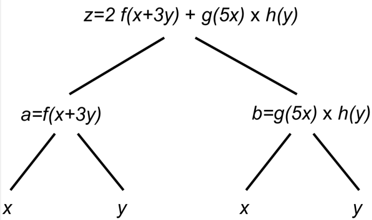
With:

$$\frac{\partial y}{\partial a}|_{x,y} = 2 \text{ and } \frac{\partial a}{\partial x}|_{x,y} = f'(a \times x + 3 \times y)$$

$$\frac{\partial y}{\partial b}|_{x,y} = 6 \times h(y) \text{ and } \frac{\partial b}{\partial x}|_{x,y} = 5 \times g'(5 \times x)$$

$$\frac{\partial a}{\partial x}|_{x,y} = g'(a \times x)$$

$$\frac{\partial b}{\partial x}|_{x,y} = 5 \times g'(5 \times x)$$



Gradient computation in MLPs: Stochastic case

Notations

- Activation function on every layer: g — Number of layer : L
- Activity of neuron i in layer l, a_i^l — Output of neuron i in layer l, $h_i^l = g(a_i^l)$, and $o_i^L = g(a_i^L)$
- Weight from a neuron j of layer $l-1$ to neuron i in layer l : w_{ij}^l
- Example considered for computing gradient (x, y)
- Squarred loss : $C(w) = \|\mathbf{o}^L - \mathbf{y}\|^2$

Gradient wrt. last layer weights

- Gradient wrt cell's ouput $\frac{\partial C(w)}{\partial o_i^L} = 2(o_i^L - y_i)$
- Gradient wrt cell's activity $\delta_i^L = \frac{\partial C(w)}{\partial a_i^L} = \frac{\partial C(w)}{\partial o_i^L}\frac{\partial o_i^L}{\partial a_i^L} = 2(o_i^L - y_i)g'(a_i^L)$
- Gradient wrt weights arriving to output cells

$$\frac{\partial C(w)}{\partial w_{ij}^L} = \frac{\partial C(w)}{\partial a_i^L}\frac{\partial a_i^L}{\partial w_{ij}^L} = \delta_i^L \times h_j^{L-1}$$

Gradient computation in MLPs: Stochastic case (continues)

Gradient wrt. last hidden layer (LHL) weights

- Gradient wrt LHL cell's activity $\delta_j^{L-1} = \frac{\partial C(w)}{\partial a_j^{L-1}} = \sum_i \frac{\partial C(w)}{\partial a_i^L} \frac{\partial a_i^L}{\partial a_j^{L-1}} = \sum_i \delta_i^L w_{ij}^L g'(a_j^{L-1})$
- Gradient wrt weights arriving to a LHL cell

$$\frac{\partial C(w)}{\partial w_{jk}^{L-1}} = \delta_j^{L-1} \times h_k^{L-2}$$

Gradient computation in MLPs

Forward propagation of activities, for an input example x

- Fill the input layer with x: $h^0 = x$
- Iterate from the first hidden layer to the last one
  - $h^l = W^l \times h^{l-1}$
  - $h^l = a(h^l)$

Backward computation of the error

- Compute the output error δ^L
- Iterate from the last hidden layer to the first one
  - Compute δ^L from δ^{L-1}

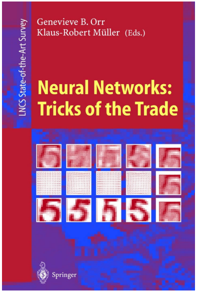
Computing gradient

- For each weight w^l_{jk} of every layer compute the gradient using δ^l_j and o^{l-1}_k

Lots of tricks to favor convergence

And more...

- Weight Initialization
- Gradient step setting
- ...
- ⇒ Despite appearances NN are still not fully usable by non experts



Outline

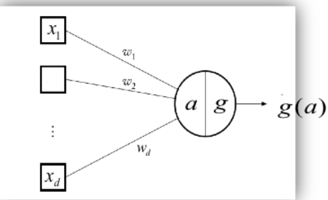
A single Neuron

One Neuron

- Elementary computation

$$\text{activation} = w^T.x = \sum_j w_j x_j + w_0$$

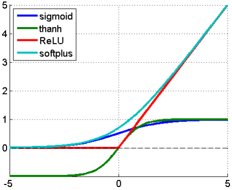
$$\text{output} = g(a(x))$$



Non linearity : g

- Sigmoide, Hyperbolic tangent, Gaussian
- Rectified Linear Unit (RELU)

$$f(x) = 0 \text{ if } x \leq 0$$
$$= x \text{ otherwise}$$



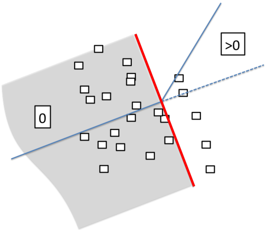
A single Neuron

One Neuron

- Elementary computation

$$\text{activation} = w^T.x = \sum_j w_j x_j + w_0$$

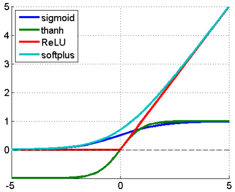
$$\text{output} = g(a(x))$$



Non linearity : g

- Sigmoide, Hyperbolic tangent, Gaussian
- Rectified Linear Unit (RELU)

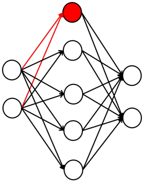
$$f(x) = 0 \text{ if } x \leq 0$$
$$= x \text{ otherwise}$$

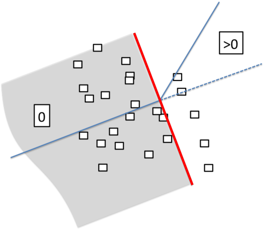


What a MLP may compute

What does a hidden neuron

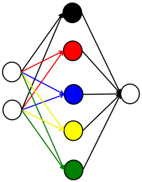
- Divides the input space in two

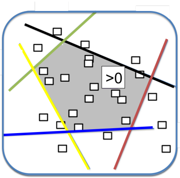




Combining multiple hidden neurons

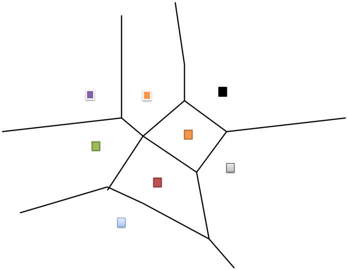
- Allows identifying complex areas of the input space
- New (distributed) representation of the input

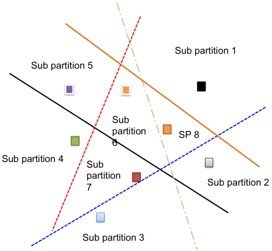




Distributed representations

Might be much more efficient than non distributed ones





MLP = Universal approximators

One layer is enough !

- Theorem [Cybenko 1989]: Let $\phi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\epsilon > 0$, there exists an integer N, such that for any function $f \in C(I_m)$, there exist real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \phi \left(w_i^T x + b_i \right)$$

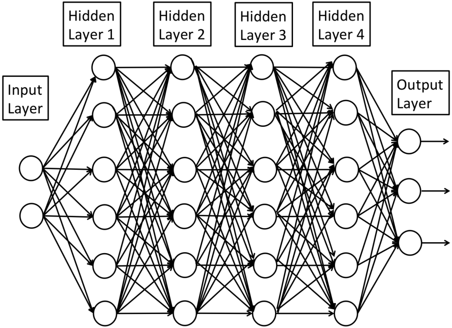
as an approximate realization of the function f where f is independent of ϕ ; that is : $|F(x) - f(x)| < \epsilon$ for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.
- Existence theorem only
- Many reasons for not getting good results in practice

Outline

What are deep models ?

NNs with more than one hidden layer !

A series of hidden layers

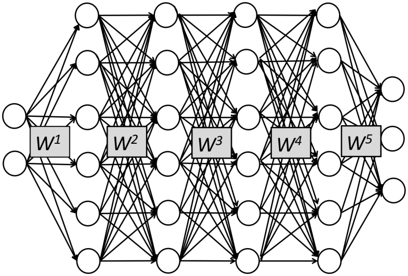


What are deep models ?

NNs with more than one hidden layer !

Computes a complex function of the input

$$y = g(W^k \times g(W^{k-1} \times g(...g(W^1 \times x))))$$

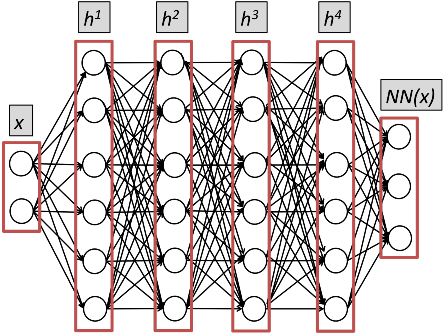


What are deep models ?

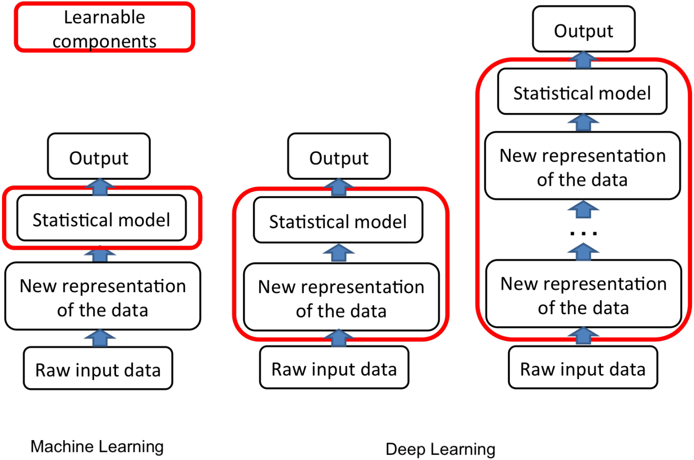
NNs with more than one hidden layer !

Computes new representations of the input

$$h^i(x) = g(W^i \times h^{i-1}(x))$$



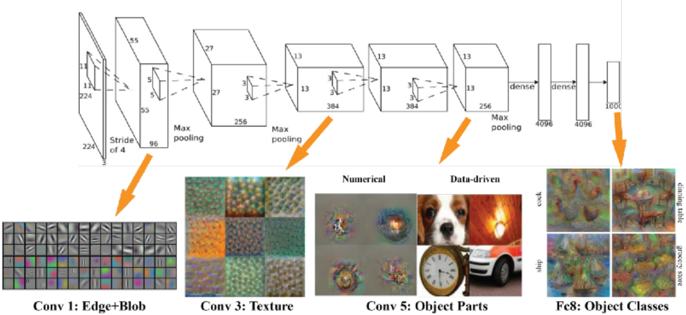
Machine Learning vs. Deep Learning



Feature hierarchy : from low to high level

What feature hierarchy means ?

- Low-level features are shared among categories
- High-level features are more global and more invariant



([Krizhevsky and al., 2012])

Examples of architectures

AlexNet [Krizhevsky and al., 2012] (top) and NetworkInNetwork [Lin and al.,2013] (bottom)