Deep-Learning:
general principles
Convolutional Neural Networks
Deep Belief Networks
Recurrent Neural Networks (RNN)

Pr. Fabien Moutarde
Robotics Lab (CAOR)
MINES ParisTech
PSL Research University

Fabien.Moutarde@mines-paristech.fr
http://people.mines-paristech.fr/fabien.moutarde

Acknowledgements

During preparation of these slides, I got inspiration and borrowed some slide content from several sources, in particular:

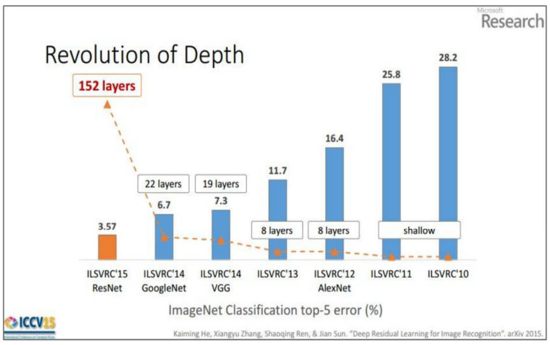
- Yann LeCun + MA Ranzato: slides on « *Deep Learning* » from the corresponding course at NYU
  http://cilvr.cs.nyu.edu/doku.php?id=deeplearning:slides:start
- Hinton+Bengio+LeCun: slides of the *NIPS'2015 tutorial on Deep Learning*
  http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf
- Fei-Fei Li + A.Karpathy + J.Johnson: Stanford course lecture slides on *« Convolutional Neural Networks »*
  http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf
- I. Kokkinos: slides of a CentraleParis course on Deep Belief Networks
  http://cvn.ecp.fr/personnel/iasonas/course/DL5.pdf

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

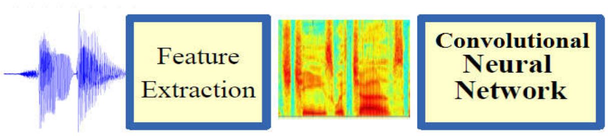
Deep-Learning recent breakthroughs

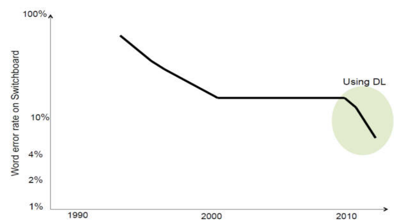
- Very significant improvement over State-of-the-Art in Pattern Recognition / Image Semantic Analysis:



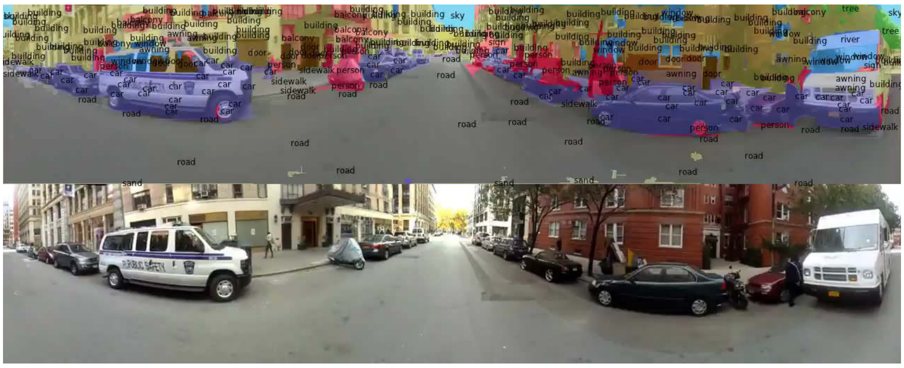
- won many vision pattern recognition competitions (OCR, TSR, object categorization, facial expression,…)
- deployed in photo-tagging by Facebook, Google,Baidu,…

- Similar dramatic progress in Speech recognition + Natural Language Processing (NLP)





Some examples of Deep-Learning important/striking applications



[C. Farabet, C. Couprie, L. Najman & Yann LeCun: Learning Hierarchical Features for Scene Labeling, IEEE Trans. PAMI, Aug.2013.

Video analysis for self-driving cars

Photos search

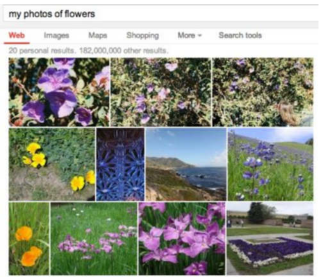
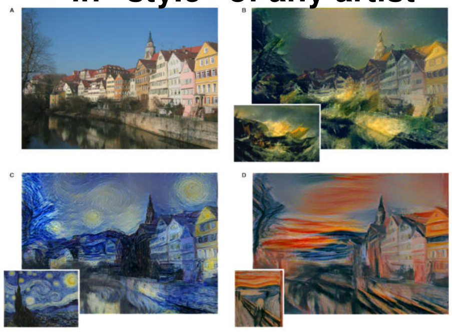


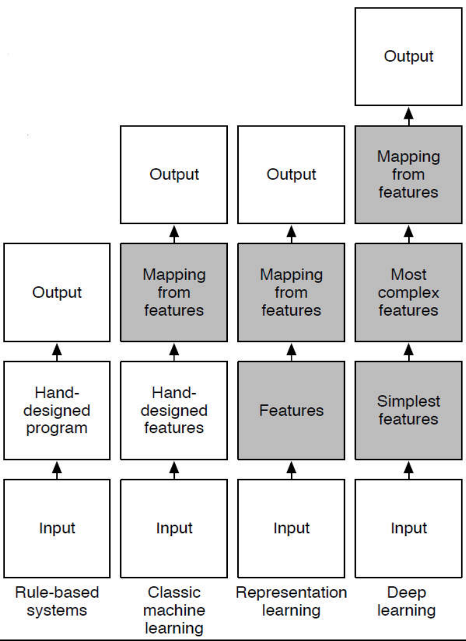
Image-to-text

'Painting' Photos in "style" of any artist



What is Deep-Learning?

Learning a ***hierarchy*** of increasingly abstract ***representations***



[Figure from Goodfellow]

Increasing level of abstraction
Each stage ~ trainable feature transform

Image recognition
Pixel → edge → texton → motif → part → object
Speech
Sample → spectral band → … → phoneme → word
Text
Character → word → word group → clause → sentence → story

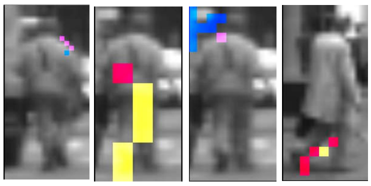
Importance of « features » in classical Machine-Learning

Traditional Machine Learning Flow

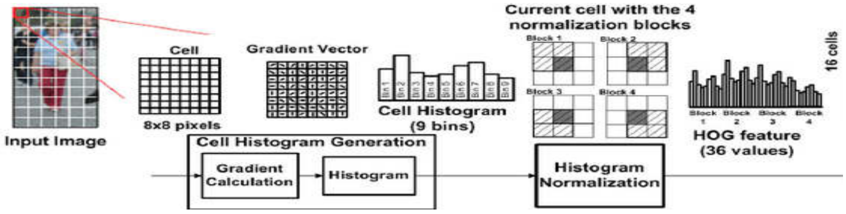
Examples of *hand-crafted* features

Haar features

Control-points features



HoG (Histogram of Gradients)



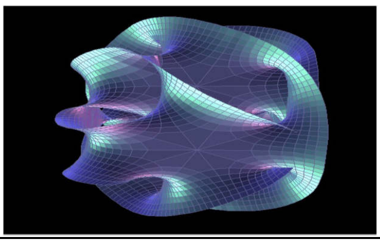
Why features should be learnt?

Real data examples for a given task are usually <u>not spreaded everywhere</u> in input space, but rather clustered on a low-dimension « <u>manifold</u> »

Example: Face images of 1000x1000 pixels
➔ « raw » examples are vectors in $R^{1000000}$!!

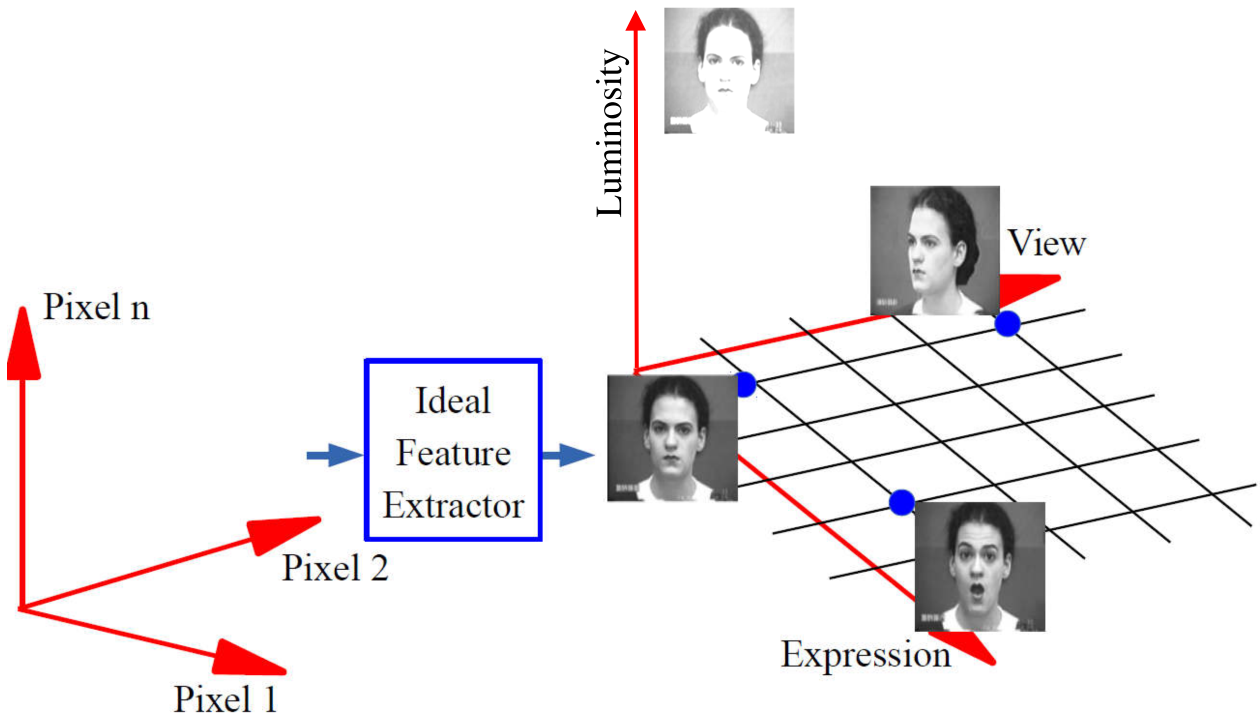
- **<u>BUT:</u>**
  - position = 3 cartesian coord
  - orientation 3 Euler angles
  - 50 muscles in face
  - Luminosity, color

➔ Set of all images of ONE person has ≤ 69 dim



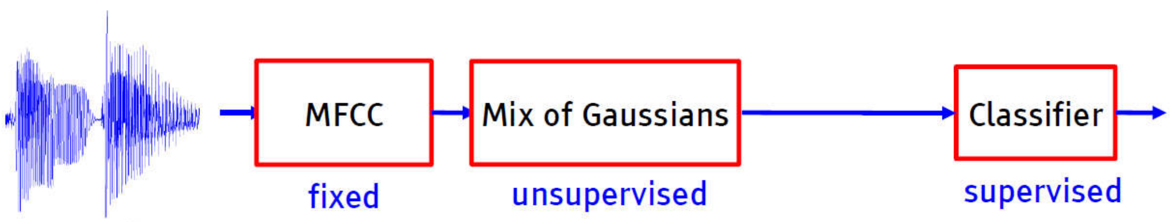
→ Examples of face images *of 1 person* <u>*are all in a LOW-dim manifold inside a HUGE-dim space*</u>

Good features ~ good « mapping » on manifold

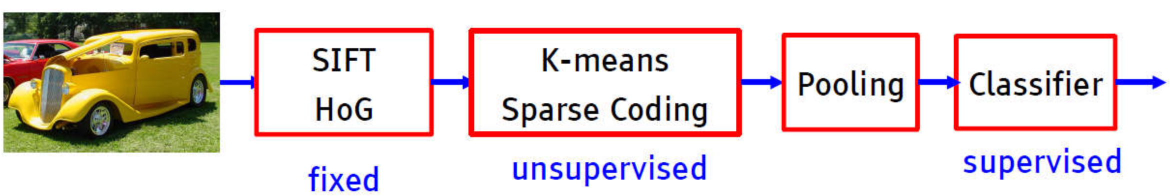


Features learning (before Deep-Learning)

- Speech recognition: early 90's – 2011



- Object Recognition: 2006 - 2012

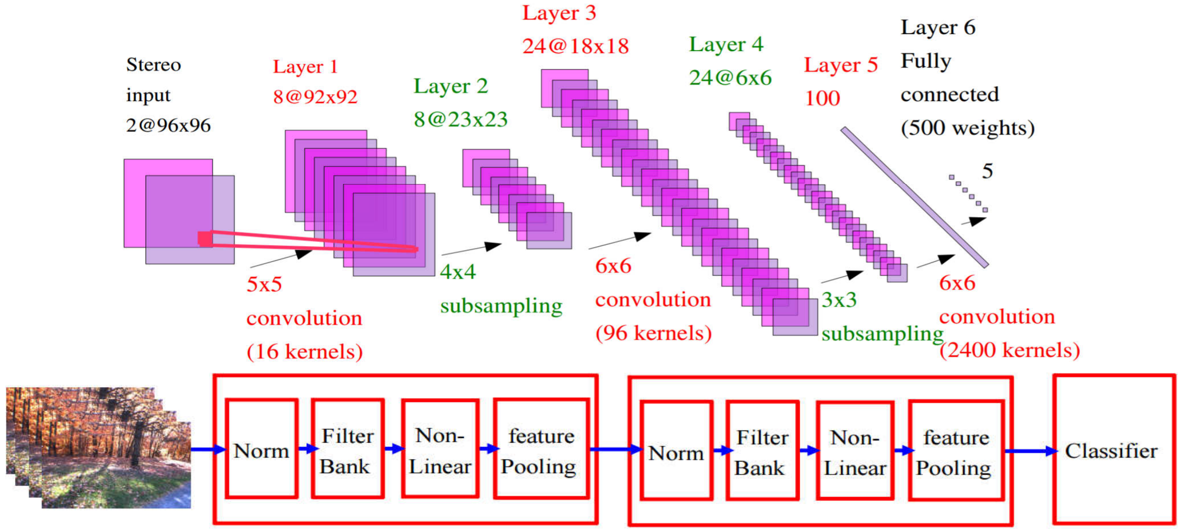


Outline

- Introduction to Deep Learning
- <u>Convolutional Neural Networks (CNN or ConvNets)</u>
  - <u>Intro + Short reminder on Neural Nets</u>
  - Convolution & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

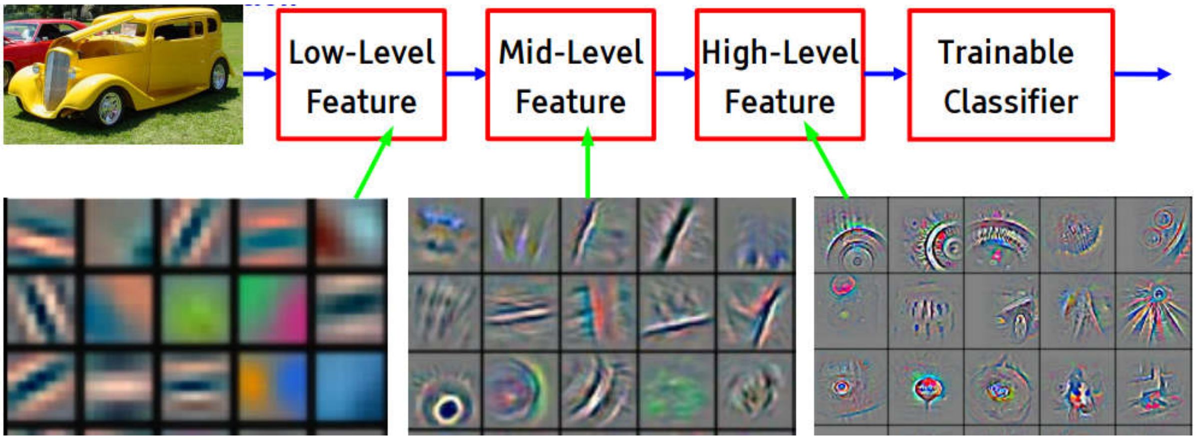
Convolutional Neural Networks (CNN, or ConvNet)

- Proposed in 1998 by Yann LeCun (french prof.@ NYU, recently appointed AI research director of Facebook)



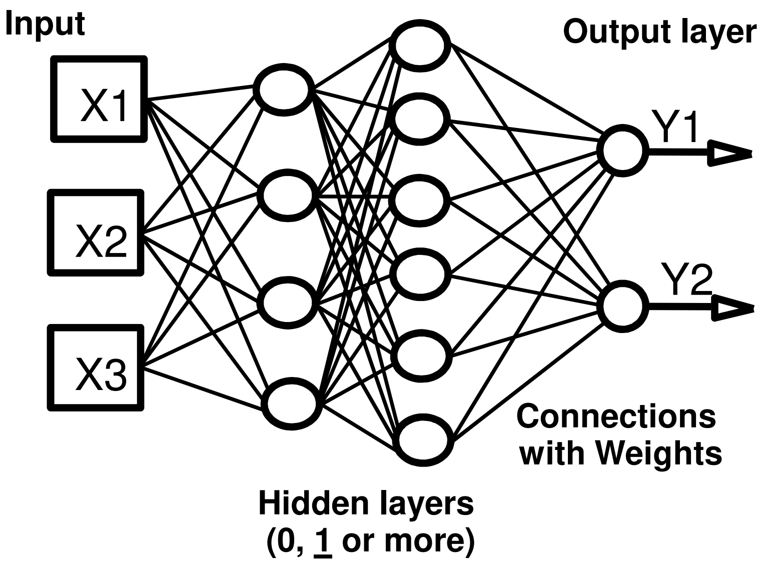
- For inputs with correlated dims (2D <u>*image*</u>, 1D signal,…)
- <u>Supervised</u> learning

CNN (2)



- Recently won many vision pattern recognition competitions/challenges (OCR, TSR, object categorization, facial expression,…)
- Deployed in photo-tagging by Facebook, Google, Baidu,…
- Also used in real-time video analysis for self-driving cars

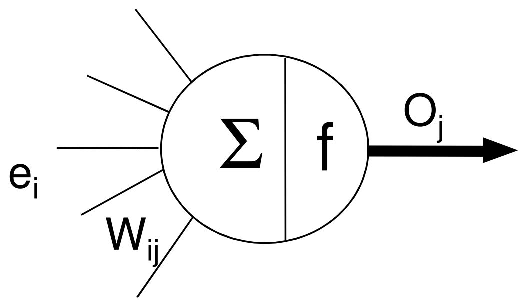
Short reminder on what is a (multi-layer) Neural Network



For "Multi-Layer Perceptron" (MLP), neurons type generally "summating with sigmoid activation"

Reminder on artificial "neurons"

PRINCIPLE

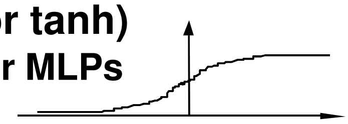


$$O_j = f\left(W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i\right)$$

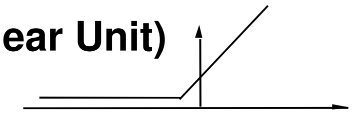
W_{0j} = "bias"

ACTIVATION FUNCTIONS

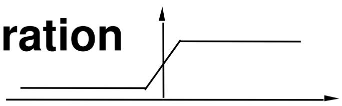
- **Threshold** (Heaviside or sign)
  → *binary* neurons
- **Sigmoïd** (logistic or tanh)
  → most common for MLPs



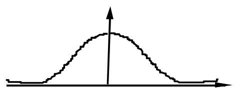
- **Identity** → *linear* neurons
- **ReLU** (Rectified Linear Unit)



- **Saturation**



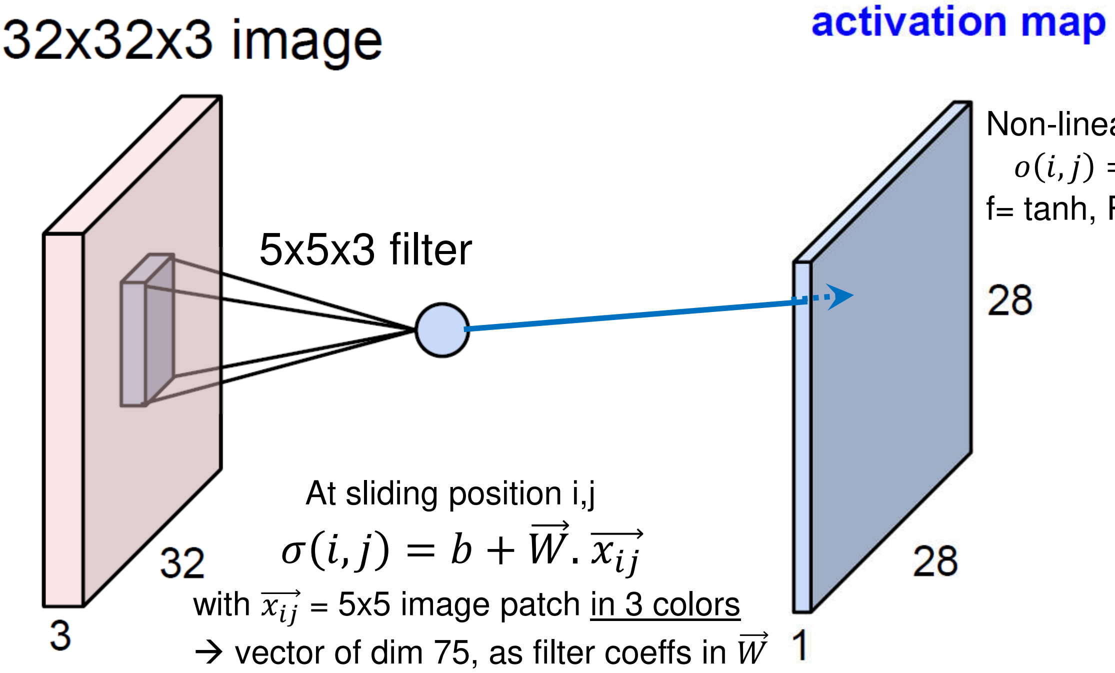
- **Gaussian**



Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

Convolution: sliding a 3D filter over image



Non-linear activation:
$o(i,j) = f\big(\sigma(i,j)\big)$
f= tanh, ReLU, …

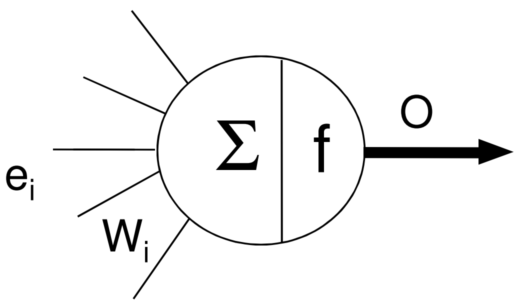
At sliding position i,j

$$\sigma(i,j) = b + \overrightarrow{W}.\overrightarrow{x_{ij}}$$

with $\overrightarrow{x_{ij}}$ = 5x5 image patch <u>in 3 colors</u>
→ vector of dim 75, as filter coeffs in $\overrightarrow{W}$

See illustrative animation at: `http://cs231n.github.io/convolutional-networks/`

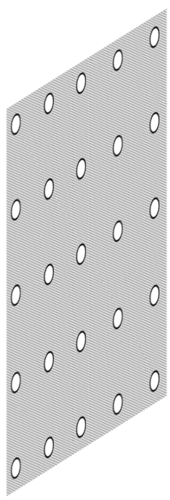
« Neural » view of convolution filters and layers



$$O = f\left(W_0 + \sum_{i=1}^{n} W_i e_i\right)$$

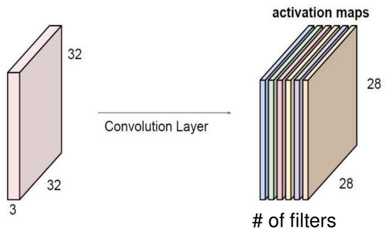
W_0 = "bias“
f = activation function

Each convolution FILTER is one set of neuron parameters



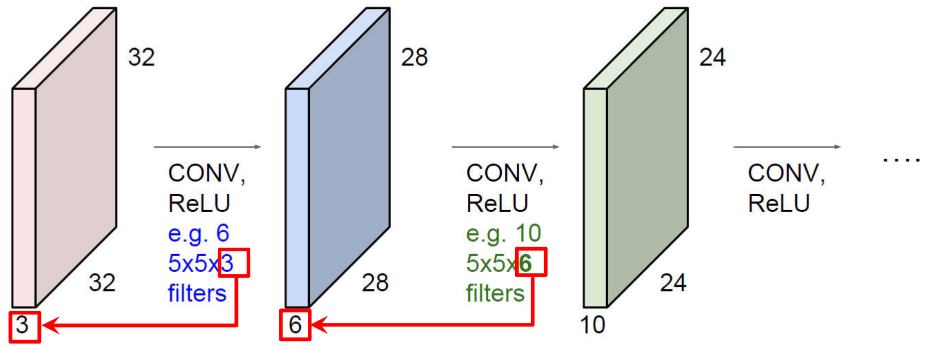
Each convolution LAYER is a set of ~`imageSize` neurons, but they all have <u>same SHARED weights</u> <u>(perform SAME convolution)</u>

Convolutional layers



One "activation map" for each convolution filter

A convNet: succession of Convolution+activation Layers

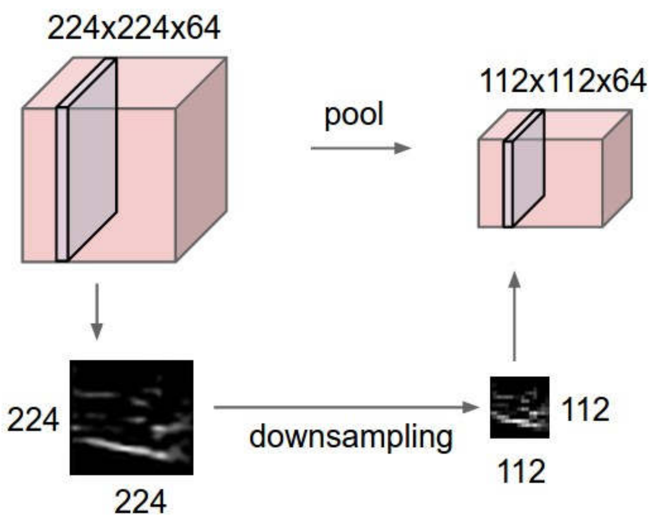


NB: each convolution layer processes *FULL DEPTH* of previous activation map (3D convolution!)

Pooling layers

Goal:

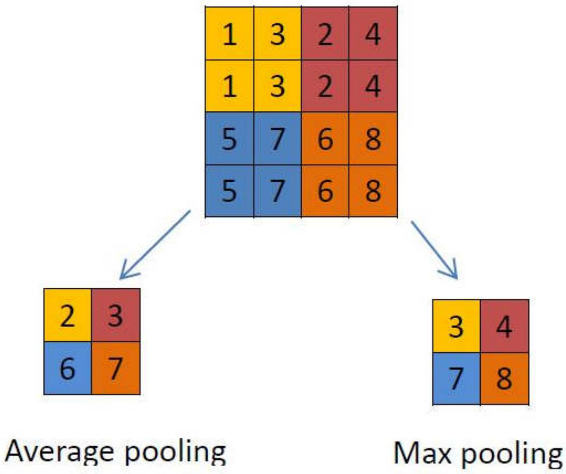
- **aggregation over space**
- **noise reduction,**
- **small-translation invariance,**
- **small-scaling invariance**



Pooling layers algorithm details

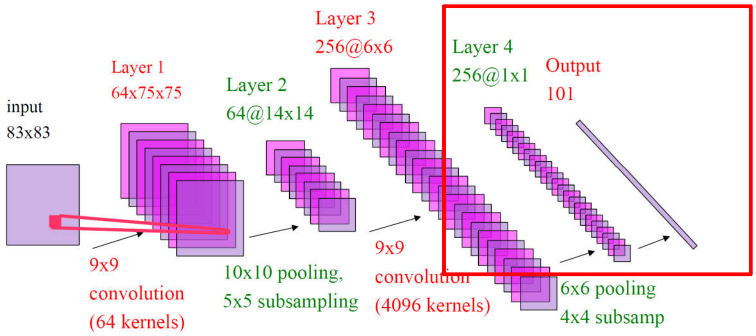
Parameters:

- pooling size (often 2x2)
- pooling stride (usually = pooling_size)
- Pooling operation: ***max***, average, Lp,…

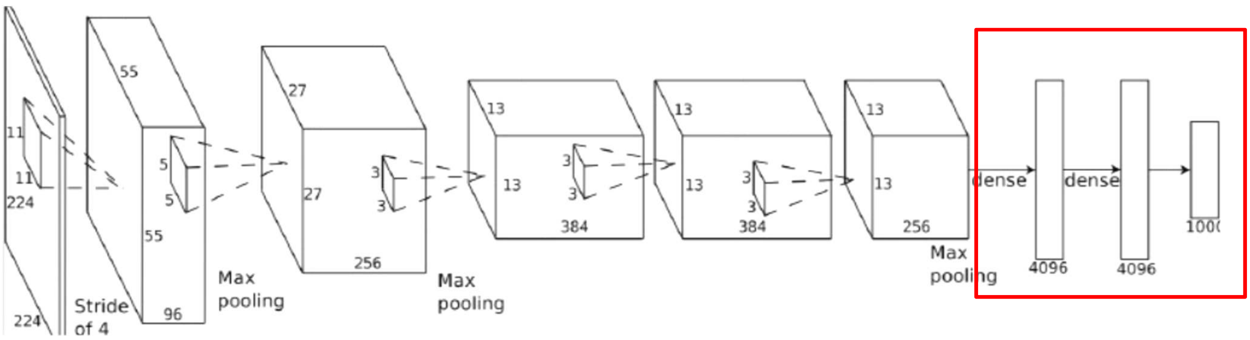


Example: 2x2 pooling, stride 2

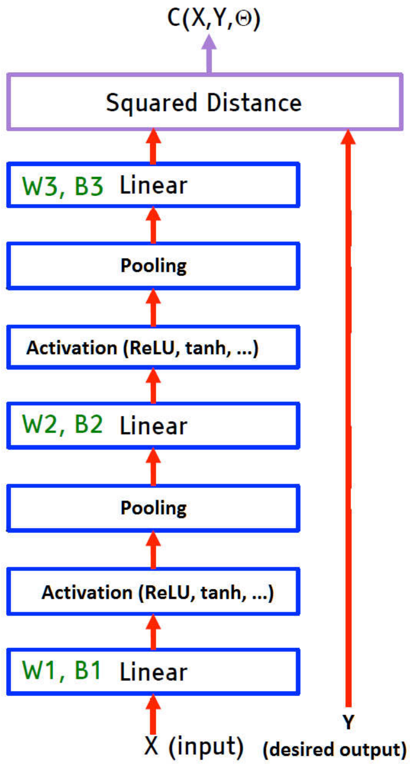
Final classification layer: often classical fully-connected MLP



AlexNet



ConvNet typical architecture: cascade of modules



Linear Module

Out = W.In+B

ReLU Module (Rectified Linear Unit)

$Out_i = 0$ if $In_i<0$

$Out_i = In_i$ otherwise

Pooling Module

Out =

Cost Module: Squared Distance

$C = ||In1 - In2||^2$

Objective Function

$L(\Theta)=1/p\ \Sigma_k\ C(X_k,Y_k,\Theta)$

Linear $\Theta = (W1,B1,W2,B2,W3,B3)$

Outline

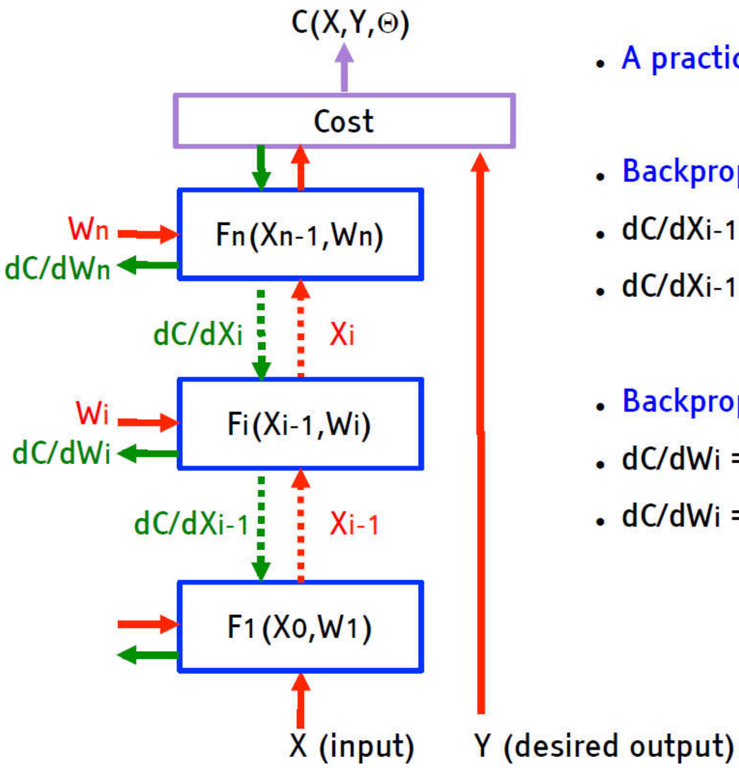
- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

ConvNet training

All successive layers of a convNet forms a Deep neural network (with weigh-sharing inside each conv. Layer, and specific pooling layers).

- **Training by <u>Stochastic Gradient Descent</u> (SGD), using *back-propagation*:**
  - **Input 1 random training sample**
  - **Propagate**
  - **Calculate error (loss)**
  - **Back-propagate through all layers from end to input, to compute gradient**
  - **Update convolution filter weights**

Computing gradient through cascade of modules



- A practical Application of Chain Rule
- Backprop for the state gradients:
- $dC/dX_{i-1} = dC/dX_i \, . \, dX_i/dX_{i-1}$
- $dC/dX_{i-1} = dC/dX_i \, . \, dF_i(X_{i-1},W_i)/dX_{i-1}$
- Backprop for the weight gradients:
- $dC/dW_i = dC/dX_i \, . \, dX_i/dW_i$
- $dC/dW_i = dC/dX_i \, . \, dF_i(X_{i-1},W_i)/dW_i$

Recall of back-prop principle

Smart method for efficient computing of gradient (w.r.t. weights) of a Neural Network cost function, based on chain rule for derivation.

Cost function is $Q(t) = \Sigma_m \text{loss}(Y_m,D_m)$, where m runs over training set examples

Usually, $\text{loss}(Y_m,D_m) = ||Y_m-D_m||^2$ [quadratic error]

Total gradient:

$$W(t+1) = W(t) - \lambda(t)\ \text{grad}_W(Q(t)) + \mu(t)(W(t)-W(t-1))$$

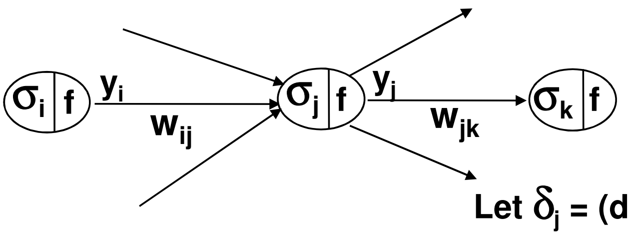
Stochastic gradient:

$$W(t+1) = W(t) - \lambda(t)\ \text{grad}_W(Q_m(t)) + \mu(t)(W(t)-W(t-1))$$

where $Q_m=\text{loss}(Y_m,D_m)$, is error computed on only ONE example randomly drawn from training set at every iteration and $\lambda(t)$ = learning rate (fixed, decreasing or adaptive), $\mu(t)$ = momentum

Now, how to compute dQ_m/dW_{ij}?

Backprop through fully-connected layers: use of chain rule derivative computation



$dE_m/dW_{ij} = (dE_m/d\sigma_j)(d\sigma_j/dW_{ij}) = (dE_m/d\sigma_j)\ y_i$

Let $\delta_j = (dE_m/d\sigma_j)$. Then $\boxed{W_{ij}(t+1) = W_{ij}(t) - \lambda(t)\ y_i\ \delta_j}$

(and $W_{0j}(t+1) = W_{0j}(t) - \lambda(t)\delta_j$)

If neuron j is output, $\delta_j = (dE_m/d\sigma_j) = (dE_m/dy_j)(dy_j/d\sigma_j)$ with $E_m=||Y_m-D_m||^2$

so $\boxed{\delta_j = 2(y_j-D_j)f'(\sigma_j) \text{ if neuron j is an output}}$

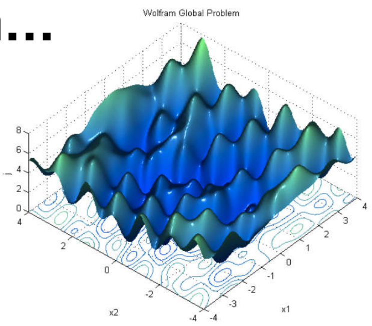
Otherwise, $\delta_j=(dE_m/d\sigma_j)=\Sigma_k (dE_m/d\sigma_k)(d\sigma_k/d\sigma_j)=\Sigma_k \delta_k(d\sigma_k/d\sigma_j) =\Sigma_k \delta_k W_{jk}(dy_j/d\sigma_j)$

so $\boxed{\delta_j = (\Sigma_k\ W_{jk}\delta_k)f'(\sigma_j) \text{ if neuron j is “hidden”}}$

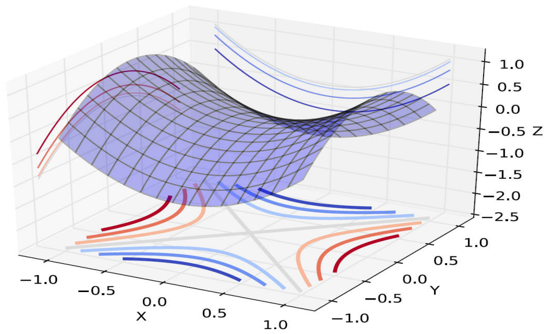
➔ all the δ_j can be computed successively from last layer to upstream layers by “error backpropagation” from output

Why gradient descent works *despites non-convexity*?

- **Local minima dominate in low-Dim…**

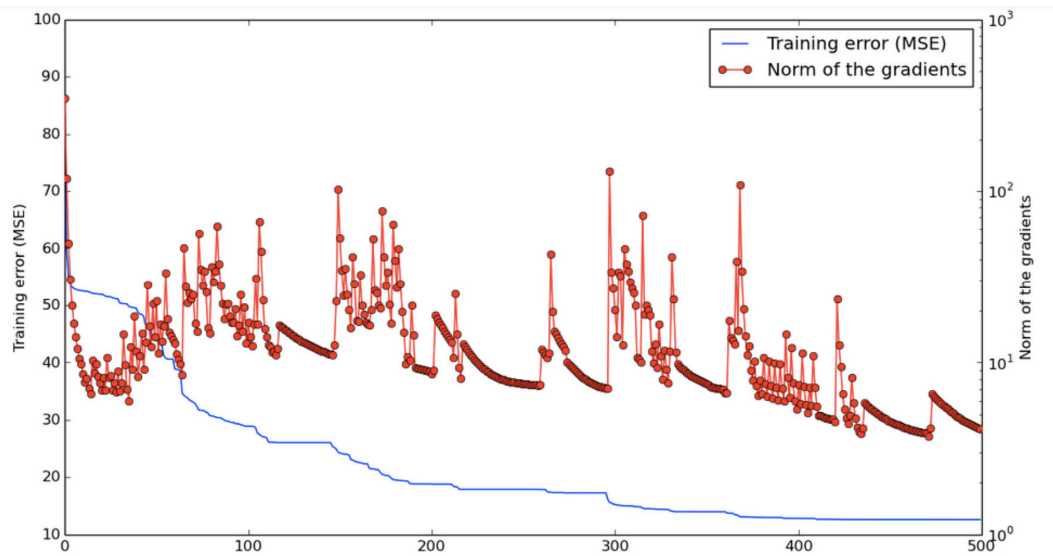


- **…but recent work has shown *saddle points* dominate in high-Dim**



- **Furthermore, most local minima are close to the global minimum**

Saddle points in training curves

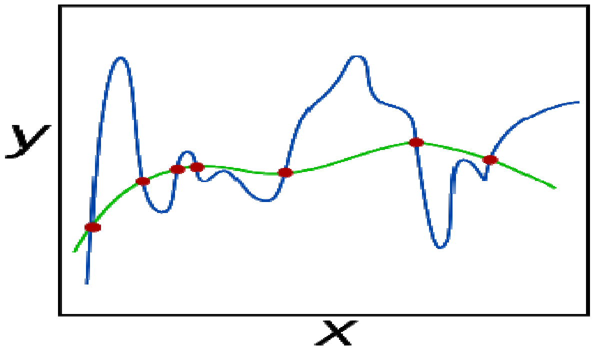


- **Oscillating between two behaviors:**
  - **Slowly approaching a saddle point**
  - **Escaping it**

Some ConvNet training « tricks »

- Importance of <u>input normalization</u>
  (zero mean, unit variance)
- Importance of <u>weights initialization</u>
  random but SMALL and prop. to 1/sqrt(nbInputs)
- Decreasing (or adaptive) <u>learning rate</u>
- Importance of <u>training set size</u>
  ConvNets often have a LARGE number of free parameters
  ➔ train them with a sufficiently large training-set !
- Avoid overfitting by:
  – Use of L1 or L2 <u>regularization</u> (after some epochs)
  – Use « *Dropout* » <u>regularization</u> (esp. on large FC layers)

Avoid <u>overfitting</u> using L1/L2 regularization (« weight decay »)



Trying to fit too many free parameters with not enough information can lead to overfitting

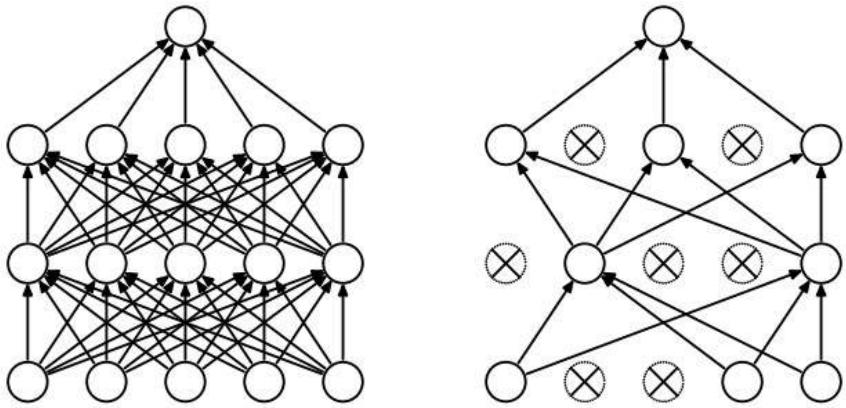
<u>Regularization</u> = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just <u>norm L2 or L1 of vector of all weights</u>:

$$K = \Sigma_m(\text{loss}(Y_m,D_m)) + \beta\, \Sigma_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2) or } p=1 \text{ (L1)}$$

→ name "Weight decay"

DropOut Regularization for convNet training



(a) Standard Neural Net (b) After applying dropout.

At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~0.5), or re-installed with last values of weights

Outline

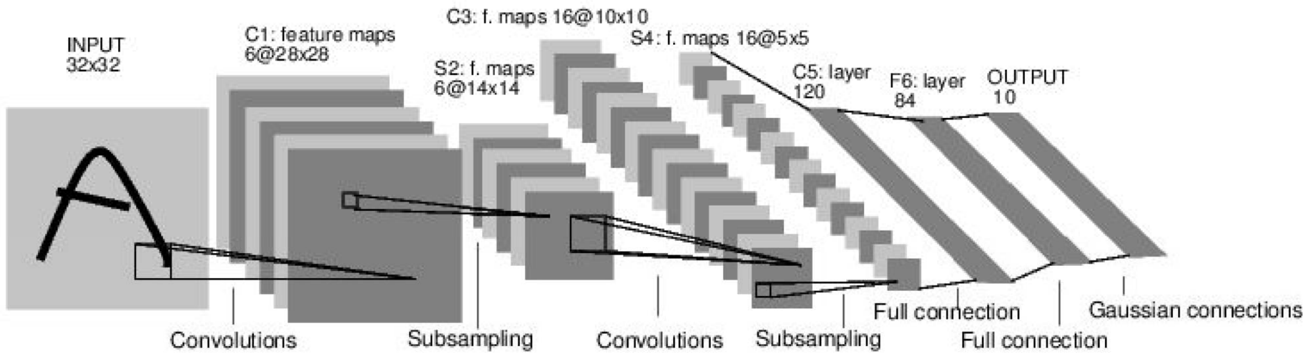
- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- <u>Useful pre-trained convNets and coding frameworks</u>
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

Examples of successful ConvNets

- **LeNet:** 1st successful applications of ConvNets, by Yann LeCun in 1990's. Used to read zip codes, digits, etc.
- **AlexNet:** Beginning of ConvNet "buzz": largely outperformed competitors in ImageNet_ILSVRC2012 challenge. Developped by Alex Krizhevsky et al., architecture similar to LeNet (but deeper+larger, and some chained ConvLayers before Pooling). 60 M parameters !
- **ZF Net:** ILSVRC 2013 winner. Developped by Zeiler&Fergus, by modif of AlexNet on some architecture hyperparameters.
- **GoogLeNet:** ILSVRC 2014 winner, developed by Google. Introduced an *Inception Module*, + AveragePooling instead of FullyConnected layer at output. Dramatic reduction of number of parameters (4M, compared to AlexNet with 60M).
- **VGGNet:** Runner-up in ILSVRC 2014. Very deep (16 CONV/FC layers) → 140M parameters !!
- **ResNet:** ILSVRC 2015, "Residual Network" introducing "skip" connections. Currently ~ SoA in convNet. Very long training but fast execution.

LeNet, for digits/letters recognition
[LeCun et al., 1998]

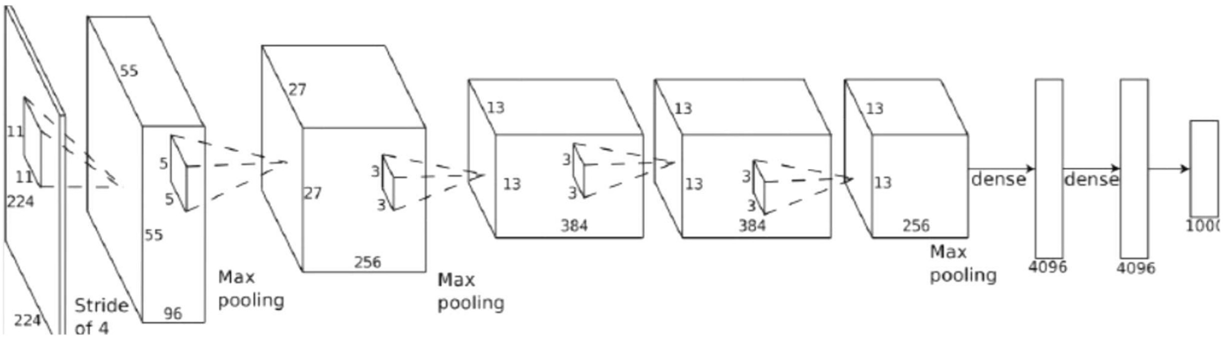
Input: 32x32 image



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

AlexNet, for image categorisation
[Krizhevsky et al. 2012]

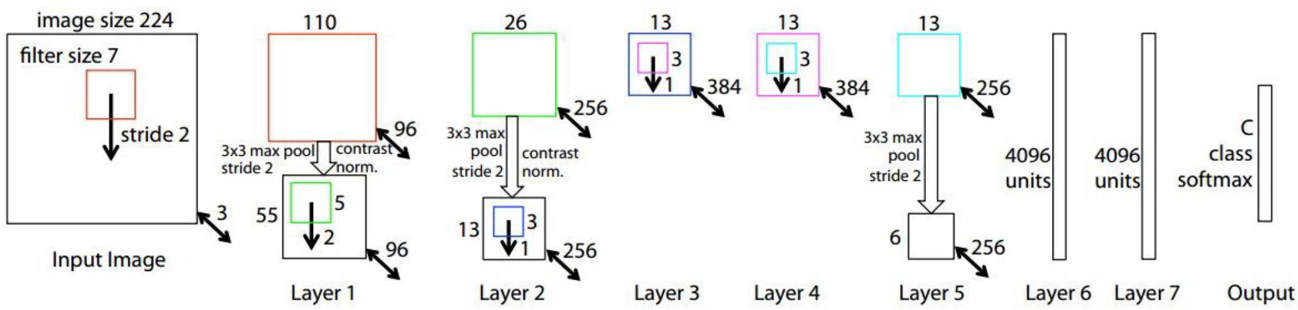
Input: 224x224x3 image



60 million parameters !...

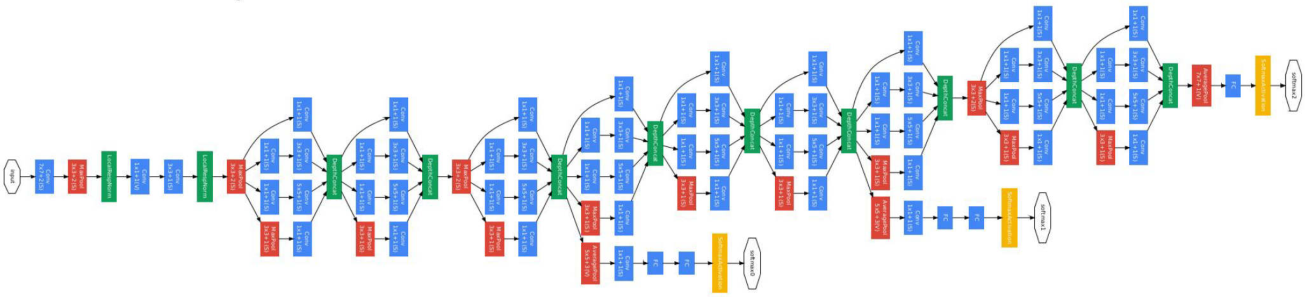
ZFnet
[Zeiler & Fergu, 2013]

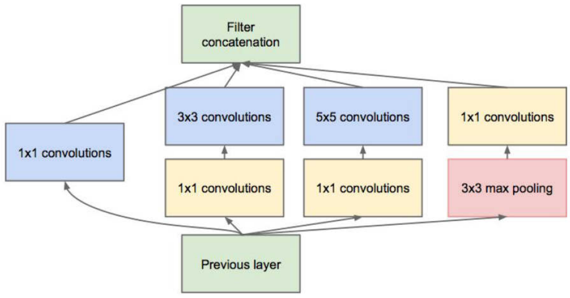
Input: 224x224x3 image



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

GoogleNet
[Szegedy et al., 2014]



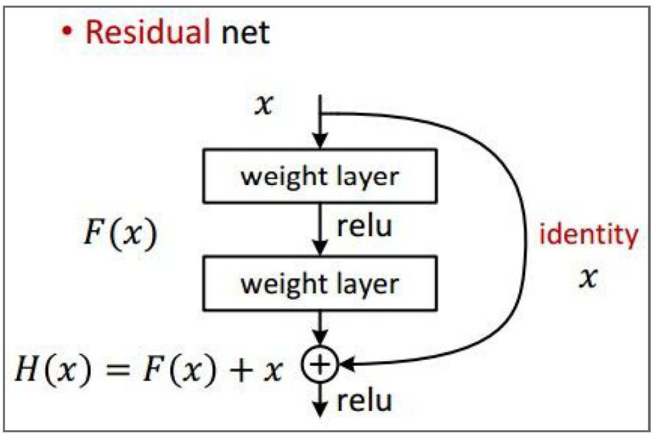


Inception module

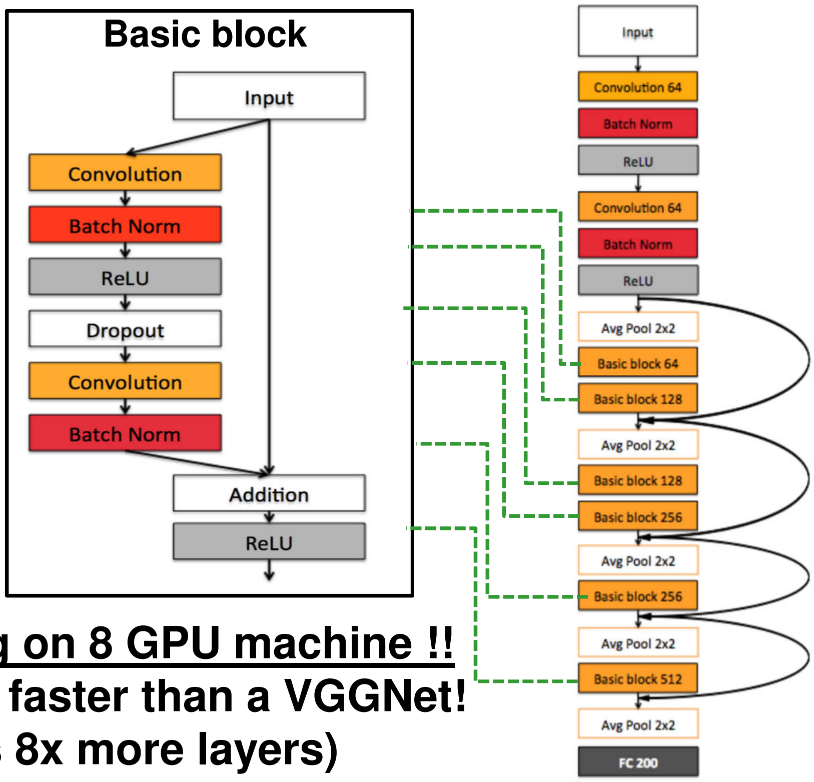
ILSVRC 2014 winner (6.7% top 5 error)

ResNet (Residual Net), by Microsoft
[He et al., 2015]

- **ILSVRC 2015 *large* winner in 5 main tracks (3.6% top 5 error)**
- **152 layers!!!**
- **But novelty = *"skip" connections***

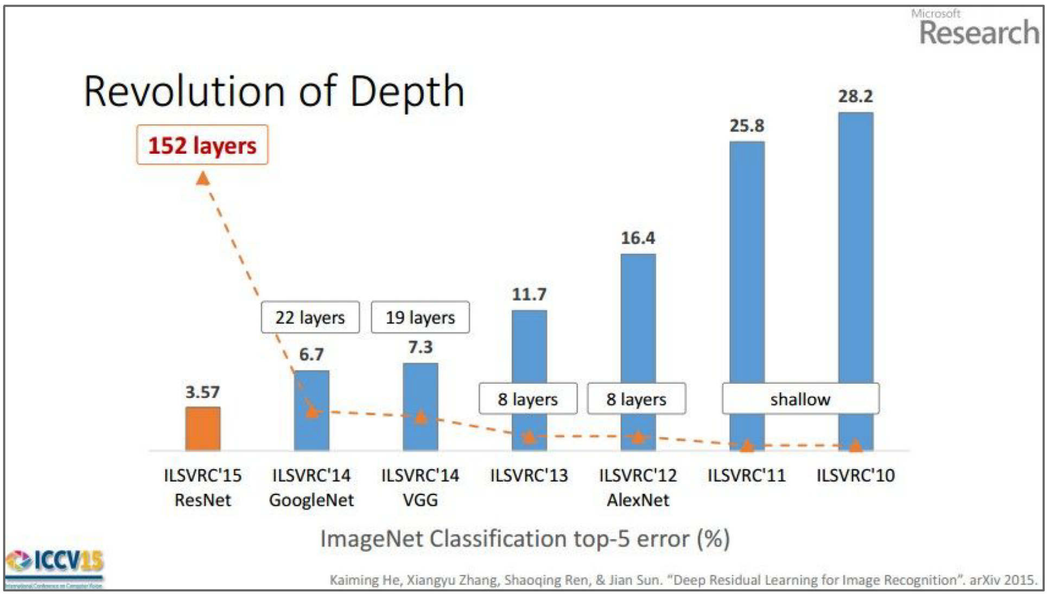


ResNet global architecture



- <u>**2-3 weeks of training on 8 GPU machine !!**</u>
- **However, at runtime faster than a VGGNet! (even though it has 8x more layers)**

Summary of recent ConvNet history



But most important is the choice of
<u>***ARCHITECTURAL STRUCTURE***</u>

Programming environments for Deep-Learning

- **Caffe** http://caffe.berkeleyvision.org/
  C++ library, hooks from Python → notebooks
- **Torch** http://torch.ch/
- **TensorFlow** https://www.tensorflow.org
- **Theano** http://www.deeplearning.net/software/theano/
- **Lasagne** http://lasagne.readthedocs.io
  lightweight library to build+train neural nets in Theano
- **KERAS** https://keras.io
  Python front-end APIs mapped either on Tensor-Flow or Theano back-end

All of them handle transparent use of GPU, and most of them are used in Python code/notebook

Example of convNet code in Keras

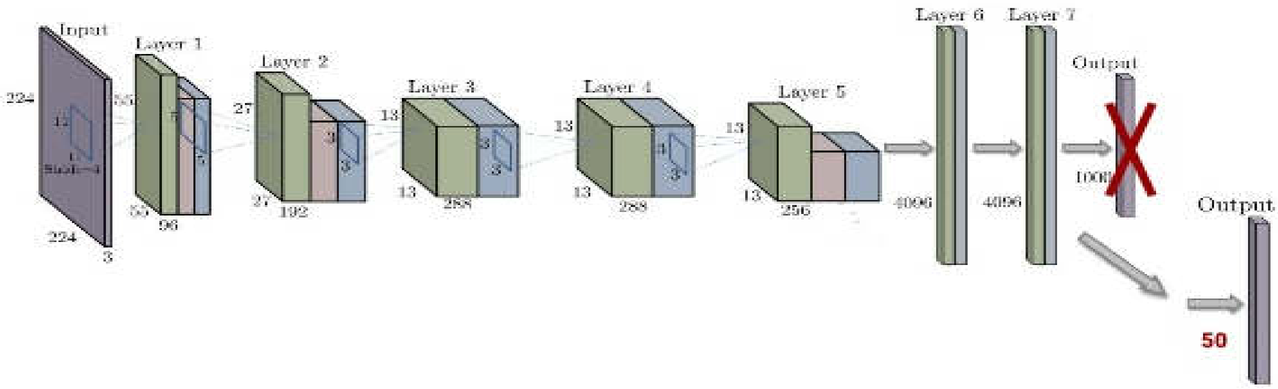
```
model = Sequential()
# Convolution+Pooling layers, with Dropout
model.add(Convolution2D(conv_depth_1, kernel_size, kernel_size,
          border_mode='valid', input_shape=(depth, height, width)))
model.add( MaxPooling2D(pool_size=(pooling_size, pooling_size)) )
model.add(Activation('relu'))
model.add(Dropout(drop_prob))
# Now flatten to 1D, and apply 1 Fully_Connected layer
model.add(Flatten())
model.add(Dense(hidden_size1, init='lecun_uniform'))
model.add(Activation('sigmoid'))
# Finally add a Softmax output layer, with 1 neuron per class
model.add(Dense(num_classes, init='lecun_uniform'))
model.add(Activation('softmax'))
# Training "session
sgd = SGD(lr=learning_rate, momentum=0.8) # Optimizer
model.compile(loss='categorical_crossentropy', optimizer=sgd)
model.fit(X_train, Y_train, batch_size=32, nb_epoch=2, verbose=1,
                          validation_split=valid_proportion)

# Evaluate the trained model on the test set
model.evaluate(X_test, Y_test, verbose=1)
```

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- <u>Transfer Learning</u>
- Deep Belief Networks (DBN)
- Autoencoders
- Recurrent Neural Networks (RNN)

Generality of learnt representation + Transfer learning



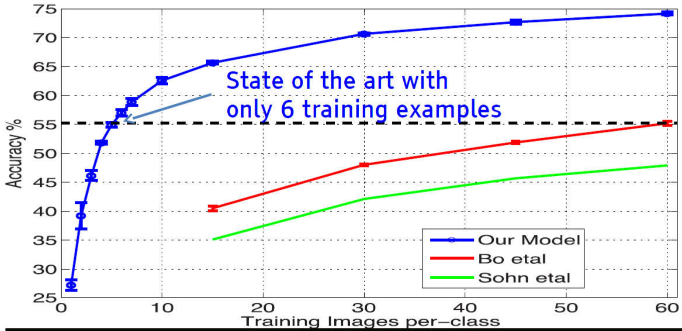
By <u>removing *last layer(s)*</u> (those for classification) of a convNet trained on ImageNet, one obtains a <u>*transformation of any input image into a semi-abstract representation,*</u> which can be used for learning SOMETHING ELSE (« <u>transfer learning</u> »):

- either by just <u>using learnt representation as features</u>
- or by creating new convNet output and perform <u>learning of new output layers + fine-tuning of re-used layers</u>

Transfer Learning and fine-tuning

- Using a CNN pre-trained on a large dataset, possible to <u>adapt it to another task, *using only a SMALL training set*</u>!

- Network first trained on ImageNet.
- Last layer chopped off
- Last layer trained on Caltech 256,
- first layers N-1 kept fixed.



- State of the art accuracy with only 6 training samples/class

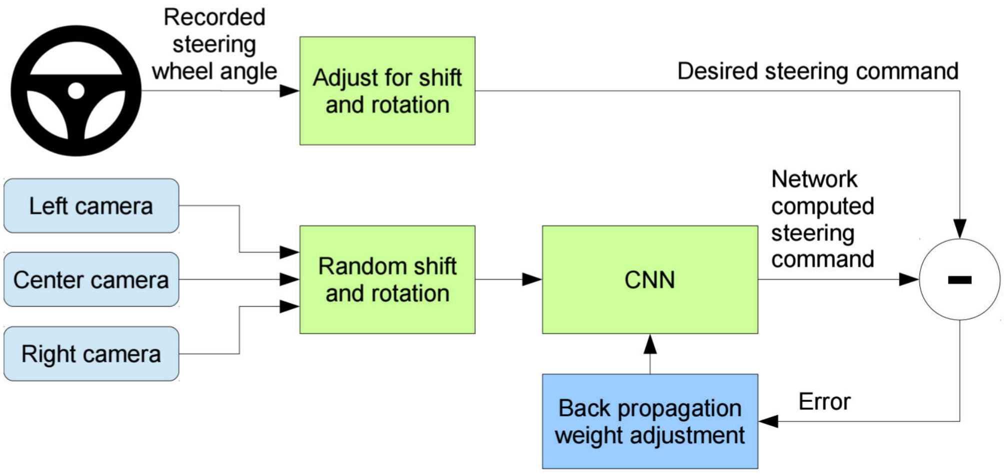
# Train	Acc % 15/class	Acc % 30/class	Acc % 45/class	Acc % 60/class
Sohn *et al.* [16]	35.1	42.1	45.7	47.9
Bo *et al.* [3]	40.5 ± 0.4	48.0 ± 0.2	51.9 ± 0.2	55.2 ± 0.3
Non-pretr.	9.0 ± 1.4	22.5 ± 0.7	31.2 ± 0.5	38.8 ± 1.4
ImageNet-pretr.	**65.7 ± 0.2**	**70.6 ± 0.2**	**72.7 ± 0.4**	**74.2 ± 0.3**

3: [Bo, Ren, Fox. CVPR, 2013]   16: [Sohn, Jung, Lee, Hero ICCV 2011]

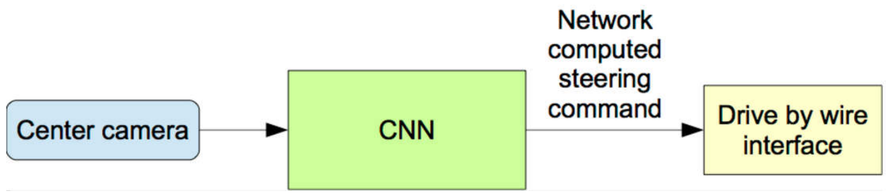
Examples of transfer-learning applications

- Recognition/classification for OTHER categories or classes
- <u>Direct control of driving wheel! (DeepDrive)</u>
- <u>Precise localisation (position+bearing) = PoseNet</u>
- … or even 3D informations from monovision!

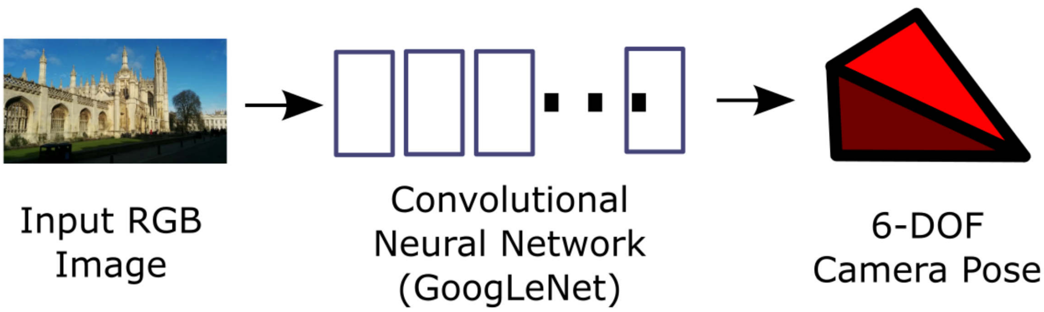
Learning to drive with transfer-learning



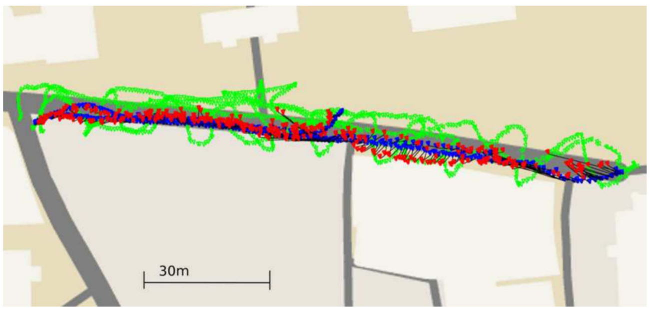
nVidia approach



Transfer-Learning for 6-DOF Camera Relocalization



[A. Kendall, M. Grimes & R. Cipolla, "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization« , ICCV'2015, pp. 2938-2946]



King's College

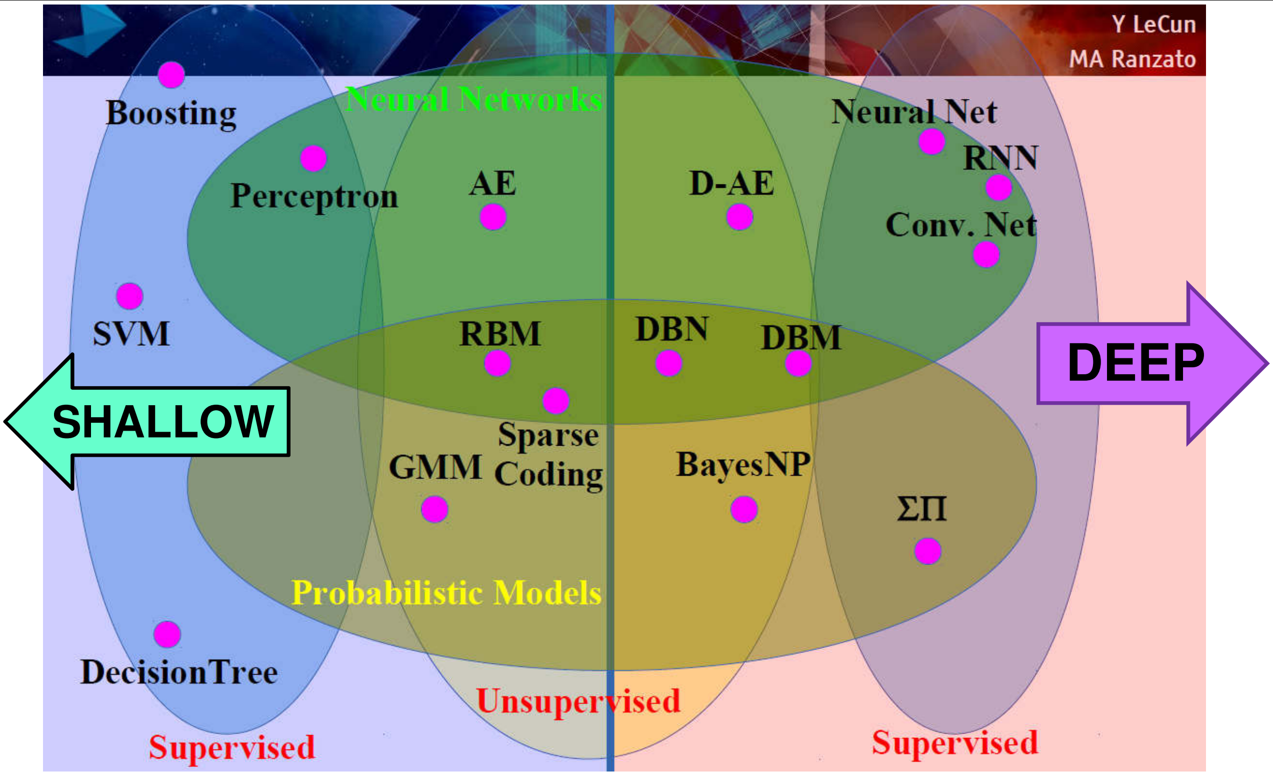
Summary and perspectives on ConvNets & Deep-Learning

- Proven advantage of learning features empirically from data
- Large ConvNets require huge amounts of <u>labelled</u> examples data for training
- Current research/progresses = finding efficient global architecture of ConvNets
- <u>Enormous potential of transfer learning on small datasets for restricted/specialized problems</u>
- Next frontier: methods for combining <u>UNsupervised deep-learning on unlabelled data</u> with supervised training on smaller labelled dataset

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- <u>Deep Belief Networks (DBN)</u>
- Autoencoders
- Recurrent Neural Networks (RNN)

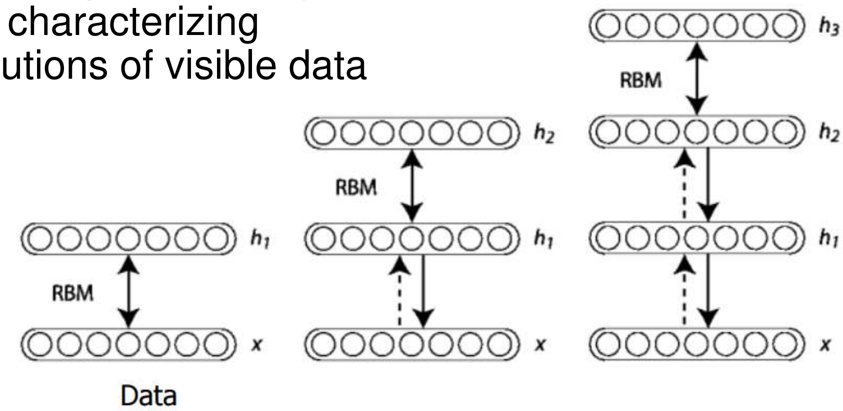
Deep vs Shallow Learning techniques overview



Deep Belief Networks (DBN)

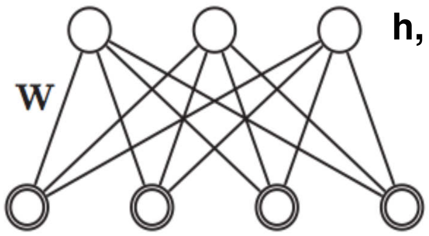
- **One of first Deep-Learning models**
- **Proposed by G. Hinton in 2006**
- **Generative probabilistic model (mostly UNSUPERVISED)**

For capturing high-order *correlations* of observed/visible data (→ pattern analysis, or synthesis); and/or characterizing *joint* statistical distributions of visible data



Greedy successive UNSUPERVISED learning of layers of Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine (RBM)



h, hidden
(~ latent variables)

v, observed

NB: connections are BI-DIRECTIONAL (with same weight)

Modelling <u>probability distribution</u> as:

$$P(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))}$$

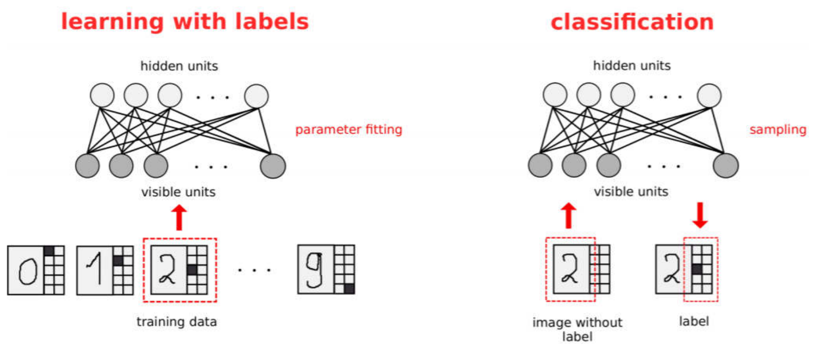
with ***<u>« Energy »</u>*** E given by

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\mathbf{v}^\top W \mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{a}^\top \mathbf{h}$$
$$= -\sum_{i=1}^{D} \sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j h_j$$

Use of trained RBM

- **Input data "completion" : set some v_i then compute h, and generate compatible full samples**
- **Generating representative samples**

- **Classification if trained with inputs=data+label**



Training RBM

Finding θ=(W,a,b) **maximizing likelihood** $\prod_{v \in S} p_\theta(v)$ **of dataset S**

⇔ **minimize NegLogLikelihood** $-\sum_{v \in S} \log(p_\theta(v))$

Independance within layers ➔ $p(v|h) = \prod_i p(v_i|h)$ **and** $p(h|v) = \prod_j p(h_j|v)$

So objective = find $\boxed{\theta_* = \underset{\theta}{\operatorname{argMin}} \left(-\sum_{v \in S} \sum_j \log(p_\theta(v_j)) \right)}$

In *binary* input case:

$$p(v_i = 1 \mid h) = \sigma(a_i + W_{:,i} h)$$
$$p(h_j = 1 \mid v) = \sigma(b_j + W_{j,:} v)$$

with $\sigma(u) = \dfrac{e^u}{e^u + 1}$

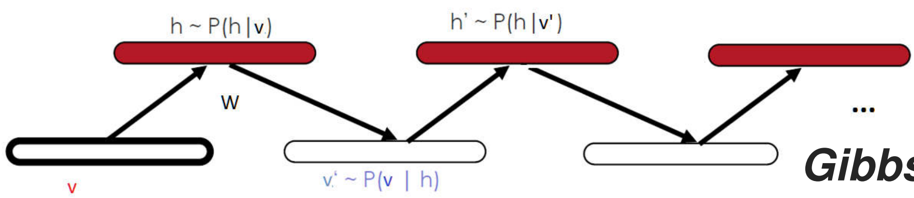
Algo: Contrastive Divergence

≈ Gibbs sampling used inside a gradient descent procedure

Contrastive Divergence algo

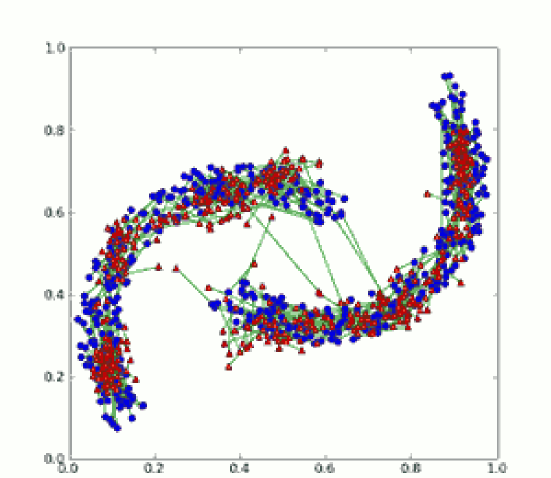
Repeat:

1. **Take a training sample *v*, compute** $p(h_j = 1 \mid v) = \sigma(b_j + W_{j,:} v)$
   and sample a vector *h* from this probability distribution
2. **Compute *positive* gradient as outer product** $G_+ = v \otimes h = v h^T$
3. **From *h*, compute** $p(v'_i = 1 \mid h) = \sigma(a_i + W_{:,i} h)$ **and sample reconstructed *v'*, then resample *h'* using** $p(h'_j = 1 \mid v') = \sigma(b_j + W_{j,:} v')$
   [Gibbs sampling single step; should theoretically be repeated until convergence]
4. **Compute *negative* gradient as outer product** $G_- = v' \otimes h' = v' h'^T$
5. **Update weight matrix by** $\delta W = \varepsilon(G_+ - G_-) = \varepsilon(v h^T - v' h'^T)$
6. **Update biases *a* and *b* analogously:** $\delta a = \varepsilon(v - v')$ **and** $\delta b = \varepsilon(h - h')$

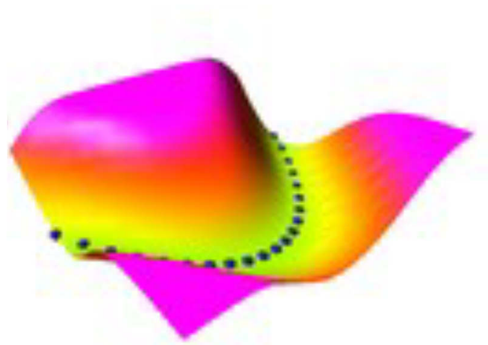


Gibbs sampling

Modeling of input data distribution obtained by trained RBM



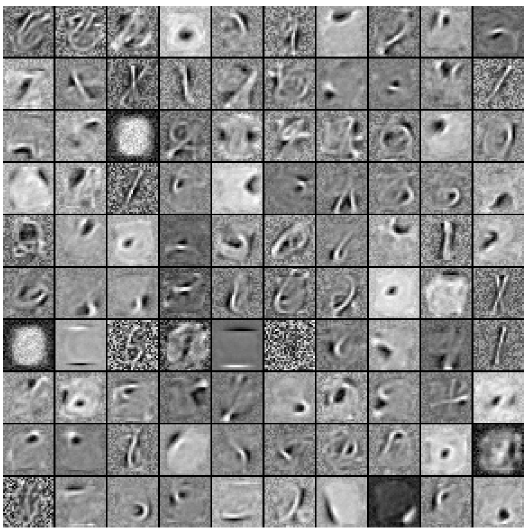
Initial data is in blue, reconstructed in red (and green line connects each data point with reconstructed one).

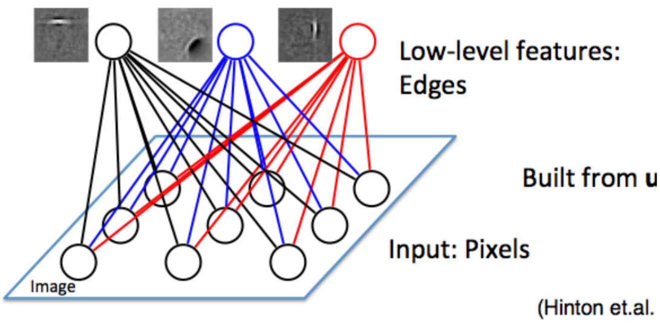


Learnt energy function:
<u>minima</u> created <u>where data points are</u>

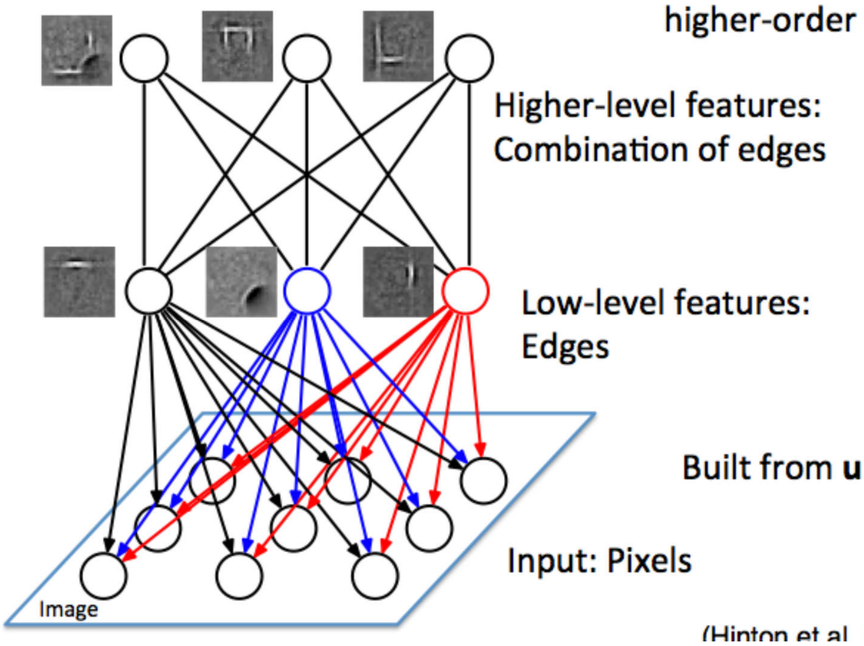
Interpretation of trained RBM hidden layer

- **Look at weights of hidden nodes → low-level features**





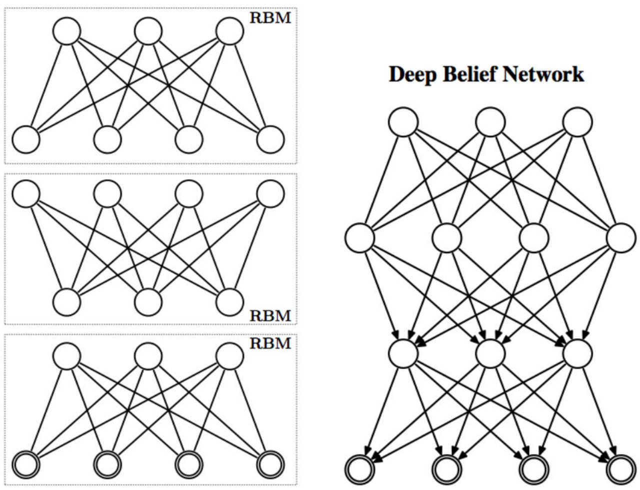
Why go deeper with DBN ?



DBN: upper layers → more « abstract » features

Learning of DBN

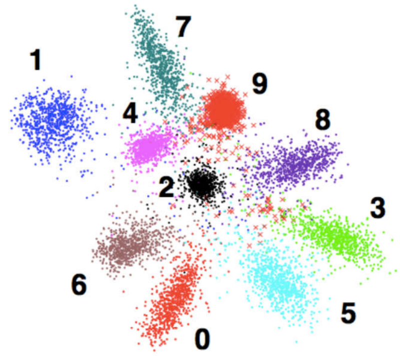
Greedy learning of successive layers

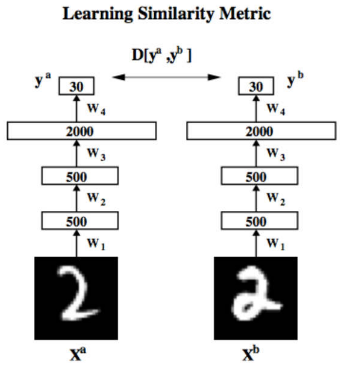


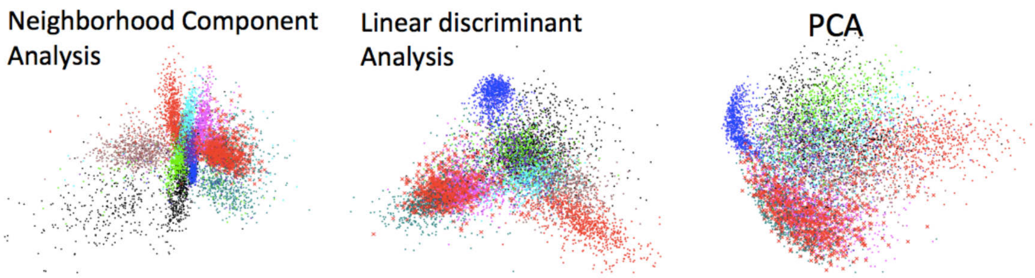
Algorithm 1 Recursive Greedy Learning Procedure for the DBN.

1: Fit parameters W^1 of the 1[st] layer RBM to data.
2: Freeze the parameter vector W^1 and use samples $\mathbf{h}^1$ from $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v}, W^1)$ as the data for training the next layer of binary features with an RBM.
3: Freeze the parameters W^2 that define the 2[nd] layer of features and use the samples $\mathbf{h}^2$ from $Q(\mathbf{h}^2|\mathbf{h}^1) = P(\mathbf{h}^2|\mathbf{h}^1, W^2)$ as the data for training the 3[rd] layer of binary features.
4: Proceed recursively for the next layers.

Using low-dim final features for clustering





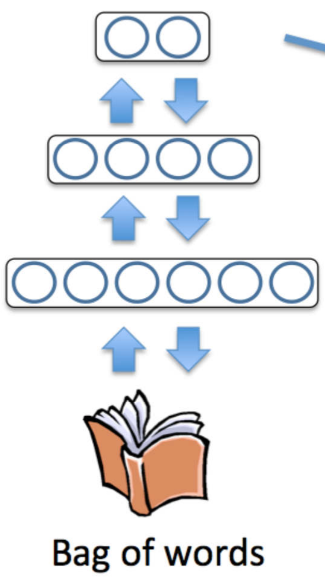


Much better results than clustering in input space or using other dimension reduction (PCA, etc…)

Example application of DBN: Clustering of documents in database

Model P(document)

Reuters dataset: 804,414 newswire stories: **unsupervised**



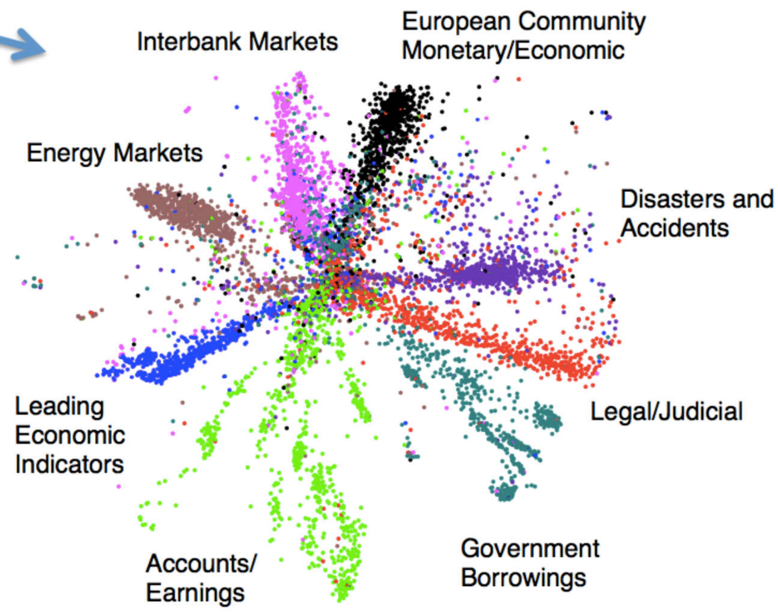
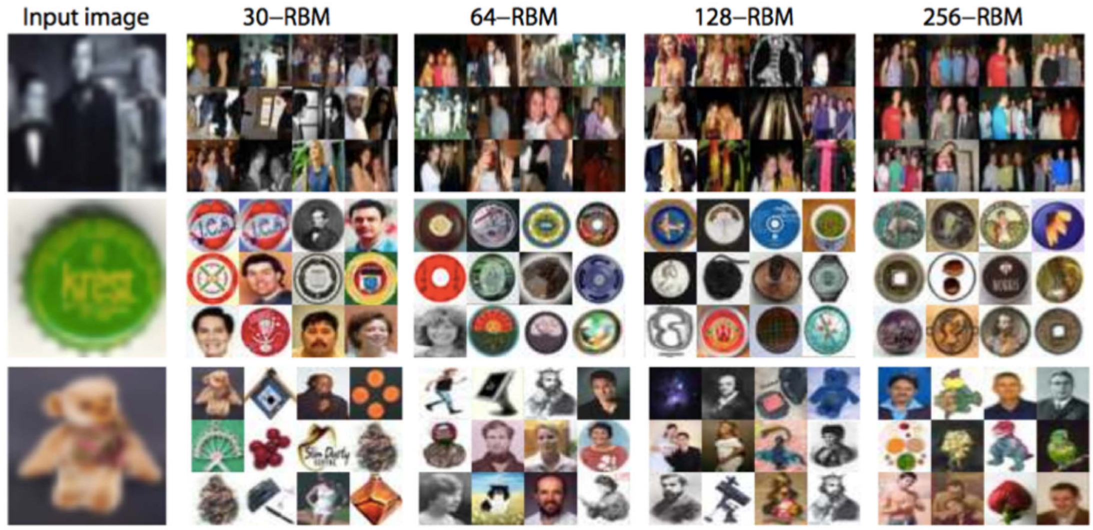


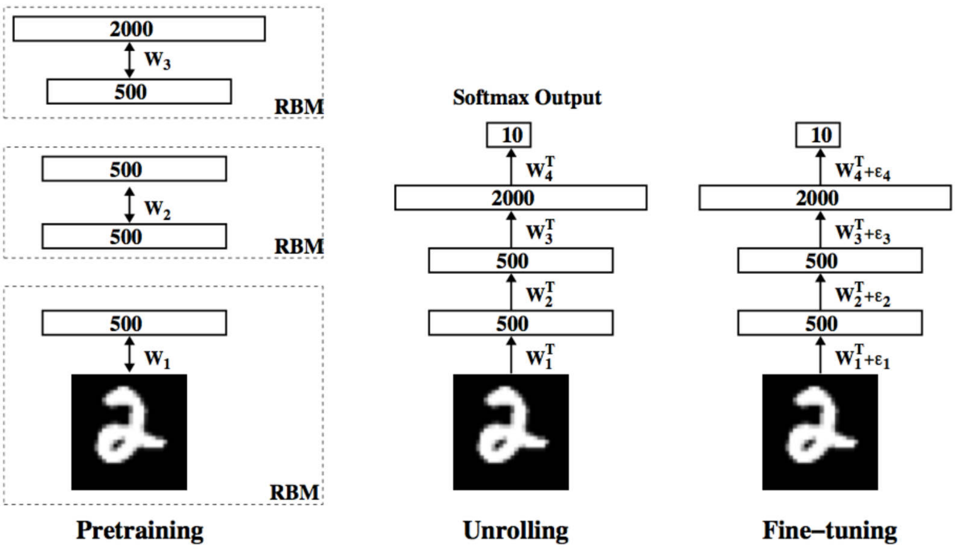
Image Retrieval application example of DBN

- Map images into binary codes for fast retrieval.



- Small Codes, Torralba, Fergus, Weiss, CVPR 2008
- Spectral Hashing, Y. Weiss, A. Torralba, R. Fergus, NIPS 2008
- Kulis and Darrell, NIPS 2009, Gong and Lazebnik, CVPR 20111
- Norouzi and Fleet, ICML 2011,

DBN supervised tuning



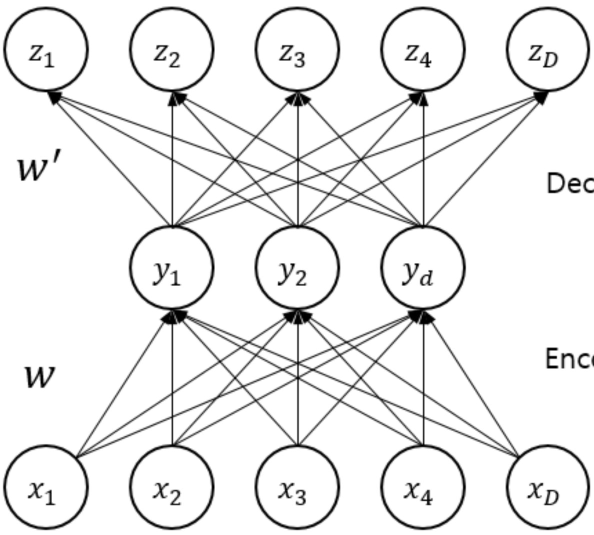
UNSUPERVISED SUPERVISED

- After layer-by-layer **unsupervised pretraining**, discriminative fine-tuning by backpropagation achieves an error rate of 1.2% on MNIST. SVM's get 1.4% and randomly initialized backprop gets 1.6%.

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- <u>Autoencoders</u>
- Recurrent Neural Networks (RNN)

Autoencoders



Decoder : $z = g_{\theta'}(y) = s(W'y + b')$

Encoder : $y = f_\theta(x) = s(Wx + b)$

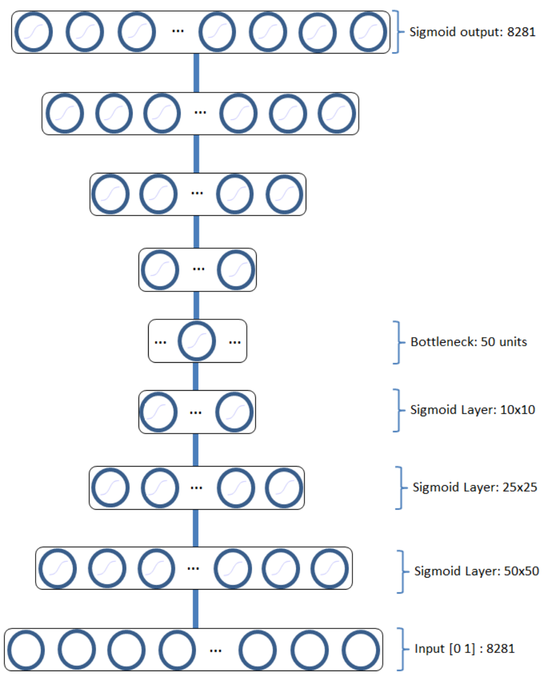
Learn W and W' to minimize: $\sum_k || z_k - x_k ||^2$

Variants of autoencoders

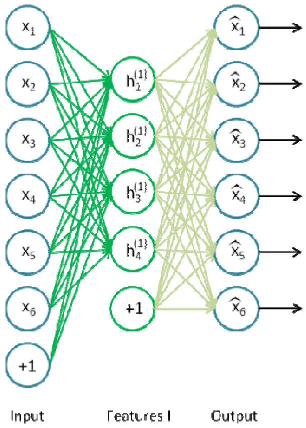
- <u>Denoising</u> autoencoders
- <u>Sparse</u> autoencoders
- Stochastic autoencoders
- Contractive autoencoders
- …

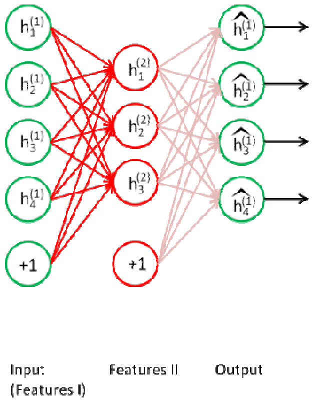
<u>Deep</u> *Stacked* Autoencoders

Proposed by Yoshua Bengio in 2007



Training of Stacked Autoencoers





etc…

Greedy layerwise training:

for each layer k, use *backpropagation* to minimize

$|| A_k(h^{(k)})-h^{(k)} ||^2$ (+ regularization cost $\lambda \Sigma_{ij} |W_{ij}|^2$)

possibly + additional term for "sparsity"

Use of Deep Autoencoders

- **Data compression / dimension reduction**
- **Learn a compact "code " ➔ Information Retrieval**
- **Noise removal**
- **Manifold learning**

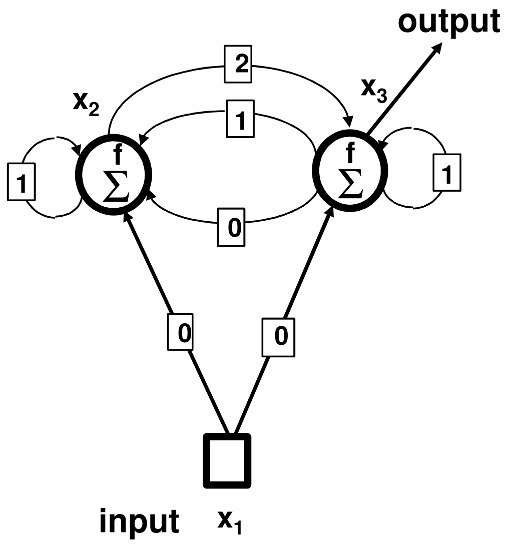
Summary and perspectives on DBN/RBM/DBM / AE

- <u>Intrinsicly UNSUPERVISED</u>
  ➔ can be used on UNLABELLED DATA
- Impressive results in <u>Image Retrieval</u>
- DBN/RBM/DBM = <u>Generative</u> *probabilistic* models
- Strong potential for enhancement of datasets
- Interest for "creative« /artistic computing?

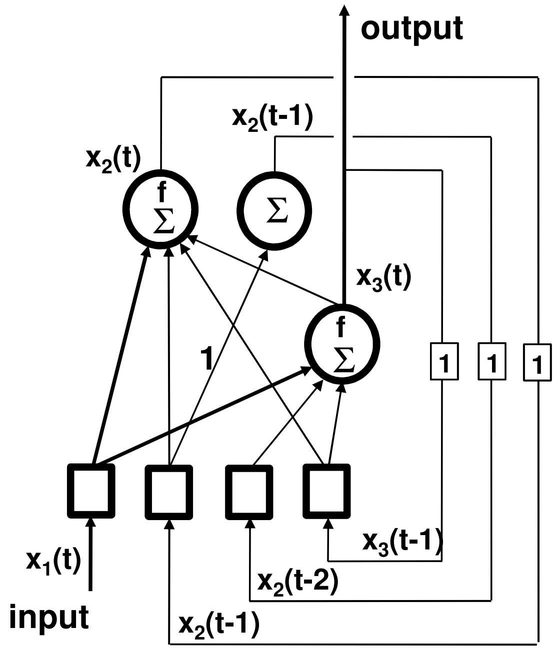
Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets and coding frameworks
- Transfer Learning
- Deep Belief Networks (DBN)
- Autoencoders
- <u>Recurrent Neural Networks (RNN)</u>

Recurrent Neural Networks (RNN)

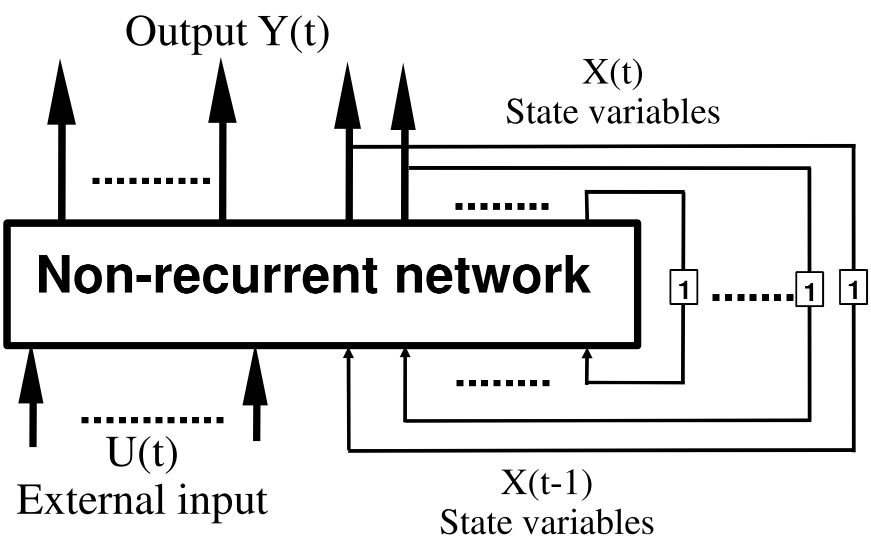


Time-delay for each connection

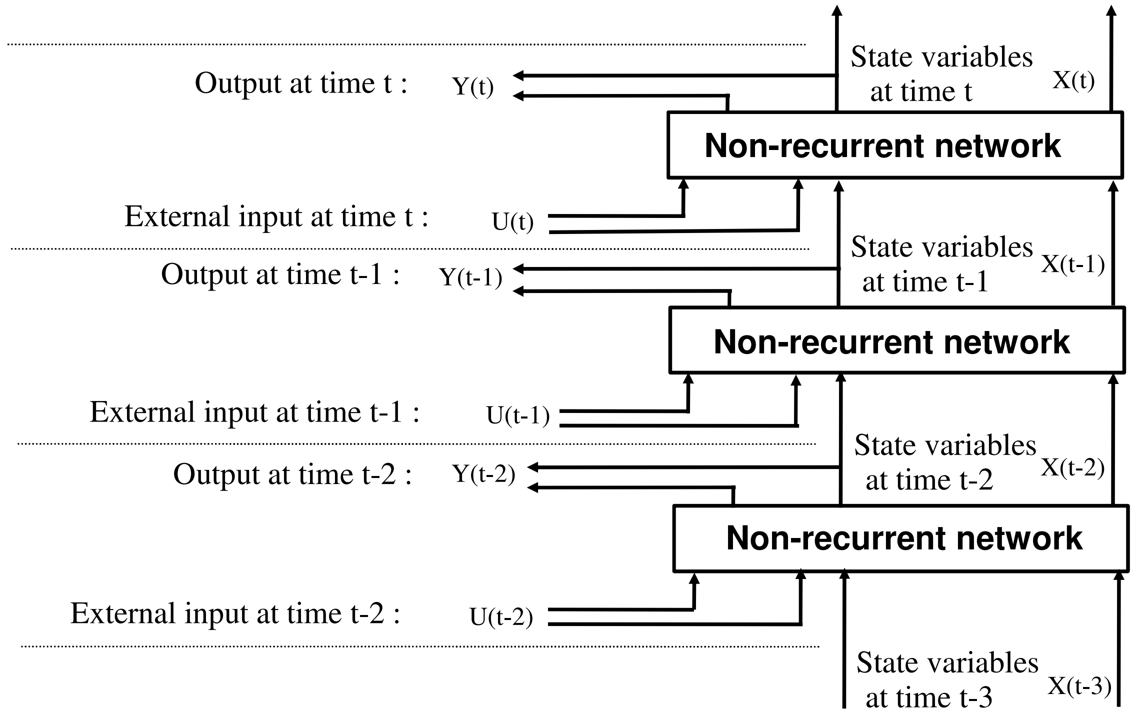


Equivalent form

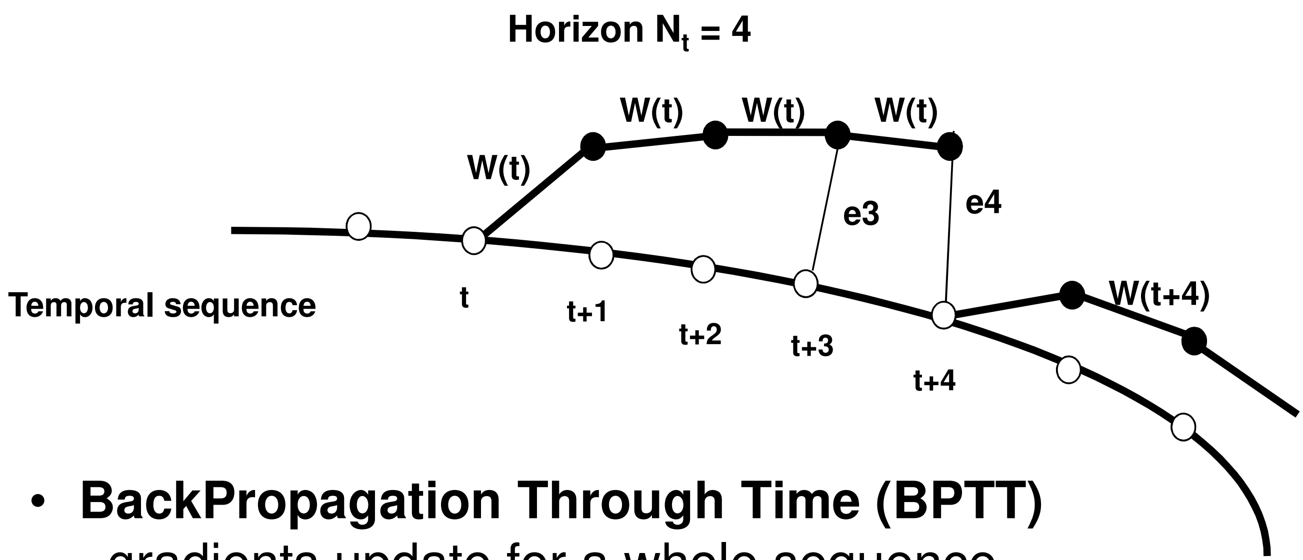
Canonical form of RNN



Time unfolding of RNN

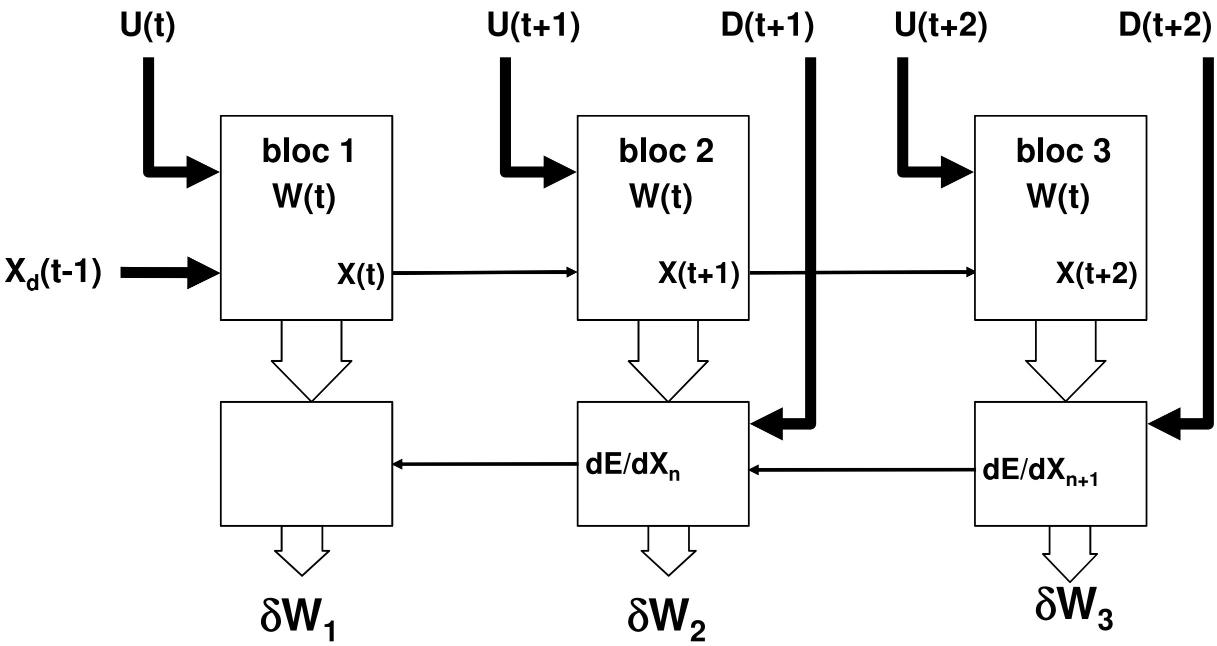


RNN training



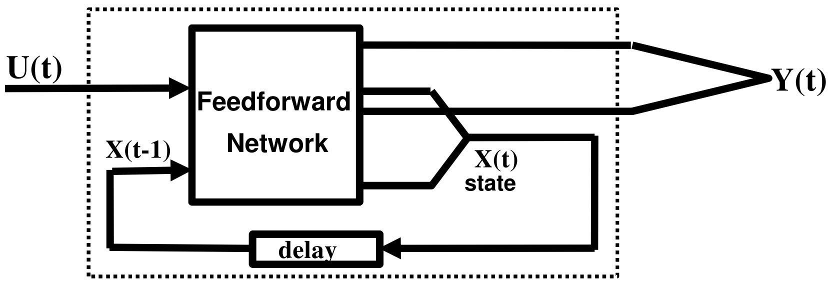
- **BackPropagation Through Time (BPTT)**
  gradients update for a whole sequence
- **Real Time Recurrent Learning (RTRL)**
  gradients update for each frame in a sequence

BackPropagation THROUGH TIME (BPTT)



$$\delta W = \delta W_1 + \delta W_2 + \delta W_3$$

BPTT algorithm



$$W(t+N_t) = W(t) - \lambda \, grad_W(E) \text{ avec } E = \Sigma_\tau (Y_\tau - D_\tau)^2$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \quad \text{and} \quad \forall t, \frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial Y_t} \frac{\partial Y_t}{\partial X_{t-1}} \frac{\partial X_{t-1}}{\partial W} \quad \text{(chain rule)}$$

$$\frac{\partial X_t}{\partial W} = \sum_{k=1}^{t-1} \frac{\partial X_t}{\partial X_{t-k}} \frac{\partial X_{t-k}}{\partial W} \qquad \frac{\partial X_t}{\partial X_{t-k}} = \prod_{j=1}^{t} \boxed{\frac{\partial X_j}{\partial X_{j-1}}}$$

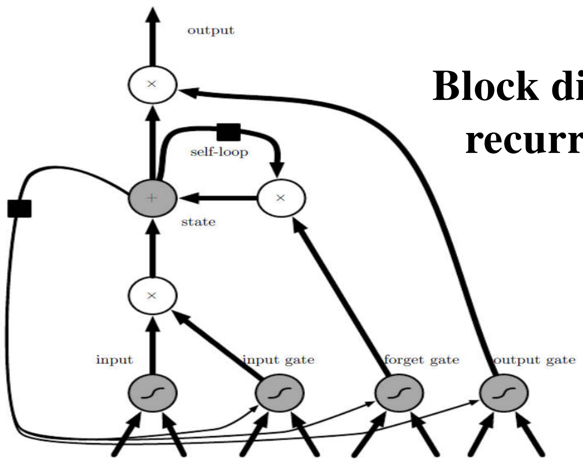
Jacobian matrix of the Feedforward net

Vanishing/exploding gradient problem

- **If eigenvalues of Jacobian matrix >1, then gradients tend to explode**
  ➔ **Learning will never converge.**

- **Conversely, if eigenvalues of Jacobian matrix <1, then gradients tend to vanish**
  ➔ **Error signals can only affect small time lags**
  ➔ **short-term memory.**

Long Short-Term Memory (LSTM)

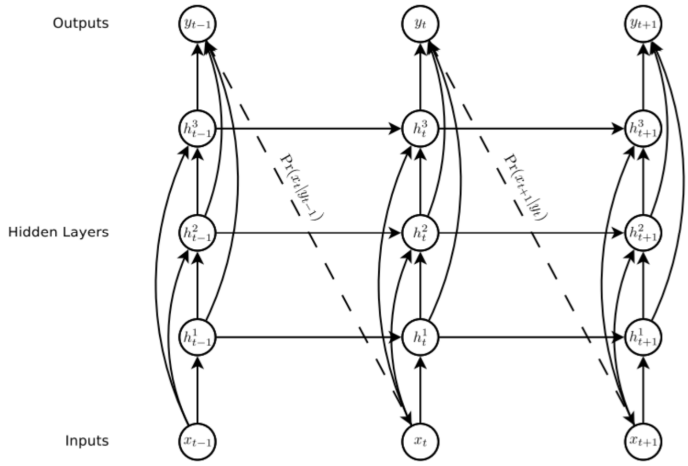


Block diagram of the LSTM recurrent network "cell"

Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

[Figure and caption taken from Deep Learning book by I. Goodfellow, Y. Bengio & A. Courville]

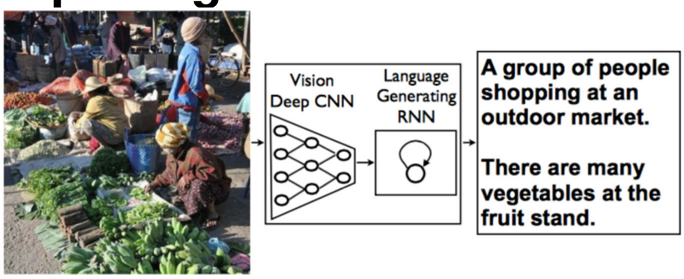
Deep RNNs



Applications of RNN/LSTM

<u>Wherever data is intrinsicly SEQUENTIAL</u>

- Speech recognition
- ***<u>Natural Language Processing (NLP)</u>***
  - Machine-Translation
  - Image caption generator



- <u>Gesture recognition</u>
- Potentially any kind of time-series!!

Summary and perspectives on Recurrent Networks

- For <u>SEQUENTIAL</u> data
  (speech, text, …, gestures, …)
- Impressive results in
  Natural Language Processing (in particular Automated Real-Time Translation)
- Training of standard RNNs can be tricky
  (vanishing gradient…)
- Increasing interest on LSTM / deep RNN

Any QUESTIONS ?