Deep Learning

Thierry Artières

Ecole Centrale Marseille

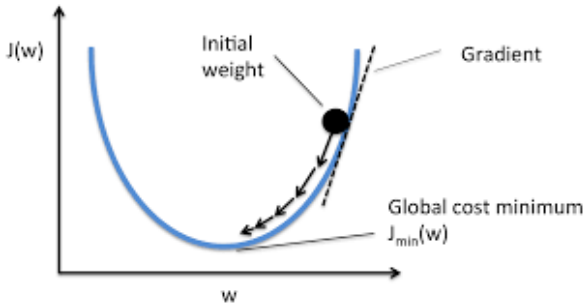
December 9, 2018

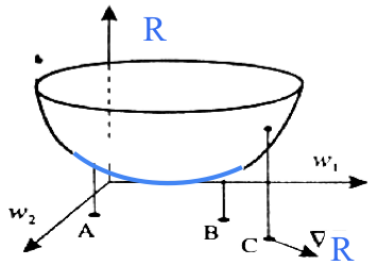
Outline

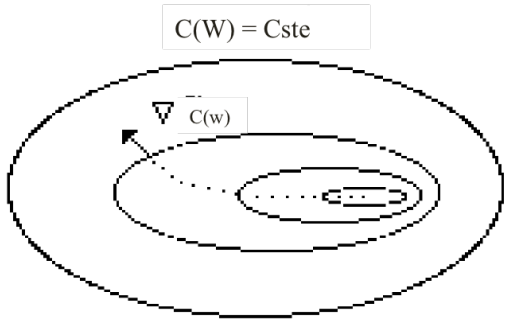
Gradient Descent Optimization

Gradient Descent Optimization

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







$\Rightarrow$ Few illustrations in these slides are taken from [LeCun et al, 1993], [Fei Fei Li lecture 6], and from *S. Ruder's blog*

Optimal learning rate and convergence speed

Second order point of view

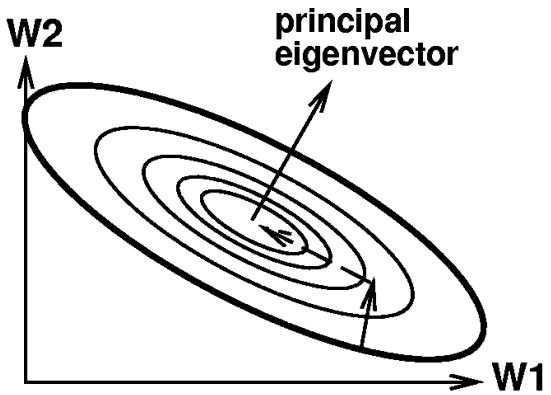
- Taylor expansion, noting $\nabla^2 C(w)$ the Hessian (a $N \times N$ matrix with N a model with parameters)

$$\nabla C(w)|_{w'} = \nabla C(w)|_w + \nabla^2 C(w)(w' - w)$$

- $\rightarrow$ optimum rule (setting $\nabla C(w)|_{w'}$ to 0):

$$w' = w - (\nabla^2 C(w))^{-1} \nabla C(w)$$

- Optimal move not in the direction of the gradient
- In Order 1 Gradient descent the optimal the optimal value of ϵ depends on eigen values of the Hessian $\nabla^2 C(w)$



[Lecun et al, 93]

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.

Optimization routines

Many SGD variants popular in DL

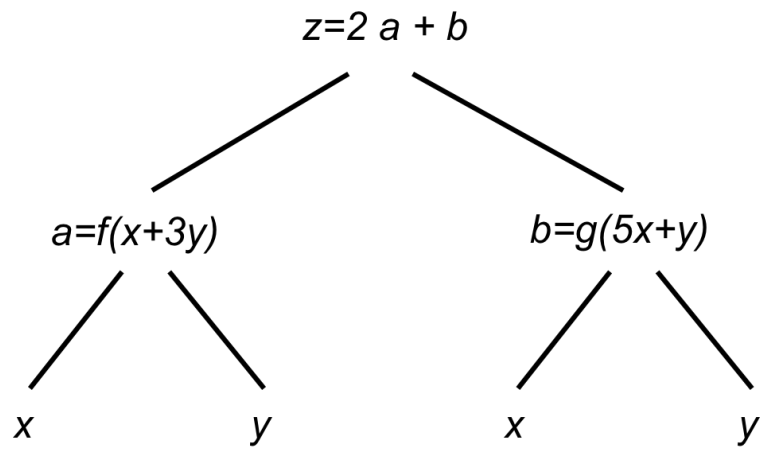
- SGD with Momentum
- Nesterov accelerated gradient
- Adragrad
- Adadelta
- RmsProp
- ...

Gradient Computation: Chain rule

Gradient of a function

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

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



Equivalent computation with the Chain rule

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

$\Rightarrow z = 2 \times a + 6 \times b$

$$\Rightarrow \frac{\partial z}{\partial x} = \frac{\partial z}{\partial a} \times \frac{\partial a}{\partial x} + \frac{\partial z}{\partial b} \times \frac{\partial b}{\partial x}$$

With:

$$\frac{\partial z}{\partial a} = 2 \text{ and } \frac{\partial z}{\partial b} = 6$$

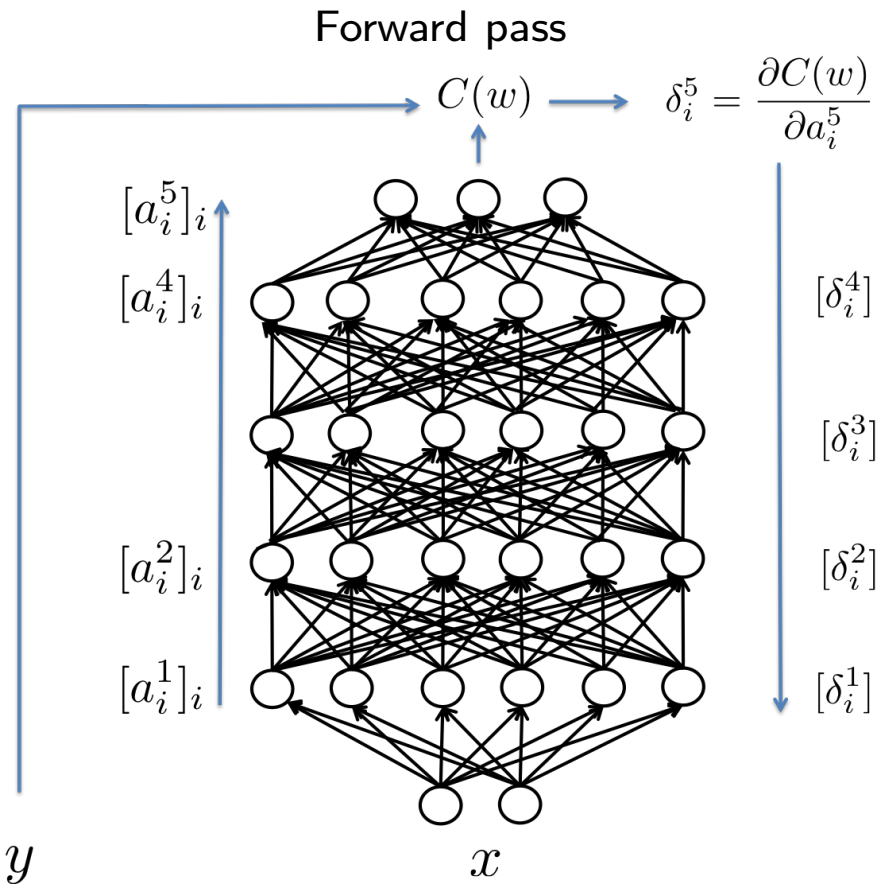
$$\frac{\partial a}{\partial x} = f'(a \times x + 3 \times y)$$

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

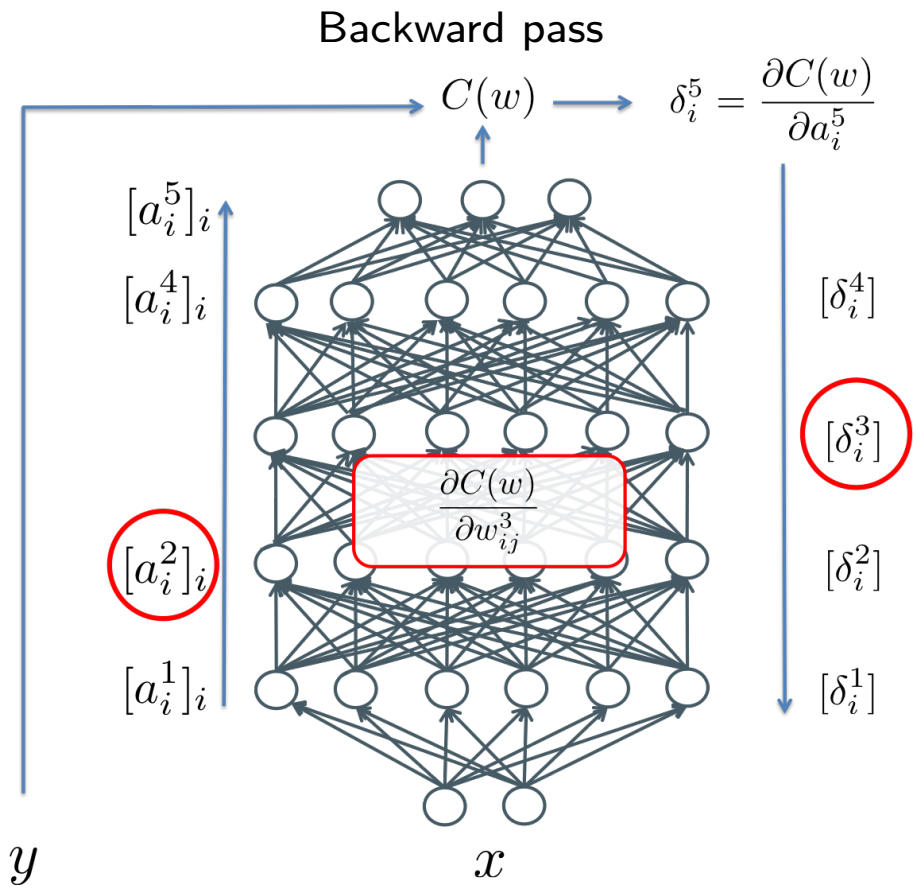
$$\frac{\partial a}{\partial y} = 3 \times f'(a \times x + 3 \times y)$$

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

BackPropagation



BackPropagation



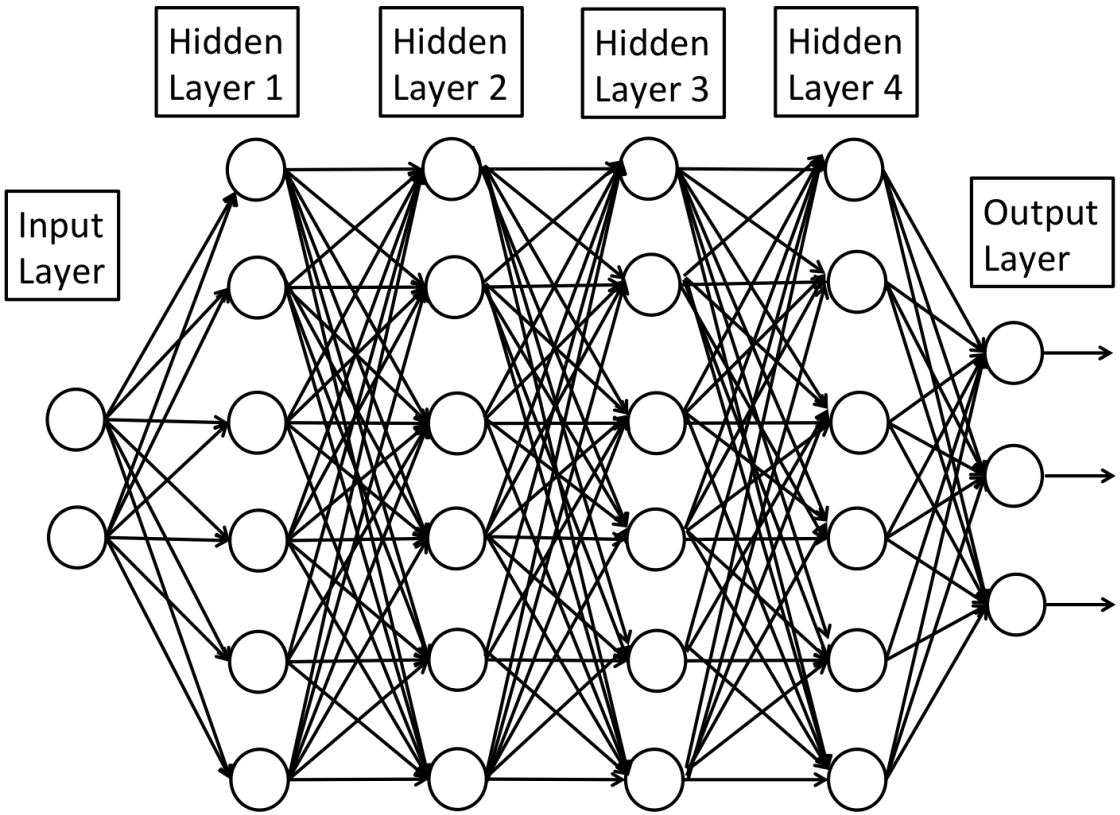
Guiding the learning through regularization

Regularization

- Constraints on weights (L1 or L2)
- Constraints on activities (of neurons in a hidden layer) $\rightarrow$ induces sparsity
  - L1 or L2
  - Mean activity constraint (Sparse autoencoders, [Ng et al.])
  - Sparsity constraint (in a layer and/or in a batch)
  - Winner take all like strategies
- Disturb learning for avoiding learning by heart the training set
  - Noisy inputs (e.g. Denoising Autoencoder, link to L2 regularization)
  - Noisy labels
- Early stopping

Outline

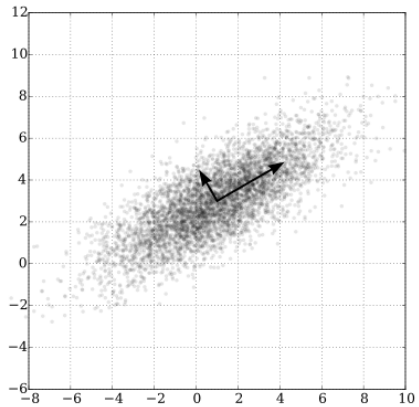
Dense architecture



Autoencoders

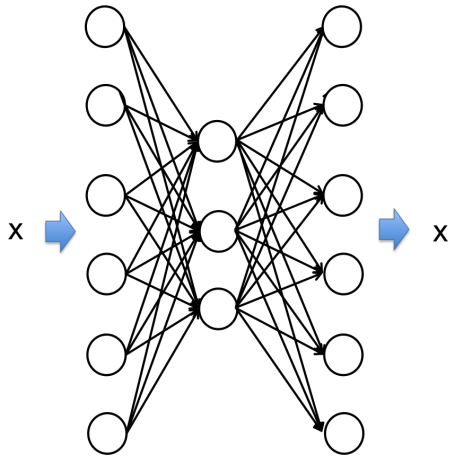
Principal Component Analysis

- Unsupervised standard (Linear) Data Analysis technique
  - Visualization, dimension reduction
- Aims at finding principal axes of a dataset



NN with Diabolo shape

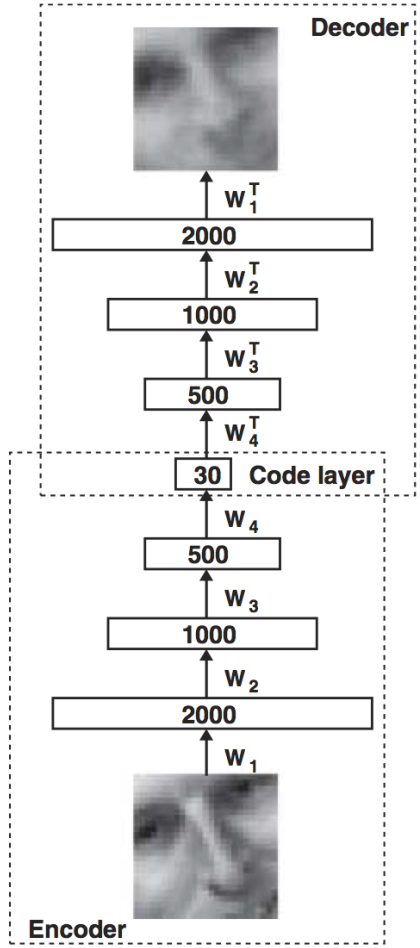
- Reconstruct the input at the output via an intermediate (small) layer
- Unsupervised learning
- Non linear projection, distributed representation
- Hidden layer may be larger than input/output layers



Deep autoencoders

Deep NN with Diabolo shape

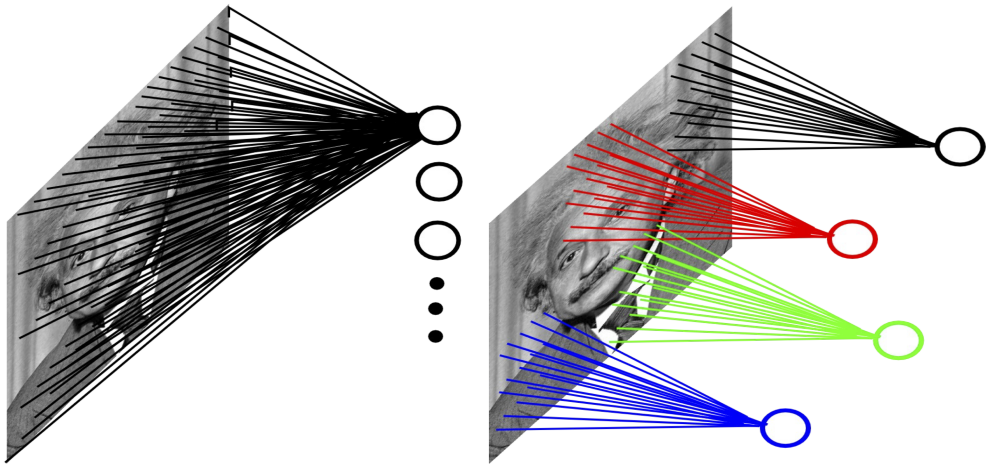
- Extension of autoencoders (figure [Hinton et al., Nature 2006])
- Pioneer work that started the Deep Learning wave



Convolutional architectures

Convolutional layers

- Exploit a structure in the data
  - Images : spatial structure
  - Texts, audio ; temporal structure
  - videos : spatio-temporal structure
- Use shared weigths

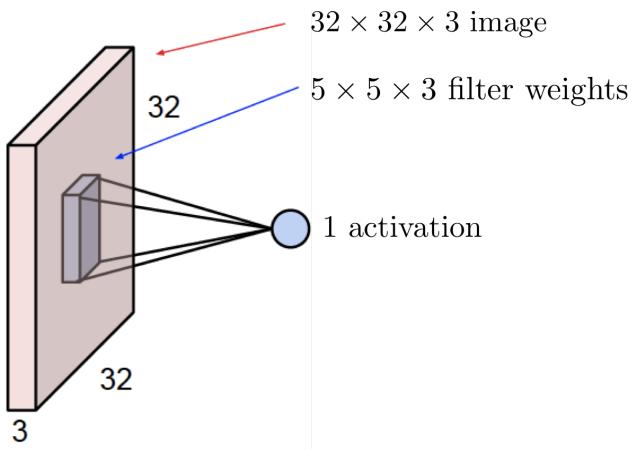


Dense vs. Locally connected

[LeCun and Ranzato Tutorial, DL, 2015]

Convolution layer

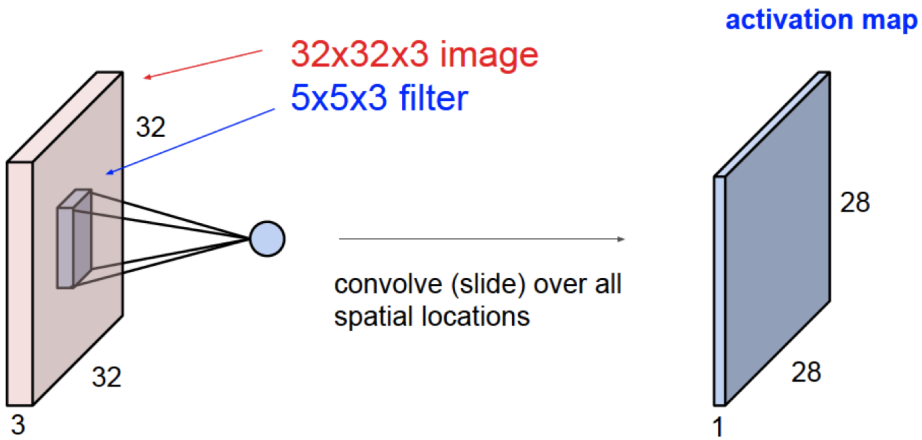
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

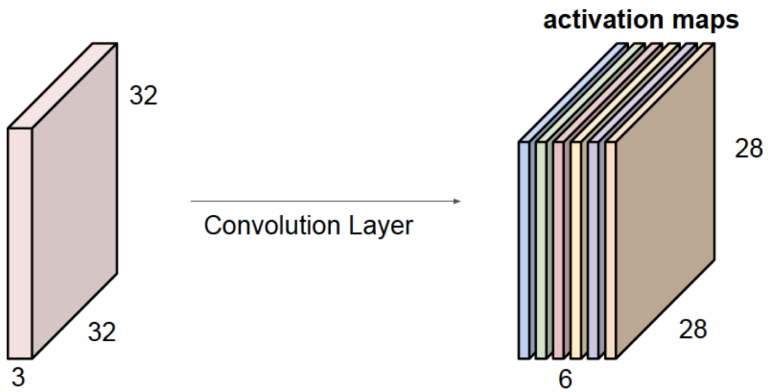
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

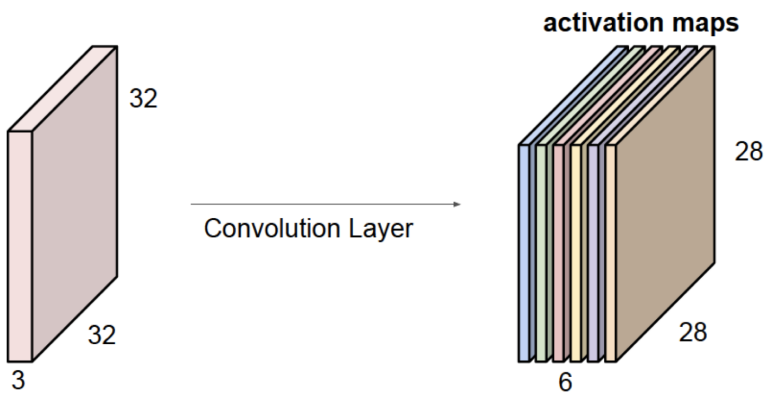
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

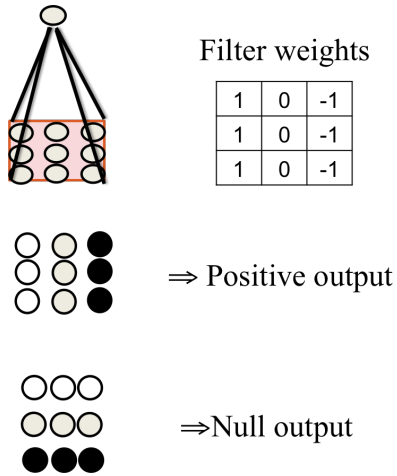
Convolution layer

Convolution of multiple inputs with several small filter yields several activation maps



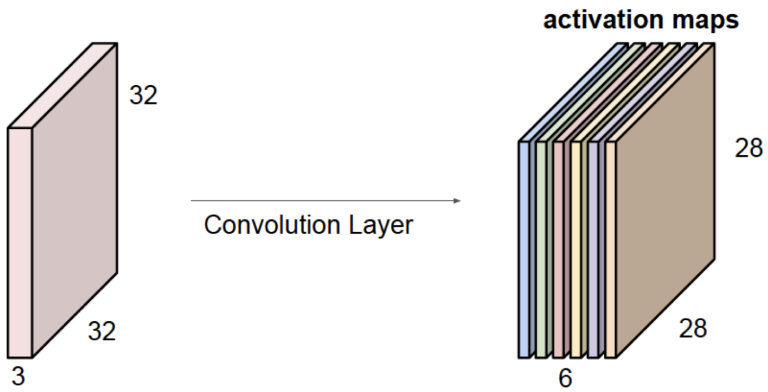
[From Fei Feil Li slides]

Example of a filter



Convolution layer

Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

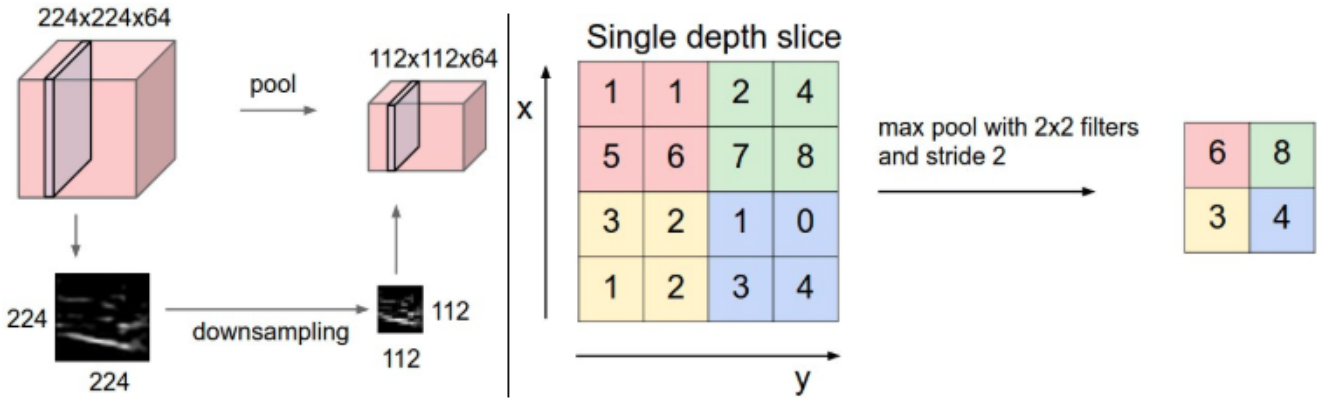
Very few free parameters but intensive computation

- Input : N_m input maps of size $H \times W$
- Output N_f filters with filter size $= h \times w$
- Parameters : $h \times w \times N_m \times N_f$
- Fwd computation :
  $\approx H \times W \times N_f \times N_m \times h \times w$
- For instance (very small case) :
  - From 3 32×32 input maps $\rightarrow$ 6 filters with filter size 3×3
  - $3 \times 3 \times 3 \times 6 = 48$ parameters
  - 165 888 operations

Convolution layer

Aggregation layer

- Subsampling layer (one per activation map) with aggregation operator
- Max pooling $\rightarrow$ brings invariance and robustness



Complexity

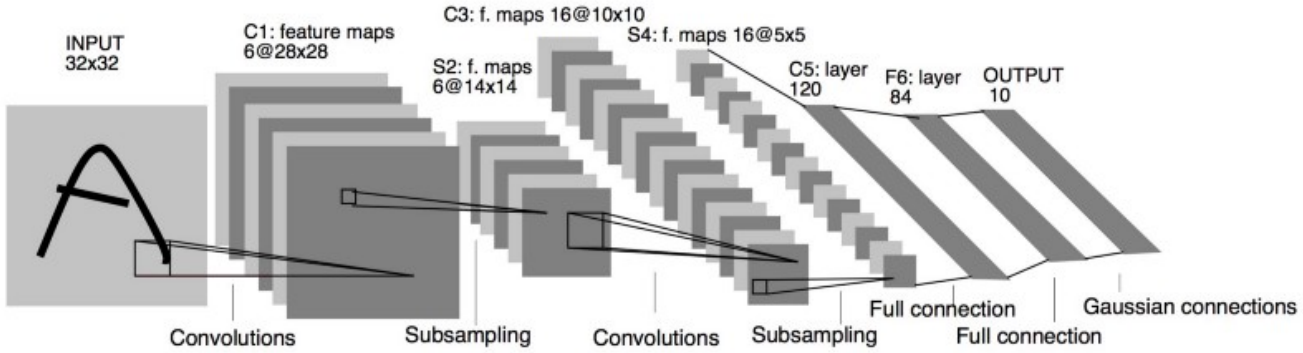
- No parameters
- Moderate computation

Old and new convolutional architectures

Convolution architectures

- Most often a mix of (convolutional + pooling) layers followed by dense layers
- Most computation effort are in propagating through convolution layers
- Most parameters are in final fully connected layers

LeNet [LeCun and al., 1997]

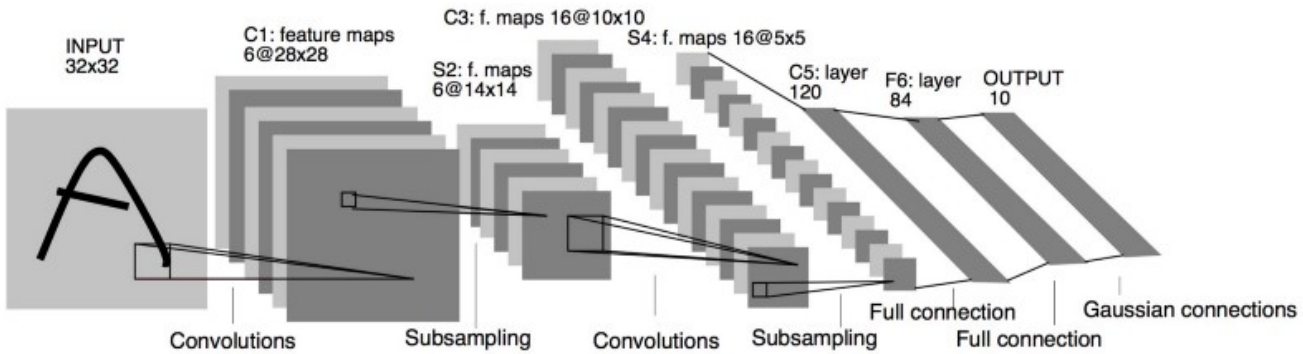


Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]

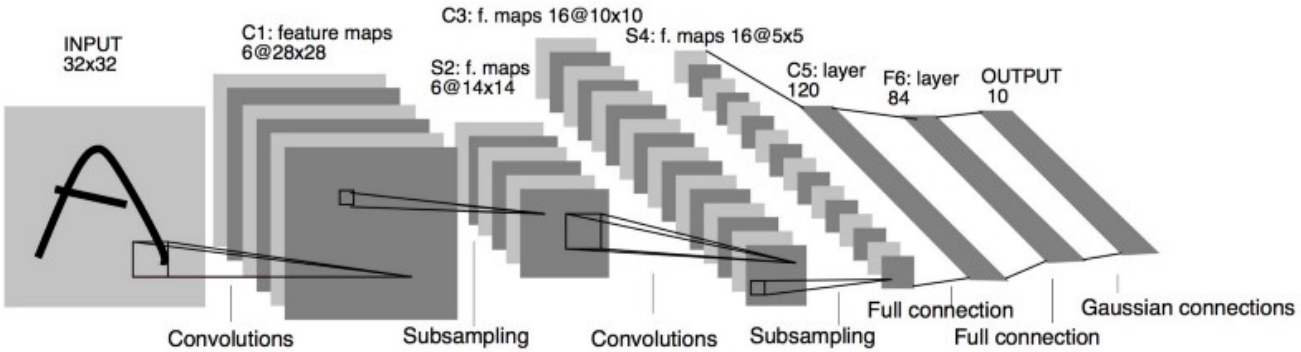


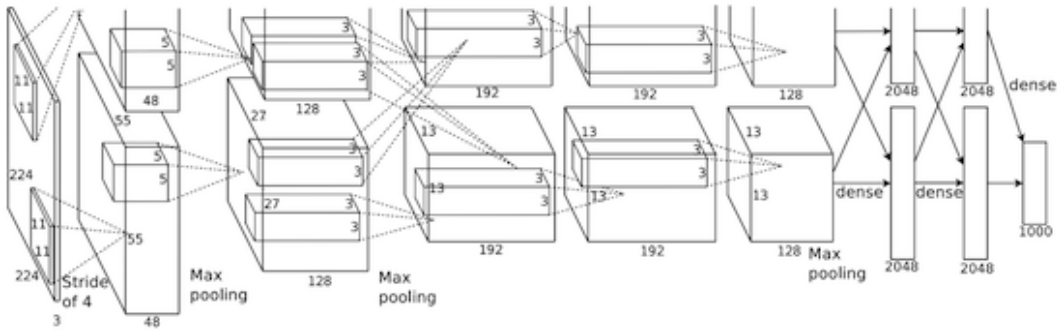
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]





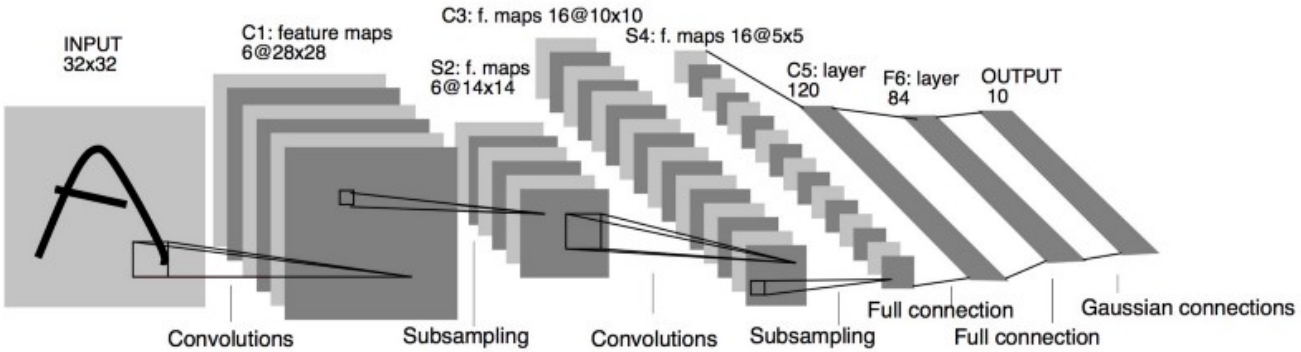
AlexNet [Krizhevsky and al., 2012]

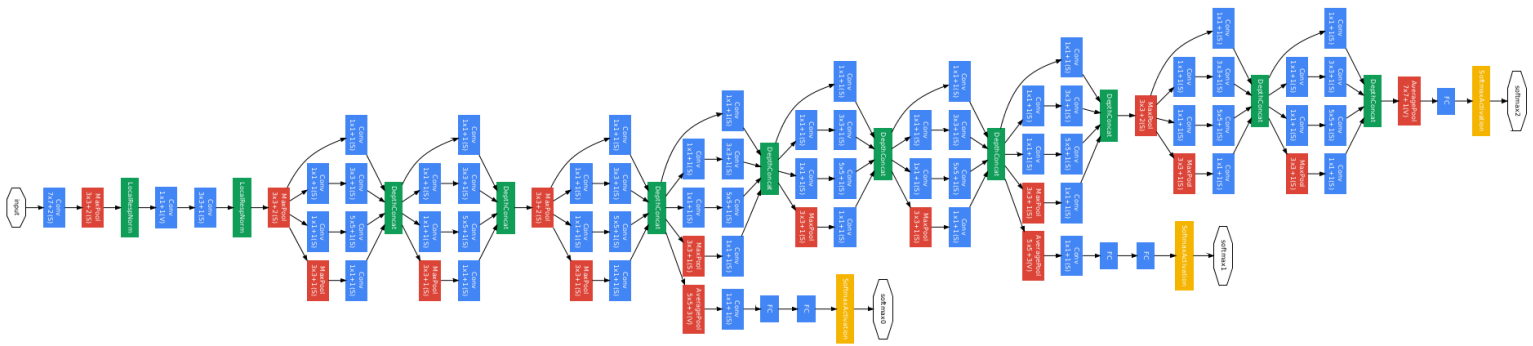
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]



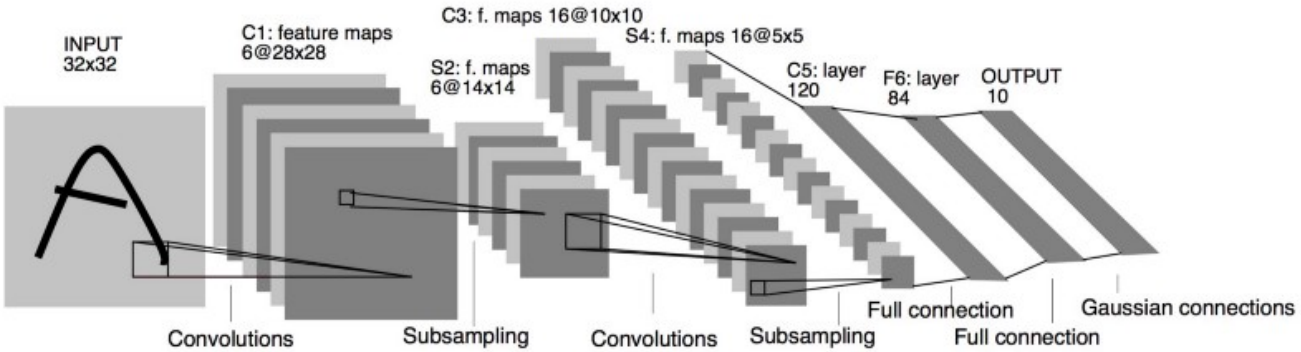


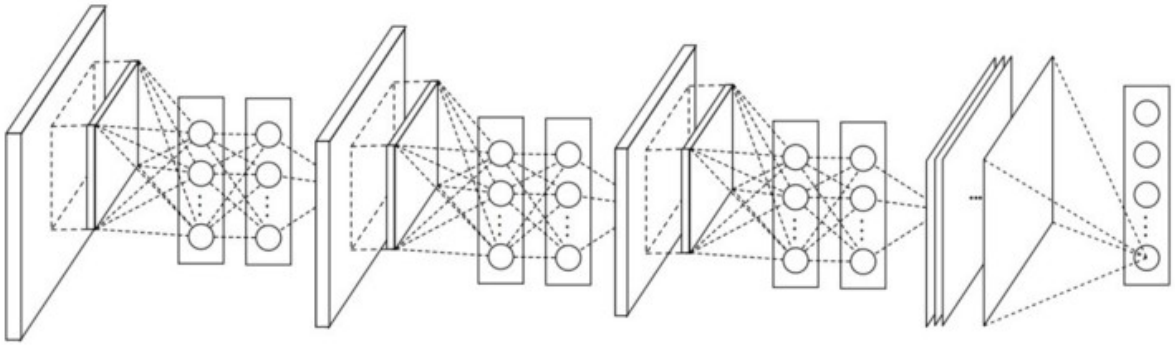
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]

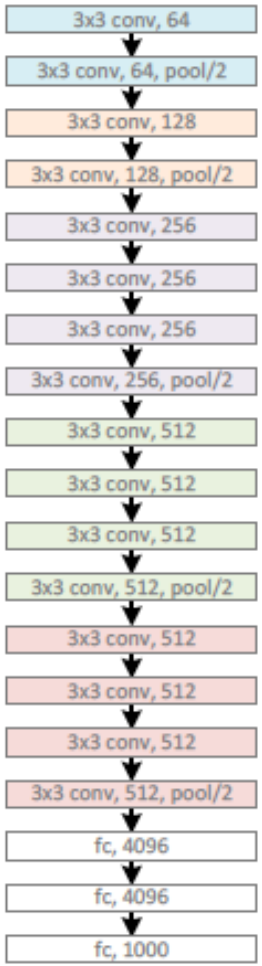




Old and new convolutional architectures

VGG idea

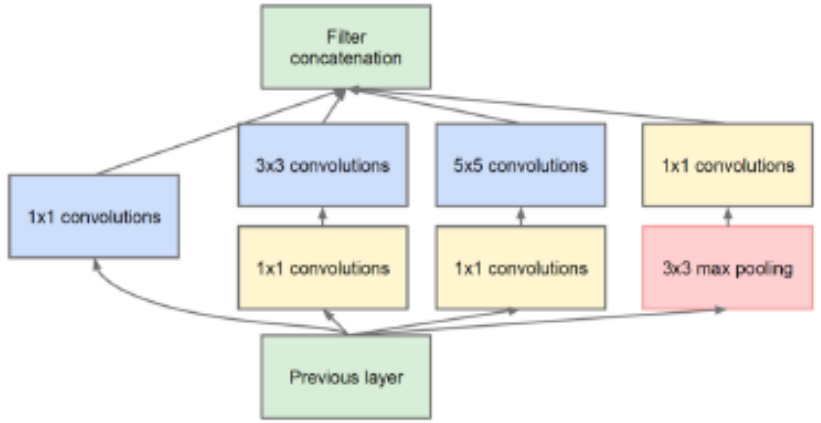
- Modular design
  - 3x3 conv as basis
  - Stack the same module
  - Same computation for each module (1/2 spatial size $=>$ 2x filters)

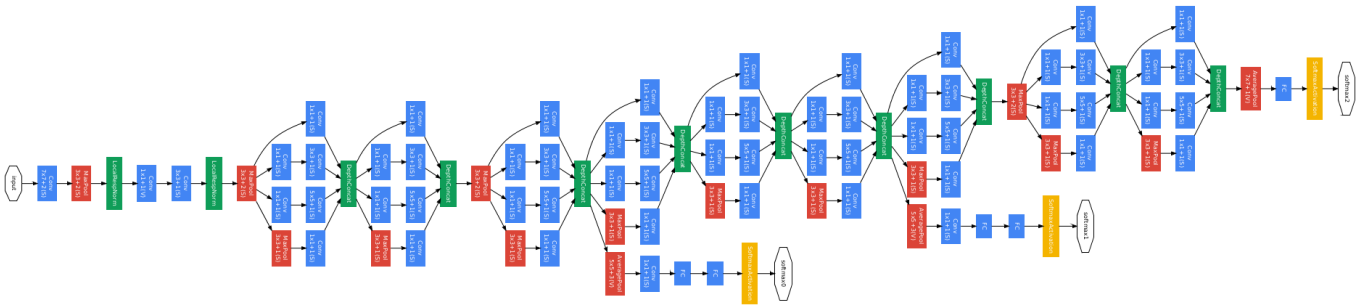


Old and new convolutional architectures

Inception idea (GoogleNet)

- Inception modules
  - Multiple branchs (1x1, 3x3, 5x5, pool)
  - Shortcuts (stand alone 1x1, merged by concat)
  - Bottleneck (reduce dim by 1x1 before expensive 3x3/5x5 convs)

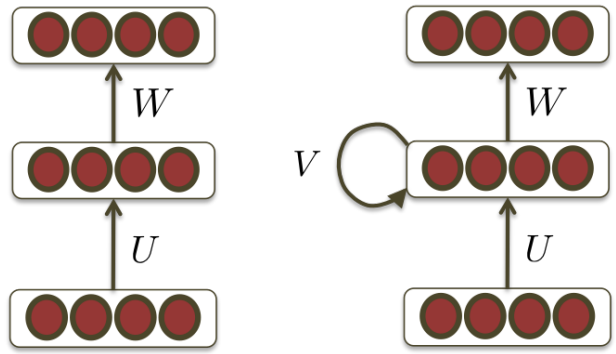




Recurrent NNs

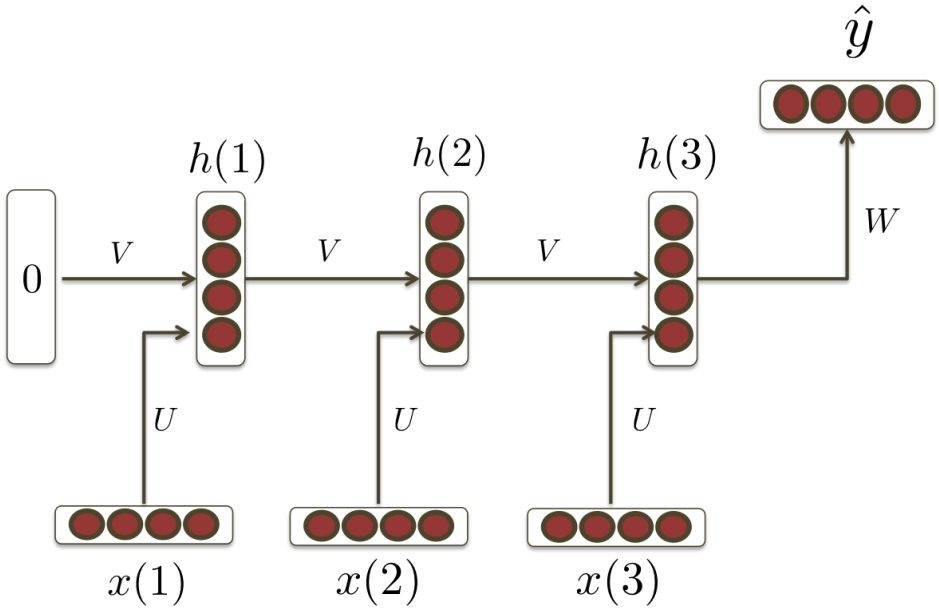
RNNs in general

- A recurrent neural network is a NN with cycles in its connections
- Today RNNs are specific recurrent architectures. Not all architectures work well..
- Shared parameters in time (whereas in space in Convolutional architectures)



MLP vs. RNN

Inference and learning through unfolding the RNN



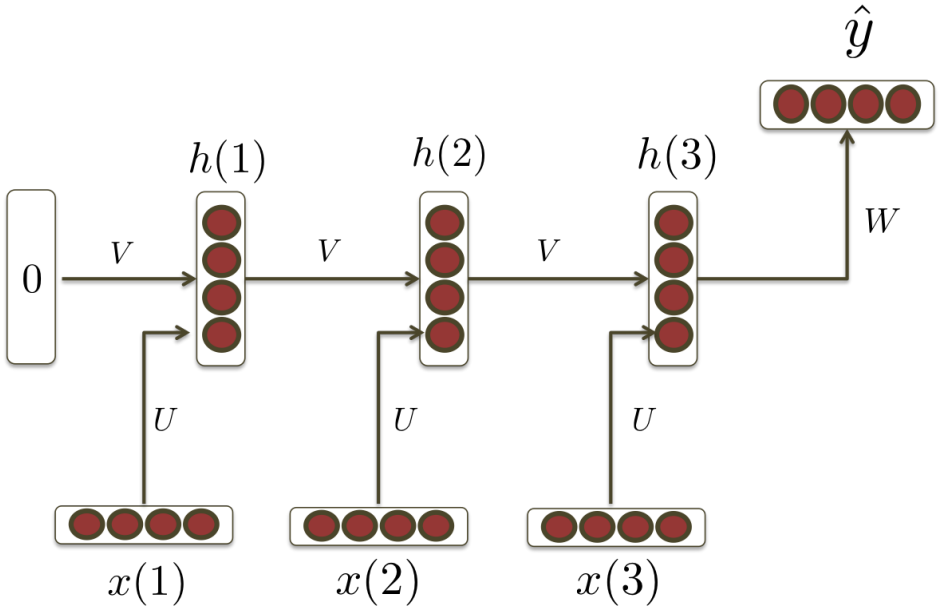
Inference: Forward propagation in the FeedForward unfolded RNN

- Start with null state $h(0) = 0$
- Iterate

$$h(t) = g(V \times h(t-1) + U \times x(t)$$
$$y(t) = g(W \times h(t))$$

- $\Rightarrow$ The final state $h(T)$ resumes the whole input

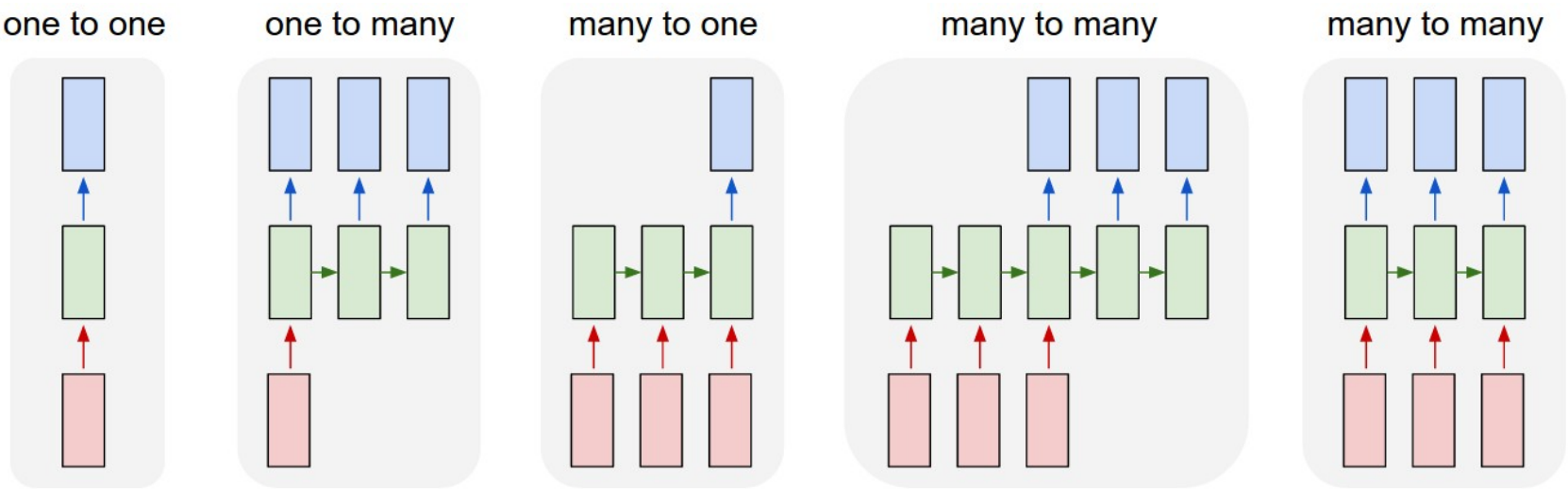
Inference and learning through unfolding the RNN



Learning: Back-propagation in the FeedForward unfolded RNN

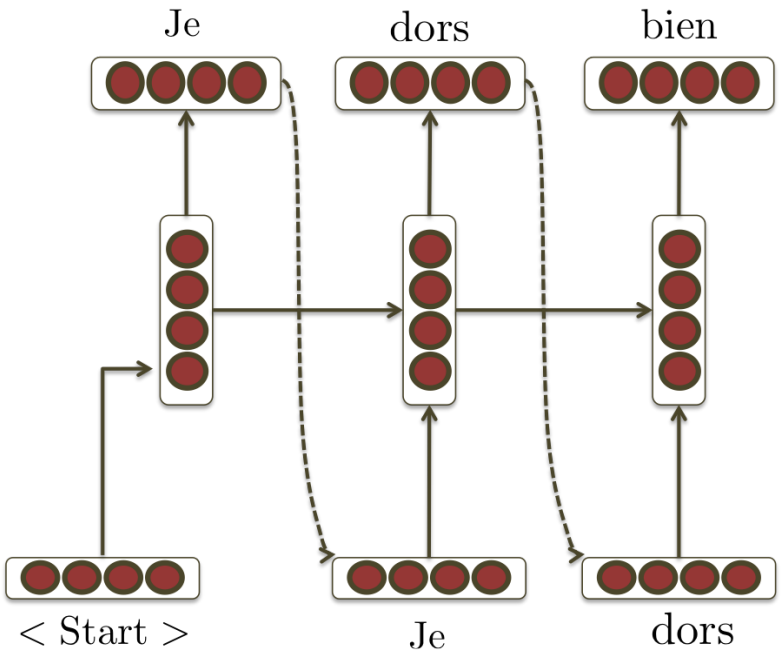
- Unfold the model
- Backpropagate the gradient in the whole network
- Sum the gradient corresponding to all shared parameters and unshared parameters (possibly the last layer)
- Apply Gradient Optimization Update rule on all parameters

Various settings



- One to One : MLP, CNN ...
- One to Many : Generation of a sequential process (speech, handwriting ...)
- Many to one : Sequence classification (e.g. activity recognition)
- Asynchronous Many to many : Machine Translation
- Synchronous Many to Many : POS tagging, Speech recognition...

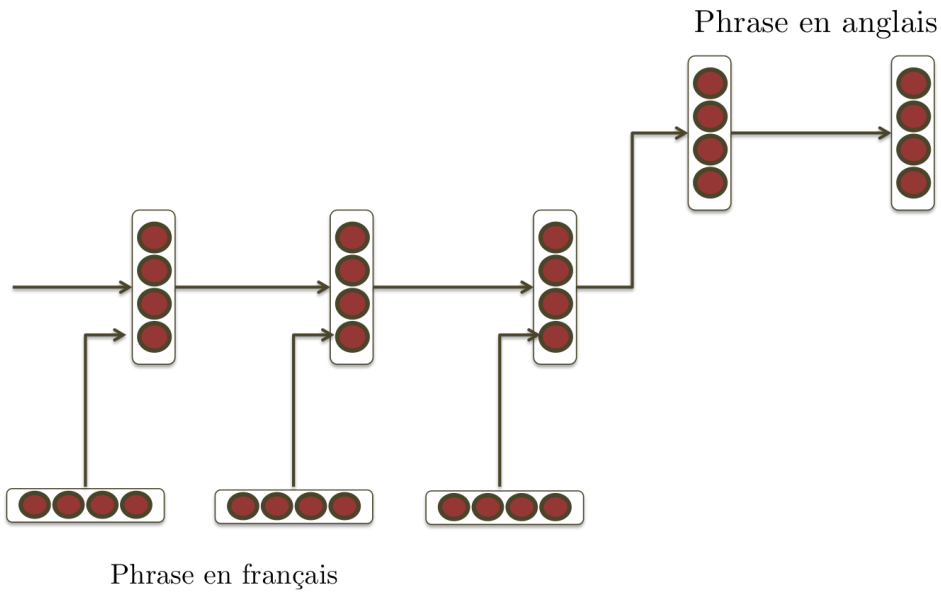
One To ManyText example



Text generation

- Example of a generation model as a one to Many model

Many to many example



Machine translation

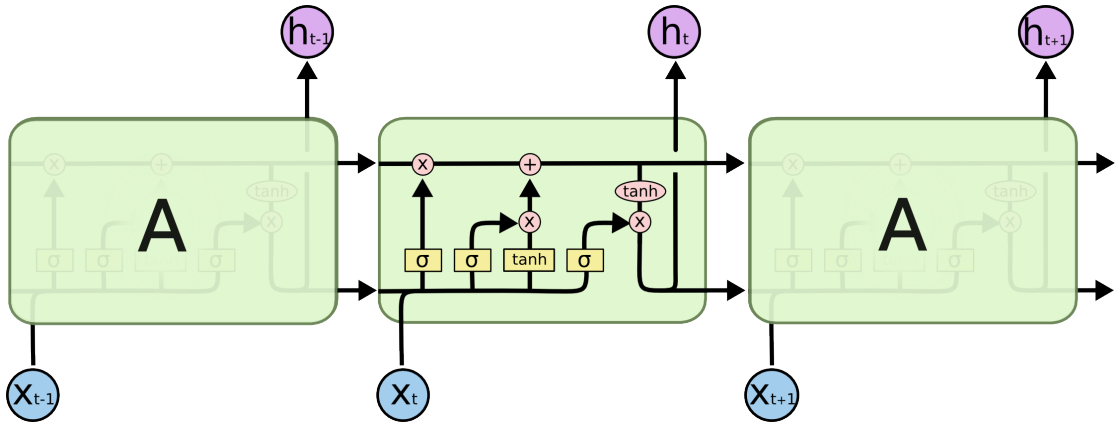
- Example of a translation model as a asynchronous Many to Many model
- The nature of language and of complex grammatical forms require to first "understand" the sentence, encoding it in a small dimensional hidden space, then to reconstruct the sentence in the target language.

LSTMs and RNNs

New units for RNNs

- Motivation:
  - Optimization problems in Recurrent Neural Networks (gradient explosion / vanishing)
  - Difficulty to capture long term dependencies
- New types of hidden cells
  - Long Short Term Memory (LSTM) [Hochreichetr 98]
  - Gated Recurrent Unit (GRU) [Cho and al., 2014]

LSTM units



Motivation

- Units that include few gates (*forget*, *input*, *output*) which allow to :
  - Stop capitilizing in the state the information about the past
  - Decide if it is worth using the information in the new input
- Depending on the input and on previous state
  - Reset the state, Update the state, Copy previous state
  - Ignore new input or fully use it to compute a new state

LSTM units

Notations

- Cell state c_t
- Forget gate f_t
- Input gate i_t
- Output o_t
- Hidden state to propagate to upper layers h_t

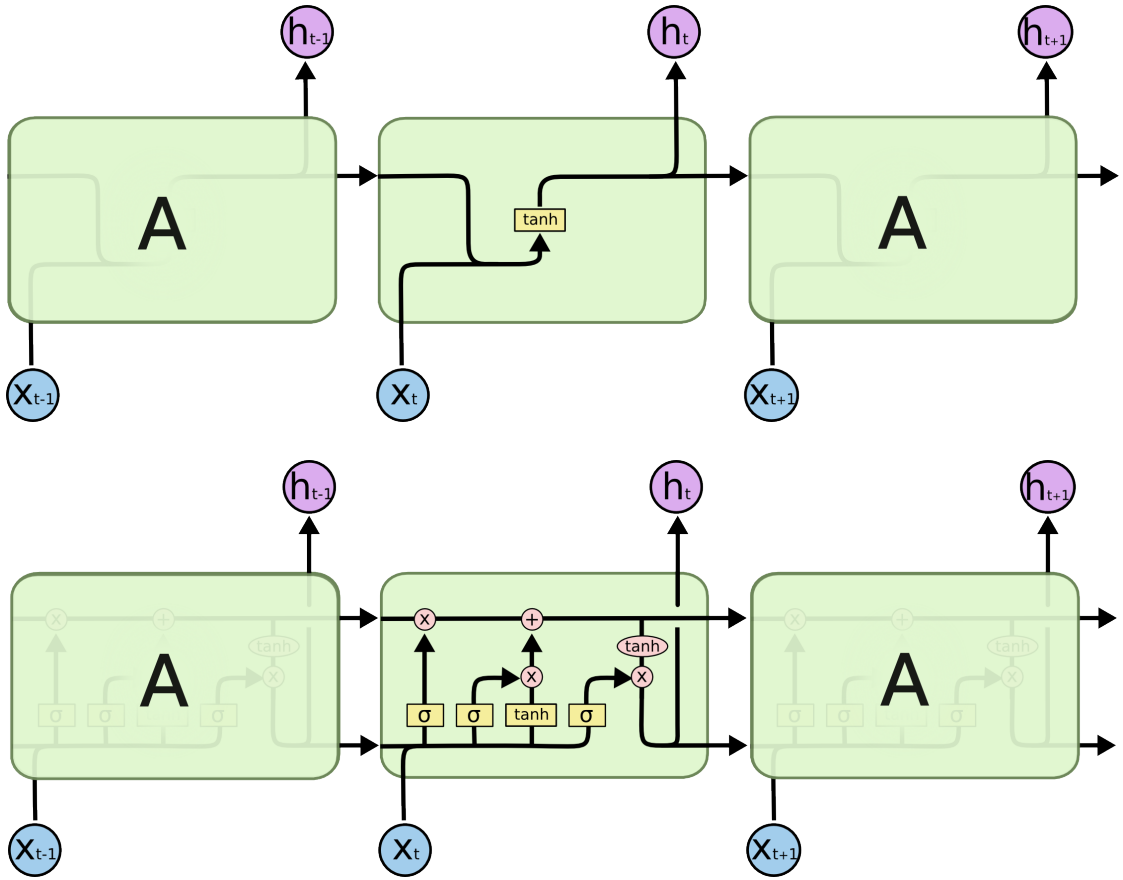
How does it work ? in words...

- Formulas

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot tanh(c_t)$$

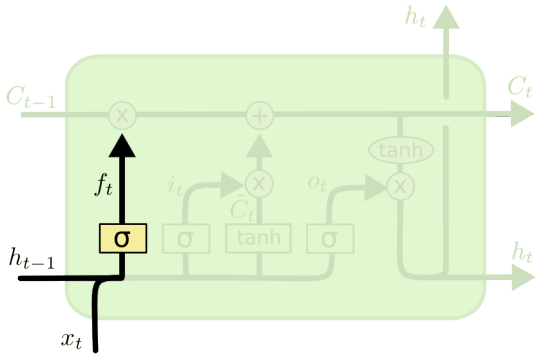
- Interpretation
  - If $f_t == 1$ and $i_t == 0$ use previous cell state
  - If $f_t == 0$ and $i_t == 1$ ignore previous cell sate
  - If $o_t == 1$ output ois set to cell sate
  - If $o_t == 0$ output is set to 0

LSTM units

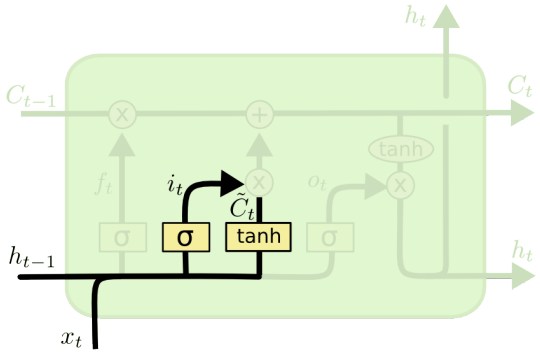


- LSTM vs traditional RNN cells

LSTM units



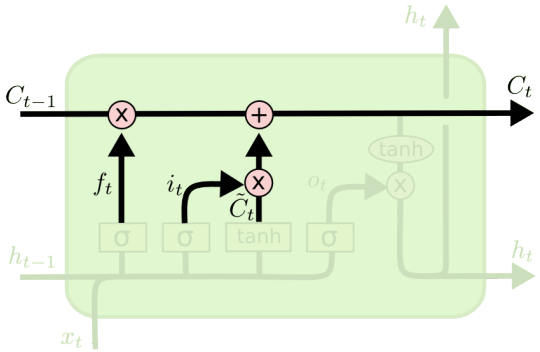
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



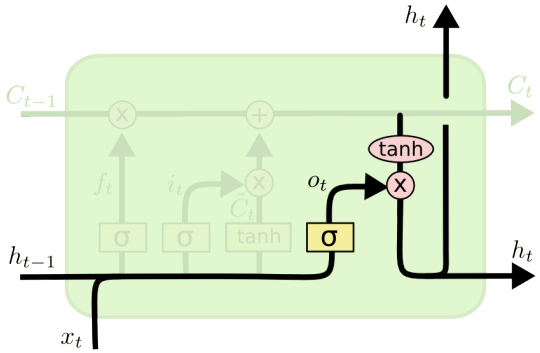
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- Forget gate f_t
- Input gate i_t
- Alternative cell state $\tilde{c}_t$

LSTM units



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



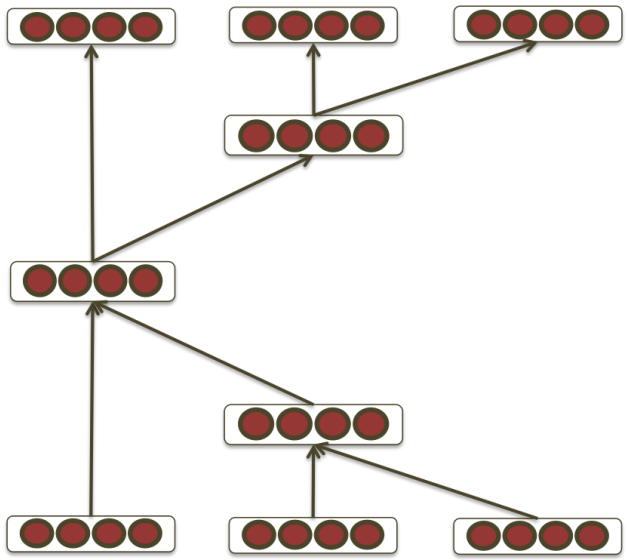
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Cell state c_t
- Output o_t
- Hidden state to propagate h_t

Recursive models

Principle

- Allows dealing with other structured data such as trees
- The model may still be unfolded and gradient may easily be computed
- Used to compute a representation using data structure (e.g. text with parse tree structure)

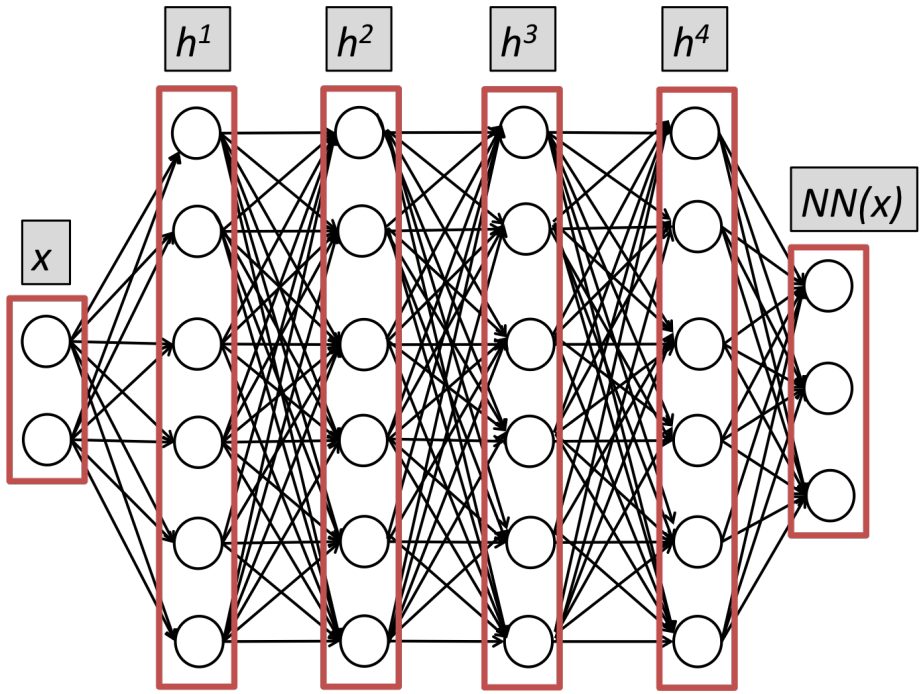


Outline

Deep Learning = Representation Learning

Hierarchy of representation spaces by successive hidden layers

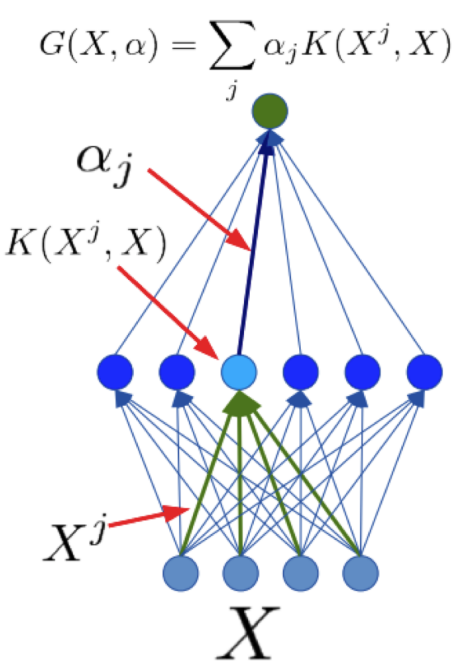
$$h^i(x) = g(W^i \times h^{i-1}(x))$$



Shallow vs deep models

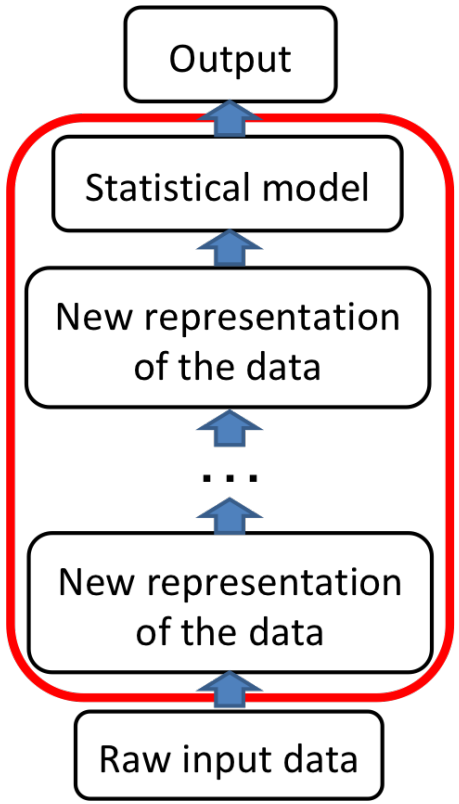
From [LeCun tutorial Statlearn]

- Neural Networks with one hidden layer are shallow models
- SVMs are shallow models



DL

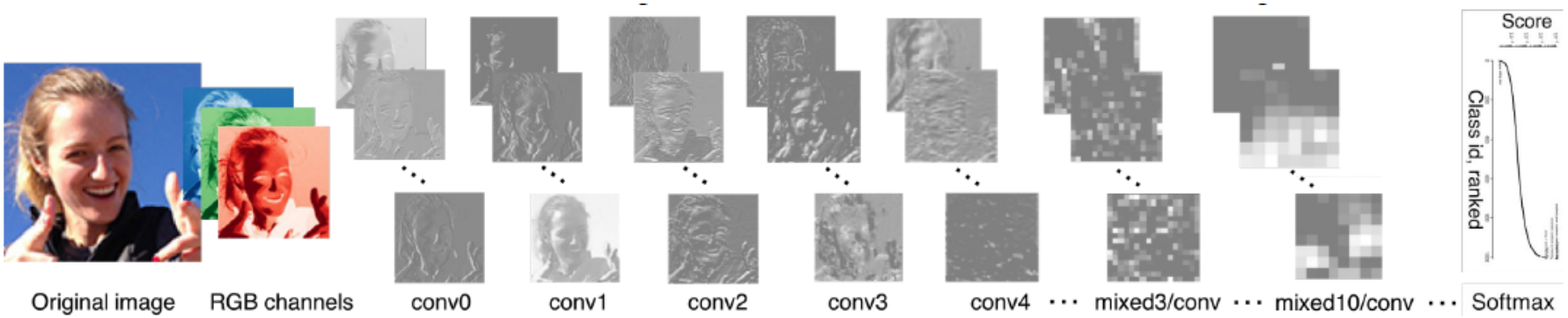
- Joint learning of a hierarchy of representations and of a prediction model



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

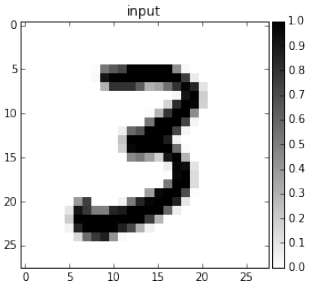


[From Taigman et al., 2014]

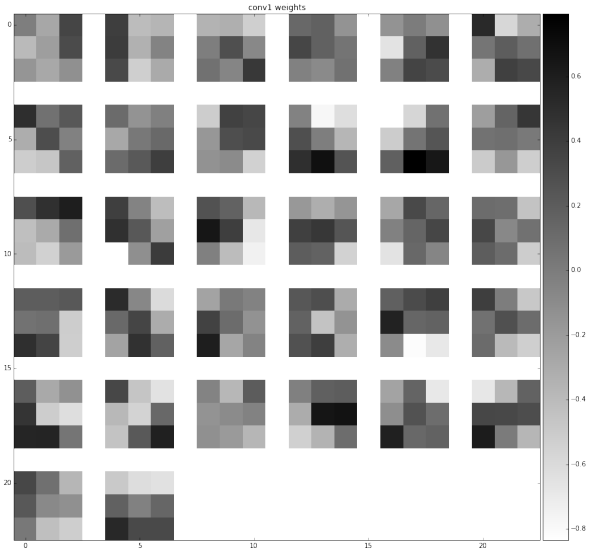
Visualizing filters and activations (primary understanding of NNs)

Mnist (toy) dataset

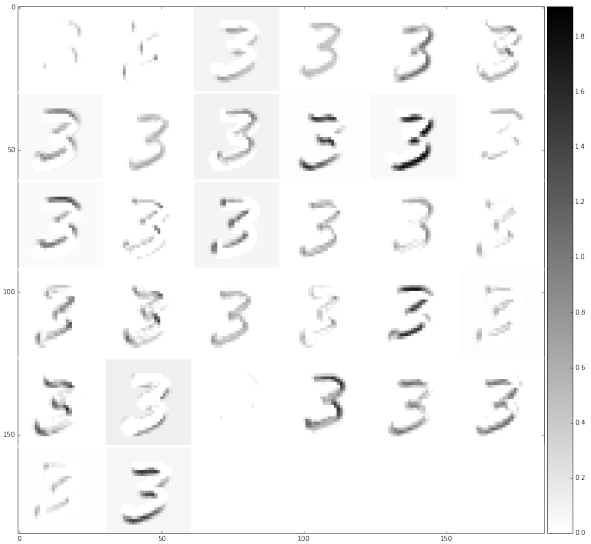
- Low resolution handwritten digit images



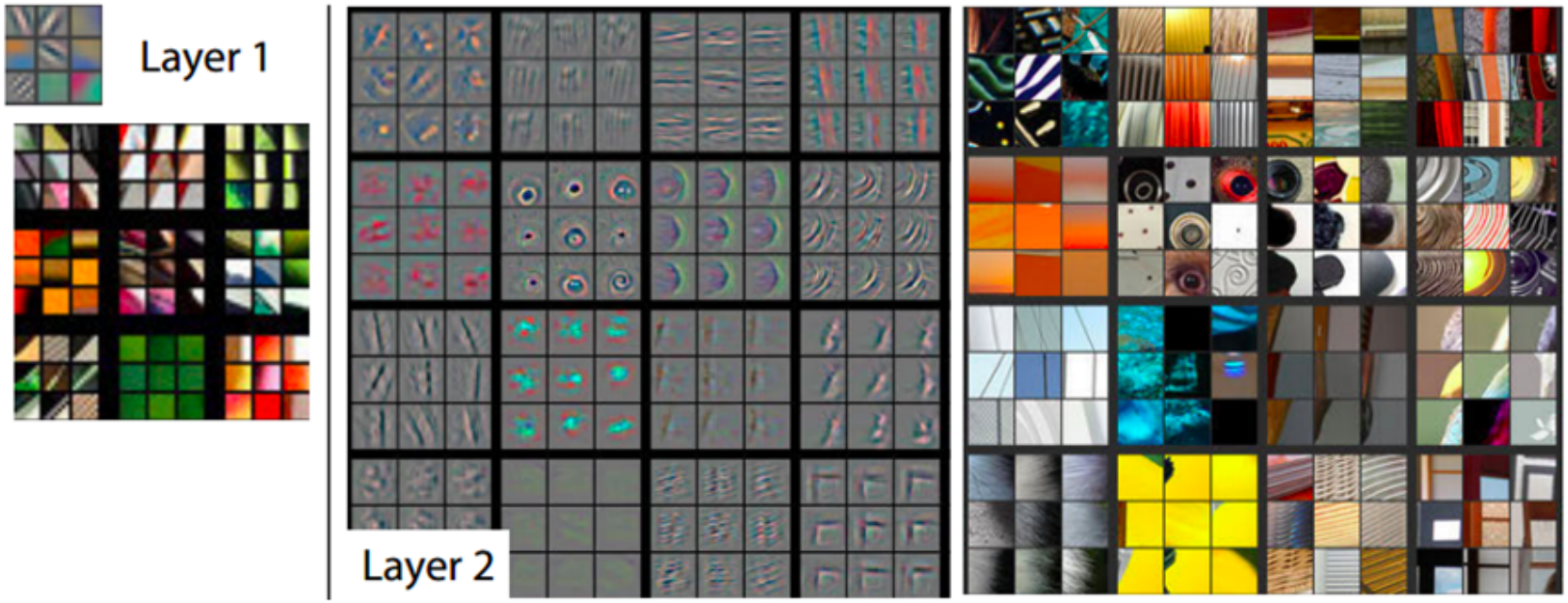
Weights of first Convolutional layer (32 maps)



Outputs of first Convolutional layer for above input



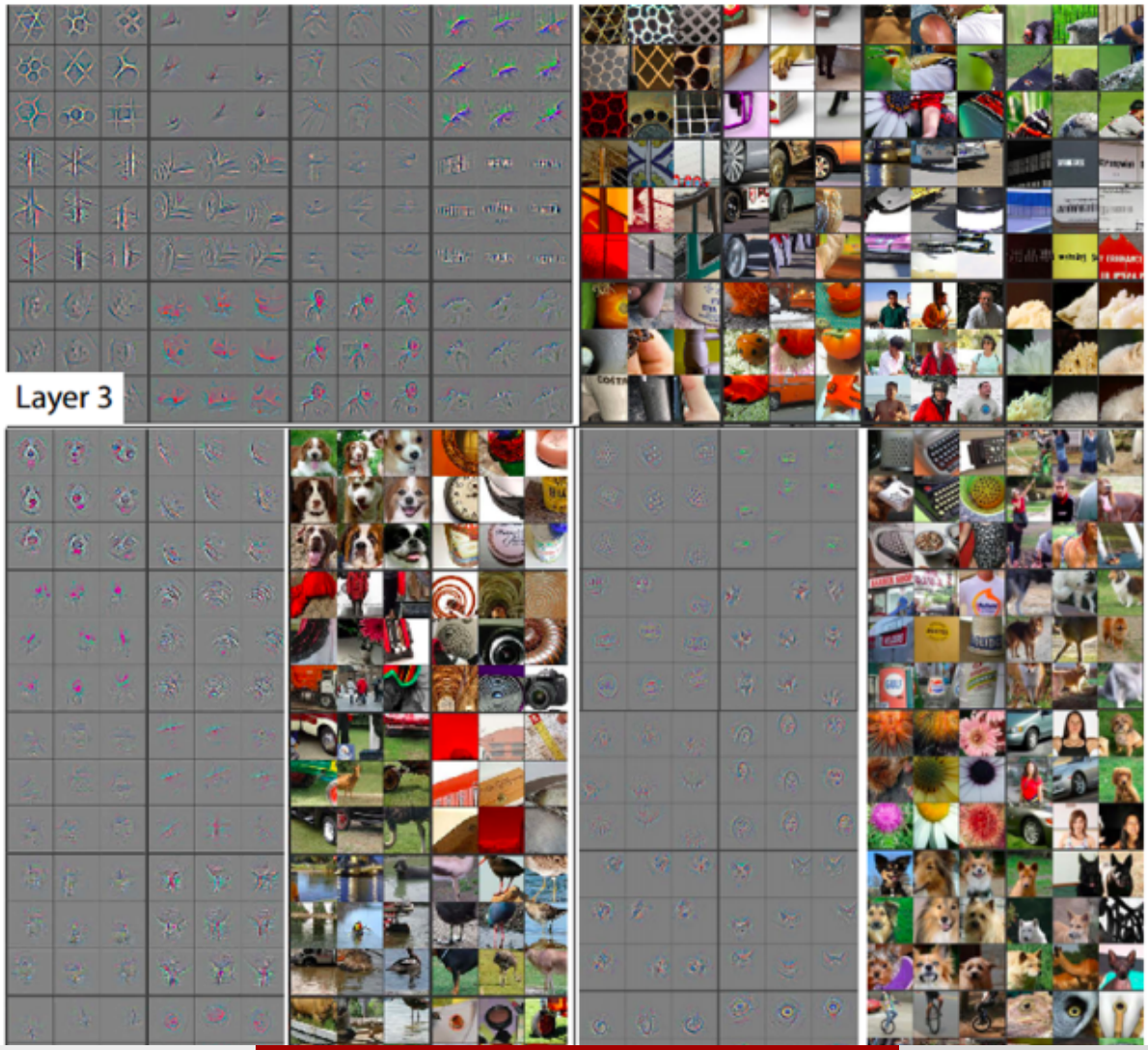
Visualizing filters and activations



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

[From Zeiler et Fergus]

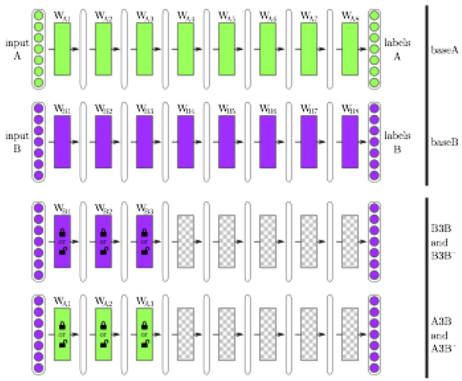
Visualizing filters and activations



Genericity of representations [Yozinski and al., 2014]

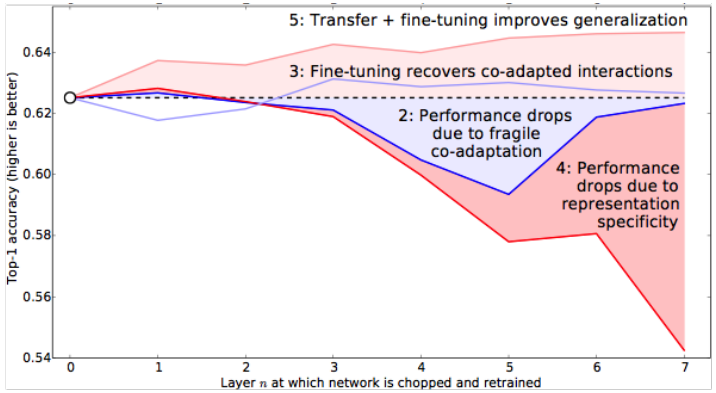
Experiments on two similar tasks

- Two DNN : Green one learned on Task A - Blue on Task B
- Reuse DNNA for Task B (and vice versa)
- Study the effect of reusing a DNN up to layer number i ...



Main results

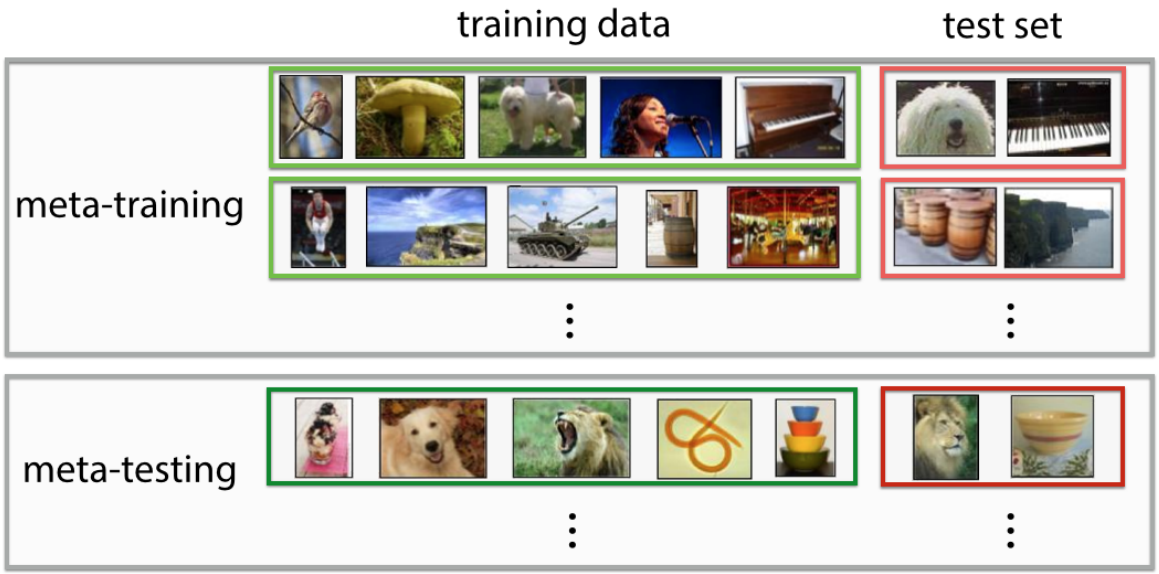
- Better to reuse DNNA and fine tune on Task B
- Lower layers learn transferable features while higher don't



Learning with few samples

Few shot learning (and zero shot learning)

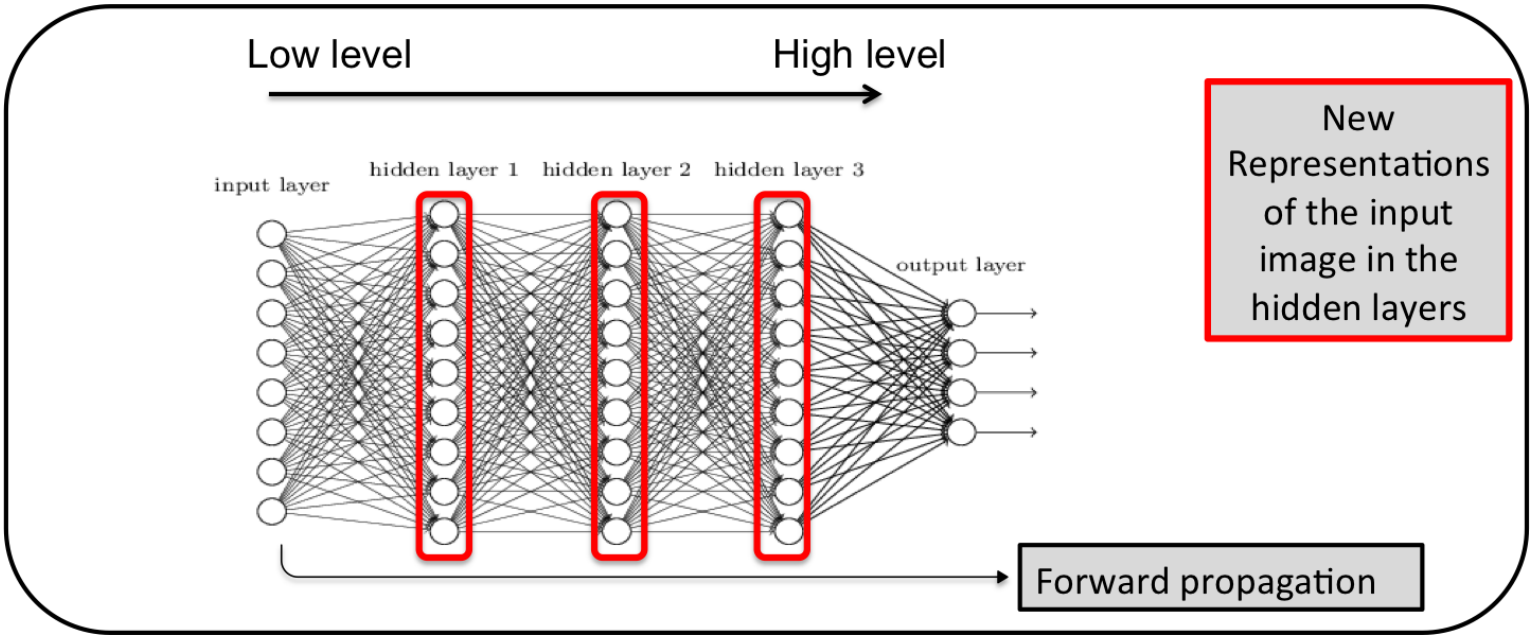
- Rely on ability to lean relavant and transferable representations
- Nearest neighbour-like rules in the learned representation space



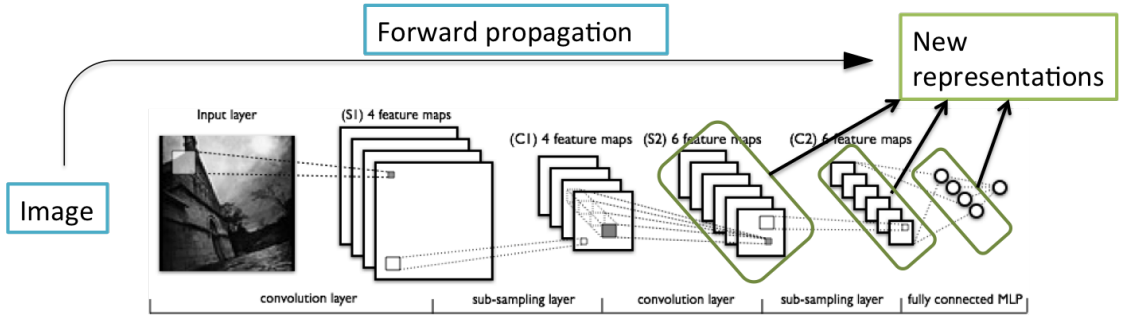
[Ravi Larochelle '17]

Extension of the embedding idea

More generally one call embedding a new representation space for any input data (image, text, signal...)



Extension of the embedding idea for images



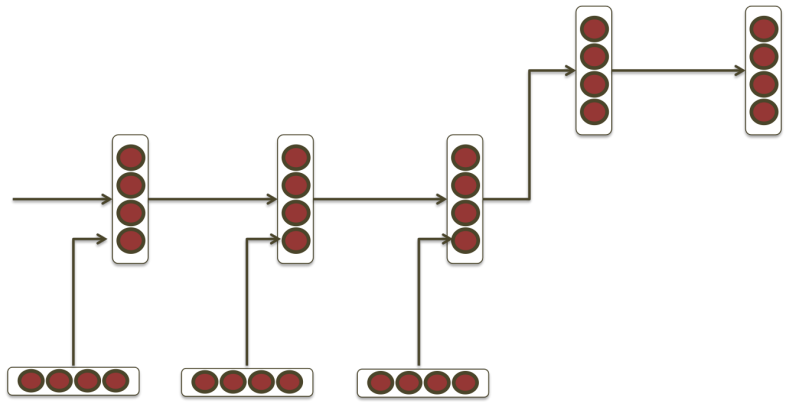
Main interest

- Many very deep architectures have been proposed by major actors (Google, Microsoft, Facebook...)
  - Using huge training corpora
  - Using huge computing resources
  - Architecture and Weights are often made publicly available
- It is better to use such models for computing high features from which one may design a classifier
  - With fine tuning (of upper layers) if enough training data are available on the target task
  - As a preprocessing if not

One goal: learning "universal" representations

Motivation : learn representations for any task

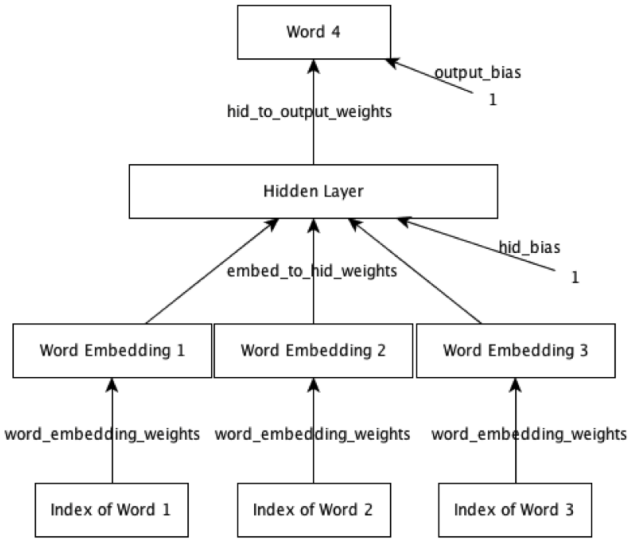
- Unsupervised or supervised
- For images, text, speech etc
  - The last layer of a CNN encodes most relevant information on the input (image)
  - The last hidden state of a RNN encodes most relevant information on the processed input sequence (e.g. sentence, signal)

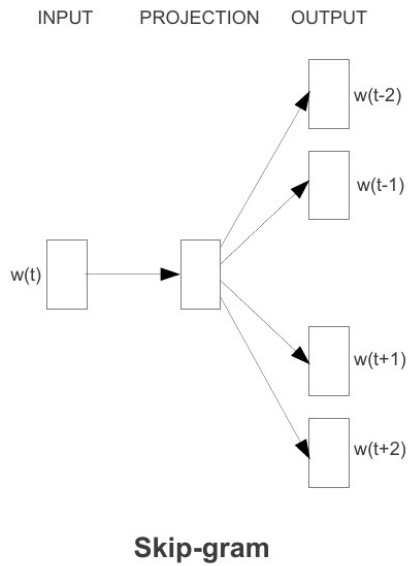


Word embeddings

Embeddings for words

- When the cardinality of the input is (very) large (e.g. NLP tasks) to allow accurate estimation from tractable corpus
- When one wants to infer some continuous representations of the input values to get insight on similarities between them



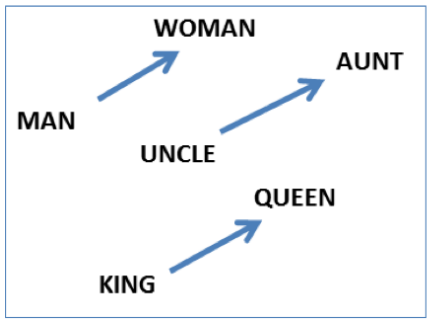


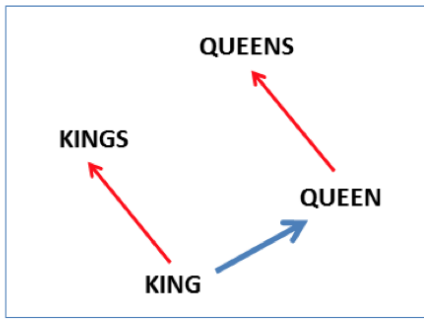
Skip-gram

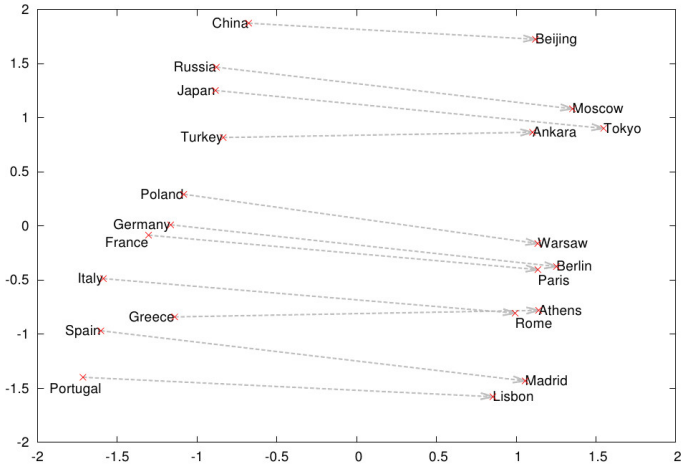
A particular interesting effect: compositionality

Idea

- $Emb('King') + Emb('Woman') - Emb('Man') \approx Emb('Queen')$
- It is an observed phenomenon which is not actually favored by the model design the learning criterion
- Similar effect reported on images (with DCGAN from [Radford et al.])







Outline

Depth in RNNs

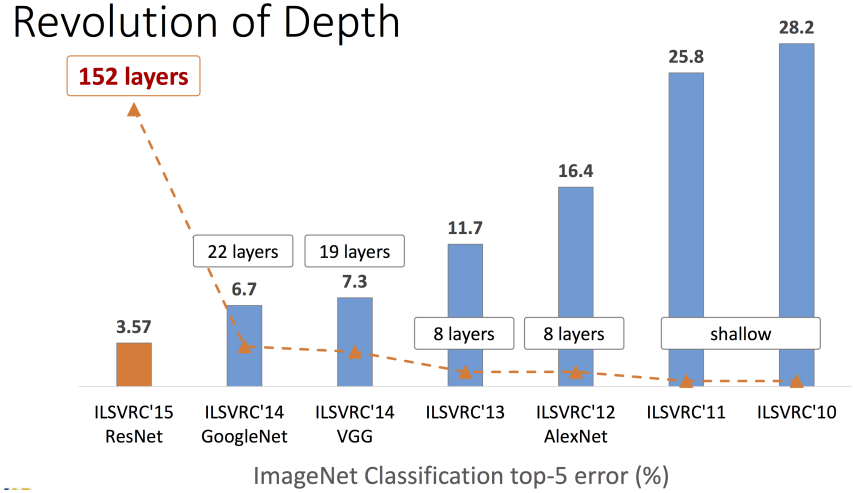
Depth in Feedforward nets

- Stacked layers in a feed forward or more complex manner (e.g. multiple paths)
- Gradient vanishing or exploding problems when backpropagating

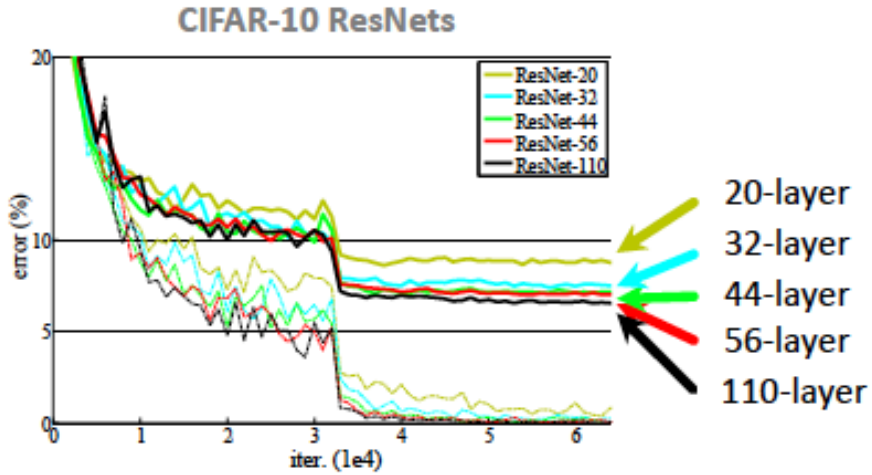
Depth in RNNs

- Stacked hidden layers as in traditional deep NNs : usual in many arhcitectures
- Long sequences $\rightarrow$ deep in time
- Both structural depths yield similar optimization problems (gradient flow)

Deep networks are powerful



[Kaiming He]



(Dense) Deep vs Shalow: Increased capacity

The power of depth [Eldan and Shamir, 2016]

- There is a simple function expressible by a 3-layer network that may not be approximated by a 2-layer network to more than a certain accuracy unless its width is exponential in the input dimension

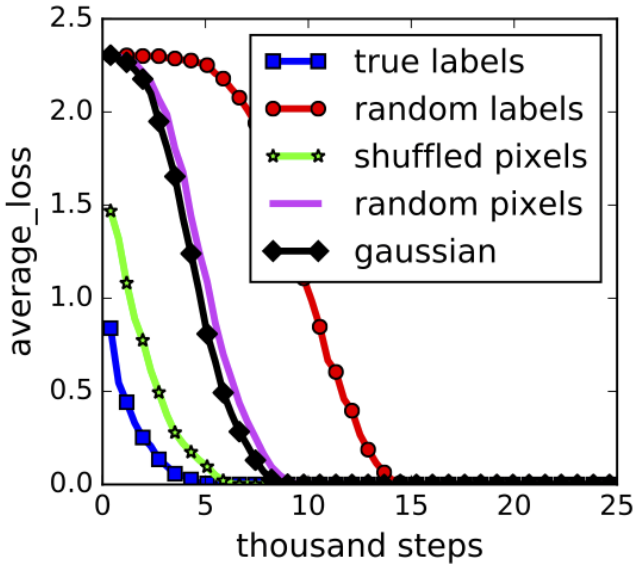
Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

- DNNs with RELU activation function $\Rightarrow$ piecewise linear function
- Capacity as a function of the number of linear regions one may divide the input space
- Exponentially more regions per parameter in terms of number of HL
  - Case of p_0 inputs and $p = 2p_0$ hidden cells per HL (with k HL) :
    - Maximum number of regions at least : $2^{(k-1)p_0} \sum_{j=0}^{p_0} \binom{2p_0}{j}$

DNNs are overparameterized

Large DNNs may even learn noise

- For instance : Learn after random permutation of the labels of the training samples
- It learns, but it takes more time...
- Note that the same (large) architectures that may learn random labels generalize well when trained on non perturbated data

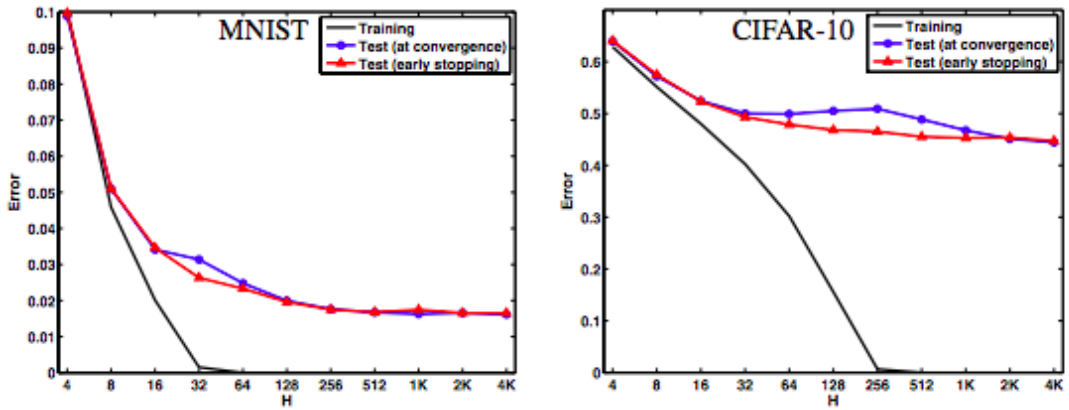


[Zhang and al., 2017]

DNNs and overfitting

Actually NNs do not easily overfit

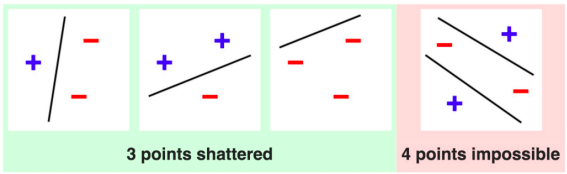
- The more you learn the better it generalizes
- Experiments on Mnist and CIFAR data (downsampled): 1 hidden layer (size H) NNs without any regularization $\rightarrow$ no overfitting observed



[Neyshabur 2017]

Capacity

Vapnik dimension



Rademacher capacity

$$R_n(H) = E_\sigma[sup_{h \in H} \frac{1}{n} \sum_{i=1}^{n} \sigma_i h(x_i)]$$

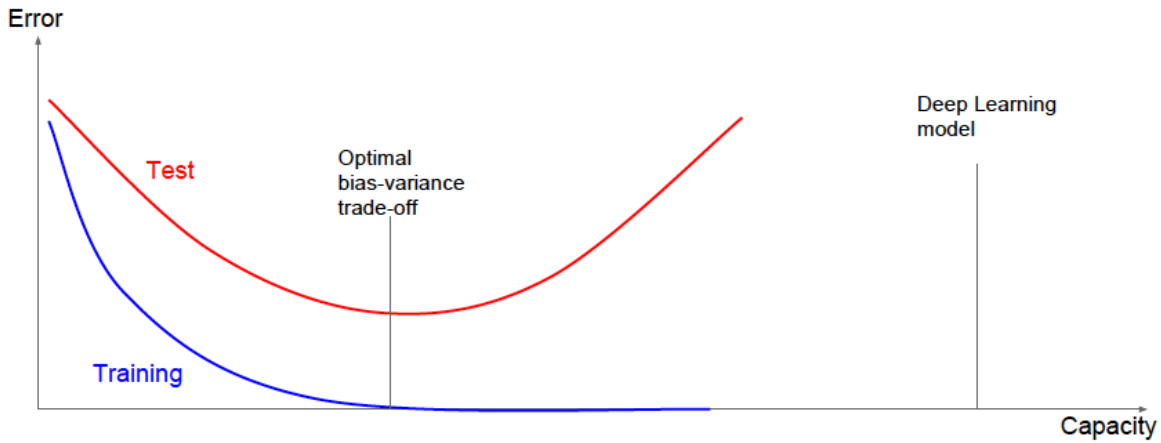
with $\sigma_i \in \{-1, 1\}$

- Clearly looks like the randomization test
- Trivial upperbound (=1): useless

DNNs' capacity

Vapnik dimension of deep NNs with ReLU

- With L hidden layer of p neurons the Vapnik dimension of deep ReLU NNs is $h = \Theta(L^2 p^2)$
- Considering classical generalization bound : $R(w) \leq R_{emp}(w) + \tilde{O}(\sqrt{\frac{L^2 p^2}{n}})$
- This does not explain generalization behavior

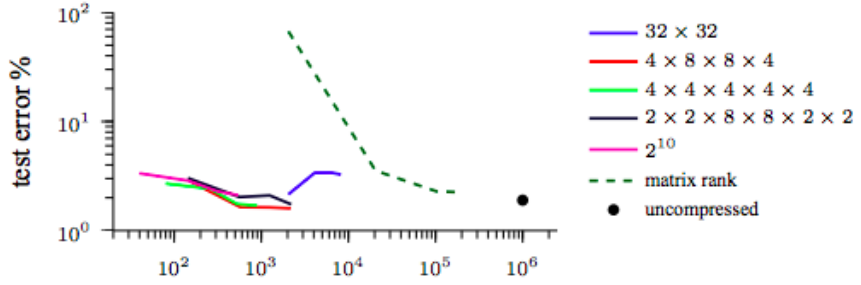


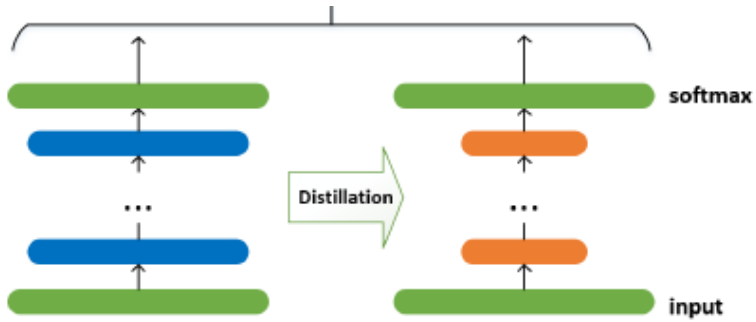
[O. Bousquet, tutorial 2017]

Deep nets do not actually need to be huge

Size helps learning but one may simplify once learned !

- Low rank tensor approximation (CP, Tucker, TensorTrain) of layer weight matrices (FC, Conv, RNN) [Novikov et al. 2015]
- Distillation strategy [Hinton et al., 2015]
  - Learn a deep and complex model f_{NN} (or en ensemble of deep models) on a dataset D
  - Create a new learning task by computing the output vectors o of f_{NN} for samples in D (better use logits than outputs of the softmax)
  - Learn a narrower model to predict o vectors for samples in D

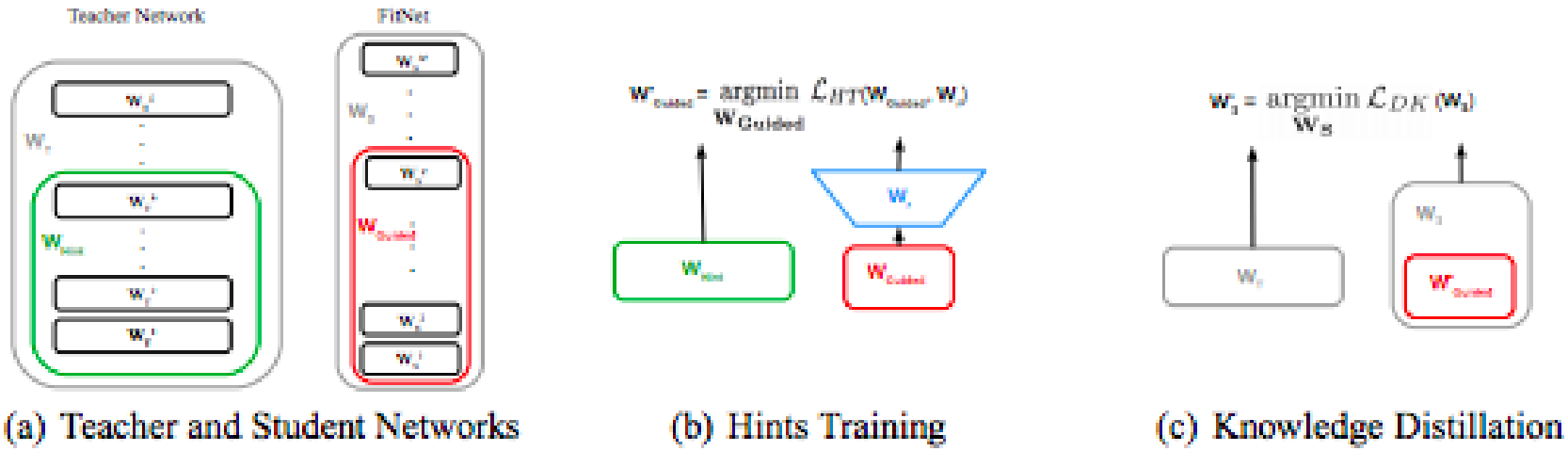




FitNets [Romero et al., 2015]

Going further in distillation with intermediate transfer

- Knowledge distillation + intermediate distillation losses



(a) Teacher and Student Networks (b) Hints Training (c) Knowledge Distillation

DL vs standard ML

Traditional Machine Learning

- Overfitting is the enemy
- One may control generalization with appropriate regularization
- Suboptimal optimization due to multiple local minima

DL: mysterious phenomenon

- Huge capacity without overfitting
- The size helps learning
- Overfitting idea should be revised for DNNs [Zhand and al., 2017] ?
- Regularization may slightly improve performance but is not THE answer for improving generalization
- Not clear what in the DNN may allow to predict its generalization ability