Réseaux de Neurones Profonds, Apprentissage de Représentations

Thierry Artières

ECM, LIF-AMU

July 5, 2017

Outline

History

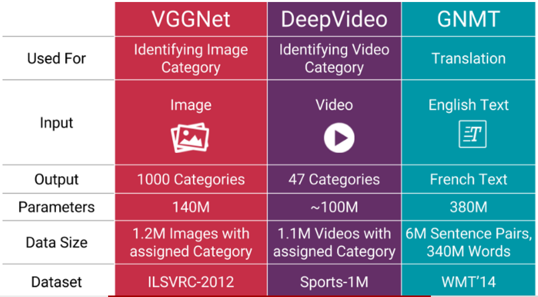
Key dates

- 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

Spectacular breakthroughs - fast industrial transfer

- Images, Vidoos, 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

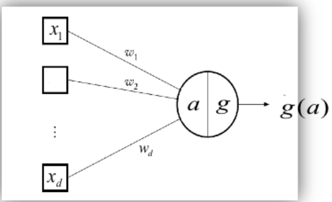
A single Neuron

One Neuron

- Elementary computation

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

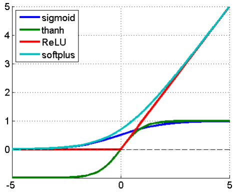
$$\text{output} = g(a(x))$$



Non linearity : g

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

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



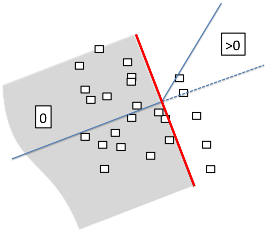
A single Neuron

One Neuron

- Elementary computation

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

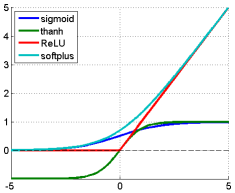
$$\text{output} = g(a(x))$$



Non linearity : g

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

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



Multi Layer Perceptron (MLP)

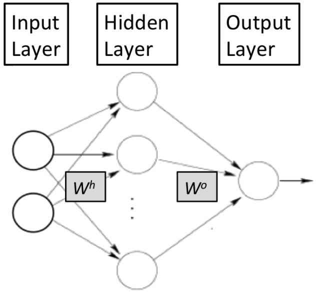
Structure

- 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



Multi Layer Perceptron (MLP)

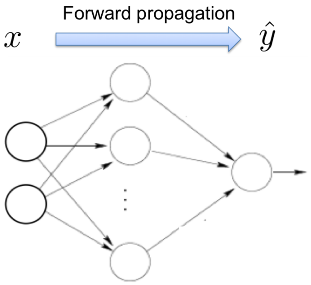
Structure

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

Function implemented by a MLP

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

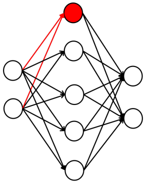
- Inference: Forward propagation from input to output layer

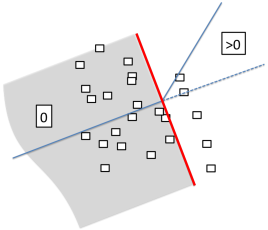


What a MLP may compute

What does a hidden neuron

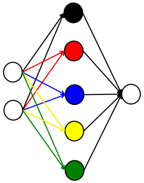
- Divides the input space in two

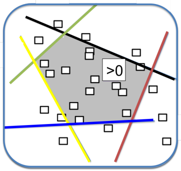




Combining multiple hidden neurons

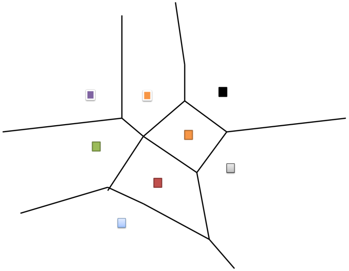
- Allows identifying complex areas of the input space
- New (distributed) representation of the input

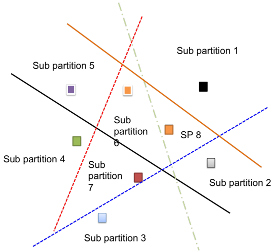




Distributed representations

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

Learning a MLP

Learning as an optimization problem

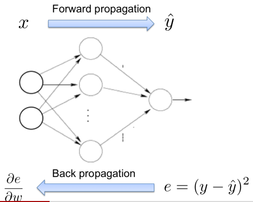
- Objective function of parameters set w for a given training set T

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

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

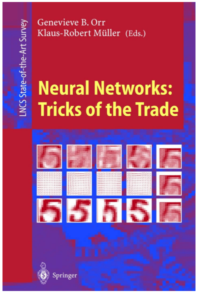
Backpropagation

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



Lots of tricks to favor good convergence

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

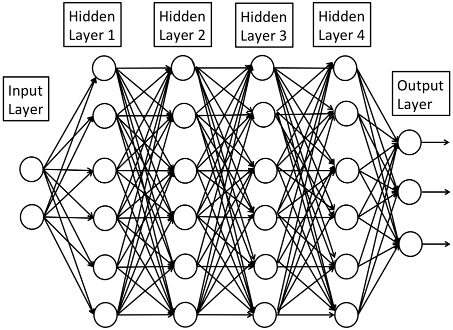

Genevieve B. Orr
Klaus-Robert Müller (Eds.)
LNCS State-of-the-Art Survey
Neural Networks: Tricks of the Trade
Springer

Outline

What are deep models ?

NNs with more than one hidden layer !

A series of hidden layers

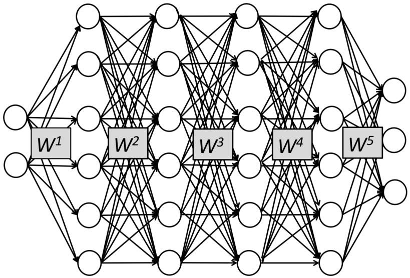


What are deep models ?

NNs with more than one hidden layer !

Computes a complex function of the input

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

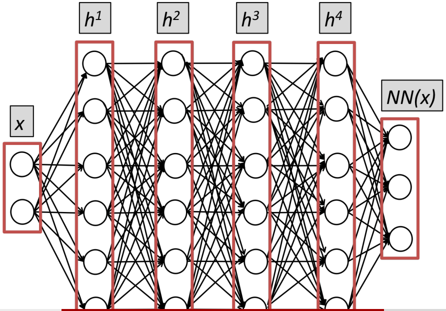


What are deep models ?

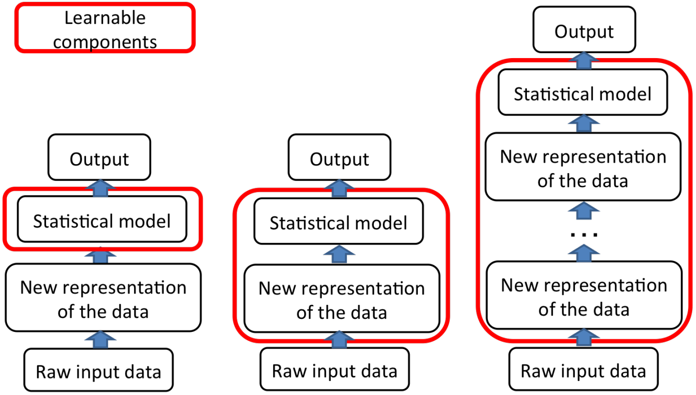
NNs with more than one hidden layer !

Computes new representations of the input

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



Machine Learning vs. Deep Learning



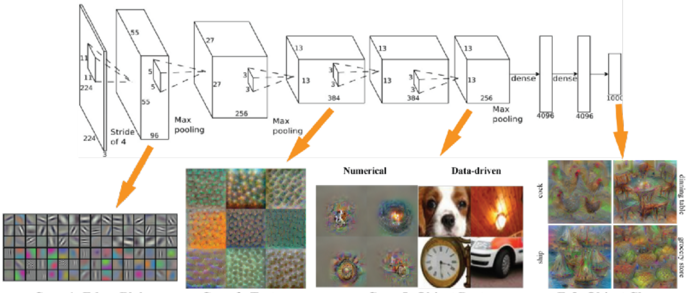
Machine Learning

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

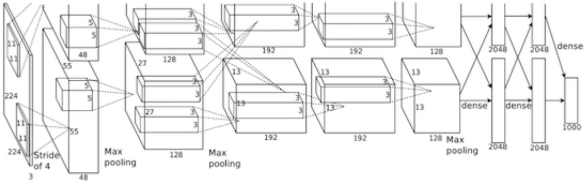


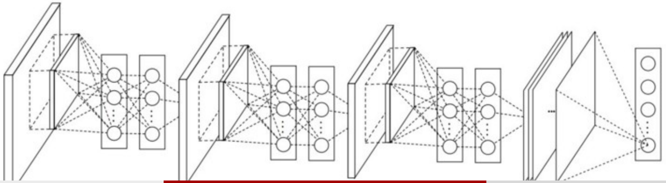
Conv 1: Edge+Blob Conv 3: Texture Conv 5: Object Parts Fc8: Object Classes

([Krizhevsky and al., 2012])

Examples of architectures

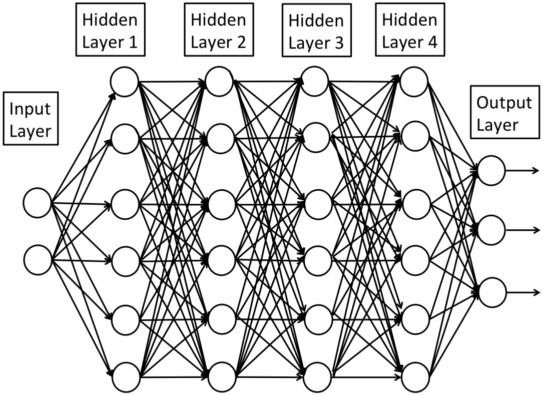
AlexNet [Krizhevsky and al., 2012] (top) and NetworkInNetwork [Lin and al.,2013] (bottom)





Outline

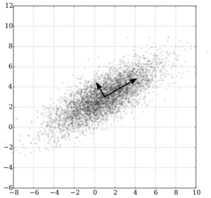
Dense architecture



Autoencoders

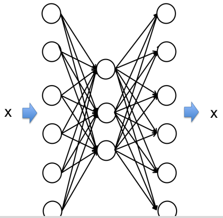
Principal Component Analysis

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



NN with Diabolo shape

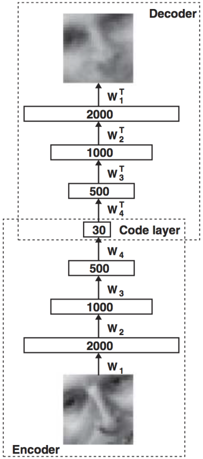
- Reconstruct the input at the output via an intermediate (small) layer
- Unsupervised learning
- Non linear projection, distributed representation
- Hidden layer may be larger than input/output layers



Deep autoencoders

Deep NN with Diabolo shape

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

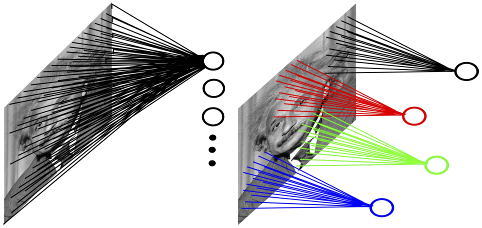


Convolutional layer

Motivation

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

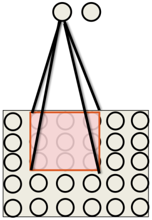
Fully connected layers vs locally connected layers



([LeCun and Bengio Tutorial DL 2015])

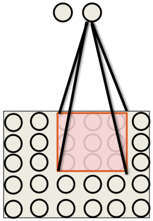
Convolution layer

Convolution layer



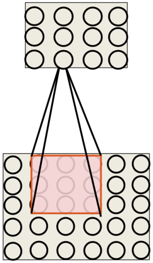
Convolution layer

Convolution layer



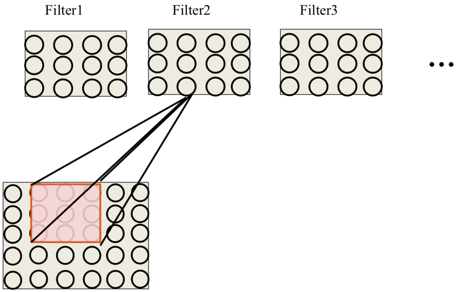
Convolution layer

Convolution layer



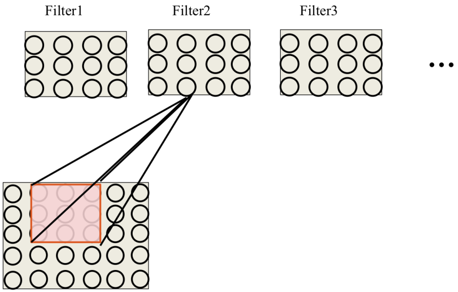
Convolution layer

Convolution layer

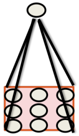


Convolution layer

Convolution layer

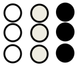


Example of a filter

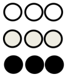


Filter weights

1	0	-1
1	0	-1
1	0	-1



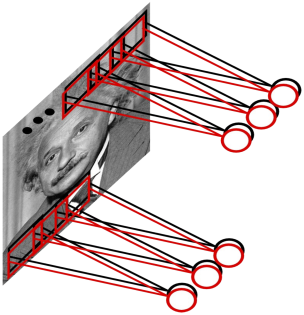
$\Rightarrow$ Positive output



$\Rightarrow$Null output

Convolution layer

Use of multiple maps



([LeCun and Ranzato Tutorial, DL, 2015])

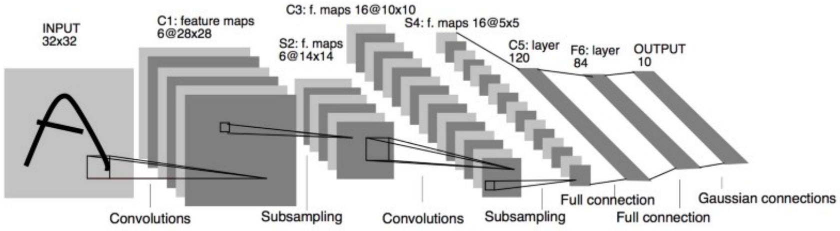
Aggregation layers

- Subsampling layers with aggregation operator
- Max pooling $\rightarrow$ brings robustness

Convolutional models

LeNet architecture [LeCun 1997]

- Most often a mix of (convolutional + pooling) layers followed by dense layers



Learning deep networks

Gradient descent optimization

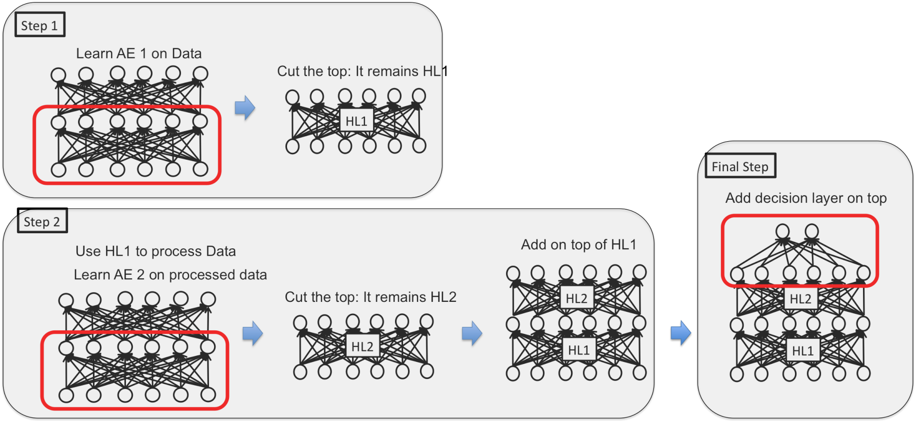
SGD, with momentum, Adagrad, Adam etc

Few strategies (considering large volumes of unlabeled data)

- Very large labeled training dataset : Fully supervised setting
- Too few labeled training samples for supervised training : Unsupervised feature learning (each layer one after the other) + fine tuning with a classifier on top
- Very few labeled training samples : Unsupervised feature learning (each layer one after the other) + flat classifier learning

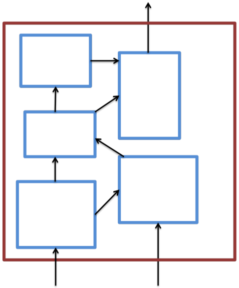
Learning deep networks

Unsupervised feature learning layer by layer


Step 1
Learn AE 1 on Data
Cut the top: It remains HL1
HL1
Step 2
Use HL1 to process Data
Learn AE 2 on processed data
Cut the top: It remains HL2
HL2
Add on top of HL1
HL2
HL1
Final Step
Add decision layer on top
HL2
HL1

More general architectures

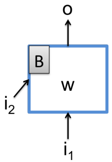
Graph of modules (better without cycles...)



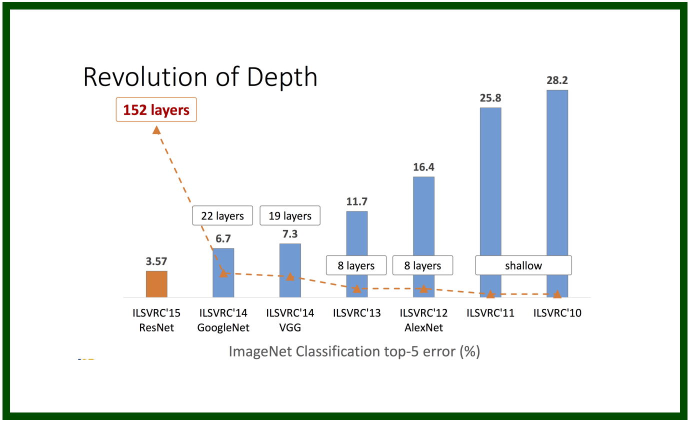
Still optimized with Gradient Descent !!

$$W = W - \epsilon \frac{\partial C(W)}{\partial W}$$

- provided functions implemented by blocks are differentiable
- and derivatives $\frac{\partial Out(B)}{\partial In(B)}$ and $\frac{\partial Out(B)}{\partial W(B)}$ are available for every block



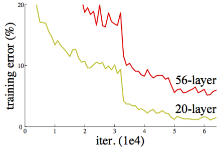
The Times They Are A Changing

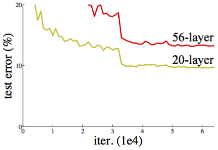


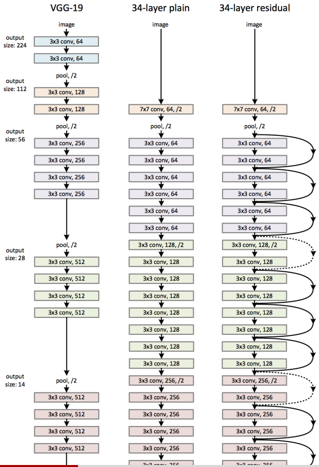
(slide from [Kaiming He])

From shalow to deep

Simply stacking layers does not work (CIFAR results) ! (figures form [He and al., 2015])







Deep vs Shalow ?

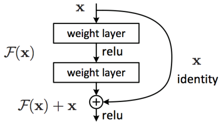
Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

- DNNs with RELU activation function $\Rightarrow$ piecewise linear function
- Complexity of DNN function as the Number of linear regions on the input data
- Case of n_0 inputs and $n = 2n_0$ hidden cells per HL (k HL) :
    - Maximum number of regions : $2^{(k-1)n_0} \sum_{j=0}^{n_0} \binom{2n_0}{j}$
- Example: $n_0 = 2$
    - Shallow model: $4n_0$ units $\rightarrow$ 37 regions
    - Deep model with 2 hidden layers with $2n_0$ units each $\rightarrow$ 44 regions
    - Shallow model: $6n_0$ units $\rightarrow$ 79 regions
    - Deep model with 3 hidden layers with $2n_0$ units each $\rightarrow$ 176 regions
- Exponentially more regions per parameter in terms of number of HL
- At least order (k-2) polynomially more regions per parameter in terms of width of HL n

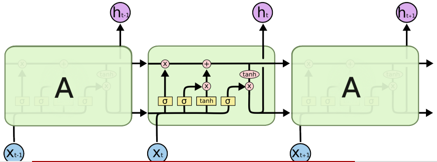
The depth alone is not enough

Making gradient flow for learning deep models

- Main mechanism : Include the identity mapping as a possible path from the input to the output of a layers
- ResNet building block [He and al., 2015]]



- LSTM (deep in time) [Hochreichter and al., 1998]



About generalization, overtraining, local minimas etc

Traditional Machine Learning

- Overfiting is the enemy
- One may control generalization with appropriate regularization

Recent results in DL

- The Overfit idea should be revised for DL [Zhand and al., 2017]
  - Deep NN may learn noise !
  - Regularization may slightly improve performance but is not THE answer for improving generalization
- Objective function do not exhibit lots of saddle points and most local minima are good and close to globale minimas [Choromanska et al., 2015]
  - Not clear what in the DNN may allow to predict its generalization ability

Favorable context

Huge training resources for huge models

- Huge volumes of training data
- Huge computing ressources (clusters of GPUs)

Advances in understanding optimizing NNs

- Regularization (Dropout...)
- Making gradient flow (ResNets, LSTM, ...)

Faster diffusion than ever

- Softwares
    - Tensorflow, Theano, Torch, Keras, Lasagne, ...
- Results
    - Publications (arxiv publication model) + codes
    - Architectures, weights (3 python lines for loading a state of the art computer vision model!)

Outline

Embedding layer

Motivation : Transformation layer for discrete/categorical inputs

- Example : a Word in a Dictionary (Natural Language Processing tasks)
- Embedding : distributed representation. Not a new idea (LSA, LDA)

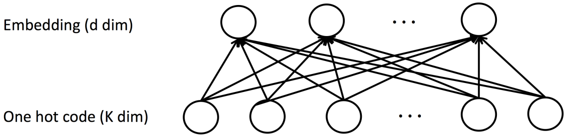
Main interests

- 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

Embedding layer: Implementation

Look up table

- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix ($K \times d$)

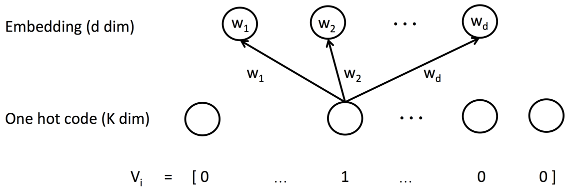


$$
\begin{array}{lllllllll}
V_1 & = & [\,1 & 0 & 0 & \dots & 0 & 0\,] \\
V_2 & = & [\,0 & 1 & 0 & \dots & 0 & 0\,] \\
 & & & & \cdots & & & \\
V_K & = & [\,0 & 0 & 0 & \dots & 0 & 1\,]
\end{array}
$$

Embedding layer: Implementation

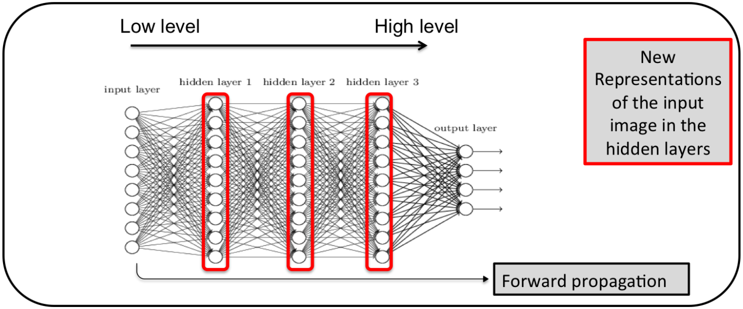
Look up table

- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix ($K \times d$)



Extension of the embedding idea

More generally one call embedding a new representation space for any input data



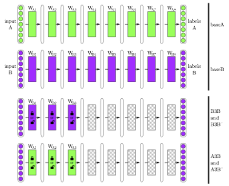
Genericity of representations [Yozinski and al., 2014]

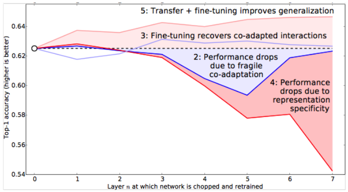
Experiments on two similar tasks

- 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* ...

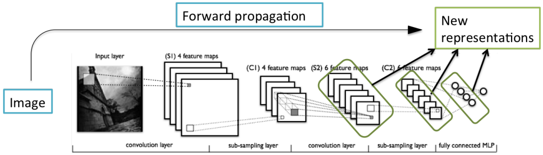
Main results

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





Extension of the embedding idea for vision tasks



Main interest

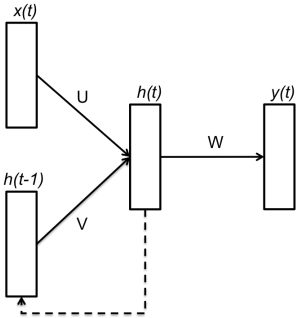
- Many very deep architectures have been proposed by major actors (Google, Microsoft, Facebook...)
  - Using huge training corpora
  - Using huge computing resources
  - Architecture and Weights are often made publicly available
- It is better to use such models for computing high features from which 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

Outline

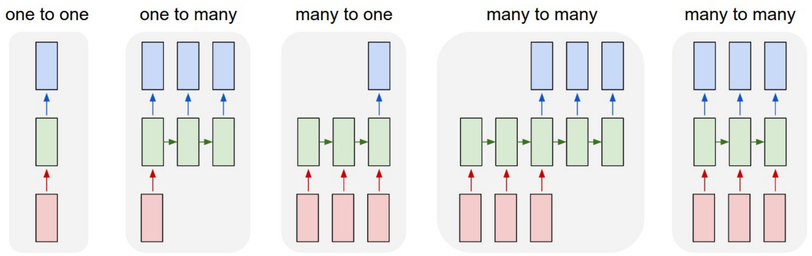
Recurrent NNs (RNNs)

RNNs in general

- May handle data of different dimension w.r.t. traditional FeedForward Models (Sequences, trees, ...)
- A recurrent neural network is a NN with cycles in its connections
- Much more powerful than acyclic models (FeedForward NNs such as MLPs)
- Not all architectures work well. Few popular ones.

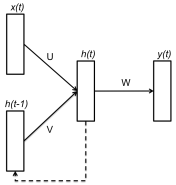


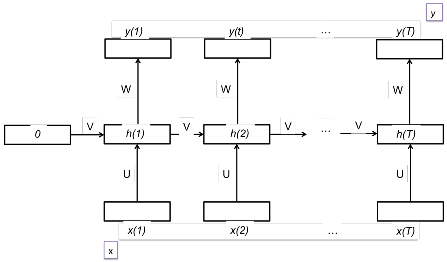
Various settings



- 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...

Inference and learning through unfolding the RNN





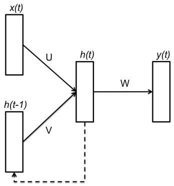
Inference: Forward propagation in the FeedForward unfolded RNN

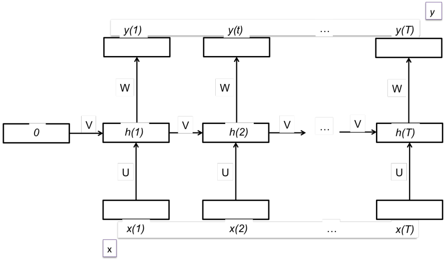
- Start with null state $h(0) = 0$
- Iterate

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

$$y(t) = g(W \times h(t))$$

Inference and learning through unfolding the RNN

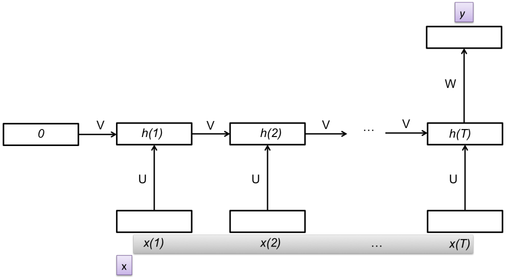




Learning: Back-propagation in the FeedForward unfolded RNN

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

Unfolding the RNN: classification tasks



Inference

- Start : $h(0) = 0$
- For $t = 1$ to T DO : $h(t) = g(V \times h(t-1) + U \times x(t))$
- Predict : $y = g(W \times h(T))$
  $\Rightarrow$ The final state $h(T)$ resumes the whole input

Depth in RNNs

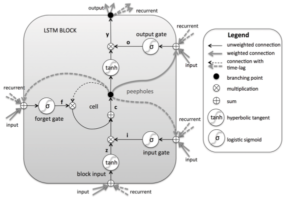
Two dimensions

- 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)

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

Outline

Adversarial learning principle

Goal

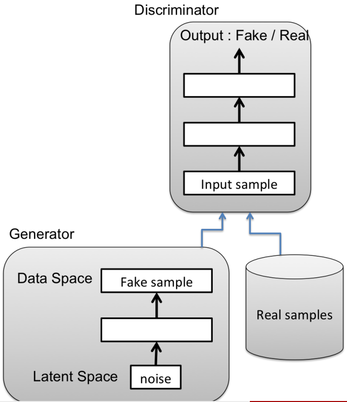
- Learn to generate complex and realistic data
- Statistical viewpoint : learn a model of the density of data / able to sample with this density
    - Postulate a parametric model : Usually not complex enough
    - Postulate a parametric form and perform optimization (e.g. Maximum Likelihood) : Intractable for complex forms $p(x) = \frac{F(x)}{Z(x}$ with $Z(x) = \sum_x F(x)$

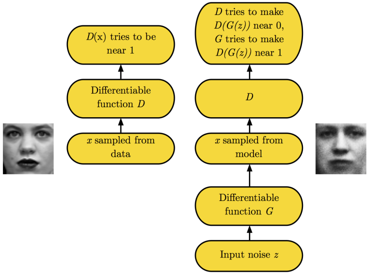
Principle

- Use a two player game
- Learn both a generator of artifical sampels and a discriminator that learns to distinguishes between true and fake samples. The generator wants to flue the discriminator
- If an equilibrium is reached the generator produces samples with the true density

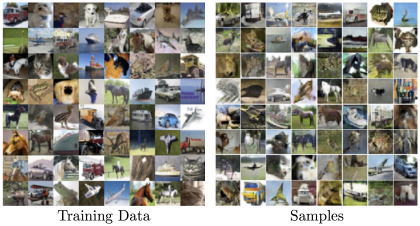
Adversarial Learning

Generative Adversarial Networks (GANs) [Goodfellow and al., 2014]

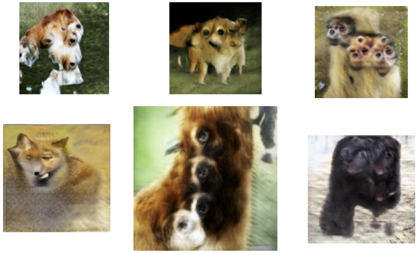



D(x) tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

Examples with GANs [Goodfellow and al., 2014]

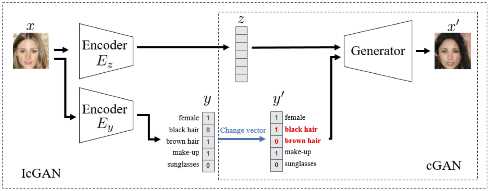


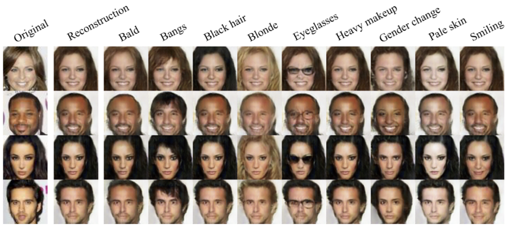
Nice samples (learning on CIFAR dataset)



Nightmare animals !

Image editing with Invertible Conditional GANs [Perarnau and al., 2016]



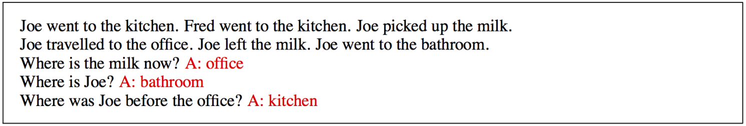


Outline

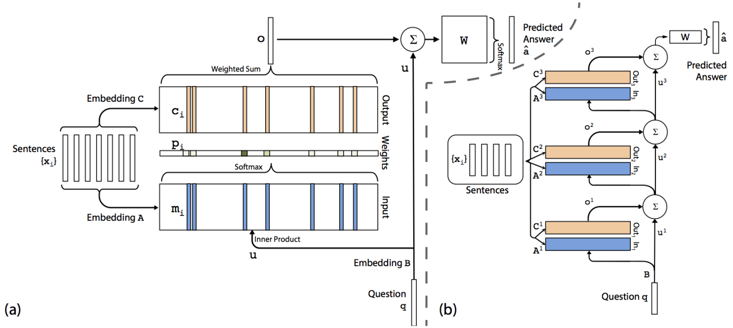
Memory Networks [Weston and al., 2015]

Principle

- Include a long-term memory that can be read and written to with the goal of using it for prediction: kind of knowledge base
- More straightforward use of the memory than in RNNs
- Ability to deal with complex question answering

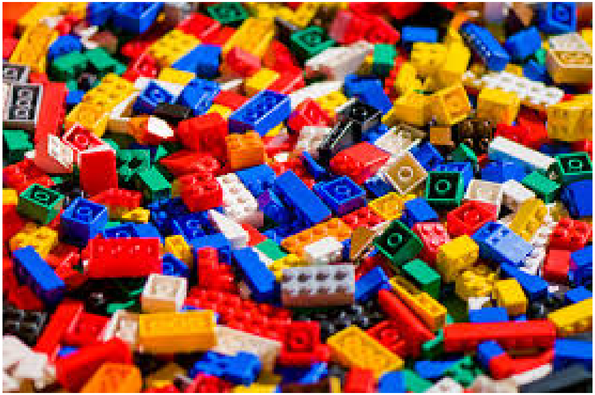

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

End to End Memory Networks [Sukhbaatar and al., 2015]



Outline

The Lego game (with nice pieces !)



Level 1: building models

Many available building bricks

- A NN is a sequence / graph of layers that process data
- Variety of layers
  - Processing layers : Dense, Convolution, Pooling
  - Activation layers
  - Normalization and regularization layers (Dropout, Batch Normalization. . .)
- Architecture bricks and specific cells
  - Residual blocks, LSTM, GRU, HighWay
- Optimization routines
  - Stochastic Gradient Descent (SGD), Momentum, Adagrad, Adam. . .

At the end

- Choosing an architecture is a combinatorial design problem
- Not many hints on how to choose an architecture

Level 2: building systems

Models available for many tasks

- Existence of models able to compute a **universal representation** of data for:
  - Text
  - Images
- Ability to learn a model that represent any structured data in fixed dimension space Sequences, trees, Graphs
- Adversarial learning for learning any probability density function on complex objects
- Attention and memory mechanisms in neural networks

What comes next

- All these models may be used as bricks in new systems for more complex tasks
- Automatic captioning, Machine translation, Text understanding. . .