Réseaux de Neurones Profonds, Apprentissage de Représentations

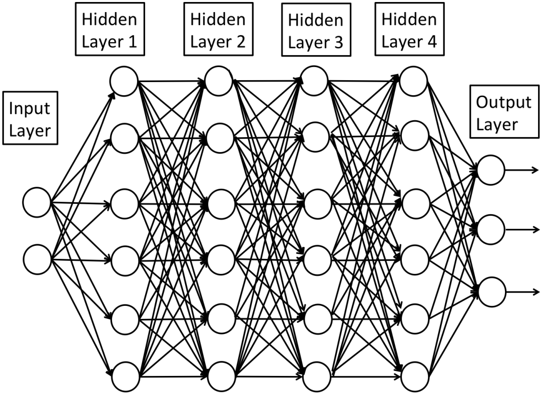
Thierry Artières

ECM, LIF-AMU

January 15, 2018

Outline

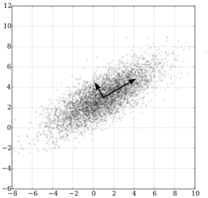
Dense architecture



Autoencoders

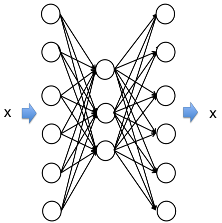
Principal Component Analysis

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



NN with Diabolo shape

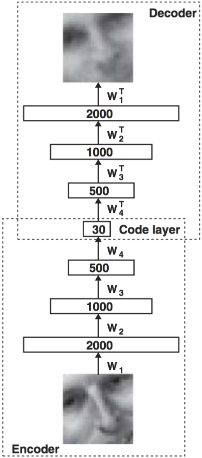
- Reconstruct the input at the output via an intermediate (small) layer
- Unsupervised learning
- Non linear projection, distributed representation
- Hidden layer may be larger than input/output layers



Deep autoencoders

Deep NN with Diabolo shape

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

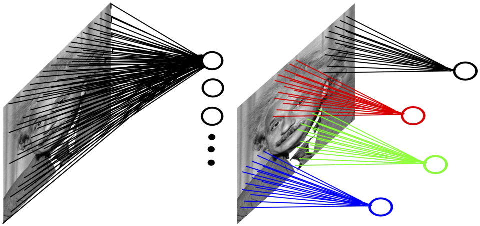


Convolutional layer

Motivation

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

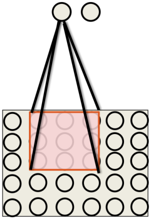
Fully connected layers vs locally connected layers



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

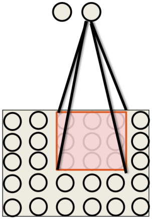
Convolution layer

Convolution layer



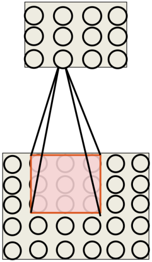
Convolution layer

Convolution layer



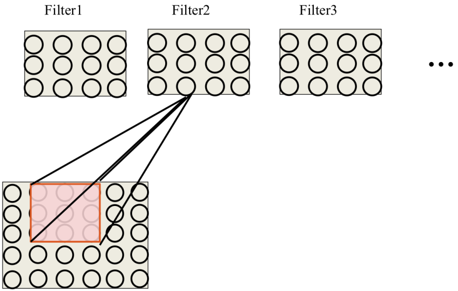
Convolution layer

Convolution layer



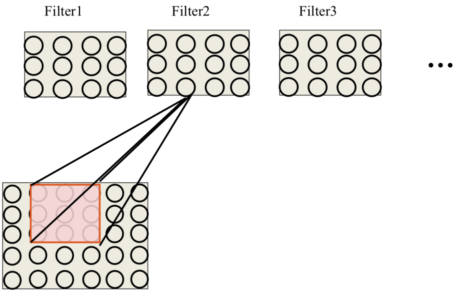
Convolution layer

Convolution layer



Convolution layer

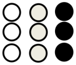
Convolution layer



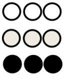
Example of a filter

Filter weights

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



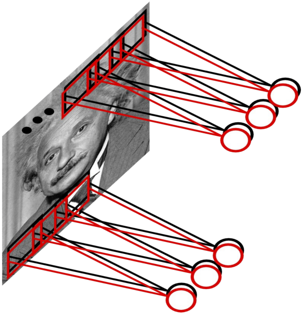
⇒ Positive output



⇒Null output

Convolution layer

Use of multiple maps



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

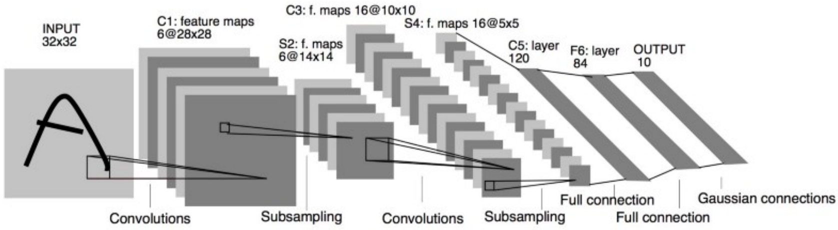
Aggregation layers

- 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



Outline

Learning deep networks

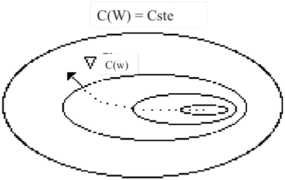
Gradient descent optimization as for MLPs

- SGD
- With momentum
- Adagrad, Adam, Adadelta etc

Gradient Descent Optimization

Gradient Descent Optimization

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



Gradient Descent: Tuning the Learning rate

Two classes Classification problem

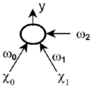
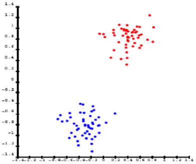
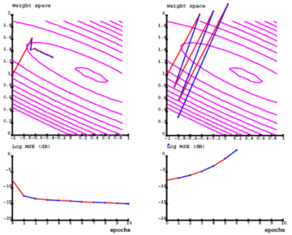


Fig. 9. Simple linear network.



Weight trajectory for two different gradient step settings.



(a) (b)

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

Images from [LeCun et al.]

Gradient Descent: Tuning the Learning rate

Effect of learning rate setting

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

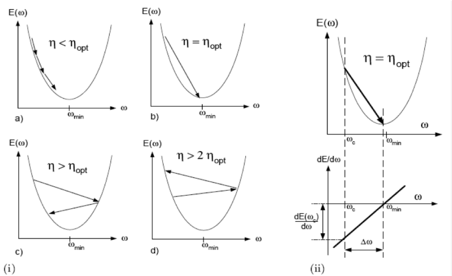


Fig. 6. Gradient descent for different learning rates.

Images from [LeCun et al.]

Gradient Descent: Stochastic, Batch and mini batchs

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

Batch vs Stochastic vs Minibatchs

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

Using Momentum

SGD with Momentum

- Standard Stochastic Gradient descent : $w = w - \epsilon \frac{\partial C(w)}{\partial w}$
- SGD with Momentum:

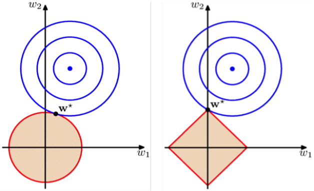
$$v = \gamma v + \epsilon \frac{\partial C(w)}{\partial w}$$

$$w = w - v$$

Guiding the learning

Regularization

- Constraints on weights (L1 or L2)



- Constraints on activities (of neurons in a hidden layer)
  - 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

Guiding the learning

Denoising autoencoders and Deep Belief Networks

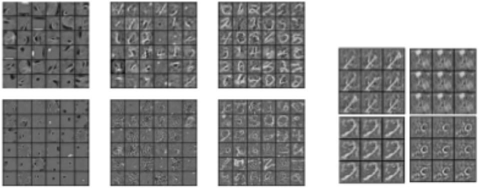
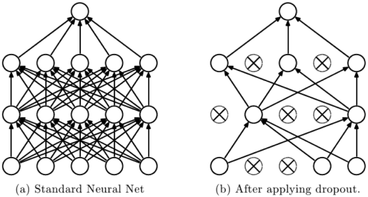


Figure 1: Activation maximization applied on MNIST. On the left side: visualization of 36 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a DBN (top) and SDAE (bottom), using the technique of maximizing the activation of the hidden unit. On the right side: 4 examples of the solutions to the optimization problem for units in the 3rd layer of the SDAE, from 9 random initializations.

- Examples of learned filters with Denoising Autoencoders (top)

Guiding the learning

Dropout



- First method that allowed learning rellay deep networks without pretraining and smart initialization
- Related to ensemble of models
- Weights are normalized at inference time

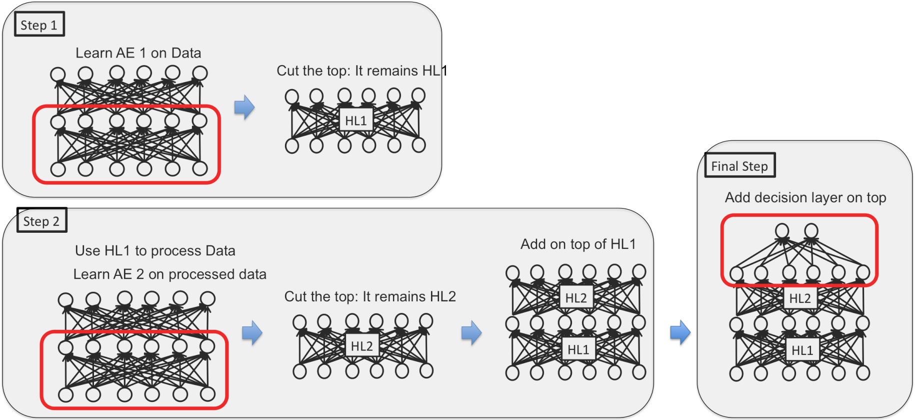
Learning deep networks: Strategies

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

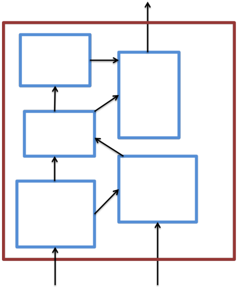
Learning deep networks

Unsupervised feature learning layer by layer



Learning more general architectures

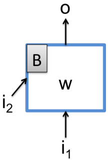
Graph of modules (better without cycles...)



Still optimized with Gradient Descent !!

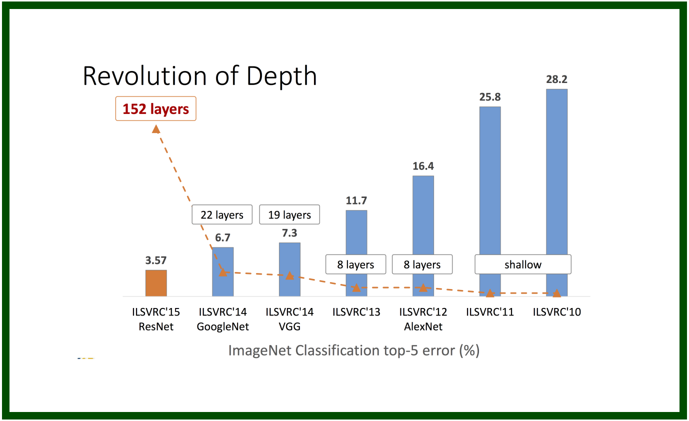
$$W = W - \epsilon \frac{\partial C(W)}{\partial W}$$

- 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



Outline

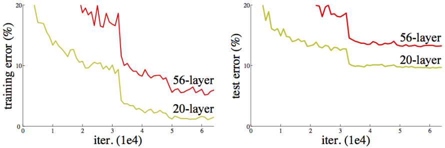
The Times They Are A Changing

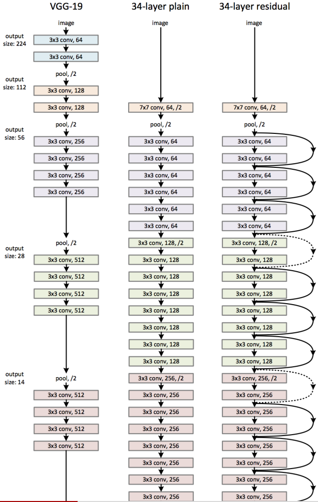


(slide from [Kaiming He])

From shalow to deep

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



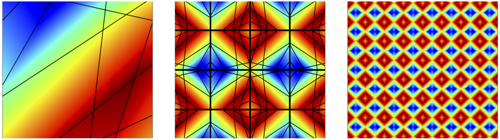


Deep vs Shalow ?

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

Deep vs Shalow ?



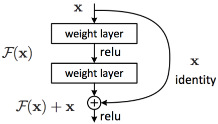
From [Pascanu and al., 2014]

- Left: Regions computed by a layer with 8 RELU hidden neurons on the input space of two dimensions (i.e. the output of previous layer)
- Middle: Heat map of a function computed by a rectifier network with 2 inputs, 2 hidden layers of width 4, and one linear output unit. Black lines delimit regions of linearity of the function
- Right: Heat map of a function computed by a 4 layer model with a total of 24 hidden units. It takes at least 137 hidden units on a shallow model to represent the same function.

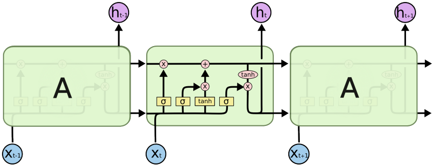
The depth alone is not enough

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!)