Etat des lieux des recherches en Deep Learning

Thierry Artières

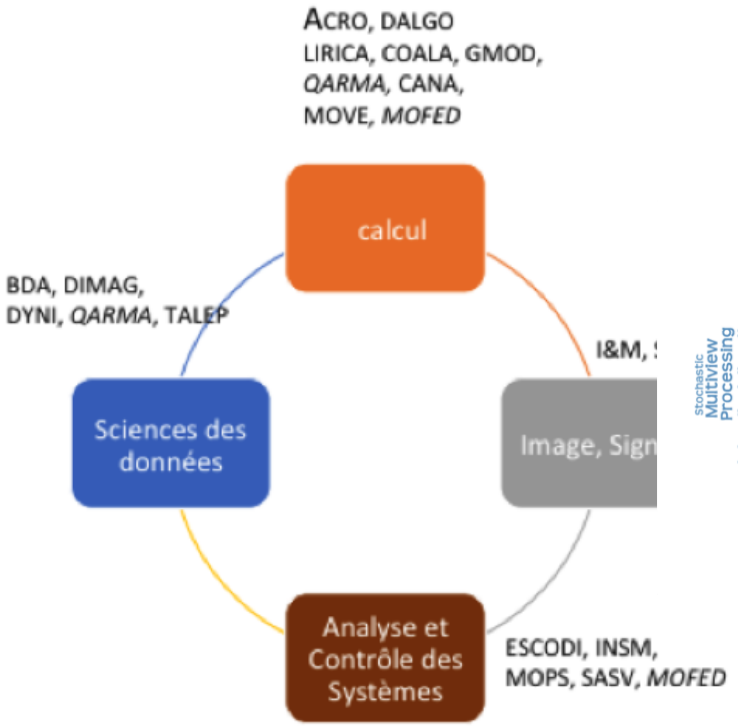
Head of Machine Learning team
Data Science Department - LIS lab
Professor at Ecole Centrale Marseille

November 15, 2018

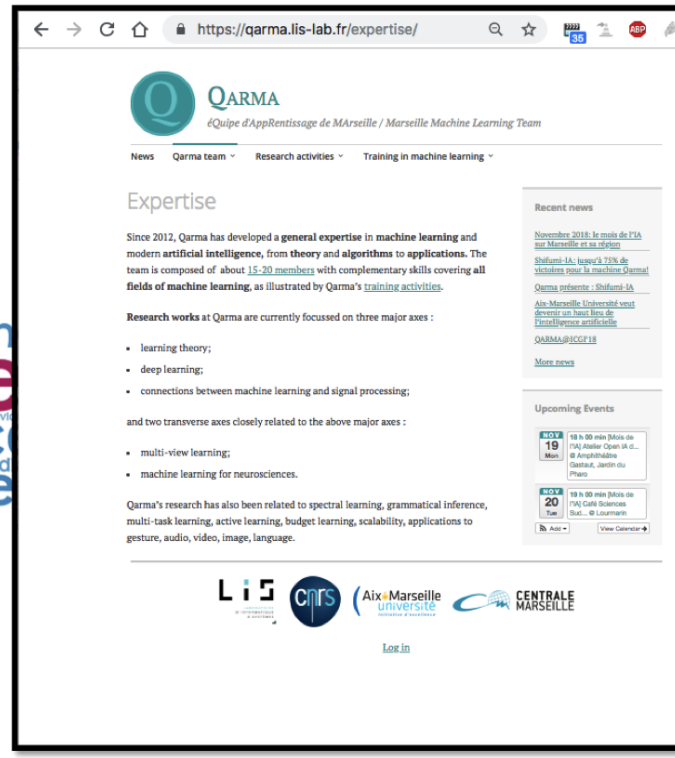
Outline

Outline

About us


ACRO, DALGO
LIRICA, COALA, GMOD,
QARMA, CANA,
MOVE, MOFED
calcul
BDA, DIMAG,
DYNI, QARMA, TALEP
I&M,
Sciences des données
Image, Sign
Analyse et Contrôle des Systèmes
ESCODI, INSM,
MOPS, SASV, MOFED


Data Science
Learning
Artificial Intelligence
Natural Language
Information Retrieval
System
Recommandation
Graph
Mining
Neural Nets
Languages
Quality
Non-IID
Cognition
Speech
Multimodal
Theory
Temporal
Information
Interdisciplinarity
Kernels
Linguistics
Representation
Sentence
Named Entities
Understanding
Environment
Analysis
Syntax
Deep learning


https://qarma.lis-lab.fr/expertise/
QARMA
éQuipe d'AppRentissage de MArseille / Marseille Machine Learning Team
News
Qarma team
Research activities
Training in machine learning
Expertise
Since 2012, Qarma has developed a general expertise in machine learning and modern artificial intelligence, from theory and algorithms to applications. The team is composed of about 15-20 members with complementary skills covering all fields of machine learning, as illustrated by Qarma's training activities.
Research works at Qarma are currently focussed on three major axes :
learning theory;
deep learning;
connections between machine learning and signal processing;
and two transverse axes closely related to the above major axes :
multi-view learning;
machine learning for neurosciences.
Qarma's research has also been related to spectral learning, grammatical inference, multi-task learning, active learning, budget learning, scalability, applications to gesture, audio, video, image, language.
Recent news
Upcoming Events
Log in

About us

Teaching

Formation initiale ECM

Parcours IA /ML en 3eme A

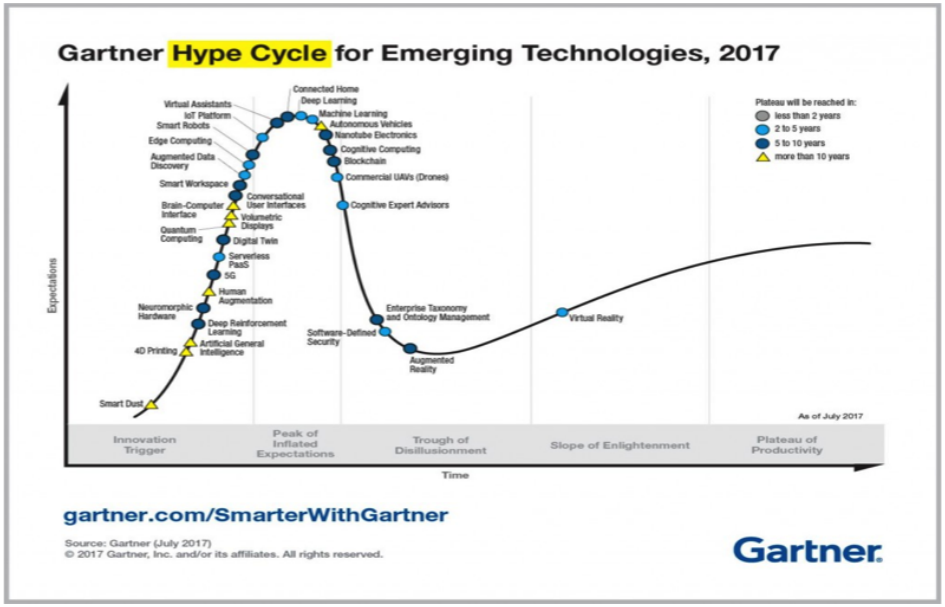
Formation continue ECM / CNRS

Machine Learning, Deep Learning…

Where is AI?

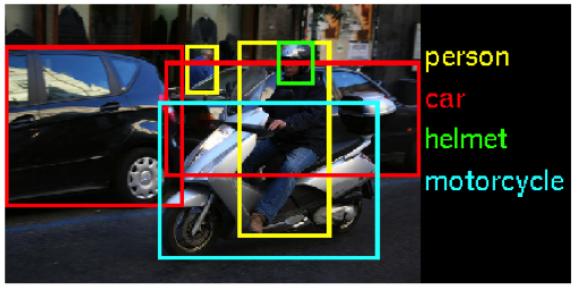
Constat

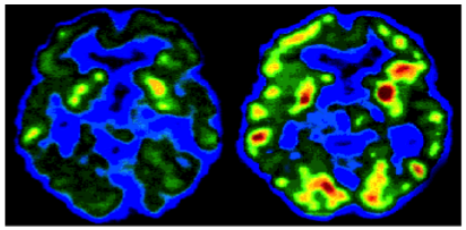
- The original AI was the General AI (IA forte)
- Today 60 years after the Dartmouth meeting
  - We have achieved some NIA results (IA faible)
  - We can start thinking more seriously about GAI
  - These are just the premises.



At the heart of AI: Machine Learning (and Deep Learning)

Which algorithms to solve these tasks ?





Machine Learning

What is it for?

- Writing programs that solve a task while we don't even know how to writre the algorithm
- Where a program takes some input and produce a corresponding output
- The program is learned from labeled data = pairs of (input, output)

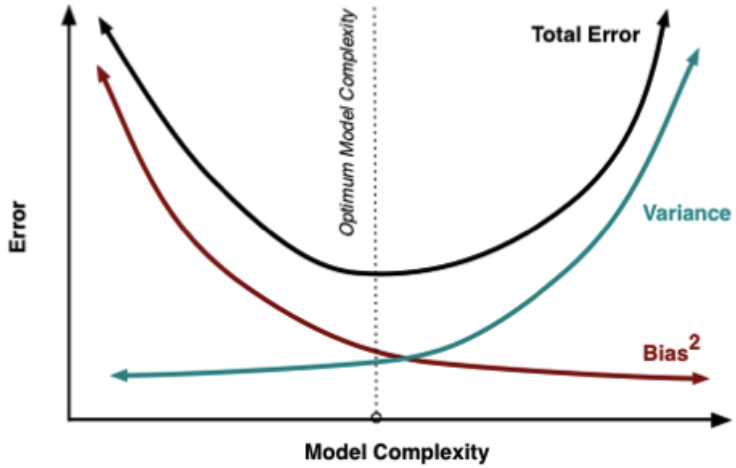
What is it?

- Algorithms that enable learning a function $f : x \in X \rightarrow y \in Y$ from a training dataset of samples
- The function must generalize well to data unseen at training time
- x and y may be discrete, continuous, vectors, matrices, tensors, sequences ...

Main difficulty

Generalization

- It is “easy” to learn models that are perfect on training data
- But is is useless



Outline

History of Neural Networks

Key dates

- 1943 : Formal neuron [McCuloch-Pitts]
- 1950 : Oragnization of neurons and learning rules [Hebb]
- 1960 : Perceptron [Rosenblatt]
- 1960 : Update rule [Widrow Hoff]
- 1969 : Limitations of the Perceptron [Minsky]
- 1980s : Back-propagation [Rumelhart and Hinton]
- 1990s : Convolutional Networks [LeCun and al.]
- 1990s: Long Short Term Memory networks [Hochreiter and Schmidhuber]
- 2006 : Paper on Deep Learning in Nature [Hinton and al.]
- 2012 : Imagenet Challenge Win [Krizhevsky, Sutskever, and Hinton]
- 2013 : First edition of ICLR
- 2013 : Memory networks [Weston and al.]
- 2014 : Adversarial Networks [Goodfelow and al.]
- 2014 : Google Net [Szegedy and al.]
- 2015 : Residual Networks [He et al.]
- ...

Deep Learning today

Spectaculary breakthroughs - fast industrial transfer

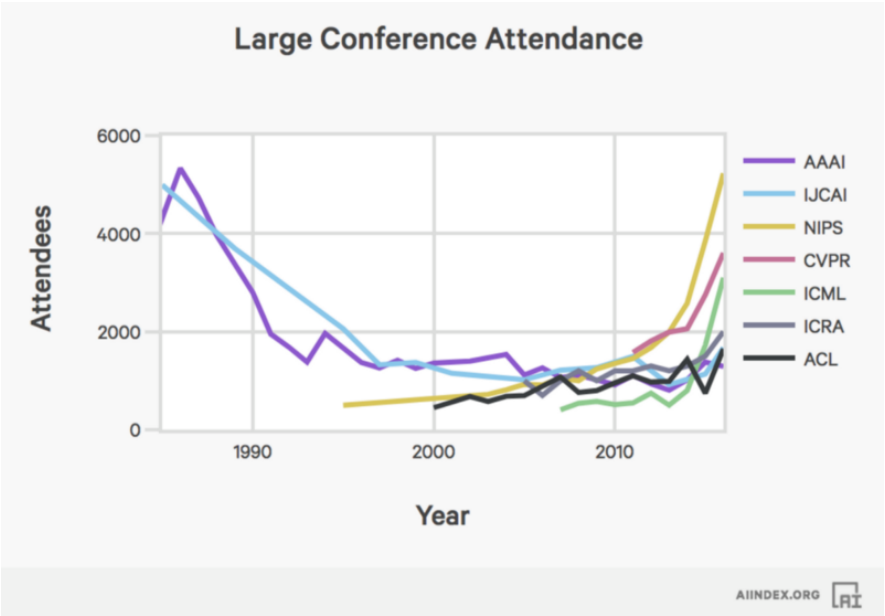
- Images, Videos, Audio, Speech, Texts
- Successful setting
  - Structured data (temporal, spatial...)
  - Huge volumes of datas
  - Huge models (millions of parameters)
  - Huge storage and computing resources (GPU, TPU)

	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

Machine Learning and Deep Learning today

Spectacular diffusion and activity

- Machine Learning and Deep Learning Conferences sold out early
- More attendees than ever seen in computer science conferences
- Exponential growth
- Semantic change in what AI means

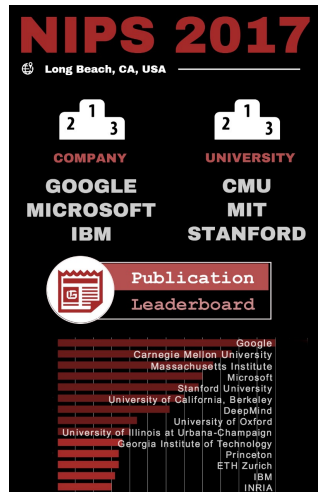


Machine Learning and Deep Learning today

Topics, trends and who's who?

- Mix between academics and companies
- Extreme popularity of Deep Learning topics
- Birth of the International Conference on Learning Representation (2014)

Rank	Day	Name	Marked	Like
1	1	Tutorials Hall A	2789	287
2	2	Deep Learning, Applications	2364	289
3	3	Deep Learning	1831	163
4	3	Reinforecment Learning, Deep Learning	1592	140
5	1	Optimization	1522	130
6	1	Tutorials Hall C	1344	135
7	1	Algorithms	1307	137
8	2	Theory	1288	83
9	2	Algorithms Optimization	1223	107
10	4	Deep Learning, Algorithms	1202	113
11	4	Deep Reinforcement Learning	1202	43
12	2	Invited talk: Kate Crawford: The Trouble with Bias	1162	71
13	3	Reinforcement Learning, Algorithms, Applications	1156	134
14	3	Invited talk: Pieter Abbeel: Deep Learning for Robotics	1087	61
15	1	Tutorials Grand Ballroom	1082	132



DL research is going very fast !!

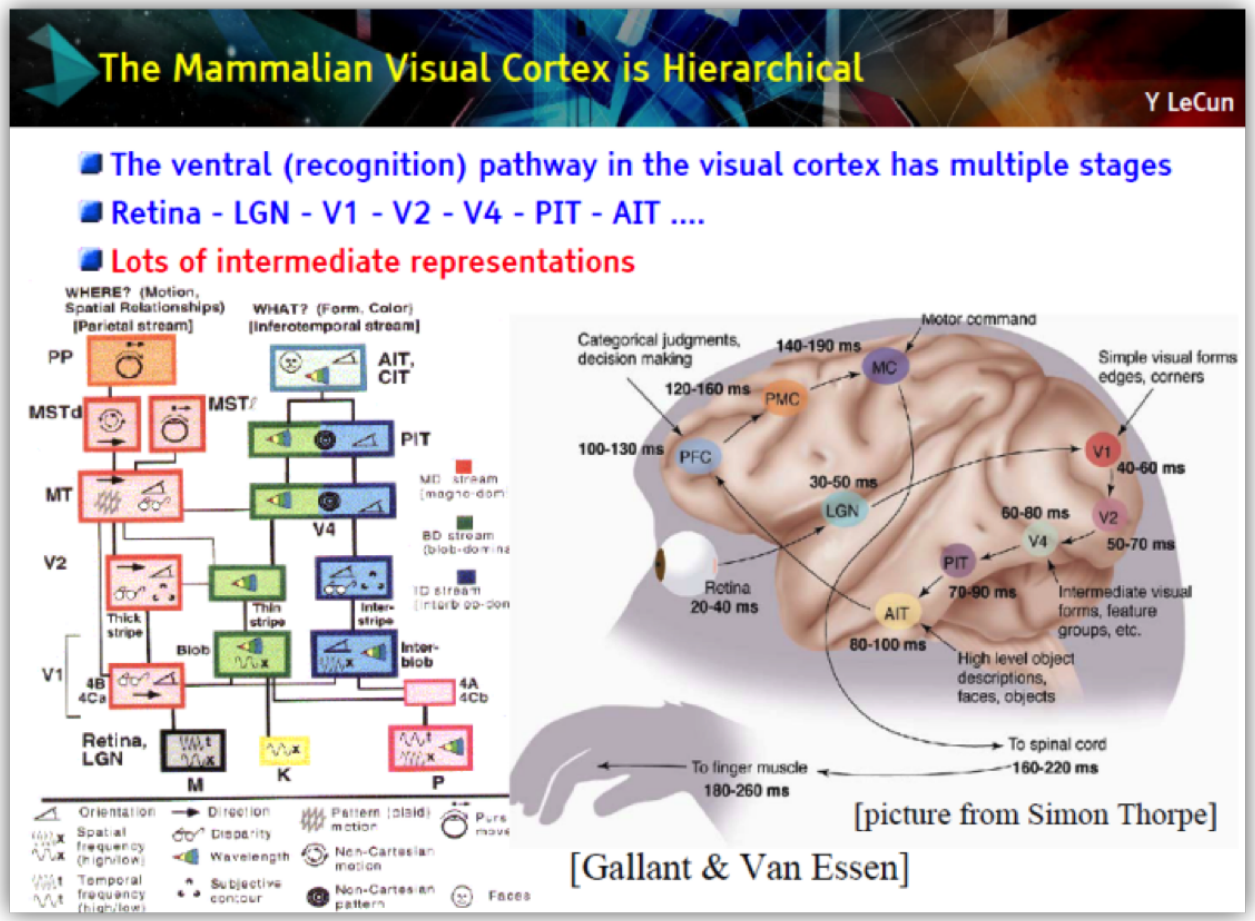
Example of an emerging topic: Generative Adversarial Networks

- First publication : 2014 by Ian J. Goodfellow, and al.
- Hundreds of publications (close to a thousand) papers since

New publication mode

- Wasserstein GANs, Martin Arjovsky and al.
  - Published on arXiv : Jan 2017
  - Published at ICML in Aout 2017
- Improved Training of Wasserstein GANs by Ishaan Gulrajani and al.
  - Published on arXiv : March 2017
  - Published at NIPS in December 2017
- Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect by Xiang Wei and al.
  - Published on Openreview : Oct 2017
  - Accepted as poster at ICLR in 2018 (April 2018)

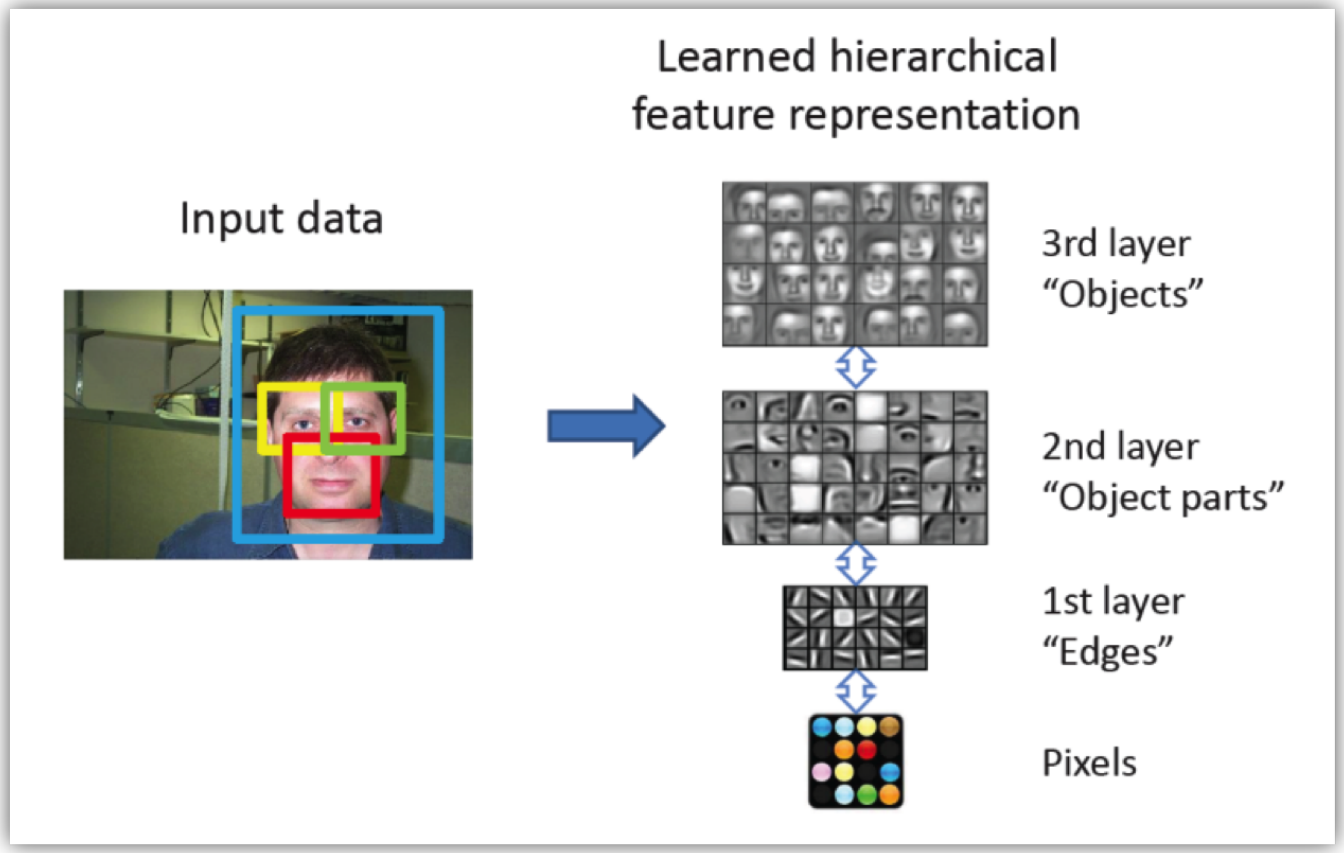
The Graal ? (but we are not there yet)



[Gallant et al., ...]

The key: features

Deep learning = Representation Learning

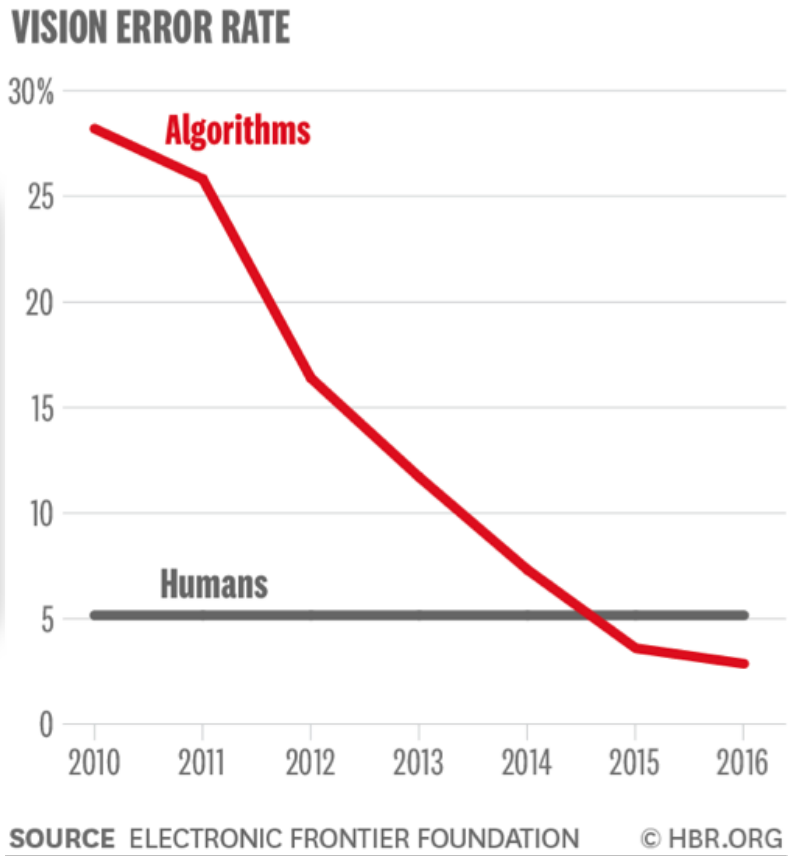


[Honglak et al.]

Spectaculary breakthrough

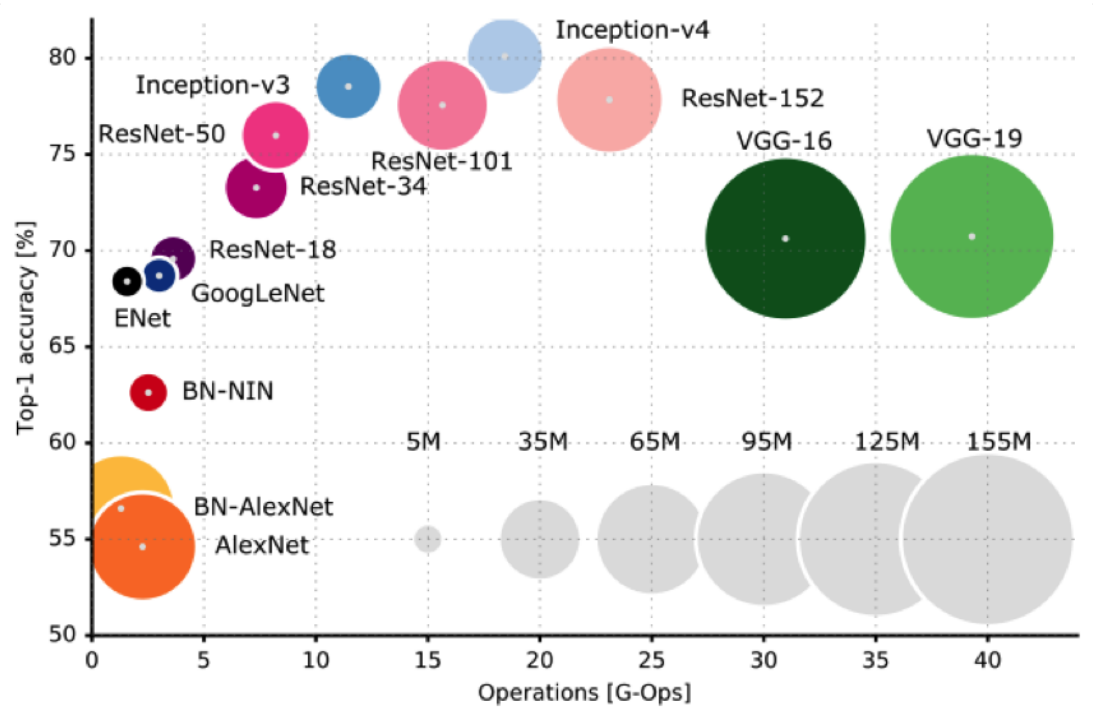
Computer vision

Real time Object recognition



Spectaculary breakthrough

Computer Vision



Spectaculary breakthrough

Speech

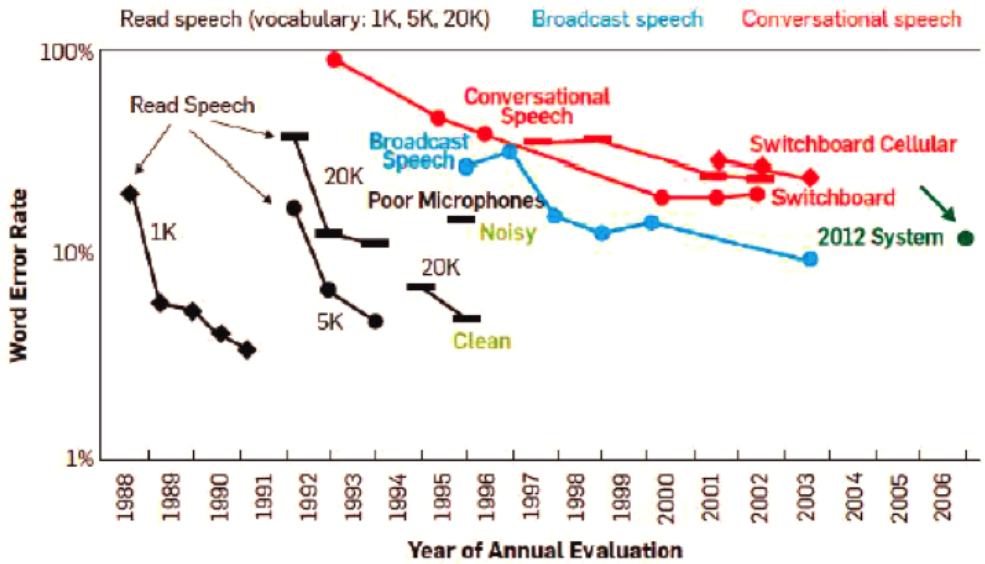
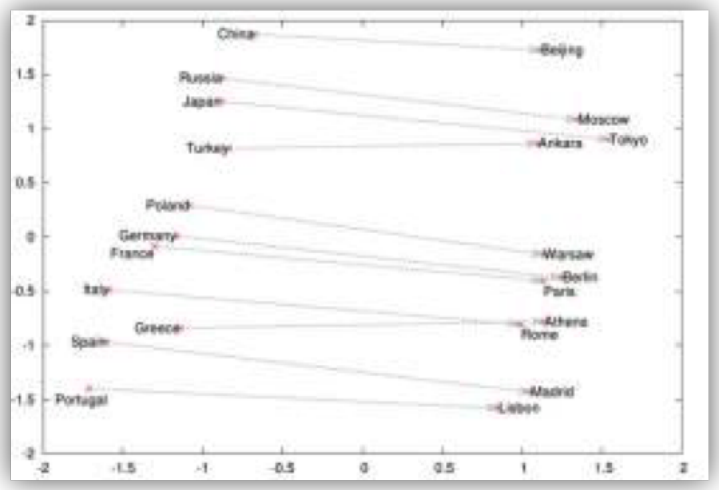


FIGURE 2.7 Historical progress on reducing the word error rate in speech recognition systems for different kinds of speech recognition tasks. Recent competency for the "difficult switchboard" task (human conversation in the wild) is marked with the green dot. SOURCE: X. Huang, J. Baker, and R. Reddy, 2014, A historical perspective of speech recognition, *Communications of the ACM* 57(1):94-103, doi:10.1145/2500887. © 2014, Association of Computing Machinery, Inc. Reprinted with permission.

Spectaculary breakthrough

Natural Language Processing

- Text representation, modeling, generation Demo
- Chat bots



Input (Head and Label)	Predicted Tails
J. K. Rowling influenced by	*G. K. Chesterton*, J. R. R. Tolkien, *C. S. Lewis*, **Lloyd Alexander**, Terry Pratchett, Roald Dahl, Jorge Luis Borges, *Stephen King*, Ian Fleming
Anthony LaPaglia performed in	*Lantana*, *Summer of Sam*, *Happy Feet*, *The House of Mirth*, Unfaithful, **Legend of the Guardians**, Naked Lunch, X-Men, The Namesake
Camden County adjoins	**Burlington County**, *Atlantic County*, *Gloucester County*, Union County, Essex County, New Jersey, Passaic County, Ocean County, Bucks County
The 40-Year-Old Virgin nominated for	*MTV Movie Award for Best Comedic Performance*, *BFCA Critics' Choice Award for Best Comedy*, *MTV Movie Award for Best On-Screen Duo*, MTV Movie Award for Best Breakthrough Performance, **MTV Movie Award for Best Movie**, MTV Movie Award for Best Kiss, D. F. Zanuck Producer of the Year Award in Theatrical Motion Pictures, Screen Actors Guild Award for Best Actor - Motion Picture
Costa Rica football team has position	*Forward*, *Defender*, *Midfielder*, **Goalkeepers**, Pitchers, Infielder, Outfielder, Center, Defenseman
Lil Wayne born in	**New Orleans**, Atlanta, Austin, St. Louis, Toronto, New York City, Wellington, Dallas, Puerto Rico
WALL-E has the genre	Animations, Computer Animation, *Comedy film*, *Adventure film*, *Science Fiction*, **Fantasy**, Stop motion, *Satire*, Drama

Spectaculary breakthrough

Games

- BackGammon, Chess, Go

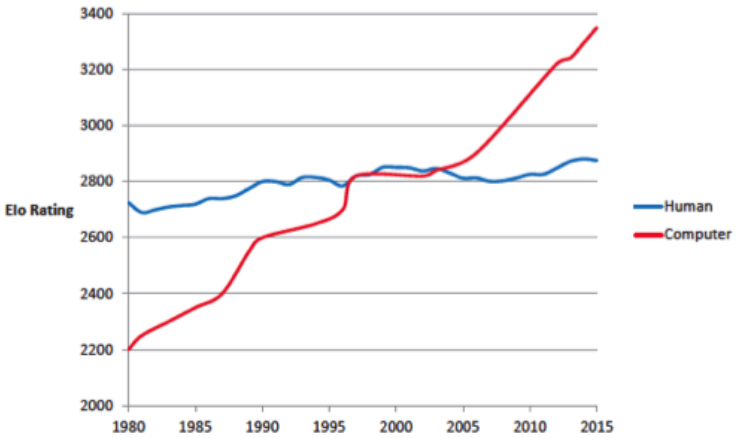
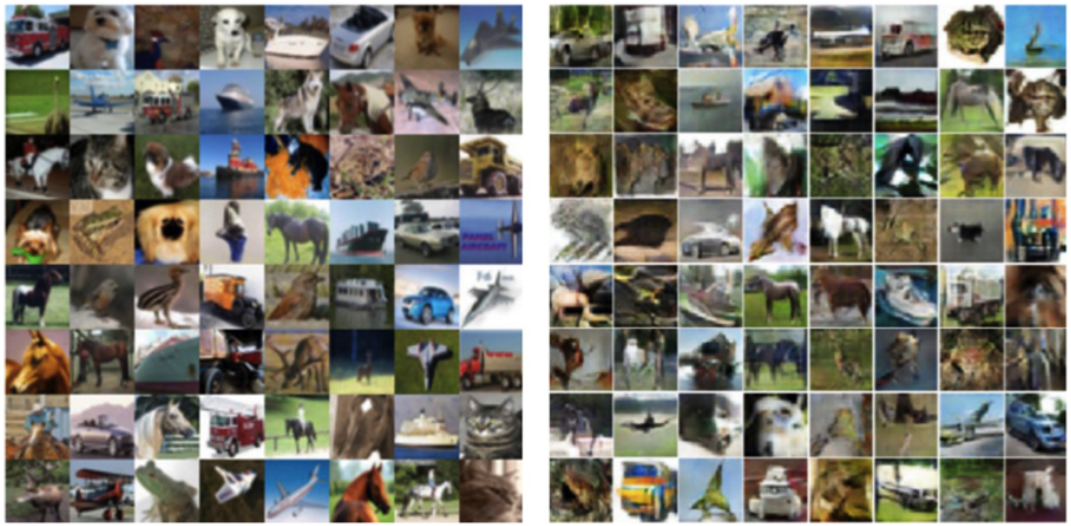


FIGURE 2.3 Elo scores—a measure of competency in competitive games—showing the chess-playing competency of humans and machines, measured over time. SOURCE: Courtesy of Murray Campbell.

Spectaculary breakthrough

Image generation

Recent Nvidia results



Training Data Samples

Spectaculary breakthrough

Should we still trust what we see?

Outline

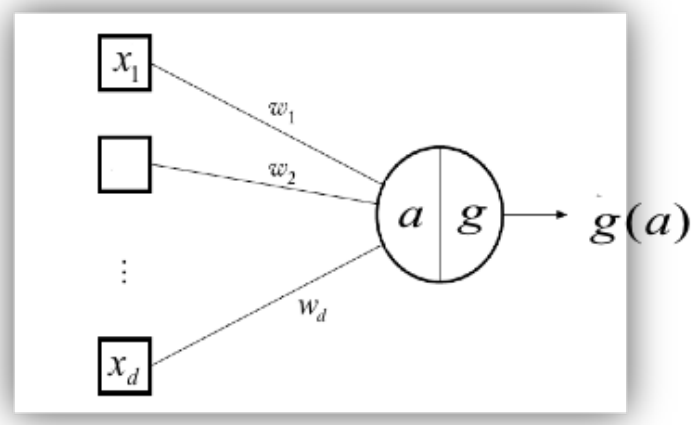
A single Neuron

One Neuron

- Elementary computation

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

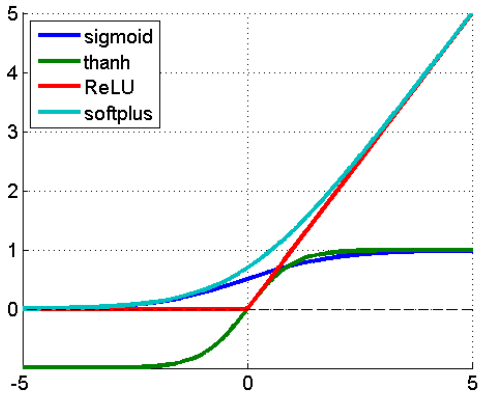
$$\text{output} = g(a(x))$$



Non linearity : g

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

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



Multi Layer Perceptron (MLP)

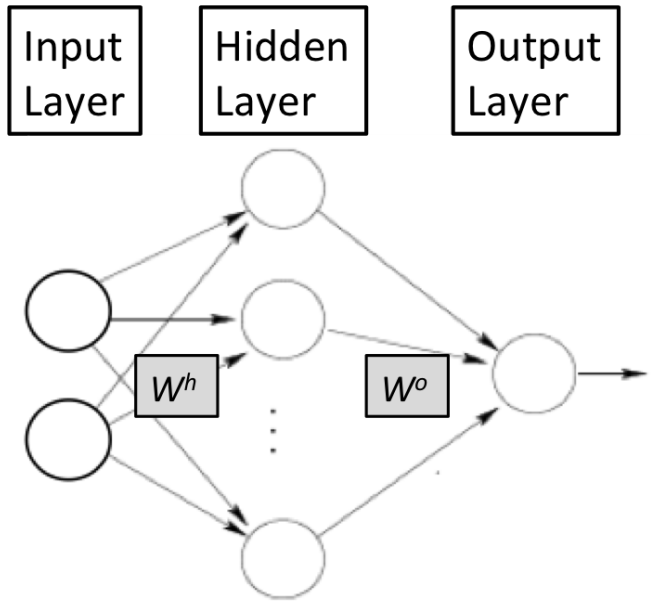
Structure

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

Function implemented by a MLP

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

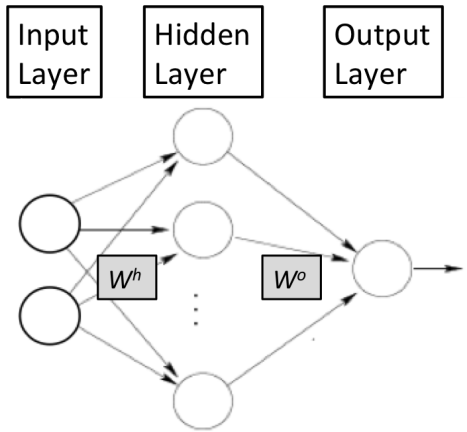
- Inference: Forward propagation from input to output layer



MLP : Forward propagation

Forward propagation of activities, for an input example x

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



MLP Usage for Regression

Notation

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

Training

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

Inference

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

MLP Usage for Classification

Training

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

Training

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

Learning a MLP

Learning as an optimization problem

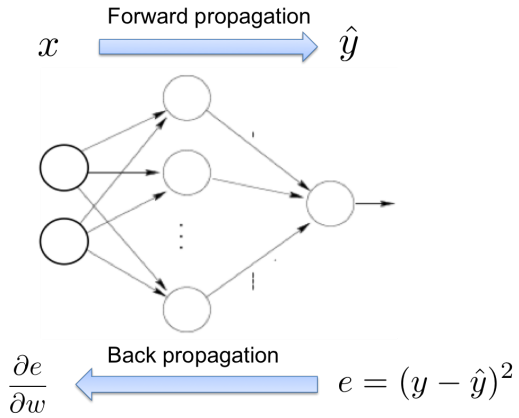
- Objective function of parameters set w for a given training set T

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

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

Backpropagation

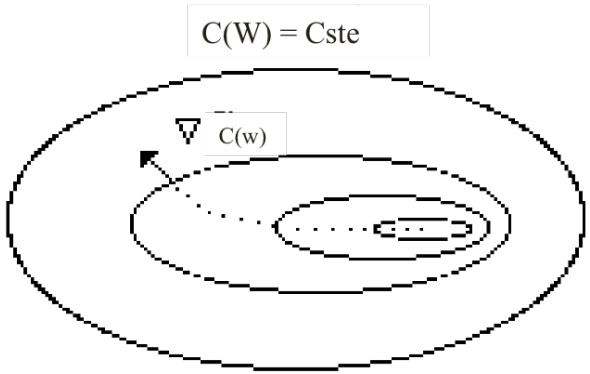
- Use chain rule for computing derivative of the loss with respect to all weights in the NN



Gradient Descent Optimization

Gradient Descent Optimization

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



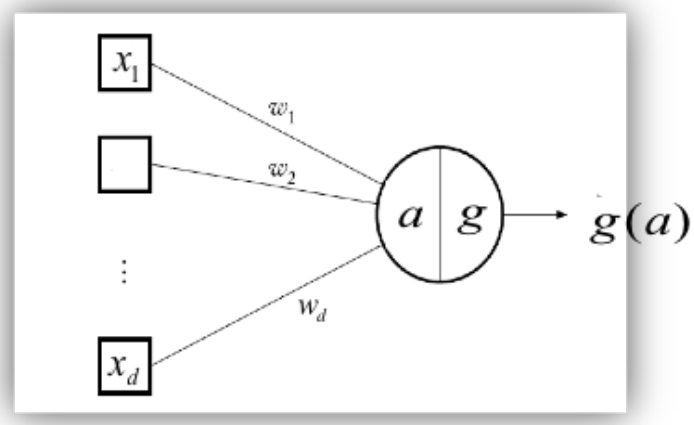
A single ReLU Neuron

One Neuron

- Elementary computation

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

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



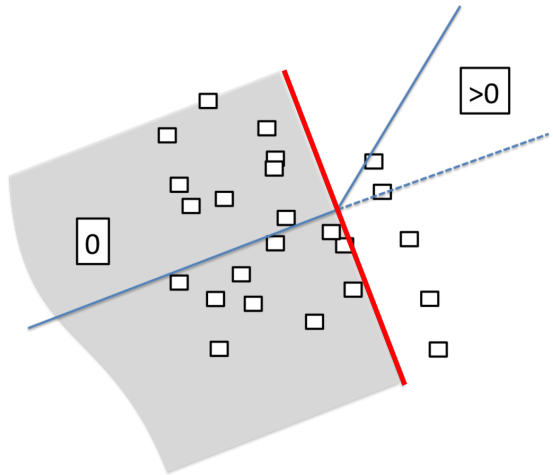
A single ReLU Neuron

One Neuron

- Elementary computation

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

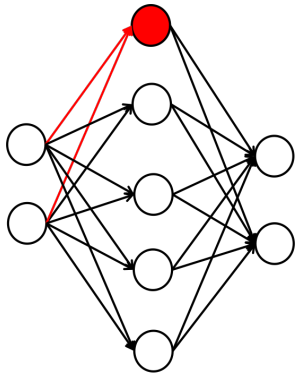
$$\text{output} = ReLU(a(x))$$

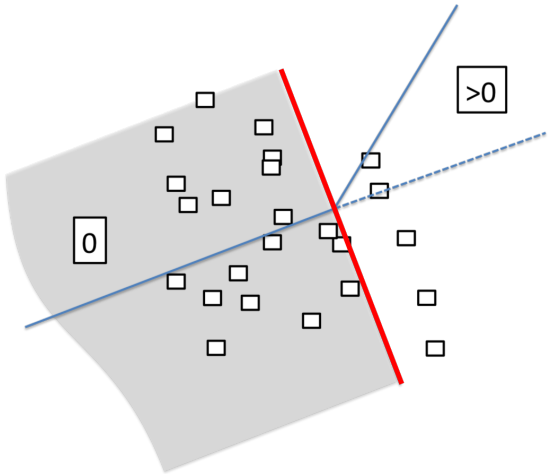


What a MLP may compute

What does a hidden neuron

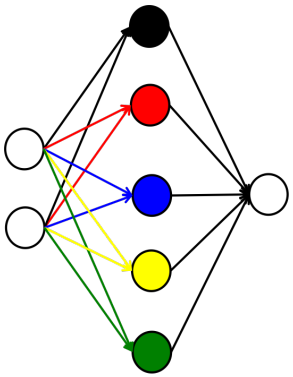
- Divides the input space in two

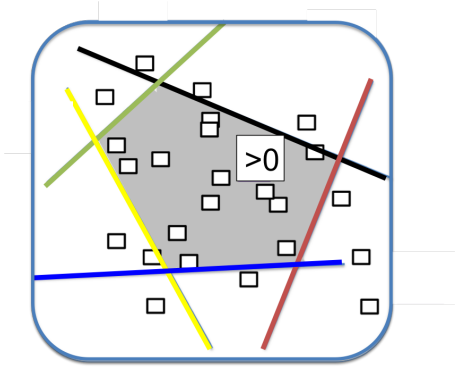




Combining multiple hidden neurons

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

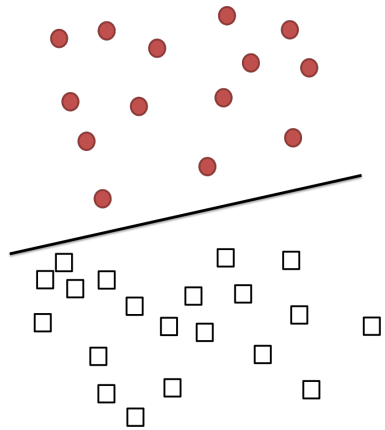




MLP and Representation Learning : Why are hidden layers useful ?

A Linear Classifier may be enough

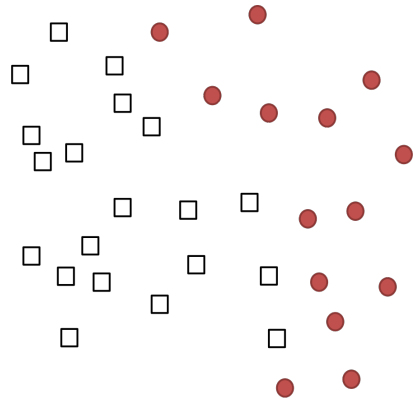
$$F_w(x) = w^T.x$$



MLP and Representation Learning : Why are hidden layers useful ?

A Linear Classifier may be enough or not

$$F_w(x) = w^T.x$$



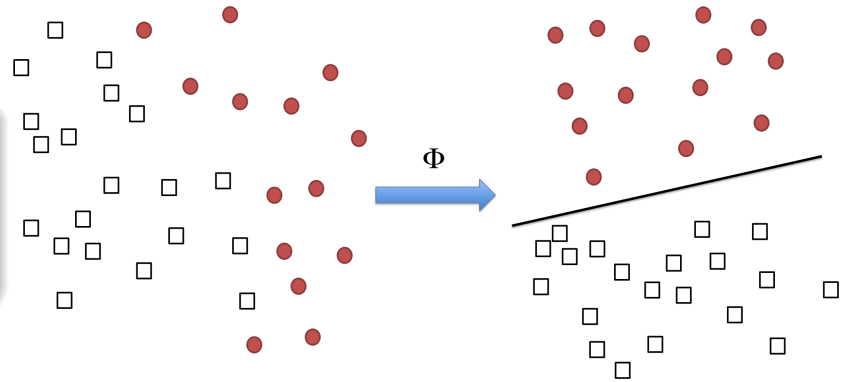
MLP and Representation Learning : Why are hidden layers useful ?

A Linear Classifier may be enough

$$F_w(x) = w^T.x$$

Otherwise use a Non linear projection followed by a Linear Classifier

$$F_w(x) = w^T.\Phi(x)$$



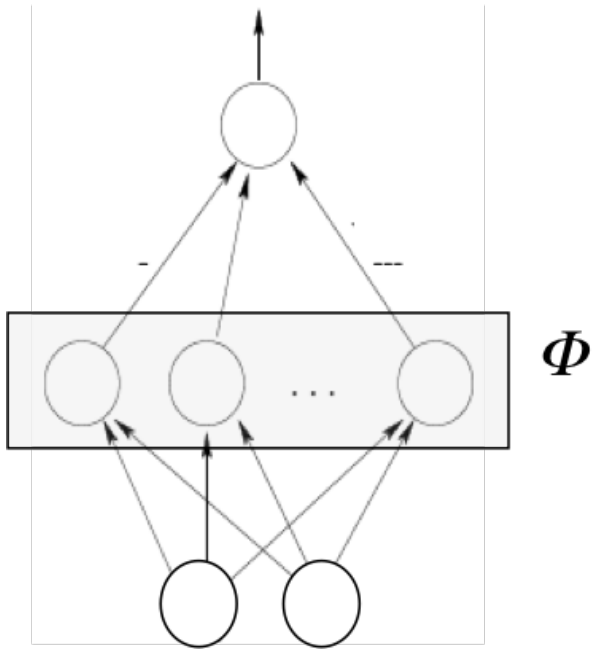
MLP and Representation Learning : Why are hidden layers useful ?

A Linear Classifier may be enough

$$F_w(x) = w^T.x$$

Otherwise use a Non linear projection followed by a Linear Classifier

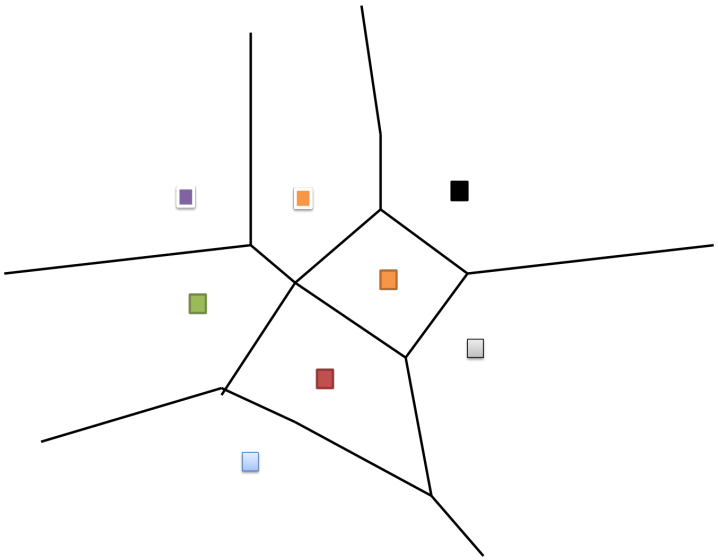
$$F_w(x) = w^T.\Phi(x)$$

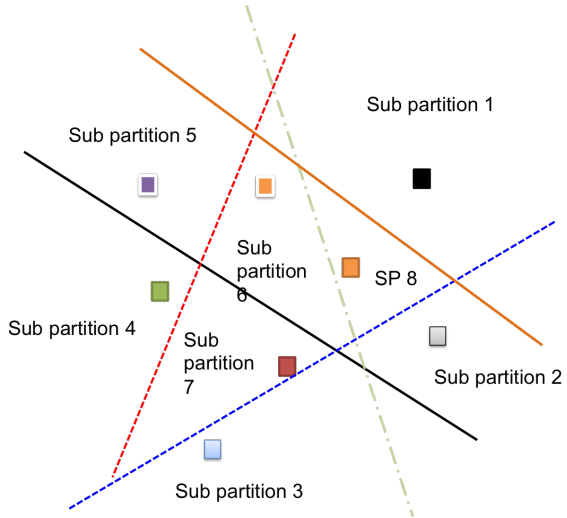


Distributed representations

Might be much more efficient than non distributed ones

Somehow the number of regions in which a NN architecture may divide the input space is a measure of its capacity





MLP = Universal approximators

One layer is enough !

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

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

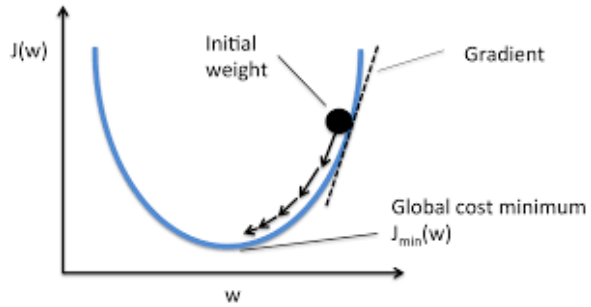
  as an approximate realization of the function f where f is independent of ϕ ; that is : $|F(x) - f(x)| < \epsilon$ for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.
- Existence theorem only
- Many reasons for not getting good results in practice

Outline

Gradient Descent Optimization

Gradient Descent Optimization

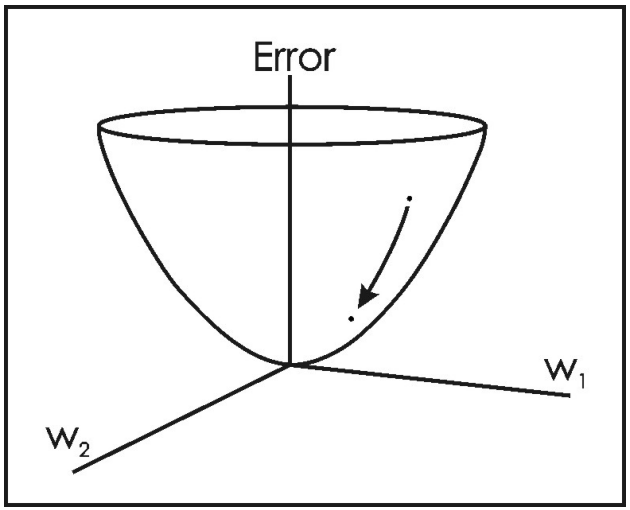
- Initialize Weights (Randomly)
- Iterate (till convergence)
  - Restimate $\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}}|_{\mathbf{w}_t}$

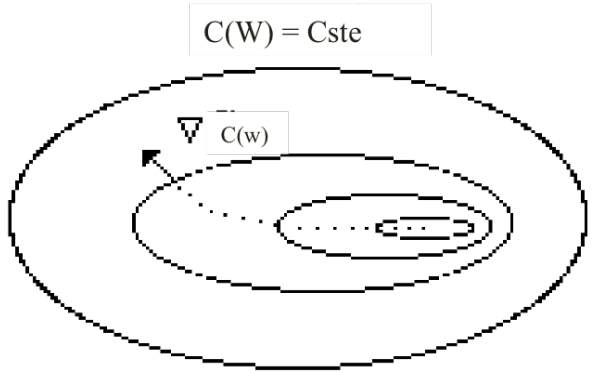


$\Rightarrow$ Few illustrations in these slides are taken from [LeCun et al, 1993], [Fei Fei Li lecture 6], and from *S. Ruder's blog*

Error surface

Surface error and gradient in weight space





Gradient Descent: Tuning the Learning rate

Two classes Classification problem

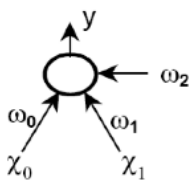
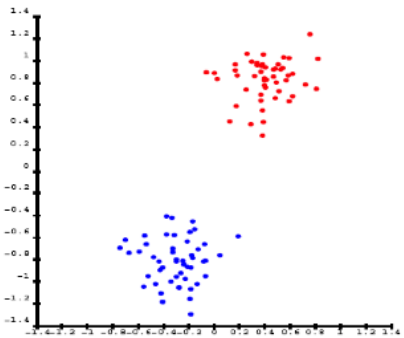
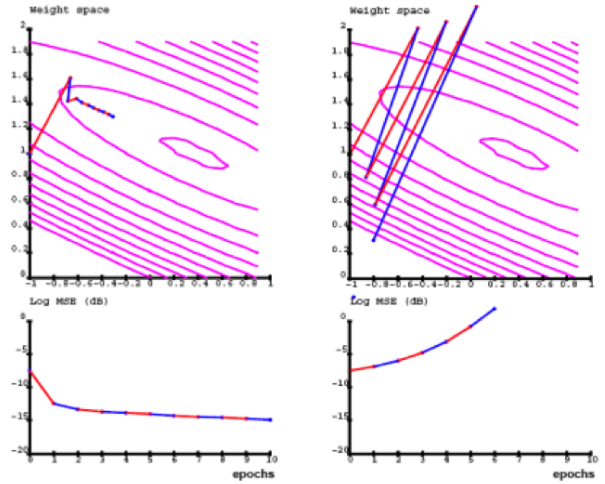


Fig. 9. Simple linear network.



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 for 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

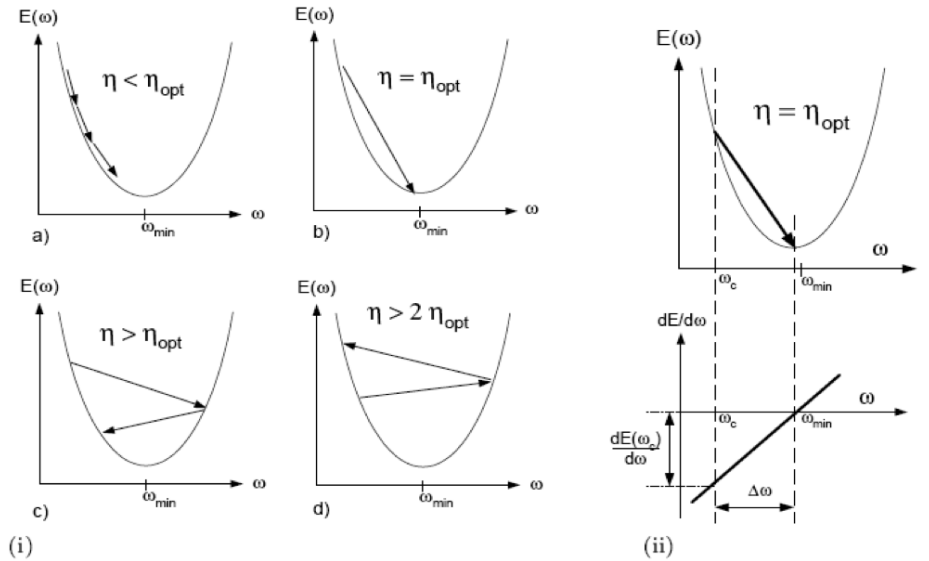


Fig. 6. Gradient descent for different learning rates.

Images from [LeCun et al.]

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)

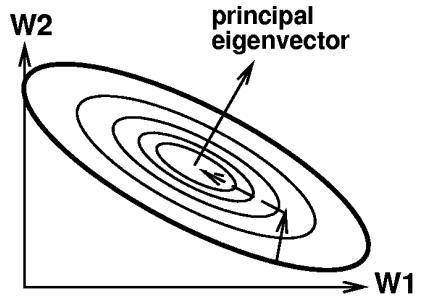
$$C(w') = C(w) + (w' - w)^T \nabla C(w) + \frac{1}{2}(w' - w)^T \nabla^2 C(w)(w' - w)$$

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

- 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
- Said differntly: Not a identical step in every direction !
- In Order 1 Gradient descent the optimal the optimal value of ϵ depends on eigen values of the Hessian $\nabla^2 C(w)$
- The optimal value depends on the highest eigen value ($\hat{\epsilon} = \frac{1}{\lambda_{max}}$) of the Hessian



From [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.
  - Optimize the GPU computation ability

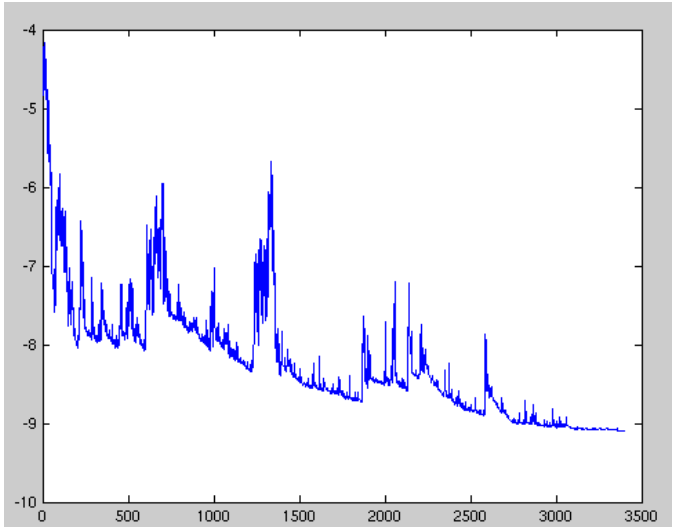
Using Momentum

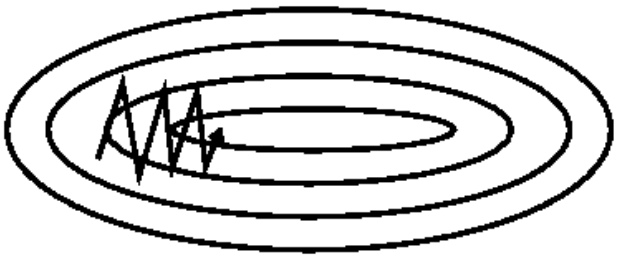
SGD with Momentum

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

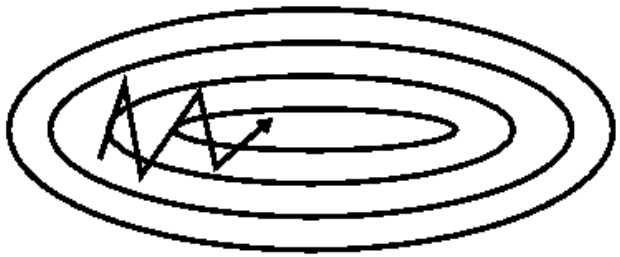
$$v = \gamma v + \epsilon \frac{\partial C(w)}{\partial w}$$

$$w = w - v$$





SGD standard



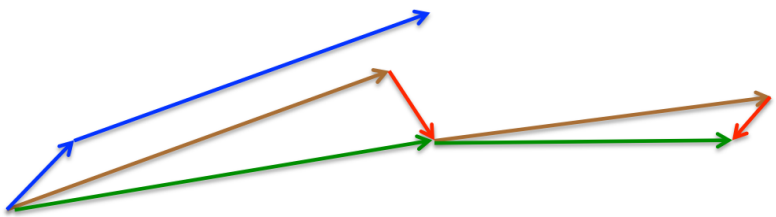
SGD avec momentum

Nesterov Accelerated Gradient

Principle

- Idea: Better anticipate when to slow down by looking forward

$$v_{t+1} = \gamma v_t + \epsilon \nabla C(w)|_{w_t - \gamma v_t}$$
$$w_{t+1} = w_t - v_{t+1}$$



- Blue vectors: standard momentum
- Brown vectors: jump
- Red vectors: correction
- Green vectors: accumulated gradient

Adagrad

Reminder: Optimally one needs to adapt the learning rate to every weight

- Define $g_{t,i} = \frac{\partial C(w)}{\partial w_i}$ the derivative wrt a single weight value w_i
- $w_{t+1,i} = w_{t,i} - \frac{\epsilon}{\sqrt{G_{t,ii}+\gamma}} g_{t,i}$
  - where $G_{t,ii}$ is a diagonal matrix with i^{th} element equal to $\sum_t g_{t,i}^2$
  - γ is a very small value to avoid numerical exceptions
  - Standard value $\epsilon = 0.01$
- Variants that aim at minimizing the aggressive feature of Adagrad: Adadelta , Adam, and RmsProp

Gradient Computation: Chain rule

Gradient of a function

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

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

Equivalent computation with the Chain rule

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

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

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

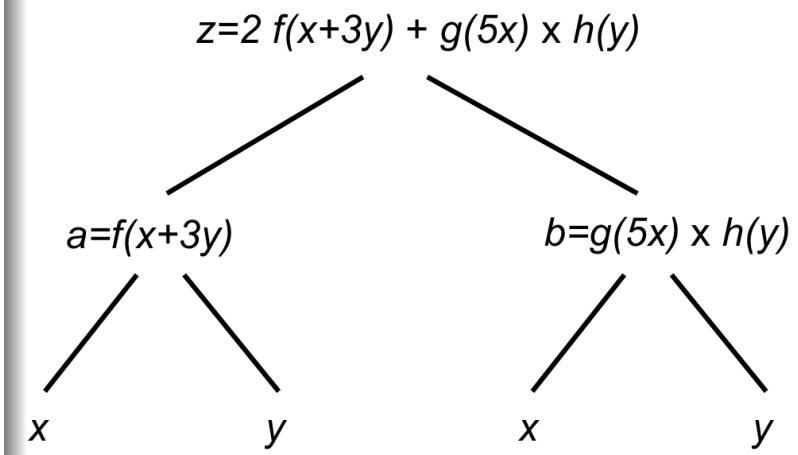
With:

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

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

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

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



Gradient computation in MLPs: Stochastic case

Notations

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

Gradient wrt. last layer weights

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

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

Gradient computation in MLPs: Stochastic case (continues)

Gradient wrt. last hidden layer (LHL) weights

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

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

Gradient computation in MLPs

Forward propagation of activities, for an input example x

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

Backward computation of the error

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

Computing gradient

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

Guiding the learning through reguarization

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

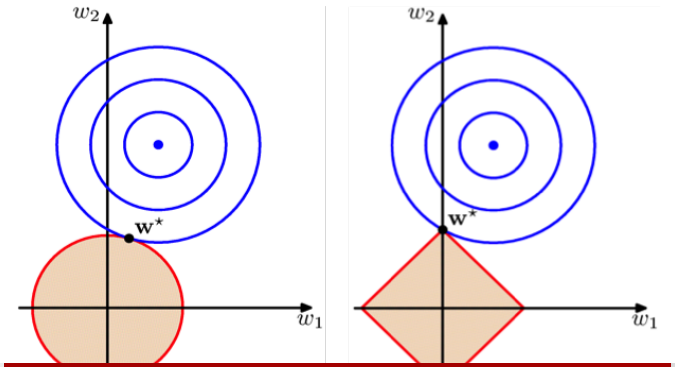
Constraints on weights

L2 norm on weights (known as Weight Decay)

- Penalizing the weights through adding a weighted L2 norm $\lambda \|w\|^2$ to the loss
- It is equivalent to defining a family of models such that $\|w\|^2 \leq C_\lambda$ with C_λ increasing when λ decreases
- L2 norm penalization $\leftrightarrow$ diminishing the space of functions implemented with the network architecture

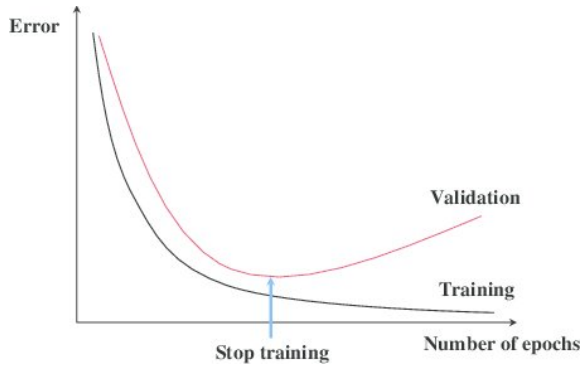
L2 and L1 norms

- L2 norm move useless weights to 0 (without reaching 0)
- L1 norm set useless weights to 0



Early stopping and callbacks

Principle

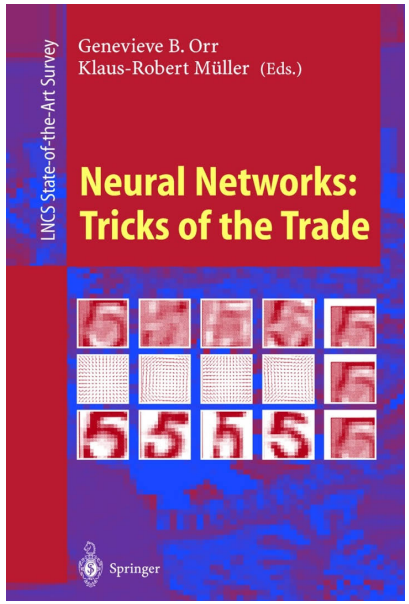


- Early stopping monitors performance (loss) on validation set
- Stopes before it reaches a plateau and starts increasing
- Related to the idea that the implemented model's capacity increases with the number of iteration
  - Think of small weights initialization and sigmoid activation
  - $\Rightarrow$ at the beginning the model is a linear one !

Lots of tricks to favor convergence

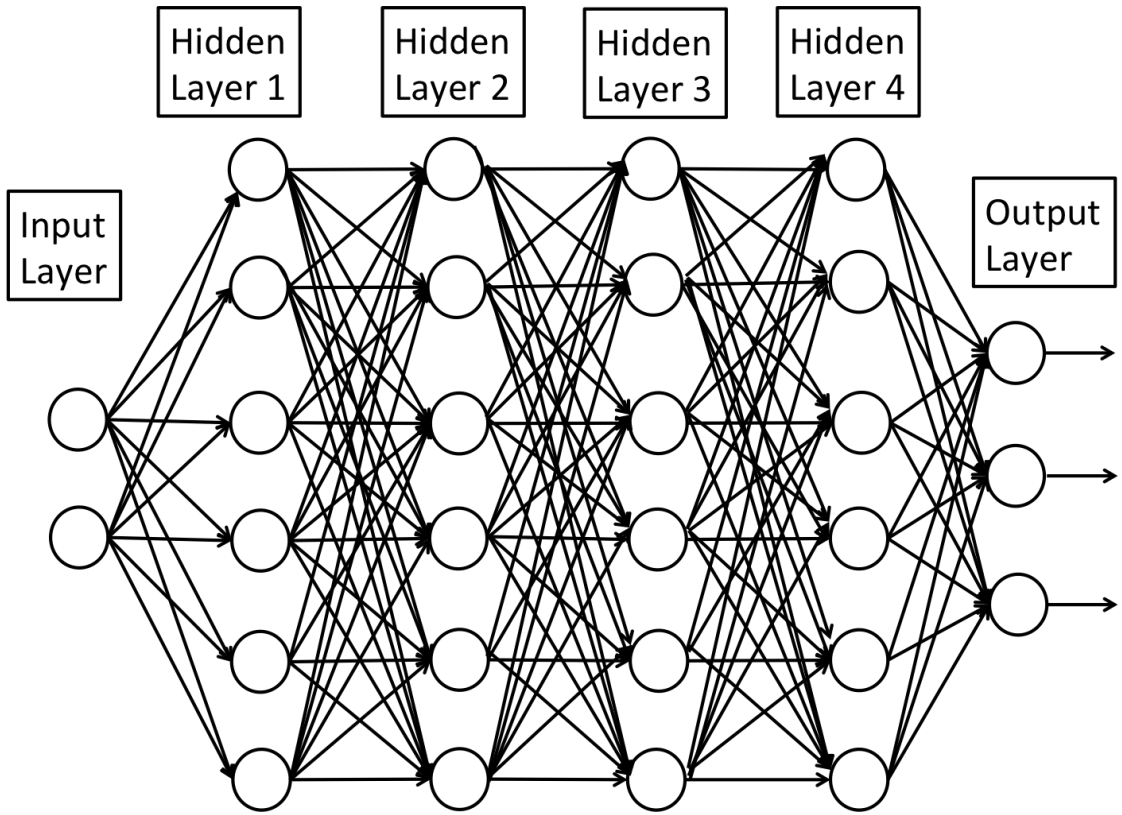
And more...

- Weight Initialization
- Gradient step setting
- ...
- $\Rightarrow$ Despite appearances not all is automatic



Outline

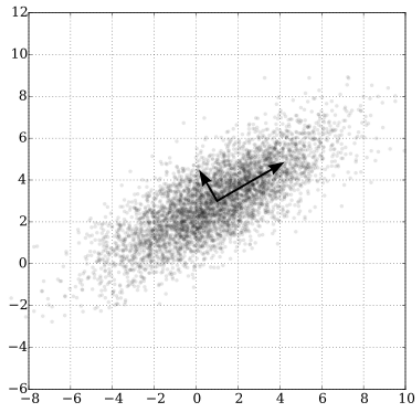
Dense architecture



Autoencoders

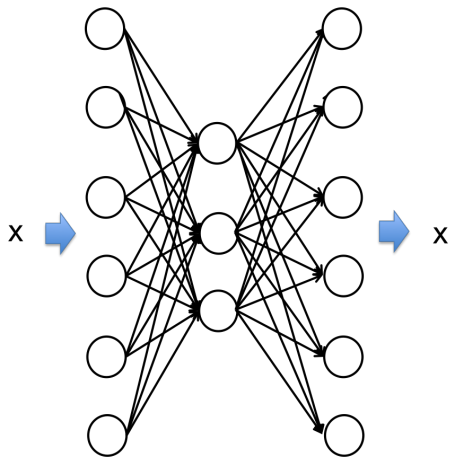
Principal Component Analysis

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



NN with Diabolo shape

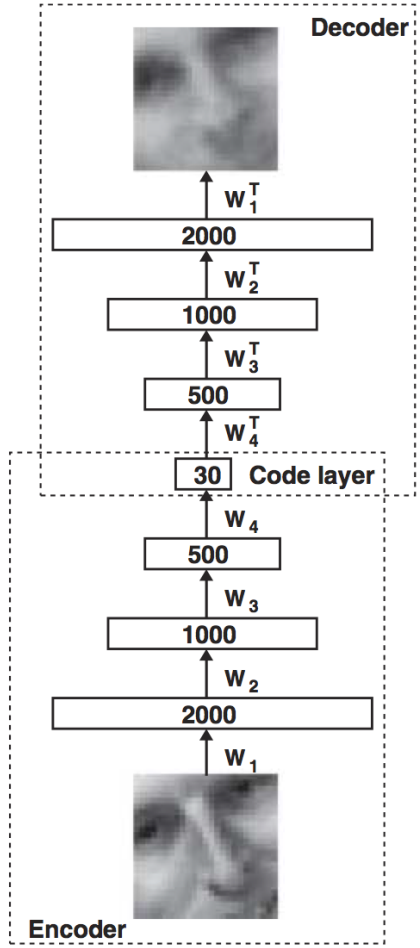
- Reconstruct the input at the output via an intermediate (small) layer
- Unsupervised learning
- Non linear projection, distributed representation
- Hidden layer may be larger than input/output layers



Deep autoencoders

Deep NN with Diabolo shape

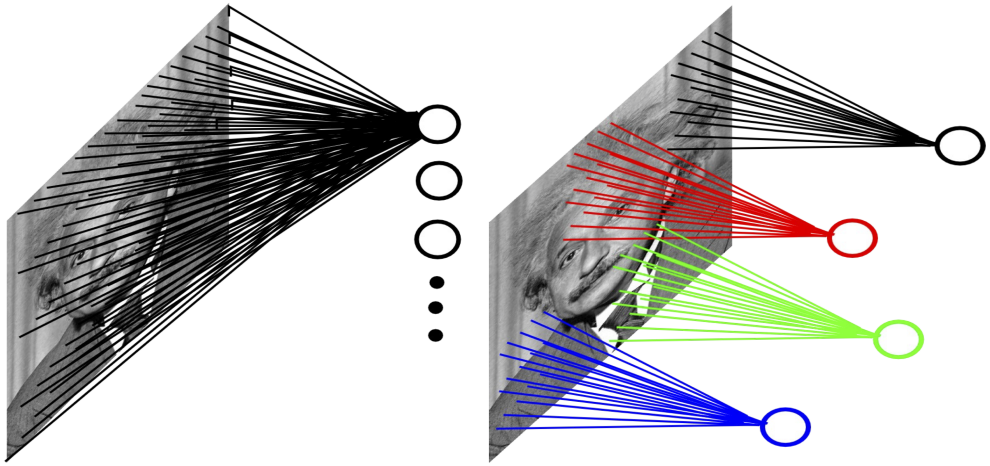
- Extension of autoencoders (figure [Hinton et al., Nature 2006])
- Pioneer work that started the Deep Learning wave



Convolutional architectures

Convolutional layers

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

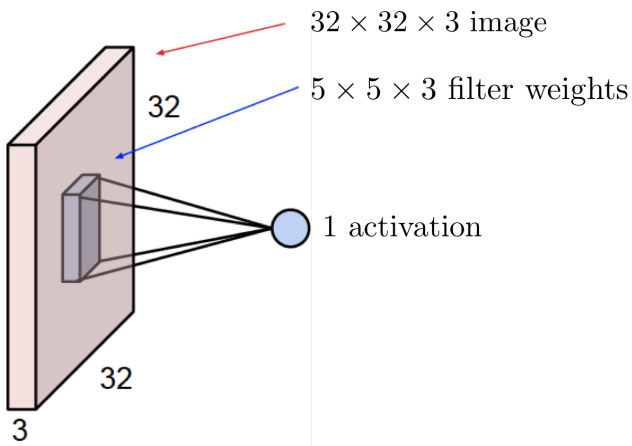


Dense vs. Locally connected

[LeCun and Ranzato Tutorial, DL, 2015]

Convolution layer

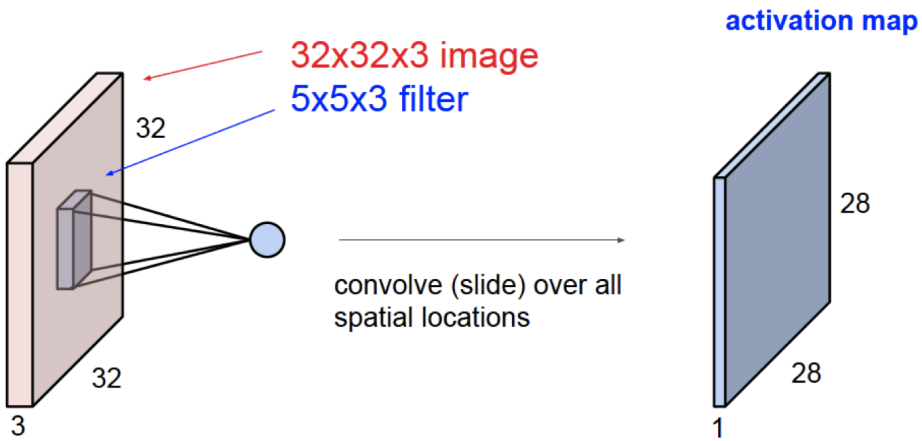
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

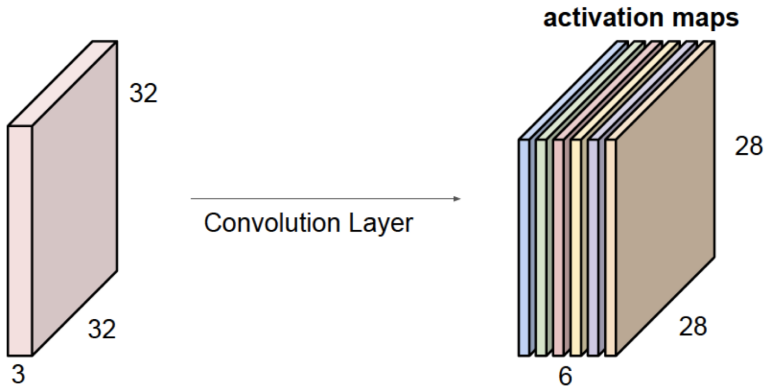
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

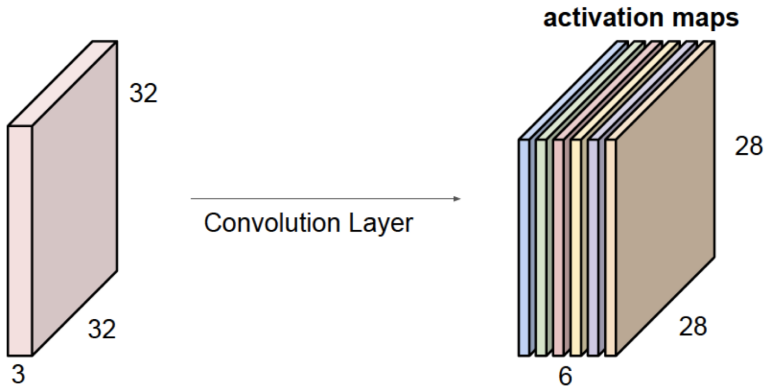
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

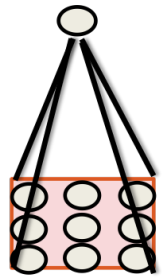
Convolution layer

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



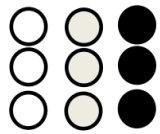
[From Fei Feil Li slides]

Example of a filter

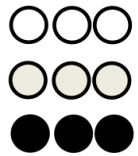


Filter weights

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



⇒ Positive output

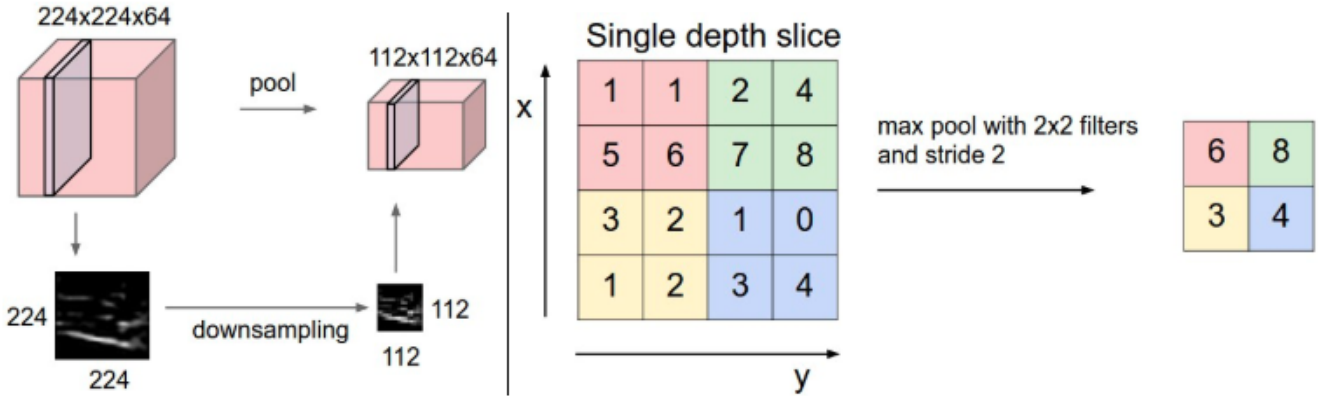


⇒Null output

Convolution layer

Aggregation layer

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

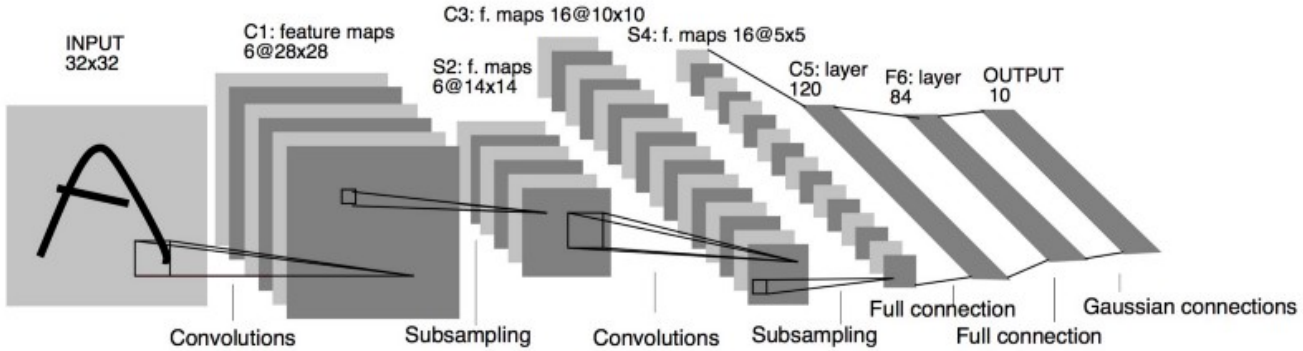


Old and new convolutional architectures

Convolution architectures

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

LeNet [LeCun and al., 1997]

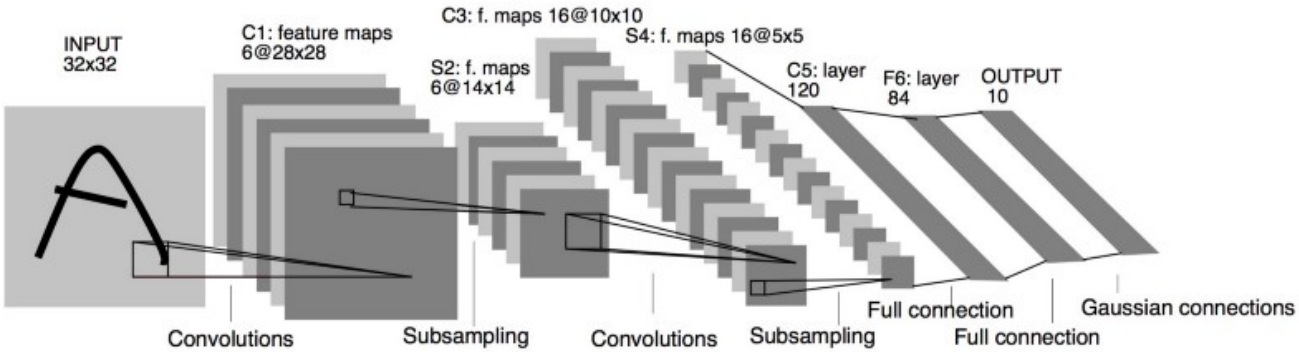


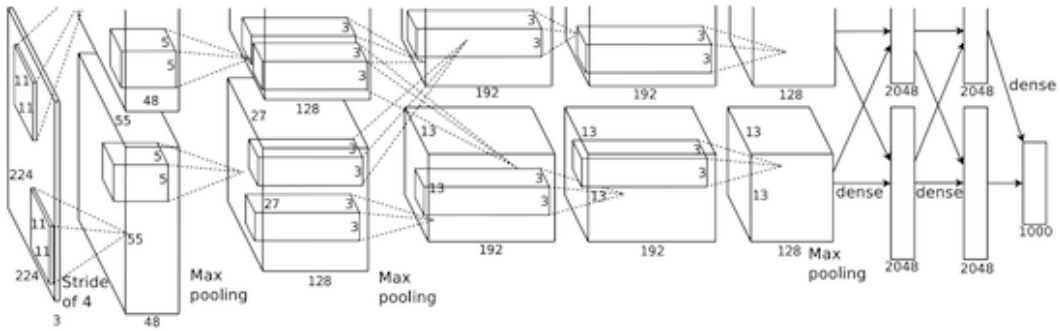
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]





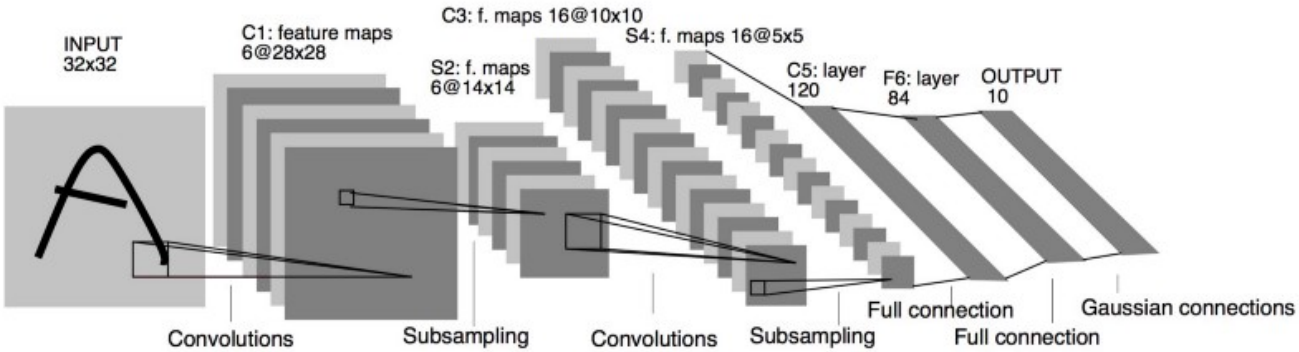
AlexNet [Krizhevsky and al., 2012]

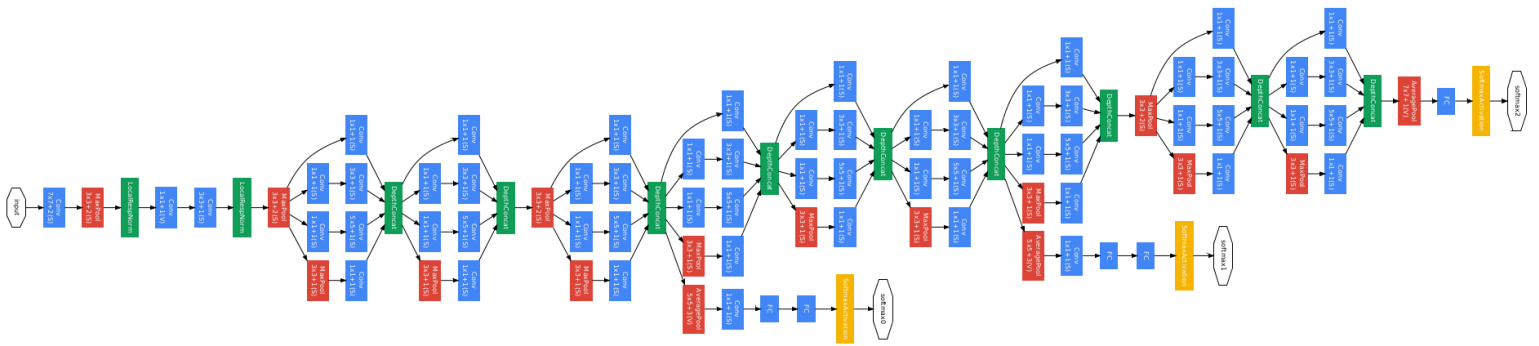
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]

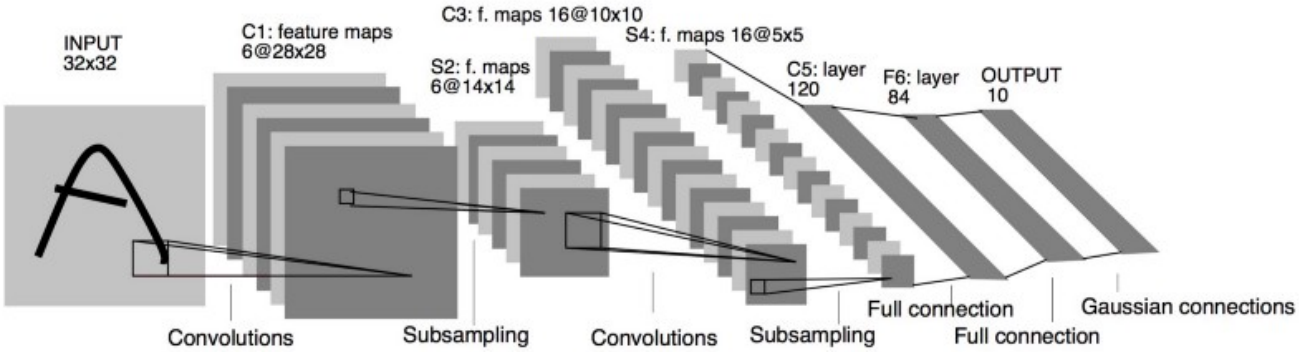


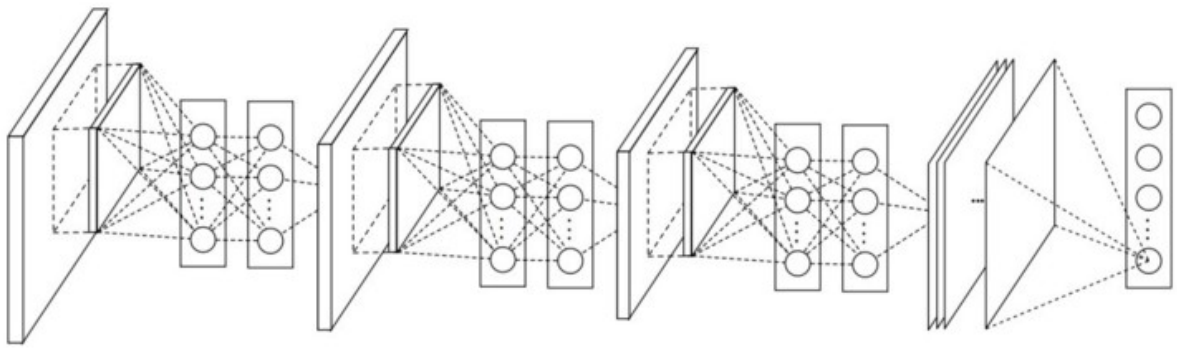


Old and new convolutional architectures

Convolution architectures

LeNet [LeCun and al., 1997]



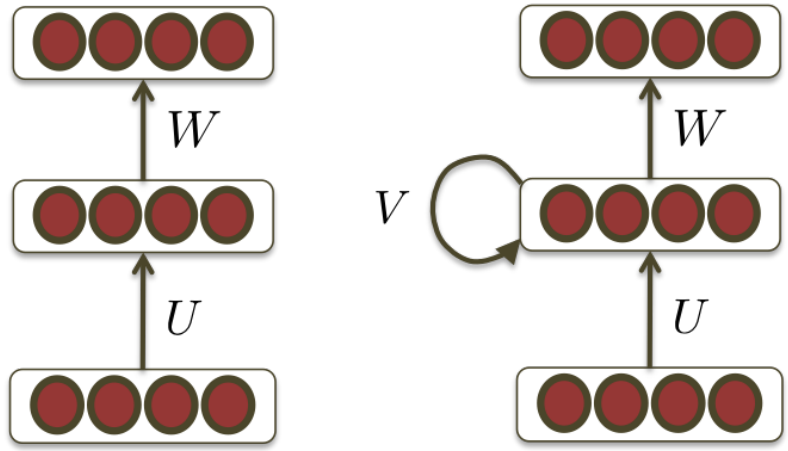


NetworkInNetwork [Lin and al.,2013]

Feedforward vs Recurrent NNs

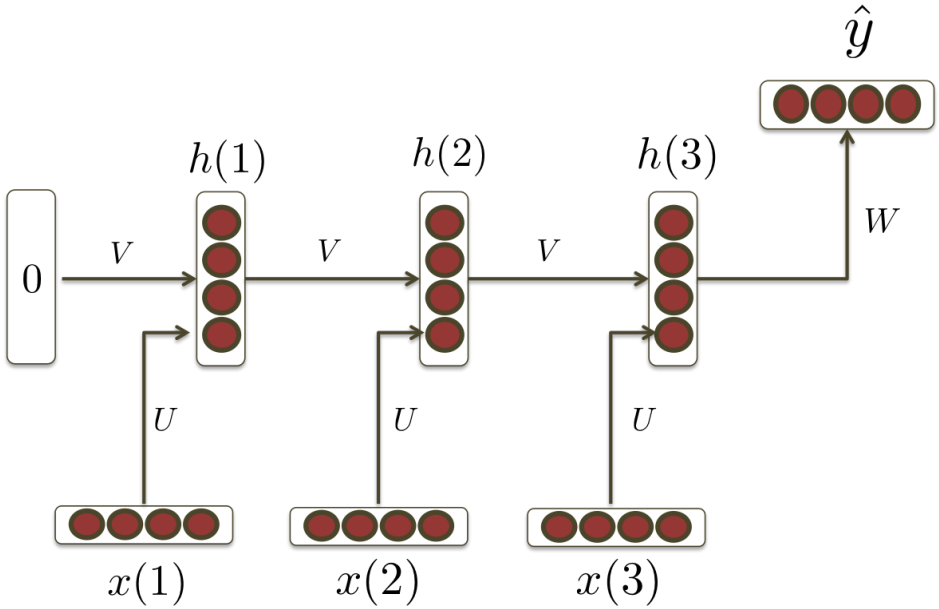
RNNs in general

- A recurrent neural network is a NN with cycles in its connections
- Today RNNs are specific recurrent architectures. Not all architectures work well..



MLP vs. RNN

Inference and learning through unfolding the RNN



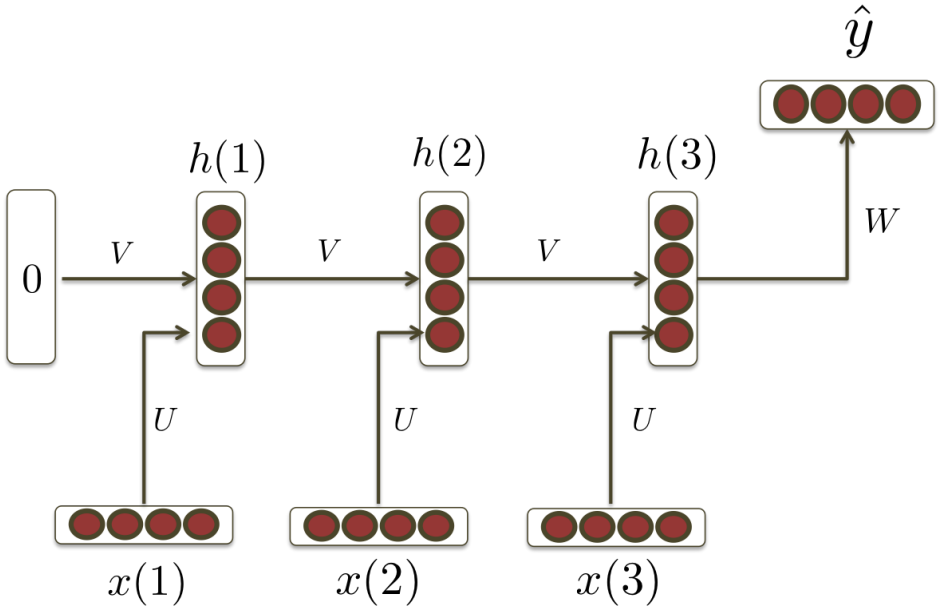
Inference: Forward propagation in the FeedForward unfolded RNN

- 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

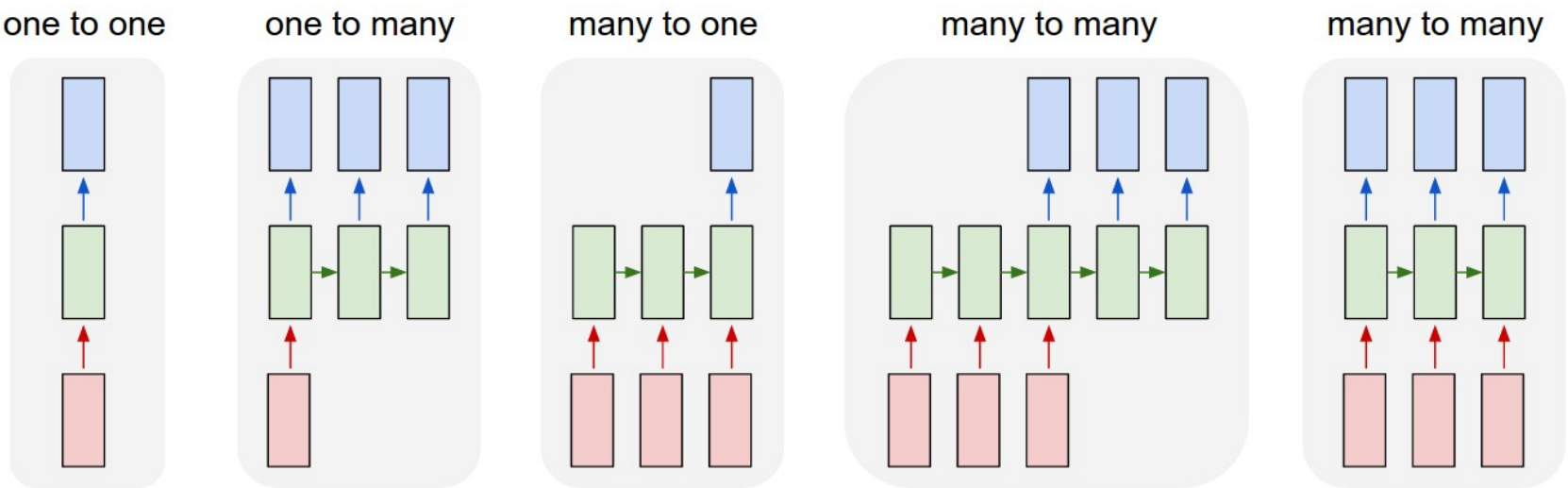
Inference and learning through unfolding the RNN



Learning: Back-propagation in the FeedForward unfolded RNN

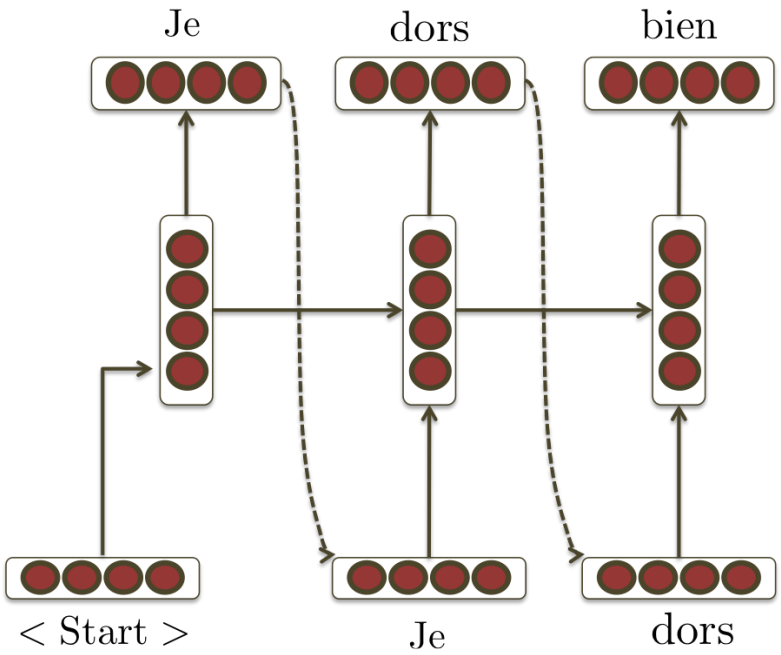
- Unfold the model
- Backpropagate the gradient in the whole network
- Sum the gradient corresponding to all shared parameters and unshared parameters (possibly the last layer)
- Apply Gradient Optimization Update rule on all parameters

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...

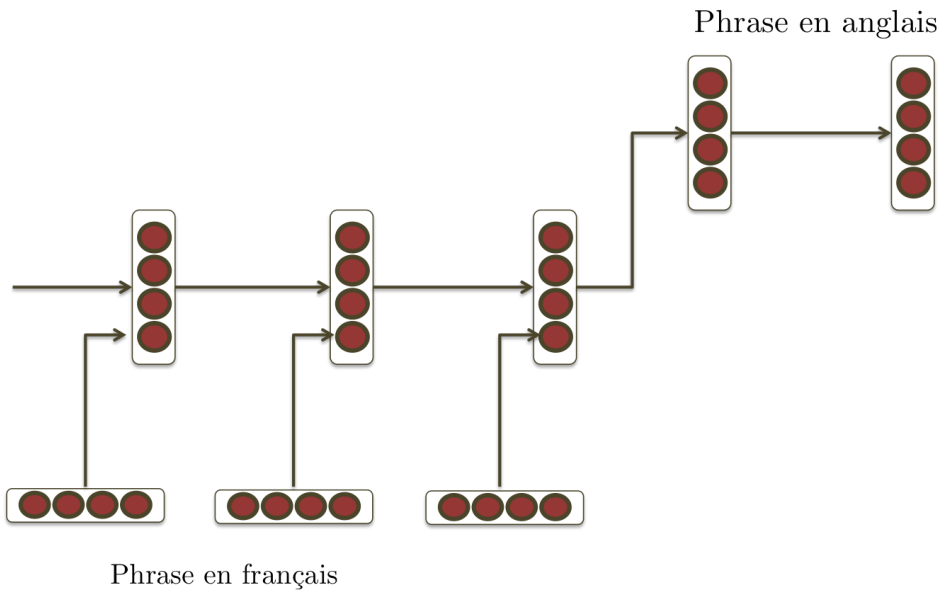
One To ManyText example



Text generation

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

Many to many example



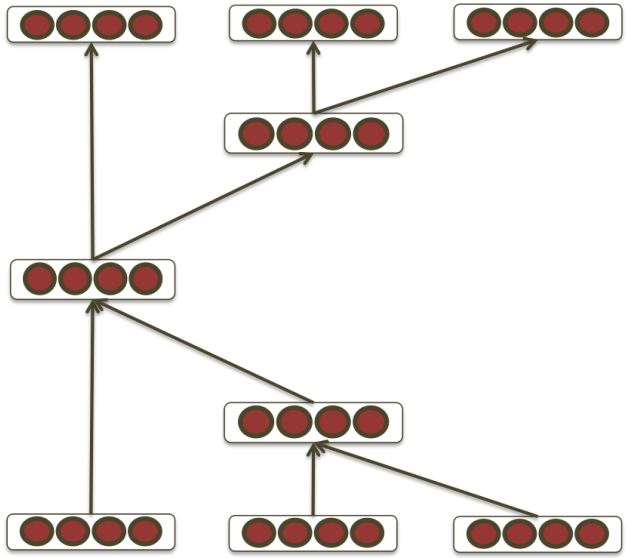
Machine translation

- 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.

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

Why now ?

Huge training resources for huge models

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

Advances in understanding optimizing NNs

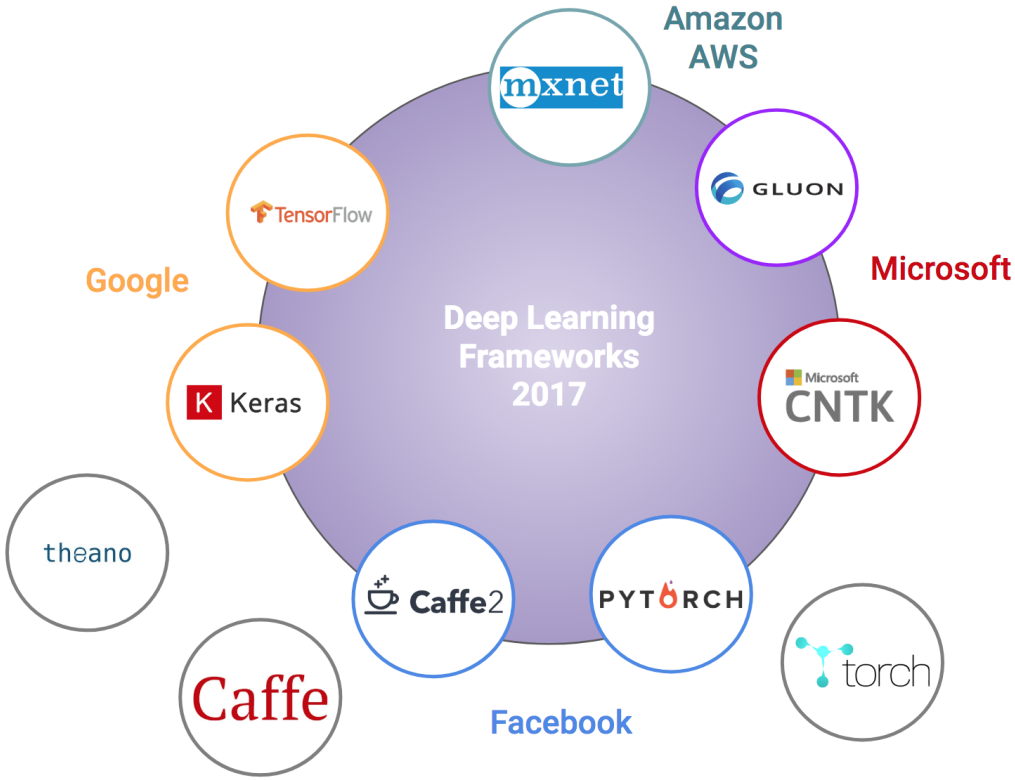
- Regularization (Dropout...)
- Making gradient flow (ResNets, LSTM, ...)

Faster diffusion than ever

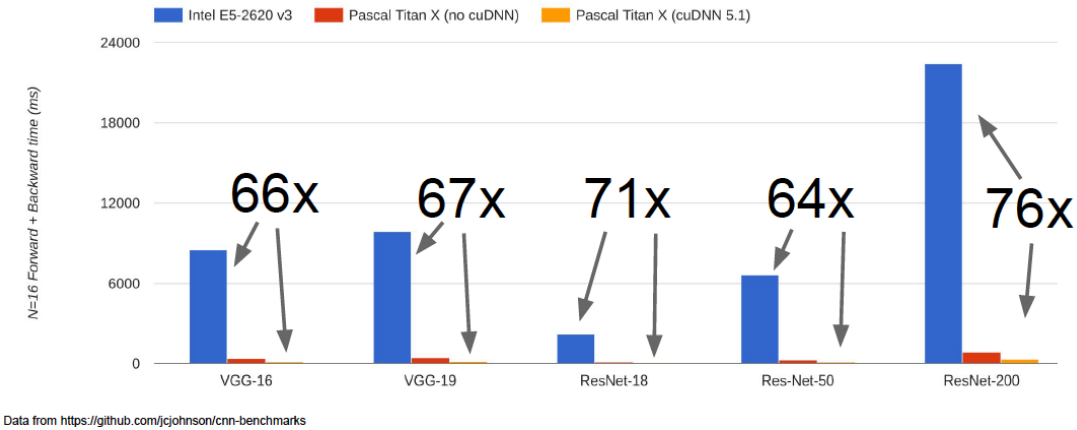
- Softwares
- Results
  - Publications (arxiv publication model) + codes
  - Architectures, weights (3 python lines for loading a state of the art computer vision model!)

Plateformes

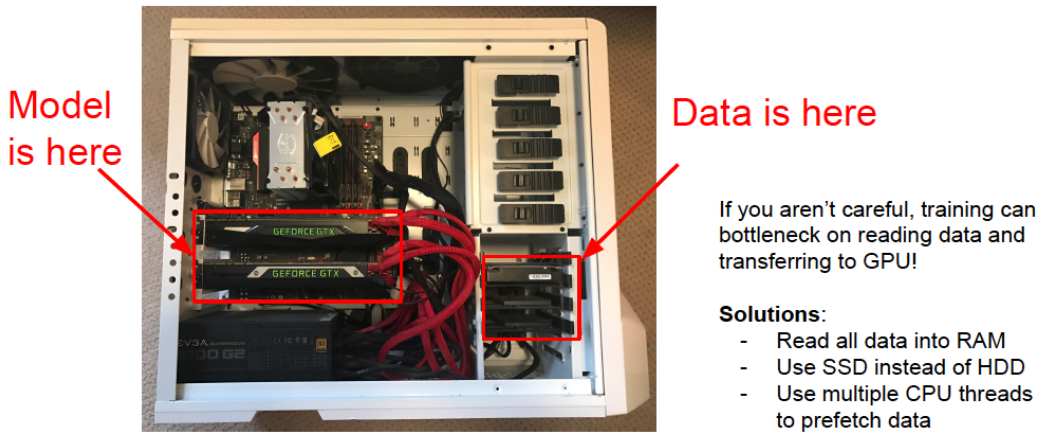
Large and active community (forums, models are available when published...) for each of these



GPU and CPU



Data from https://github.com/jcjohnson/cnn-benchmarks



If you aren't careful, training can bottleneck on reading data and transferring to GPU!

Solutions:

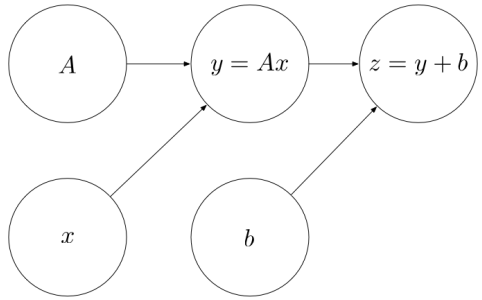
- Read all data into RAM
- Use SSD instead of HDD
- Use multiple CPU threads to prefetch data

Computation graph

Computation graph for a calculus

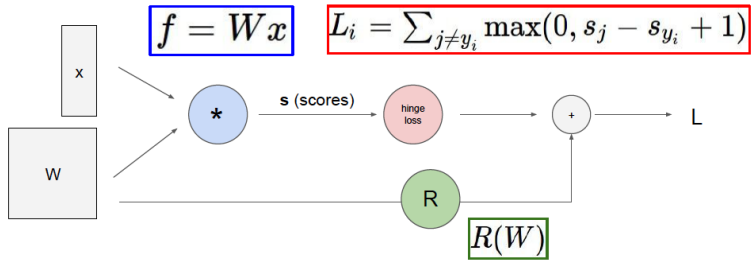
One may build a computation graph form the calculus definition

$$z = Ax + b$$



Computation graph for a calculus and a criterion

One may add a criterion (accounting for supervised learning)



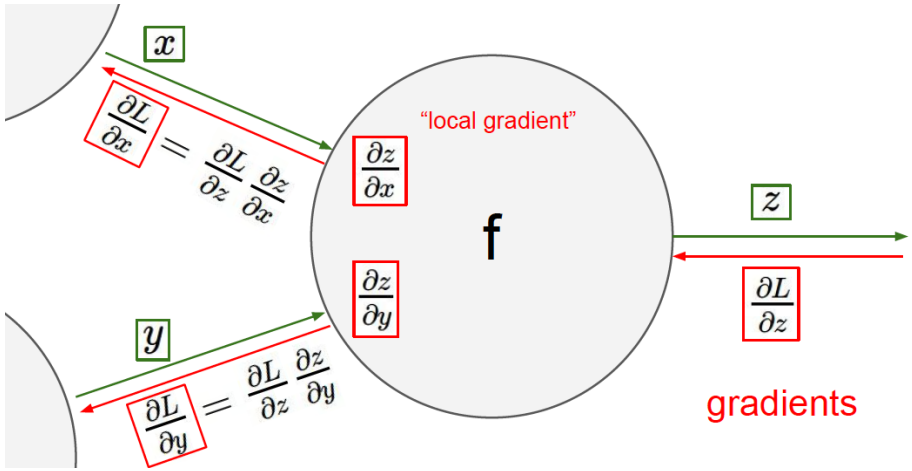
From Fei Fei Li slides

Automatic differentitation

Differentiation graph

From a computation graph one may automatically compute the backward differentiation graph !

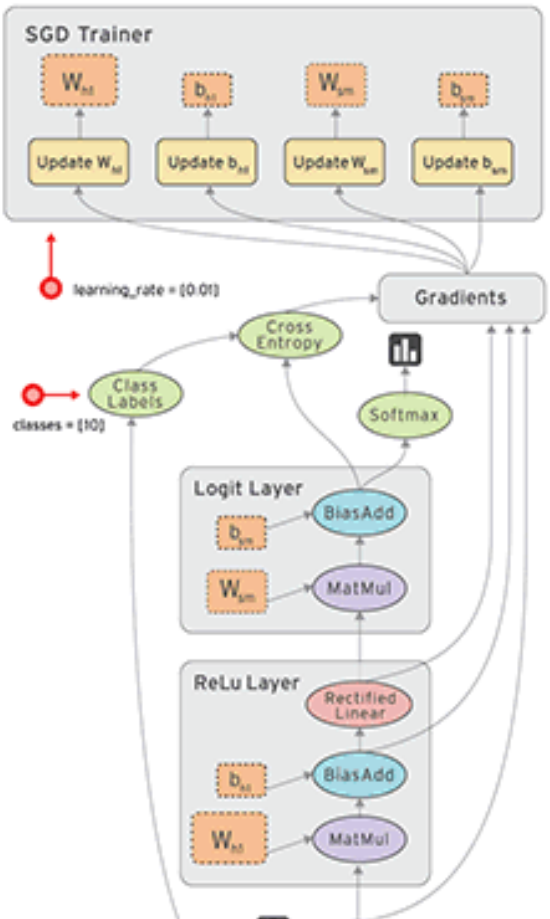
- Different rules to apply according to the operation yielding z from x and y



From Fei Fei Li slides

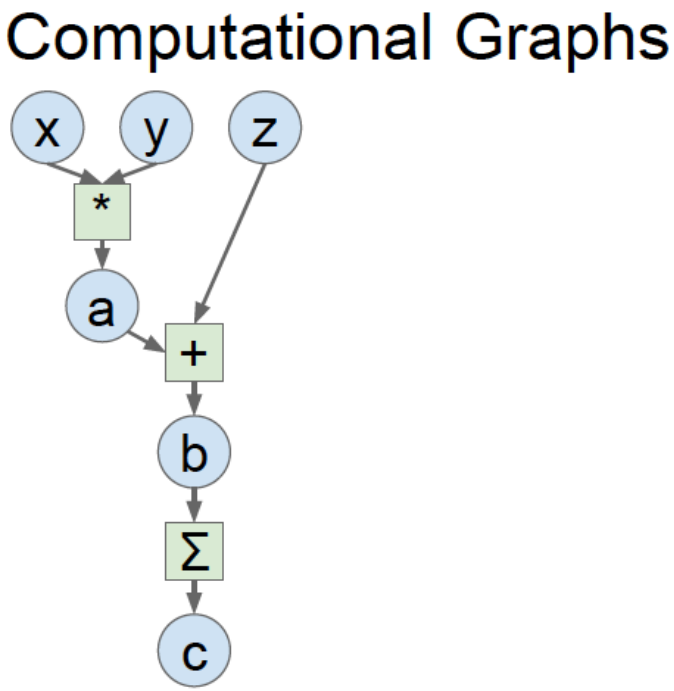
Computation graph and TensorFlow

An example from [Tensorflow doc]



Computation graph and Pytorch

Another example



PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Computation graph and Pytorch

Another example

```
[3]  import torch
     from torch.autograd import Variable
```

```
     N, D = 3, 4
     x=Variable(torch.randn(N,D).cuda(),requires_grad=True)
     y=Variable(torch.randn(N,D).cuda(),requires_grad=True)
     z=Variable(torch.randn(N,D).cuda(),requires_grad=True)
```

```
[6]  a = x* y
     b = a + z
     c = torch.sum(b)

     c.backward()

     print (x.grad.data)
     print (y.grad.data)
     print (z.grad.data)
```

```
tensor([[-2.4946, -1.7749, -2.8303, -1.0450],
        [ 1.8087, -0.8123,  1.4324, -0.7497],
        [ 0.4153, -0.7573, -0.3054,  1.8146]], device='cuda:0')
tensor([[-0.2363, -1.8247, -3.2515,  4.3729],
        [-0.6283,  1.9725, -3.6697, -1.4272],
        [ 0.8991, -0.2417, -0.2456, -2.4684]], device='cuda:0')
tensor([[2., 2., 2., 2.],
        [2., 2., 2., 2.],
```

Example (pytorch)

Mnist Classifier (model definition)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Example (pytorch)

Mnist Classifier (model training)

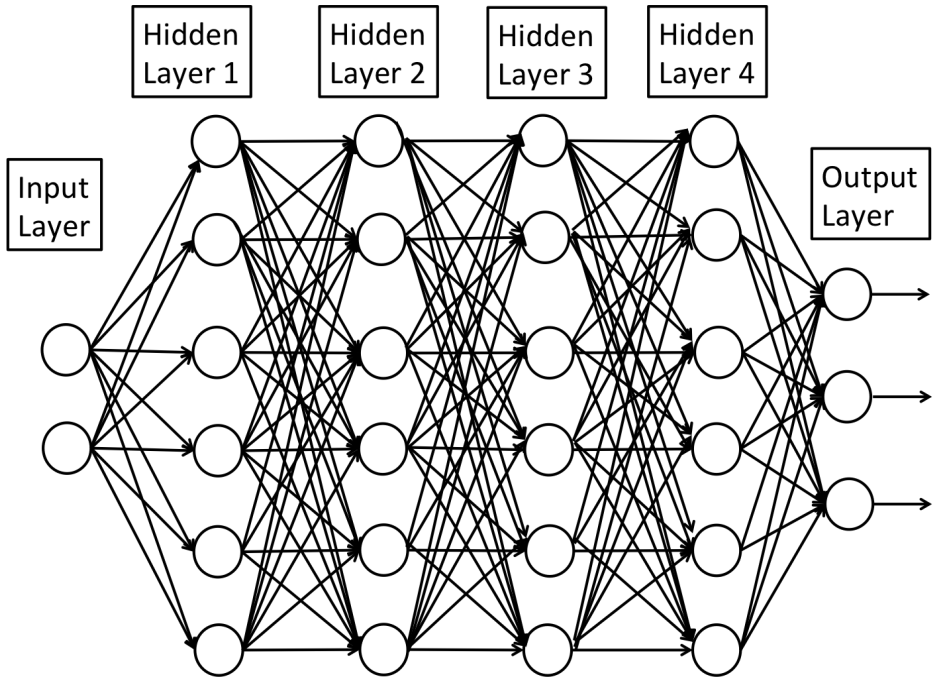
```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

Outline

Deep Learning = Representation Learning

Hierarchy of representation spaces by successive hidden layers

A series of hidden layers

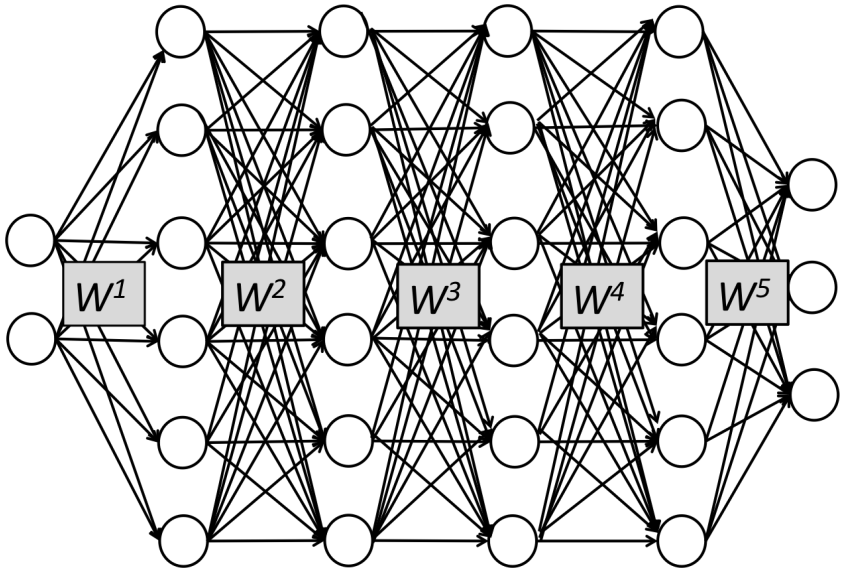


Deep Learning = Representation Learning

Hierarchy of representation spaces by successive hidden layers

Computes a complex function of the input

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

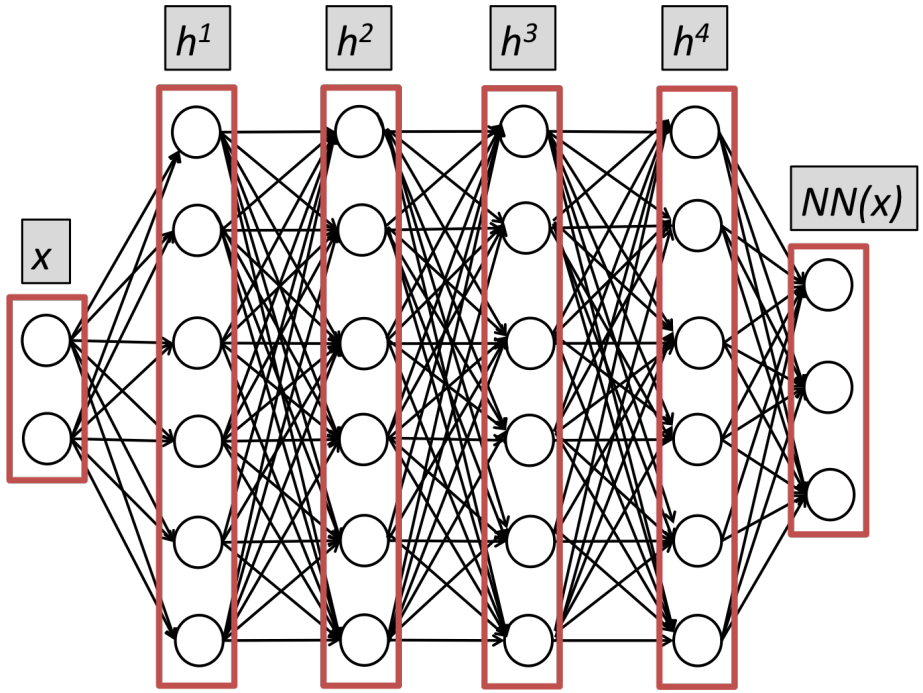


Deep Learning = Representation Learning

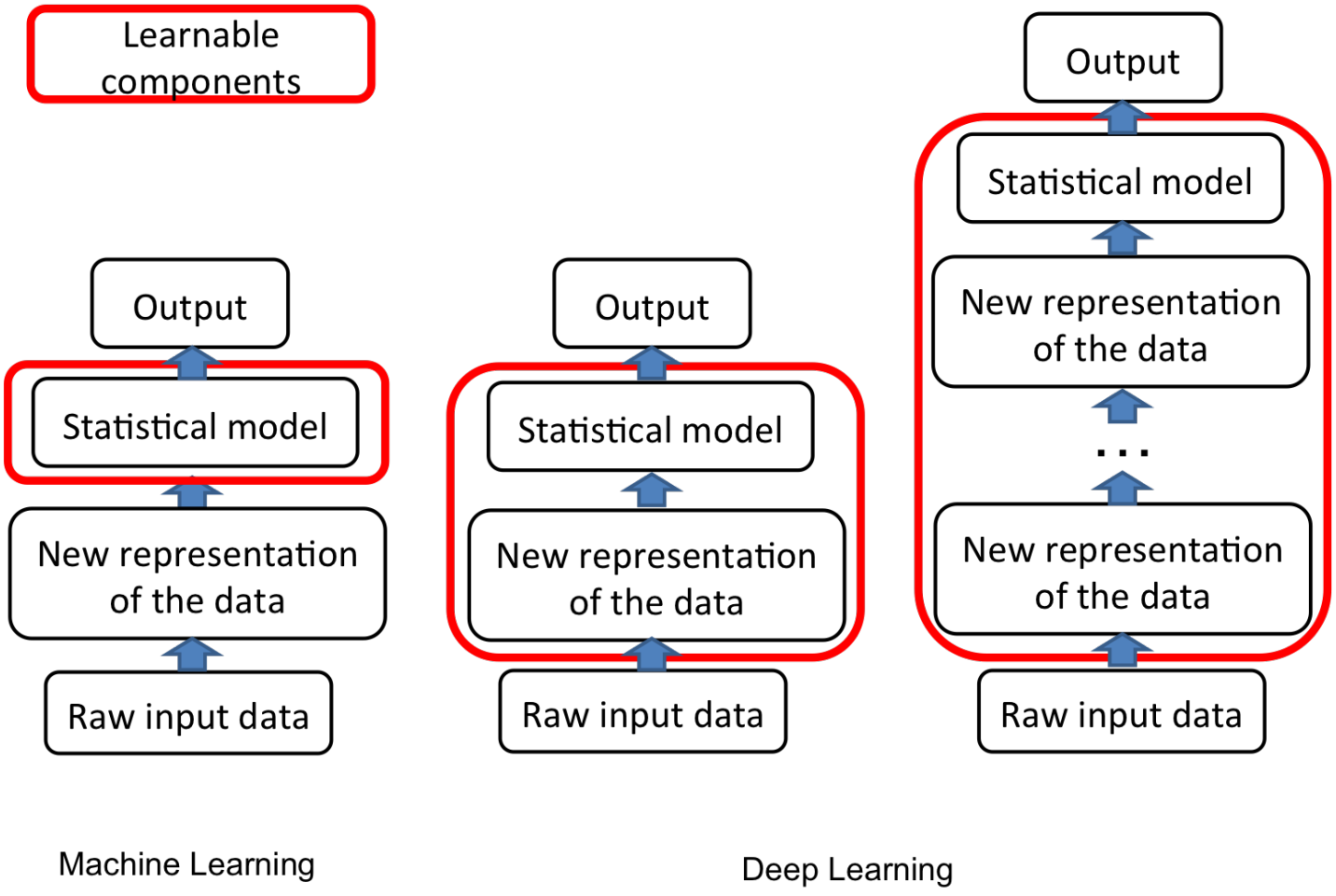
Hierarchy of representation spaces by successive hidden layers

Computes new representations of the input

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



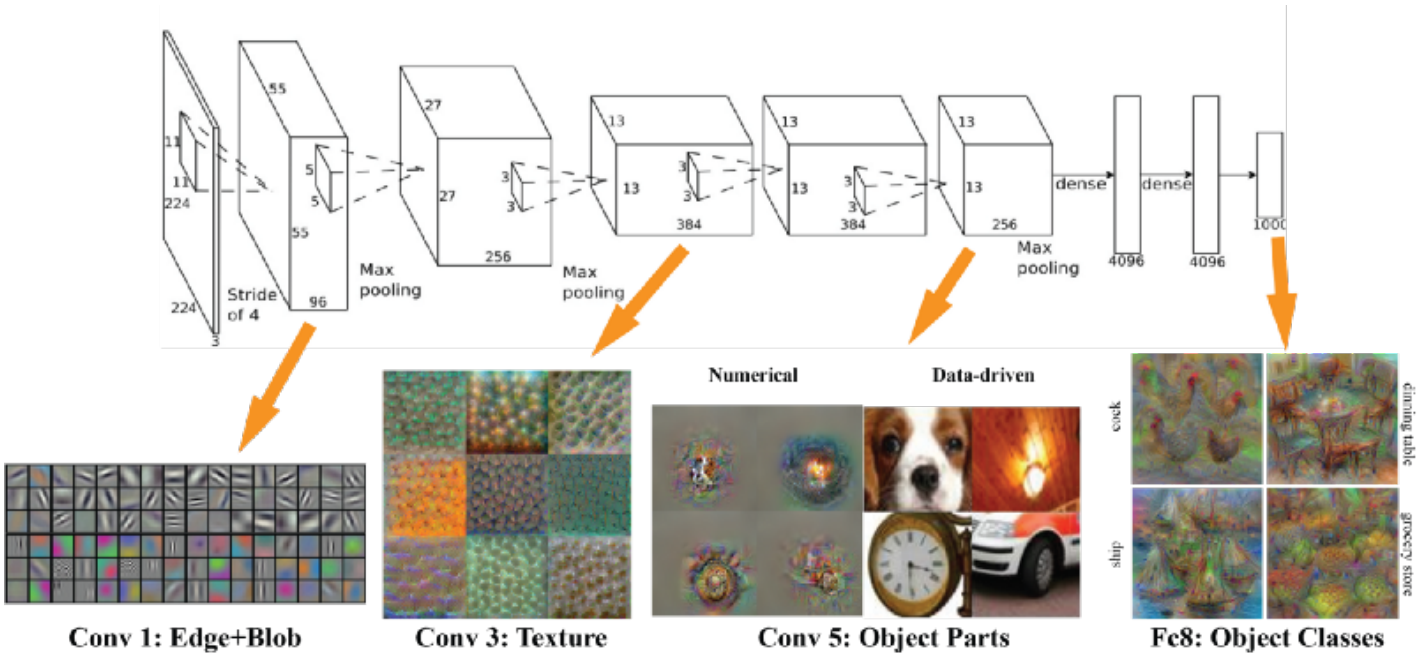
Machine Learning vs. Deep Learning



Feature hierarchy : from low to high level

What feature hierarchy means ?

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

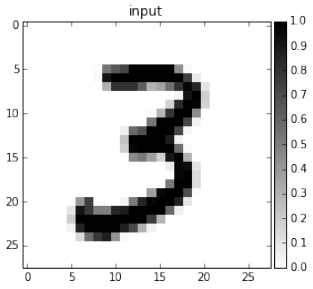


([Krizhevsky and al., 2012])

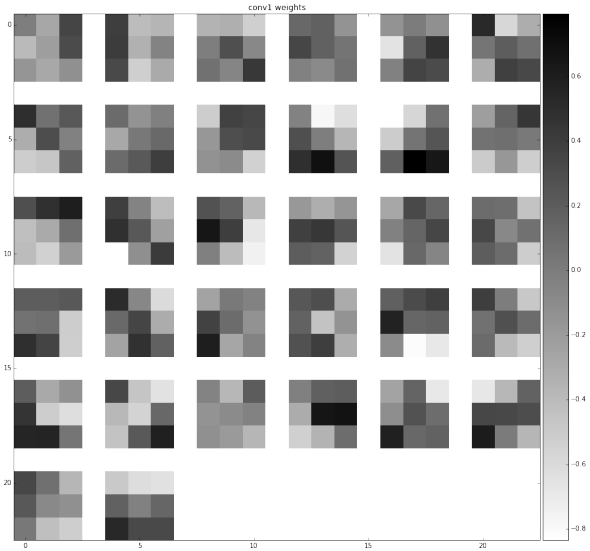
Visualizing filters and activations

Mnist (toy) dataset

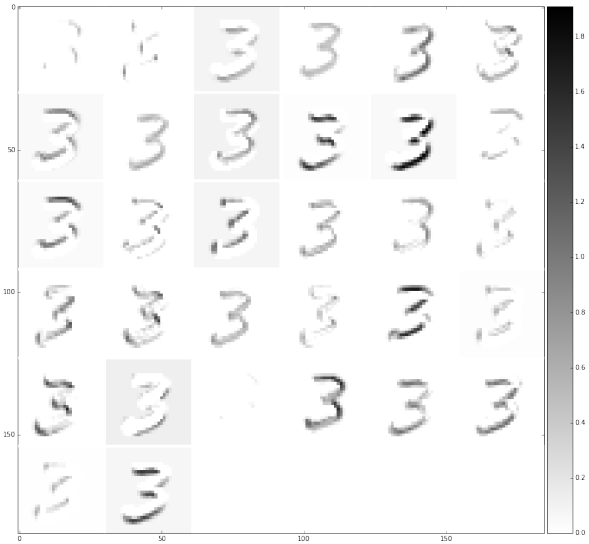
- Low resolution handwritten digit images



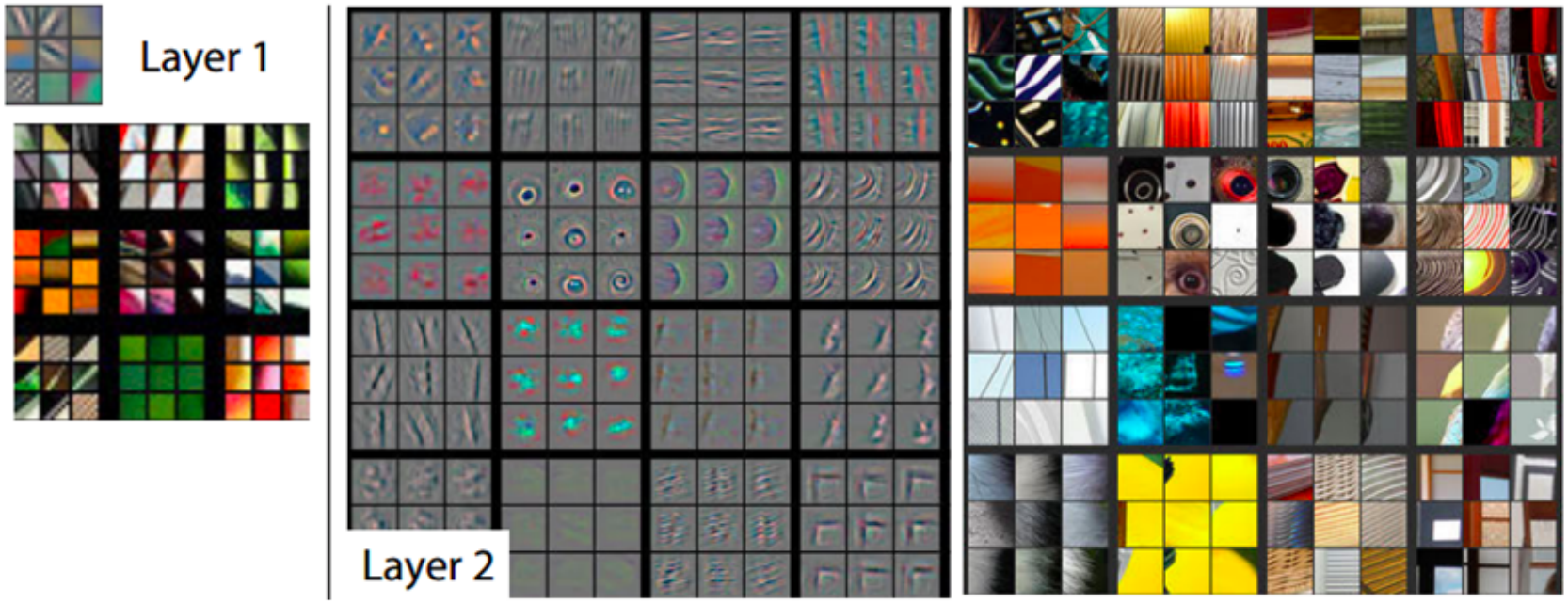
Weights of first Convolutional layer (32 maps)



Outputs of first Convolutional layer for above input



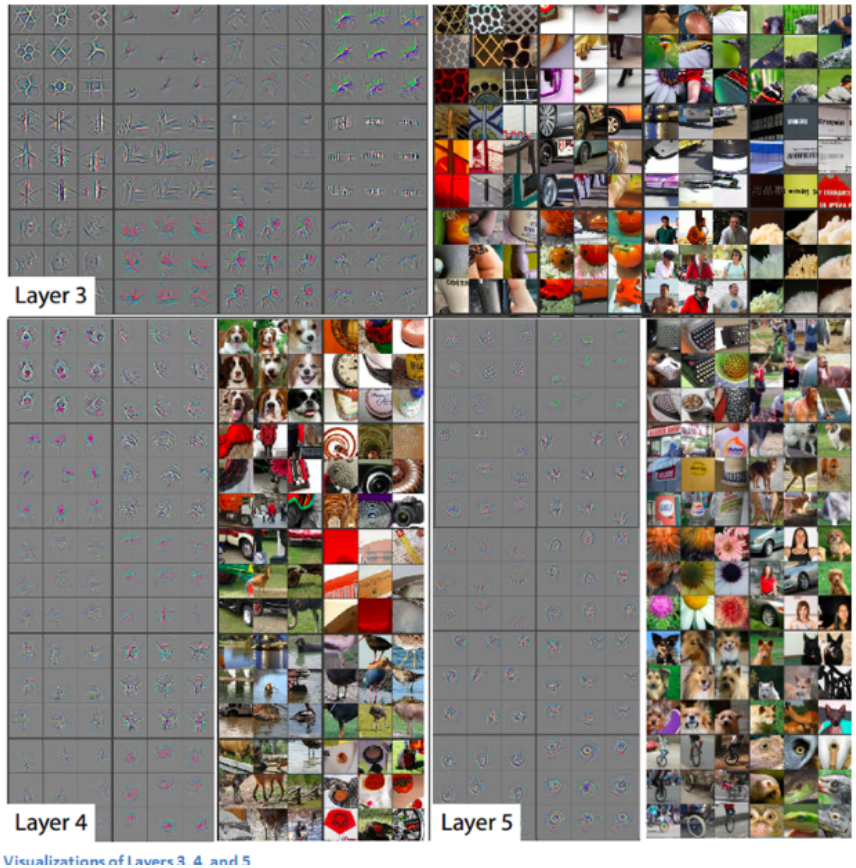
DConvnet



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]

DConvnet



[From Zeiler et Fergus]

Embedding layer for text representation

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)

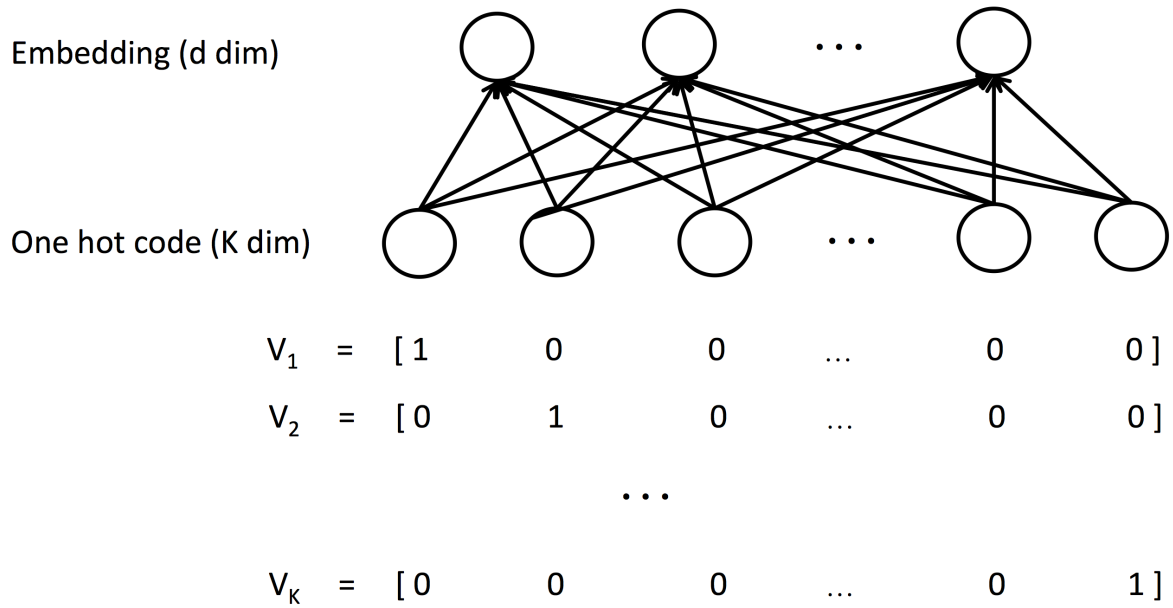
Main interests

- 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

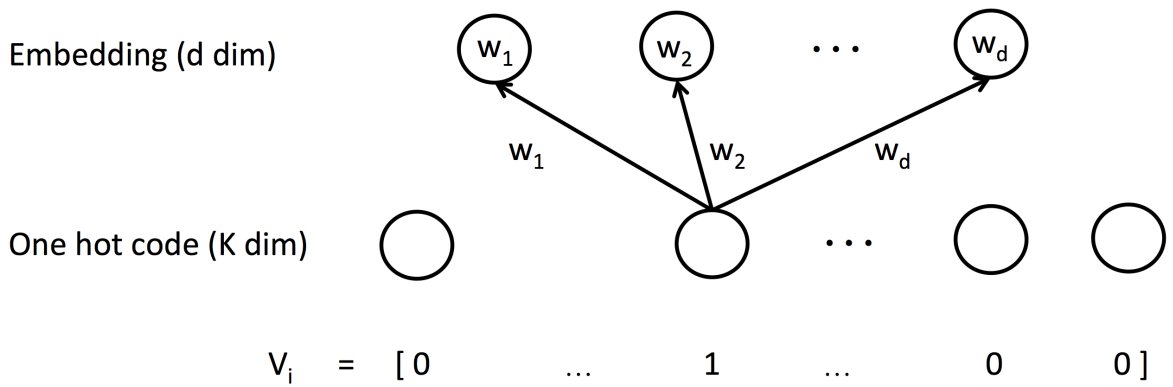
- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix ($K \times d$)



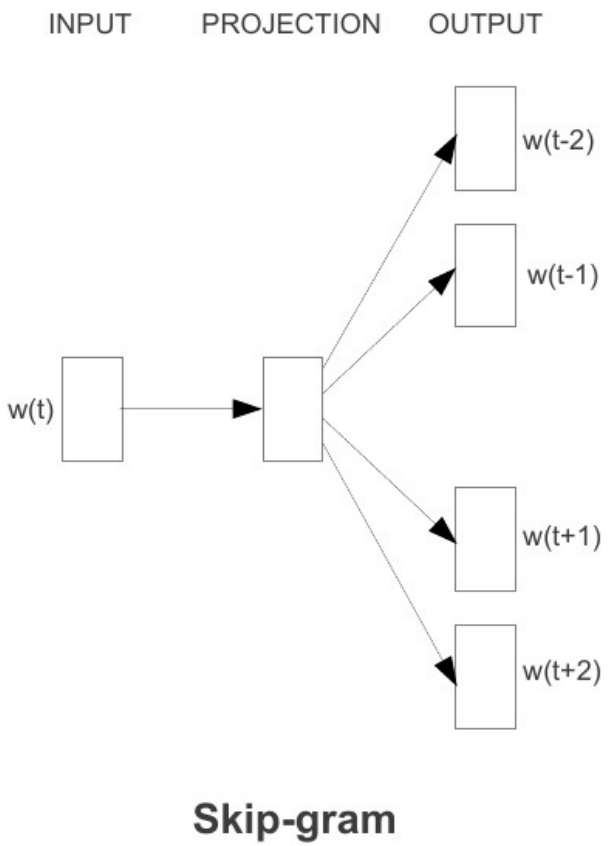
Embedding layer: Implementation

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 ($K \times d$)



Architecture SkipGram [Mikolov et al., 2013]



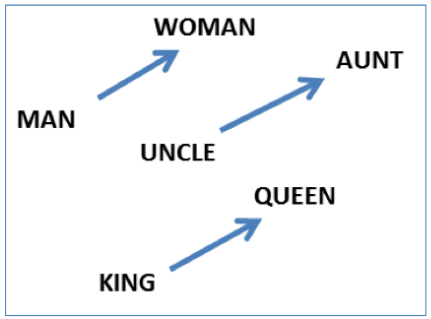
- Utilise une representation pour chaque mot en entrée et idem en sortie
- Modèle

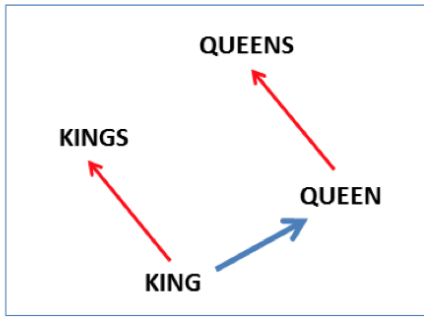
$$P(w_{t+d}|w_t) \propto exp((w_{t+d})^T w_t))$$

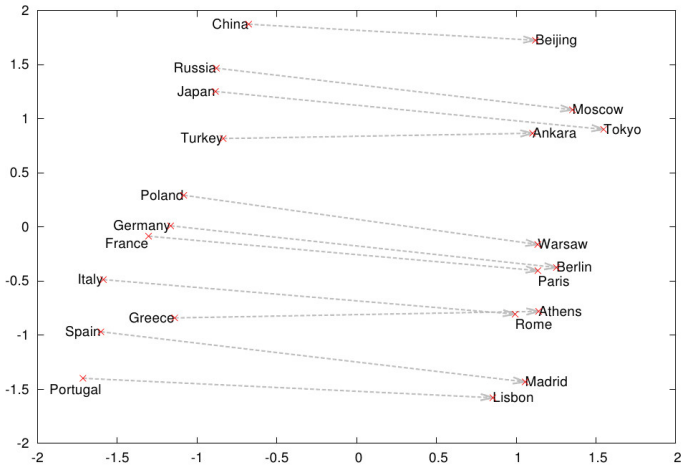
A particular interesting effect: compositionality

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

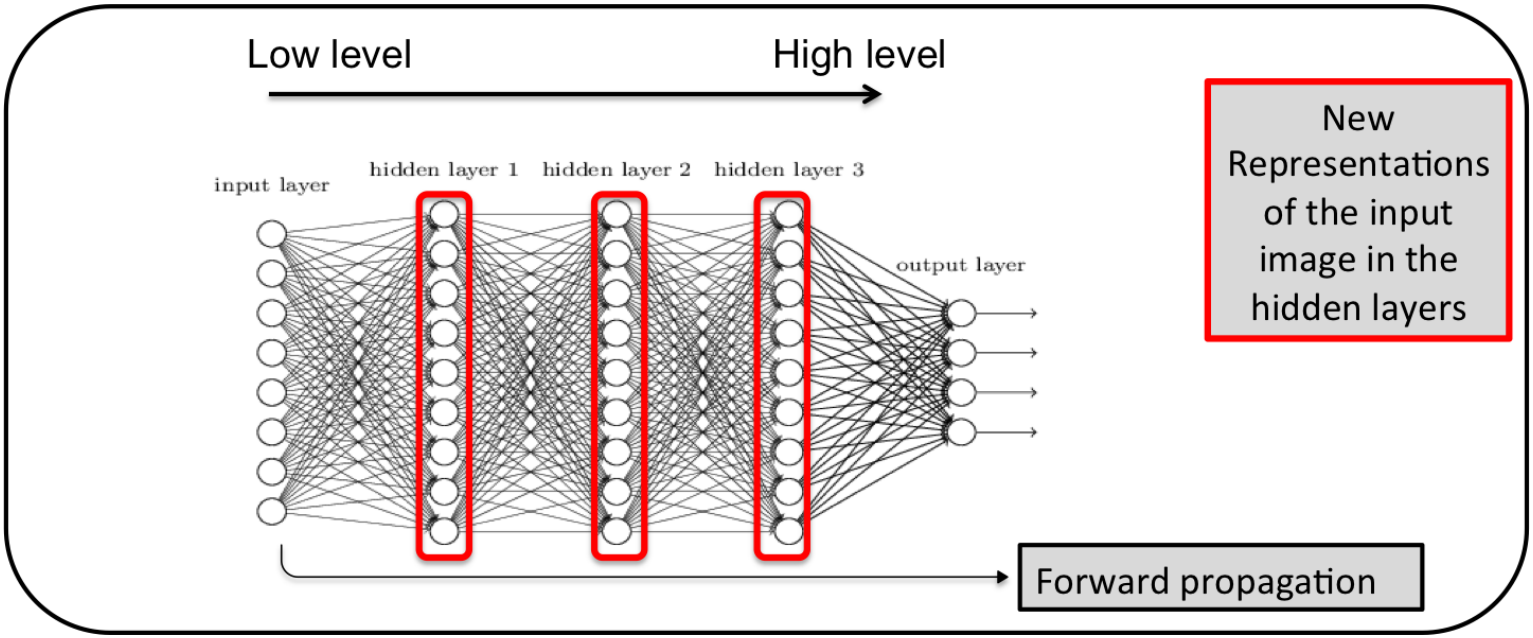




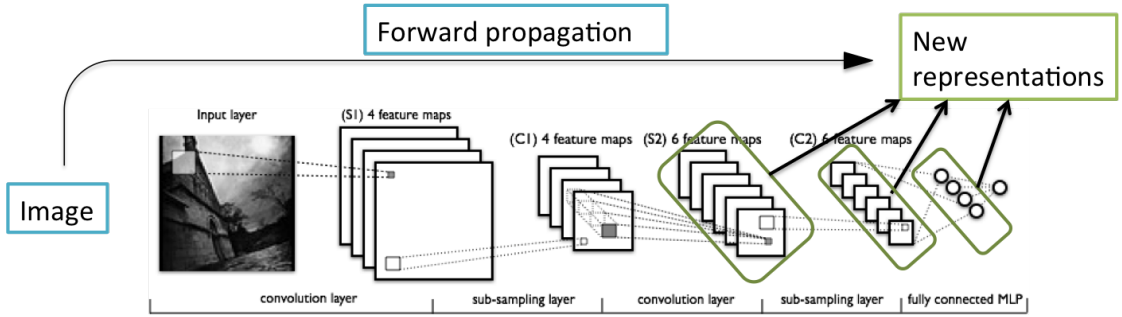


Extension of the embedding idea

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



Extension of the embedding idea for images



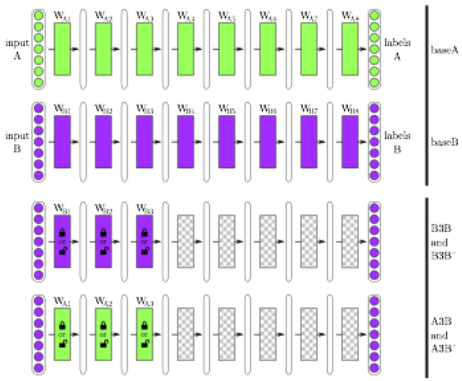
Main interest

- 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

Genericity of representations [Yozinski and al., 2014]

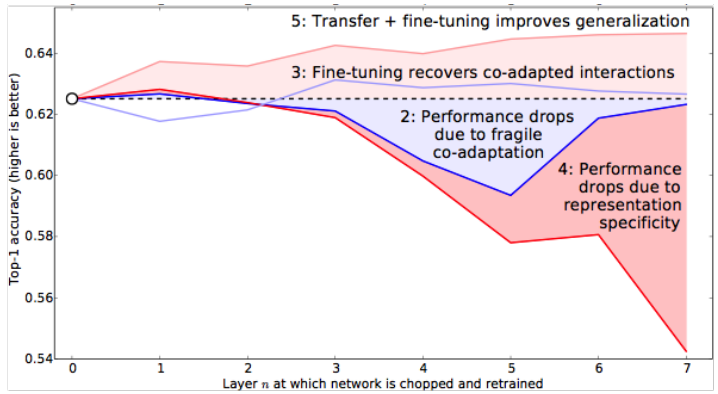
Experiments on two similar tasks

- 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



Outline

Depth in RNNs

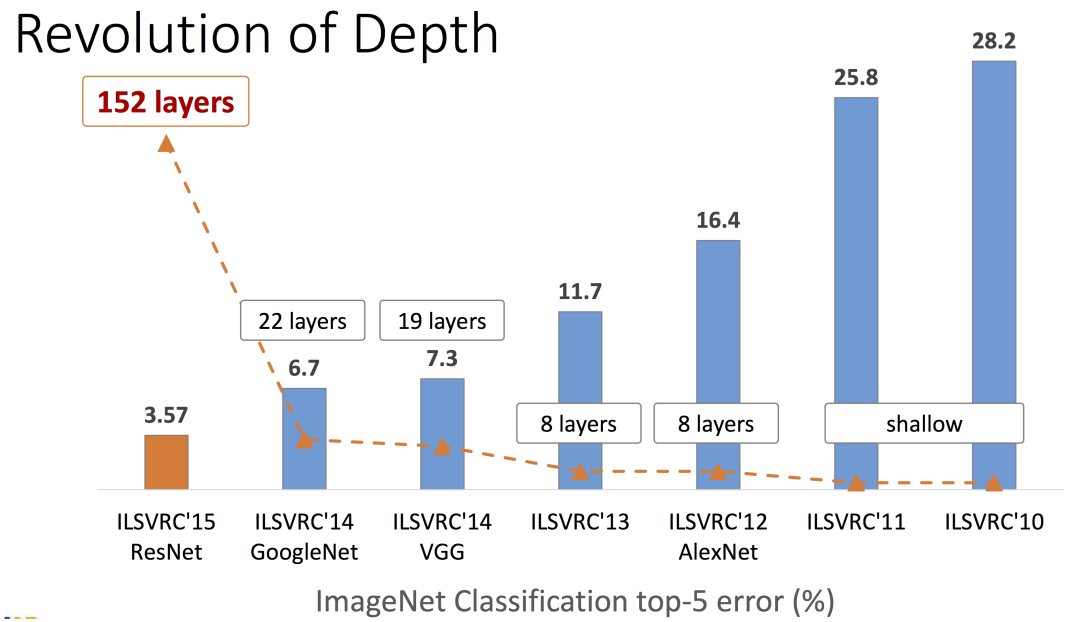
Depth in Feedforward nets

- 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)

The Times They Are A Changing



(slide from [Kaiming He])

Deep vs Shalow: Increased capacity?

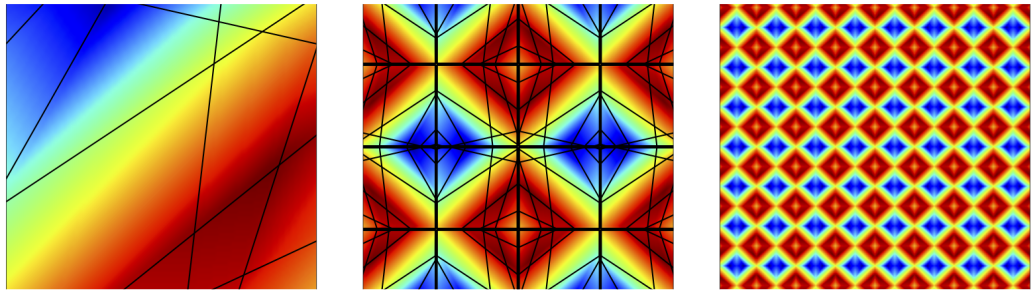
Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

- DNNs with RELU activation function $\Rightarrow$ piecewise linear function
- Complexity of DNN function as the Number of linear regions on the input data
- Exponentially more regions per parameter in terms of number of HL

In more details

- 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
- At least order (k-2) polynomially more regions per parameter in terms of width of HL n

Deep vs Shalow ?



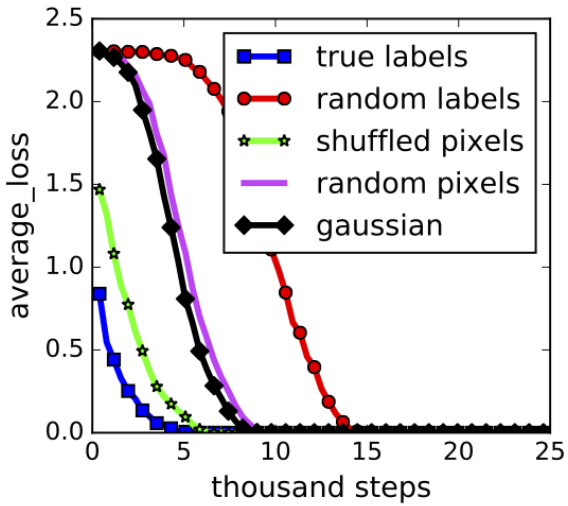
From [Pascanu and al., 2014]

- 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.

DL in the view of generalization, overtraining, local minimas etc

Traditional Machine Learning and NNs

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



[Zhand and al., 2017]

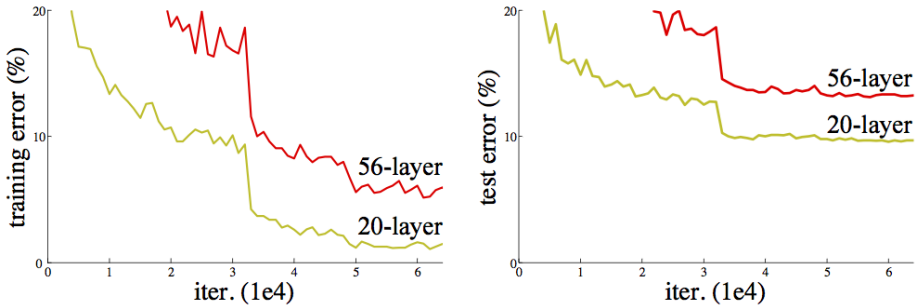
Recent results in DL

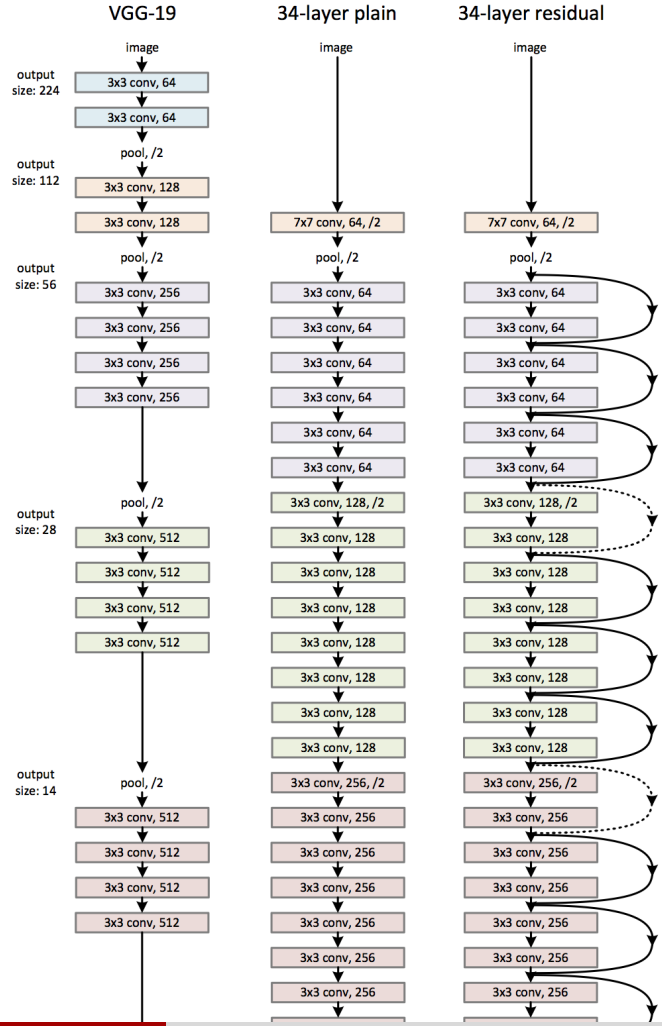
- The Overfit idea should be revised for DL [Zhand and al., 2017]
  - Deep NN may learn noise !
  - Regularization may slightly improve performance but is not THE answer for improving generalization
- Objective function do not exhibit lots of saddle points and most local minima are good and close to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

From shalow to deep

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

...





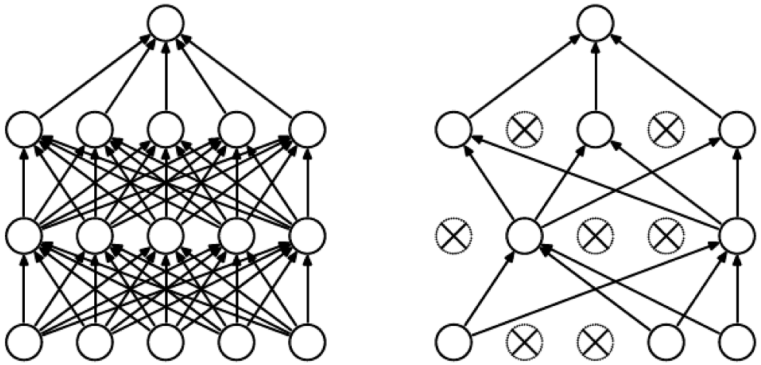
The depth alone is not enough

Main answers

- Regularization (Dropout)
- Make the gradient flow with activation monitoring (Batch Normalization)
- Make the gradient flow with structural constraints (Identity mapping)

Dropout [Hinton 2012]

Principle



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

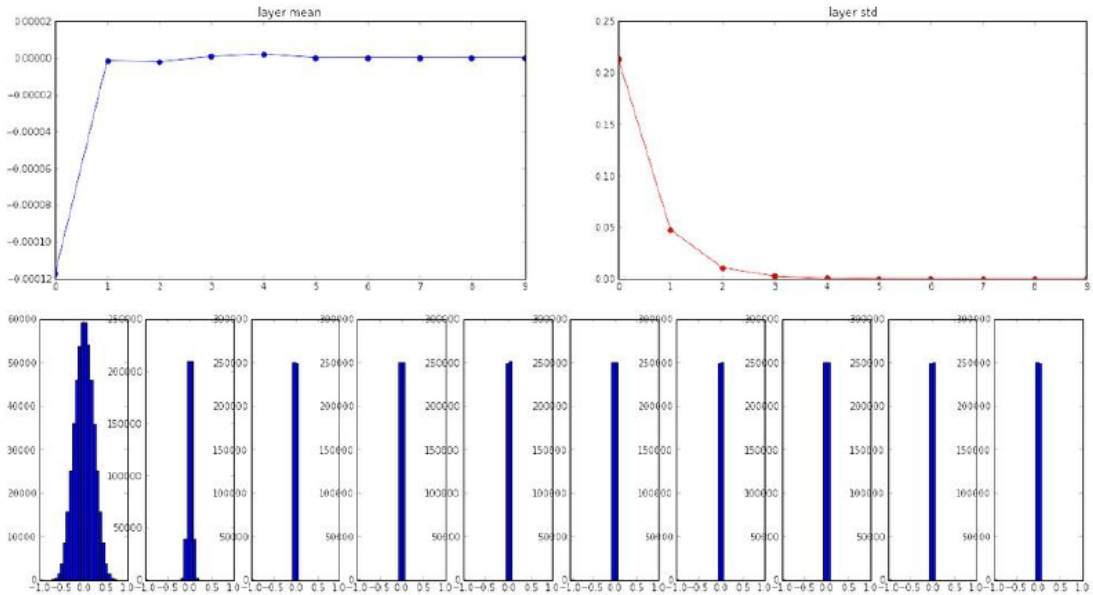
- First method that actually allowed learning deep networks without pretraining and smart initialization
- Related to ensemble of models
- Weights are normalized at inference time

Problems with activity propagation in deep NNs [He et al., 2016]

Few slides from *Fei Fei Li*

Standard initialization schema for MLPs

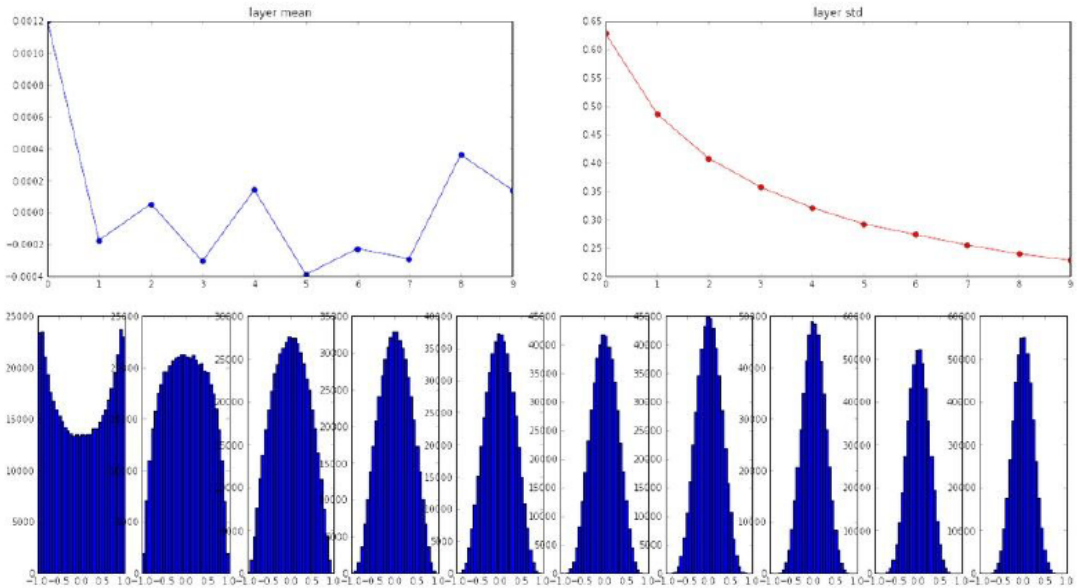
- 10 layers networks (500 neurones each, with tanh)
- Initialization : gaussian random with small (std=0.01) values (what if all null initialization?)
- All activations at 0
- What about the gradient ?



Smarter initialization

Good (but not enough)

- 10 layers networks (500 neurones each, with tanh)
- Xavier initialization : random gaussian with std dev $= \frac{1}{N_{previouslayer}}$
- Much better behavior but fails with RELU activation (assuming normalized inout data)

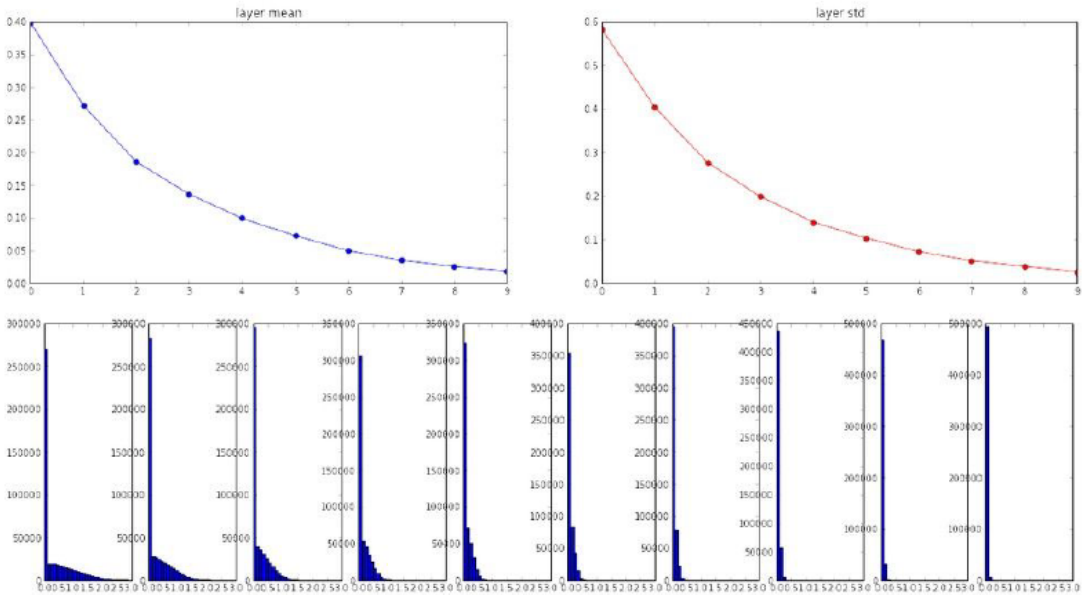


From Fei Fei Li's slides

Smarter initialization

Good (but not enough)

- 10 layers networks (500 neurones each, with tanh)
- Xavier initialization : random gaussian with std dev $= \frac{1}{N_{previouslayer}}$
- Much better behavior but fails with RELU activation (assuming normalized inout data)

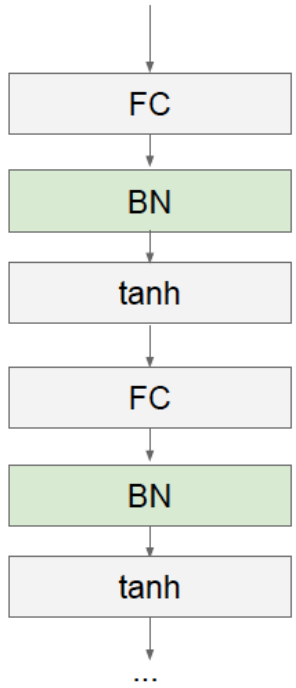


From Fei Fei Li's slides

Batch Normalization

Main idea

- Usually inputs to neural networks are normalized to either the range of [0, 1] or [-1, 1] or to mean=0 and variance=1
- BN essentially performs Whitening to the intermediate layers of the networks.
- Usually placed before nonlinearities

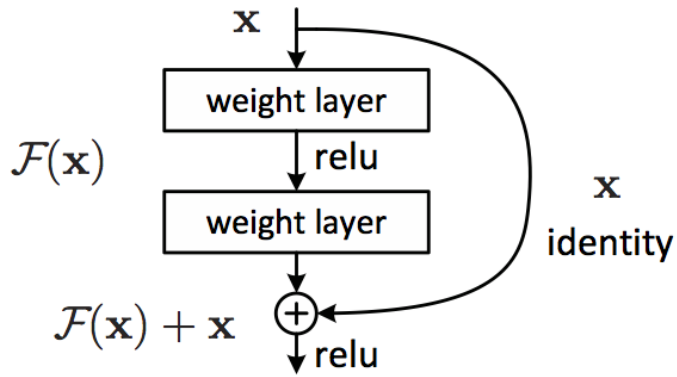


From Fei Fei Li's slides

Residual Networks

Principle

- Include identity mapping in the model
- ResNet building block [He and al., 2015]]



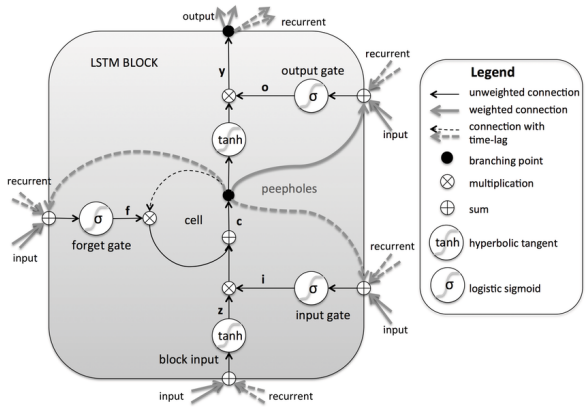
- Every layer becomes close to the output ($\Rightarrow$ not far in the backpropagation process)

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]

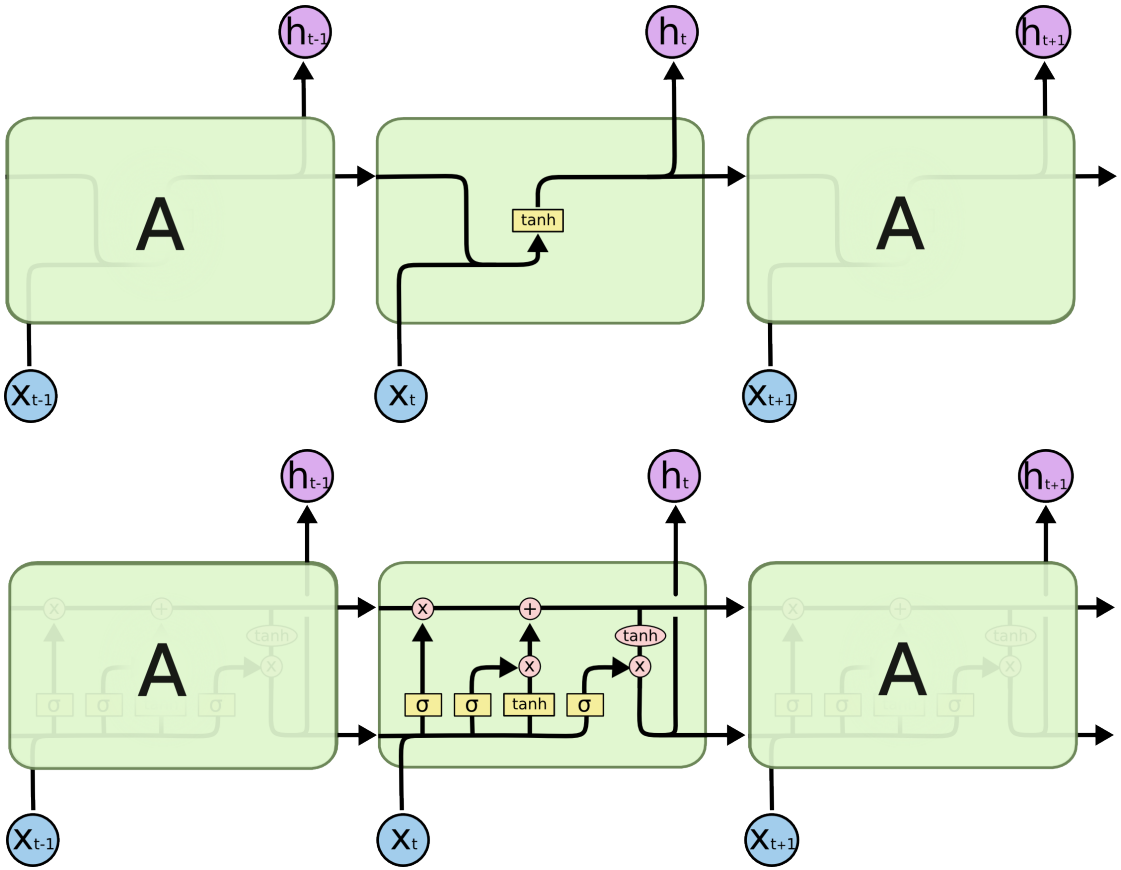
LSTM units



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

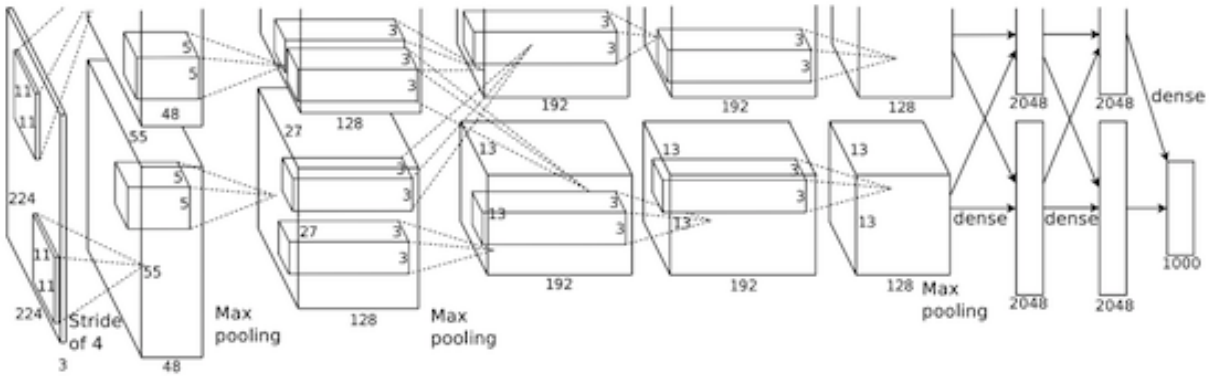
LSTM units

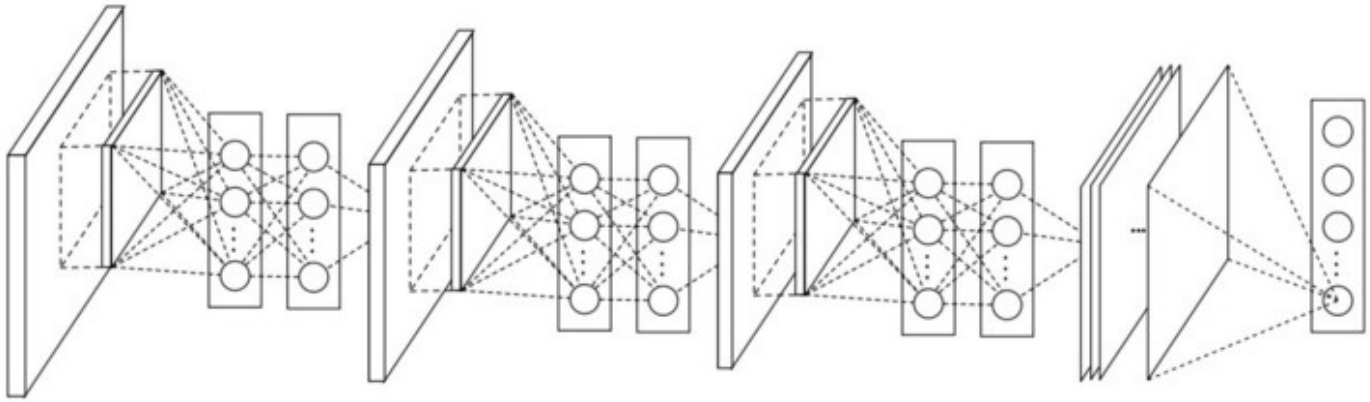


- LSTM layers may be stacked as well as standard RNN layers ($h_t = LSTM(x_t, h_{t-1}, c_{t-1})$ is input to the upper layer)

Examples of architectures

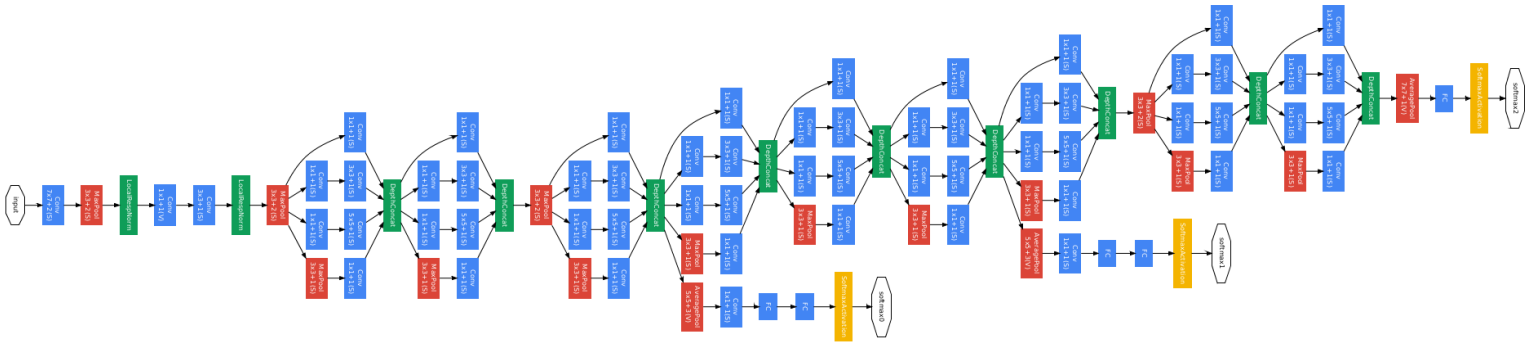
AlexNet [Krizhevsky and al., 2012] (top) and NetworkInNetwork [Lin and al.,2013] (bottom)





Looking for a good architecture: Lego game

How to reach such an architecture (GoogleNet 2014) ?



Searching for a good architecture requires making choices !!

Looking for a good architecture: Lego game

Gridsearch

- Standard Machine Learning models
  - Very few hyperparameters (regularization tradeoff, kernel width or degree etc)
  - Easier optimization problem
  - Usually much less data and much simpler models
  - $\Rightarrow$ Quite exhaustive gridsearch
- Large deep networks
  - Many choices (sequence of layers, width of layers, convolution kernel's size and strides, activation function, optimization routine and its parameters...): Not many theoretical hints
  - Harder optimization problem
  - Each try is expensive
  - $\Rightarrow$ Reuse of others' architectures whenever possible
  - $\Rightarrow$ Gain experience on **how to** design

Looking for a good architceture

Activation function

- ReLu
- Sigmoid
- Tanh
- Linear
- GRU
- LSTM

Connectivty

- Fully connected
- Convolutional
- Dilated
- Recurrent
- Recursive
- Skip/ Residual

Criterion

- MSE
- L2 reg
- Cross entropy
- Binary cross entropy
- Likelihood

Optimizer

- SGD
- Adam
- Adagrad
- Adadelta
- RMSProp

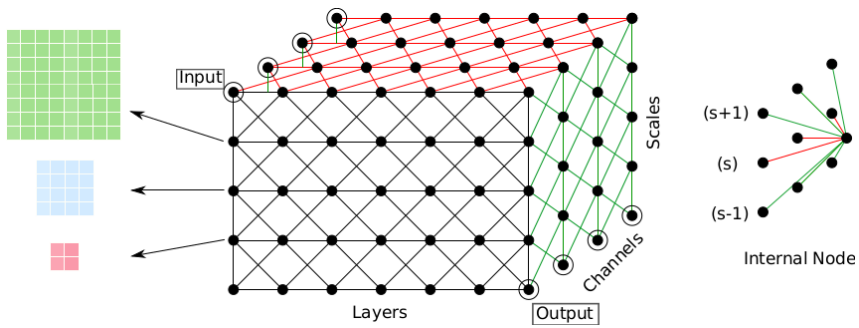
Hyper-parameters

- Learning rate
- Decay
- Layer size
- Batch size
- Dropout rate
- Gradient clipping

Looking for a good architceture

Illustration [Verbeek 2017]

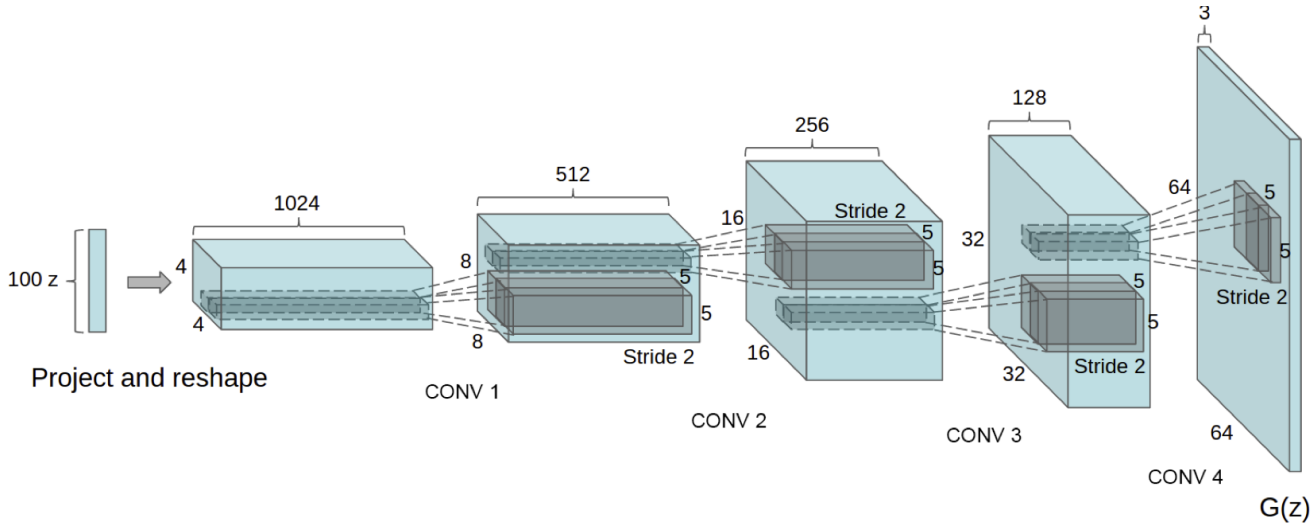
- **Simple** search (but for a large network)
  - 19 convolution layers and 5 pooling layers to set
  - Question: where to put the poooling layers? $\rightarrow$ 40 000 architectures !!
  - No question about layers' dimensions, activation function, kernels' size, pooling type etc
- Remember
  - 1 hour GPU on AWS = 1 \$
  - Learning 1 model = Few hours
    $\Rightarrow$ Expensive design !!!
- Not much alternatives



Looking for a good architecture: use others' !!

Deep Models for High resolution images [Radford 2015]

Historical attempts to scale up GANs using CNNs to model images have been unsuccessful. This motivated the authors of LAPGAN (Denton et al., 2015) to develop an alternative approach to iteratively upscale low resolution generated images which can be modeled more reliably. We also encountered difficulties attempting to scale GANs using CNN architectures commonly used in the supervised literature. However, after extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models.

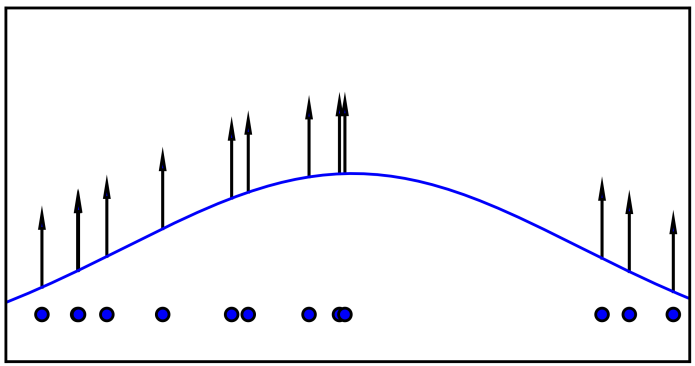


Outline

Generative models

Goal

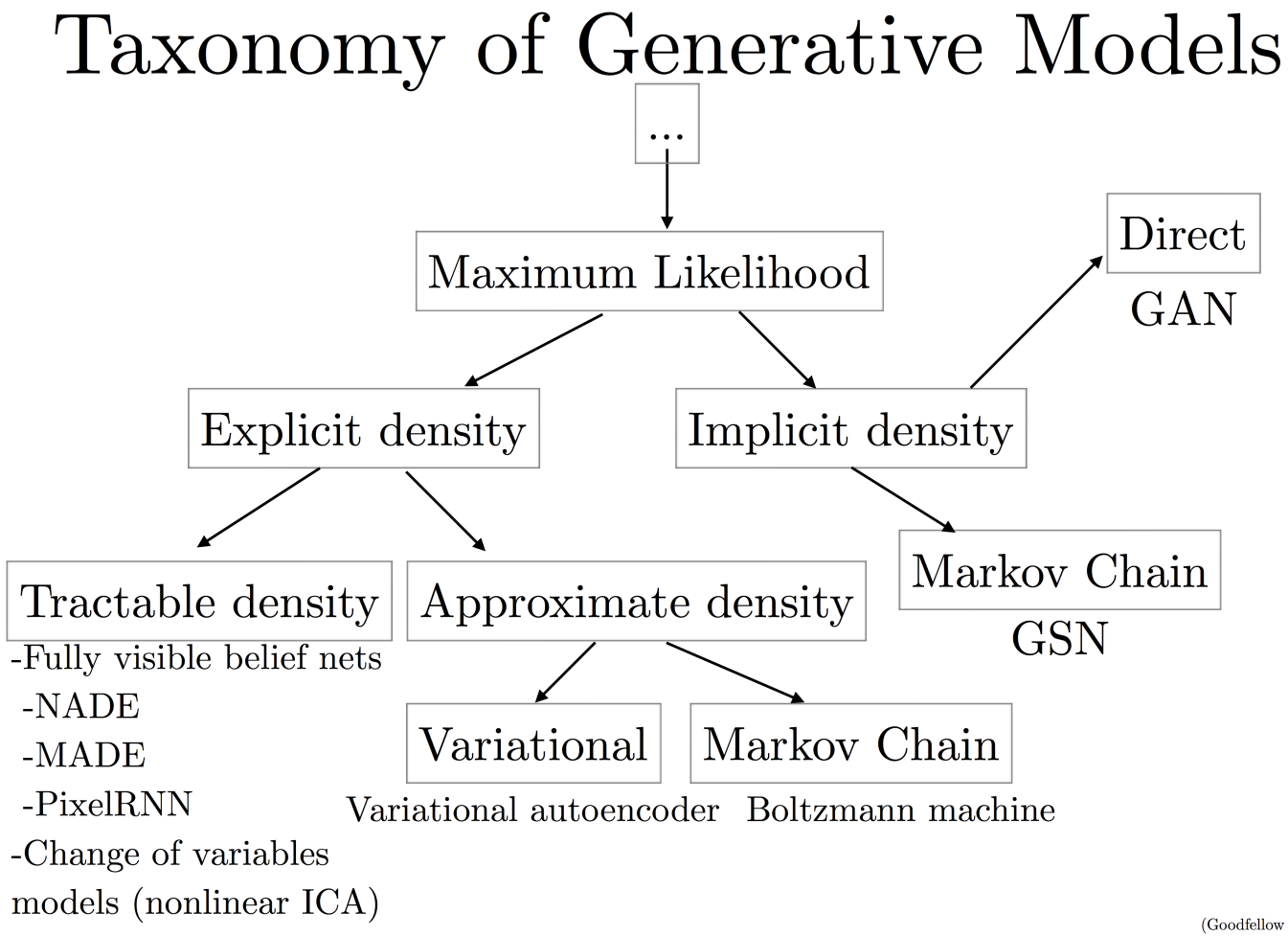
- Learn to generate complex and realistic data
- Statistical viewpoint : learn a model of the density of data / able to sample with this density
  - Postulate a parametric model : Usually not complex enough
  - Postulate a parametric form and perform optimization (e.g. Maximum Likelihood) : Intractable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



$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

Maximum Likelihood Estimation (MLE)

Adversarial learning principle



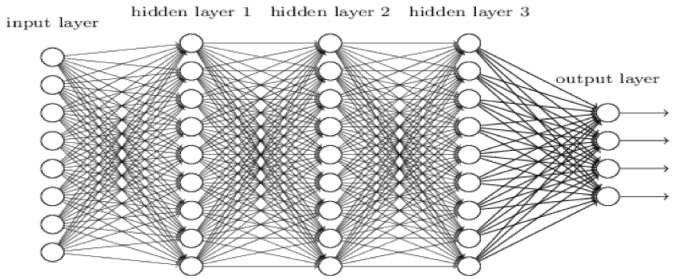
Adversarial learning principle

Principle

- Use a two player game
  - Learn both a generator of artificial samples 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: Generator

Determinitic NN as a generative model



Using a deterministic NN as a generative model

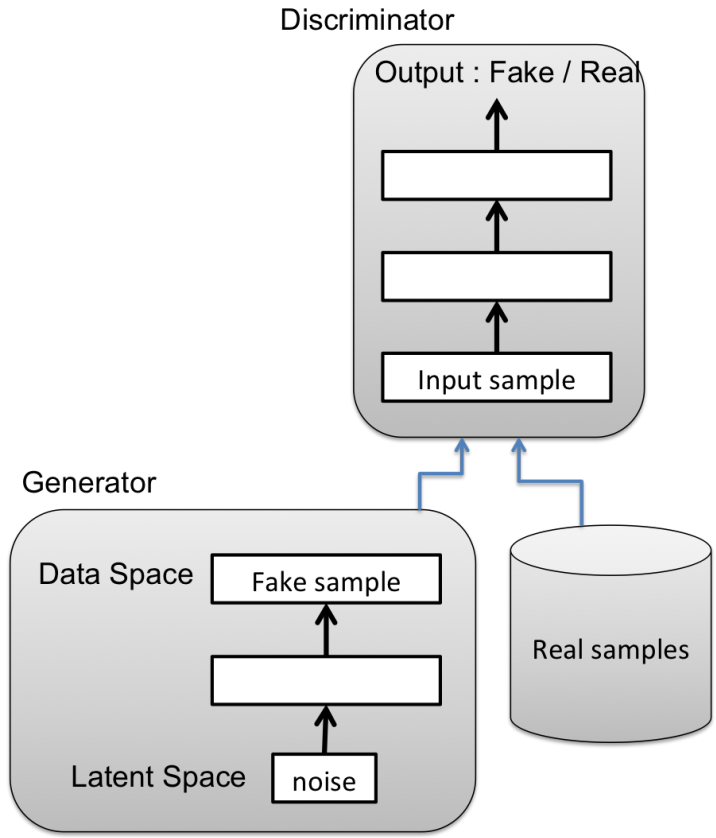
- Let note the function implemented by the model as G
- Let note the input $z \rightarrow$ The NN computes $G(z)$
- Assume z obeys a prior (noise) distribution, p_z, e.g. Gaussian distribution
- then the output x of the NN follows a distribution

$$\Rightarrow p_G(x) = \int_{z \text{ s.t. } G(z)=x} p_z(z)dz$$

Le principe de l'adversarial learning [Goodfellow and al., 2014]

Principle

- Jeu à deux joueurs: un générateur et un discriminateur
  - le discriminateur veut distinguer les exemples générés des vrais exemples
  - Le générateur veut tromper le discriminateur



Adversarial Learning criterion

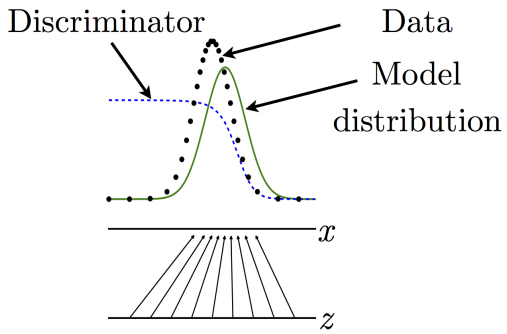
Criterion from [Goodfellow and al., 2014]

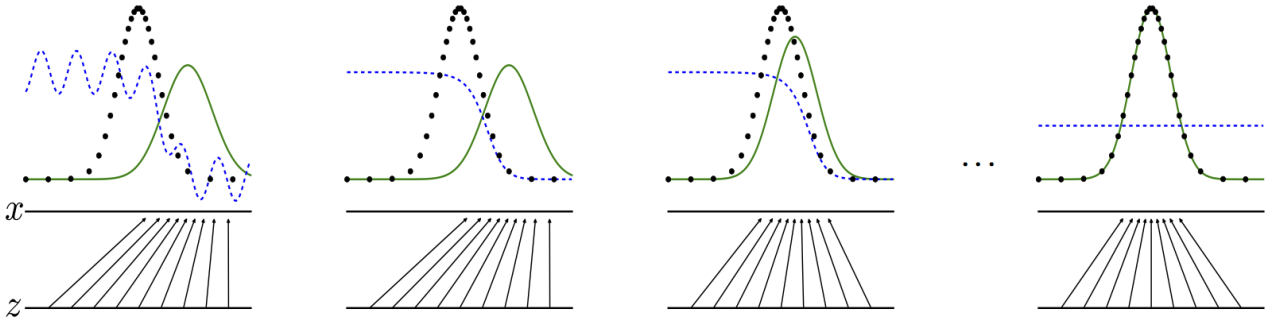
- Generator G and Discriminator D are two NNs
  - Whose parameters are noted θ_g and θ_d
- Distributions
  - p_{data} stands for the empirical distribution of the data from the training set
  - p_z is a prior noise distribution, e.g. a Gaussian distribution
  - On convergence we want $p_g = p_{data}$
- Learning criterion:

$$min_g max_d v(\theta_g, \theta_d) = \mathbf{E}_{x \sim p_{data}} [logD(x)] + \mathbf{E}_{z \sim p_z} [log(1 - D(G(z)))]$$

  - Assume G is fixed: D is trained to distinguish between fake and true samples
  - Assume D is fixed : G is trained to generate samples as realistic as possible

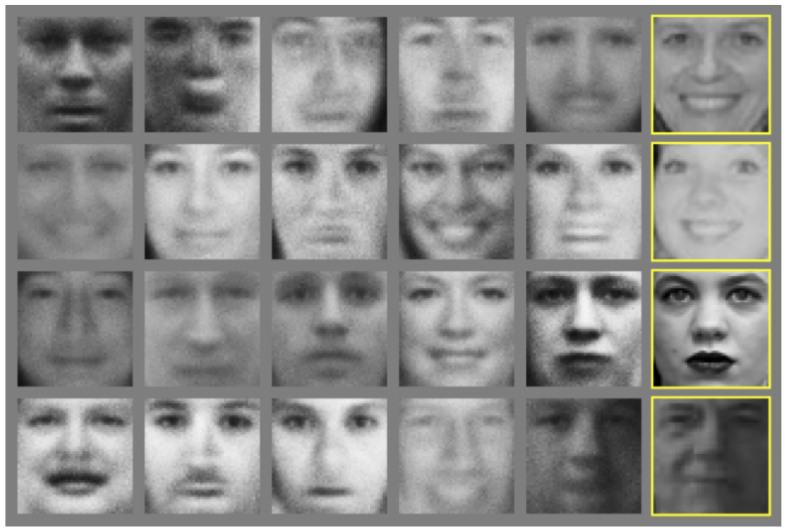
Adversarial Learning theory: What happens during Learning



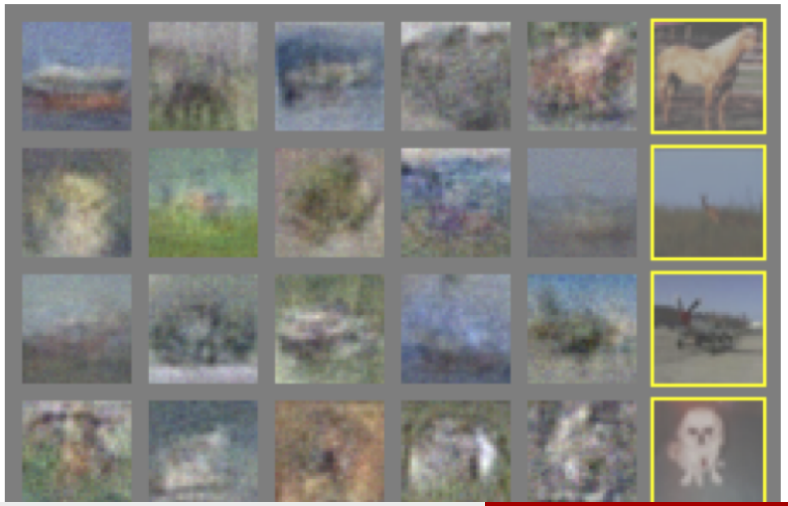


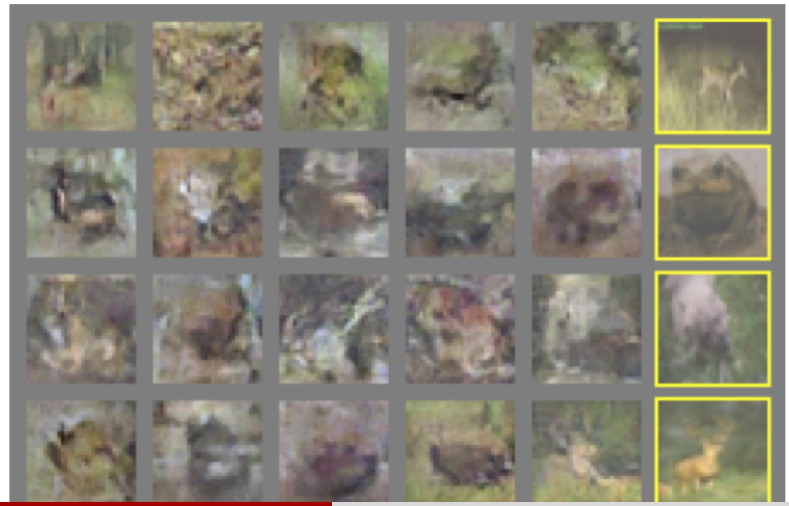
Good Examples

a)

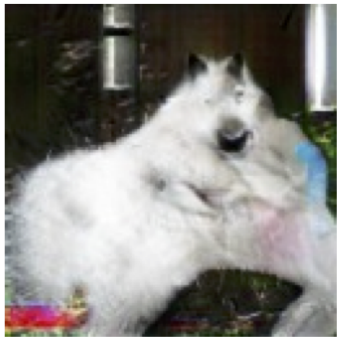


b)

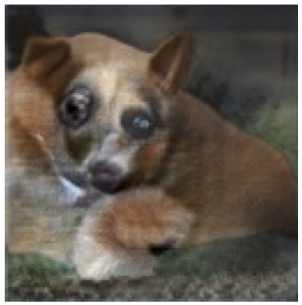
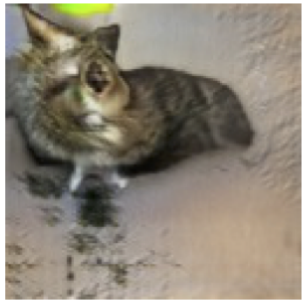




Bad examples





Interpolating with GANs [Goodfellow and al., 2014]

Idea

- The latent code space is fully occupied
- Any sample drawn by sampling with the generator should be realistic
- One may interpolate between two latent codes and see

Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

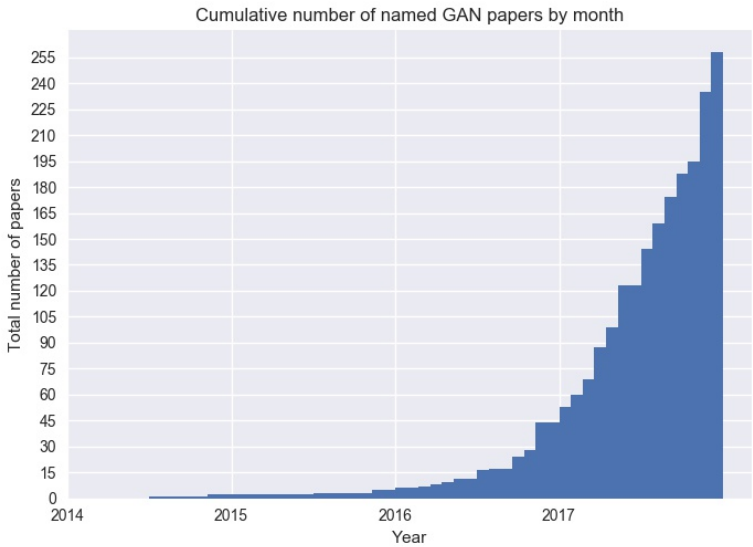
About GANs'

Known problems

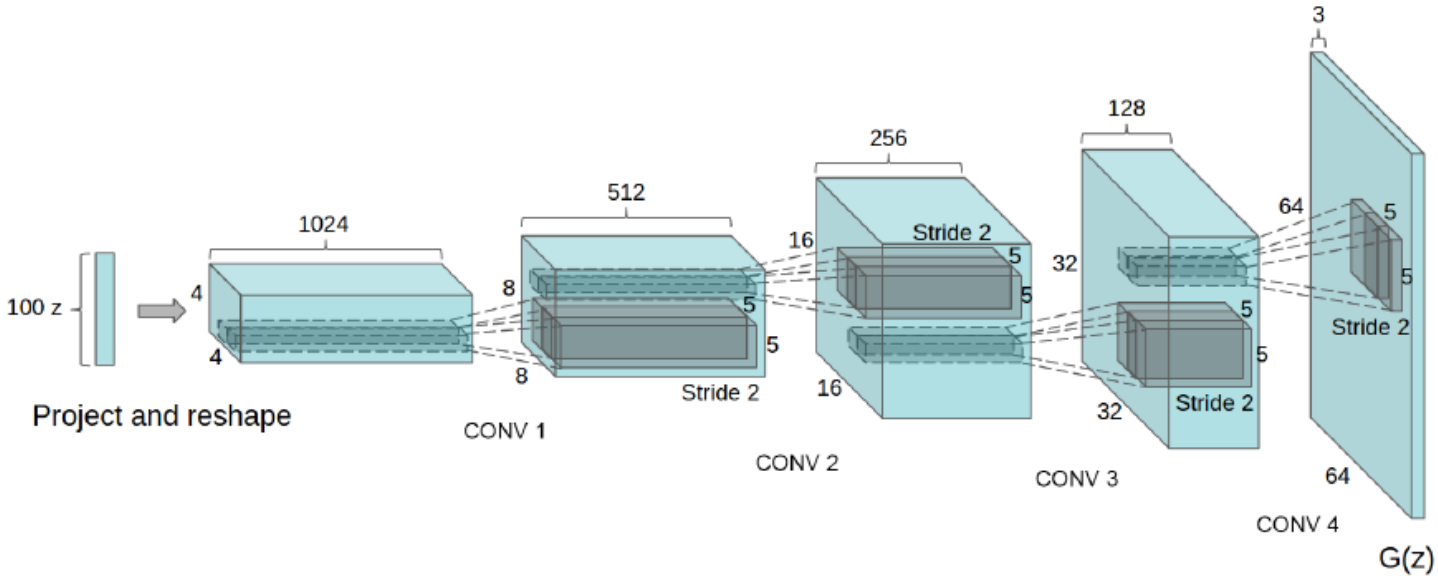
- DIfficult learning
- Very long learning
- Missing modes
- Evaluation measures

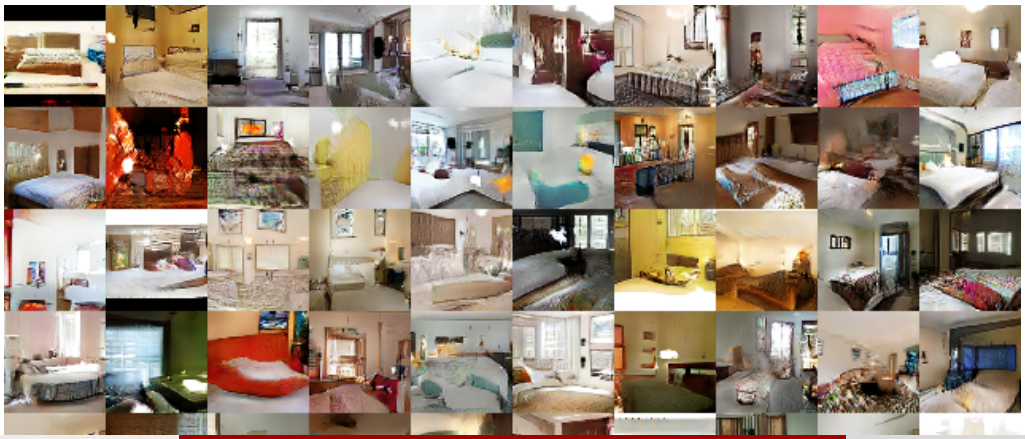
Many many variants

- Conditional
- Disantangling
- Image editing



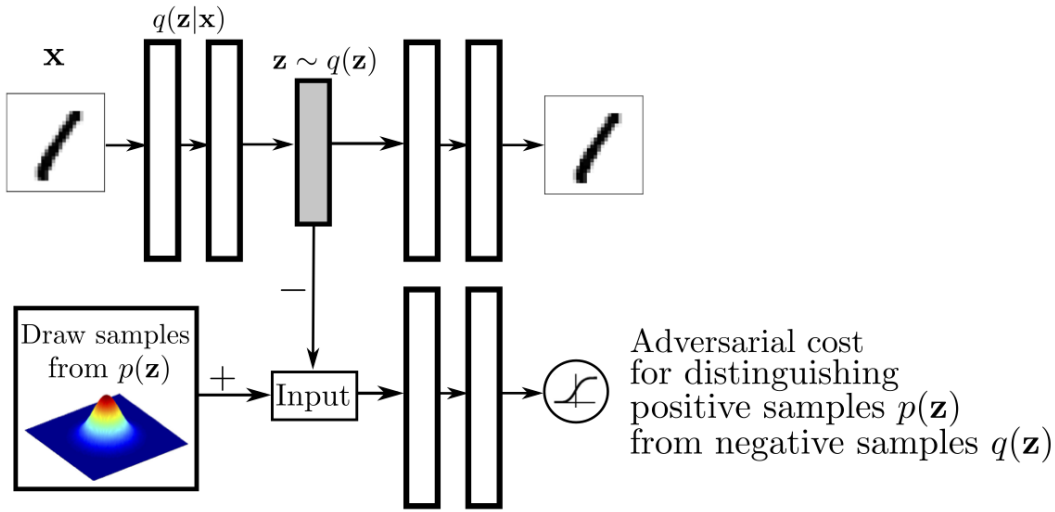
DCGANS [Radford 2015]





Remember the puppet !

Adversarial AE [Makhzani and al., 2014 ou 15]



Learning criterion

- Few definitions for $q(z|x)$: simplest = deterministic
- Learning criterion:

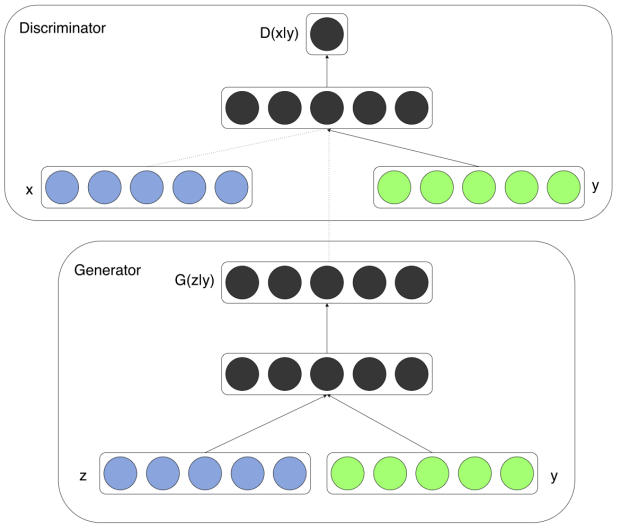
$$min_g max_d v(\theta_g, \theta_d) = \mathbf{E}_{x \sim p_{data}} \left[\|D_c(E_c(x)) - x\|^2 \right] + \mathbf{E}_{z \sim p_z} \left[logD(z) \right] + \mathbf{E}_{x \sim p_{data}} \left[log(1 - D(q(z|x))) \right]$$

Conditional GANs [Mirza and al., 2014]

Learning criterion

- Citerion

$$min_g max_d v(\theta_g, \theta_d) = \mathbf{E}_{x,y\ p_{data}}[logD(x,y)] + \mathbf{E}_{z\ p_z, y'\ p_y}\left[log(1 - D(G(z,y'),y'))\right]$$



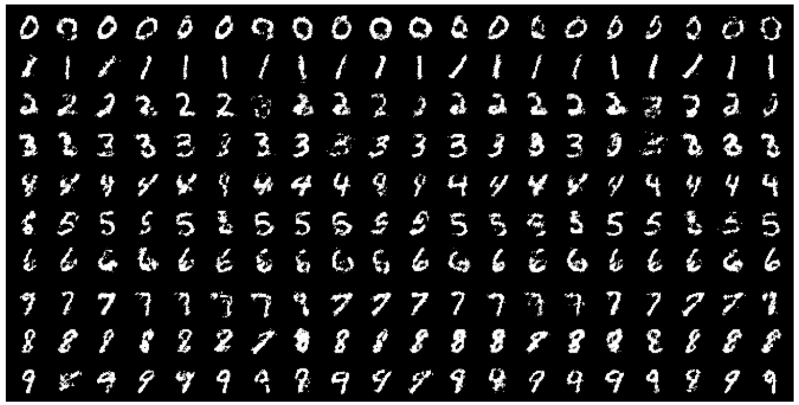
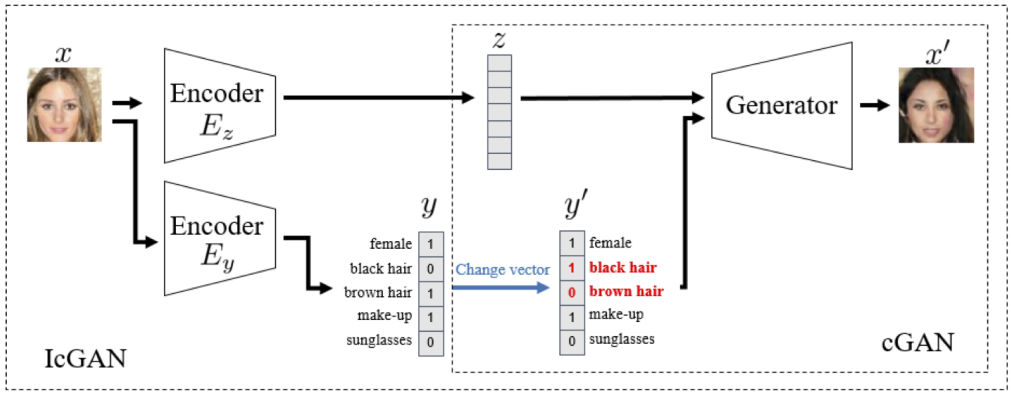
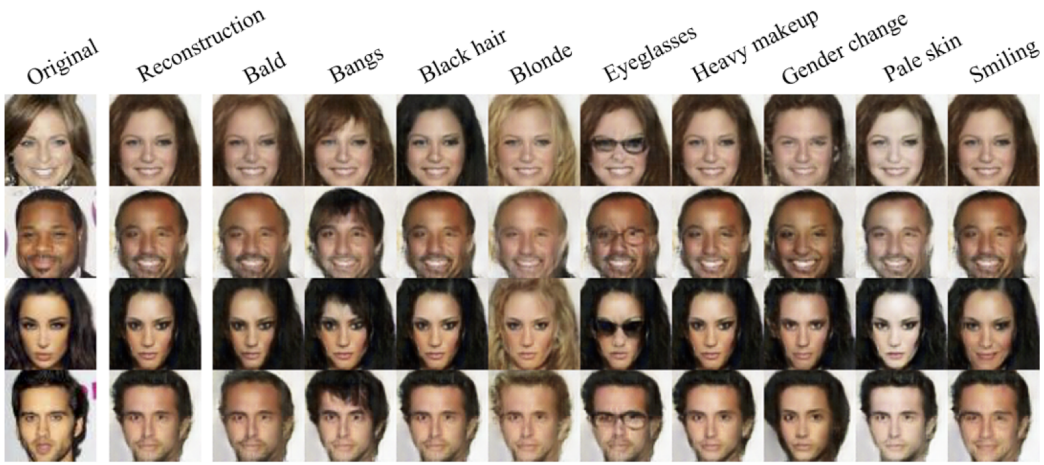


Figure 2: Generated MNIST digits, each row conditioned on one label

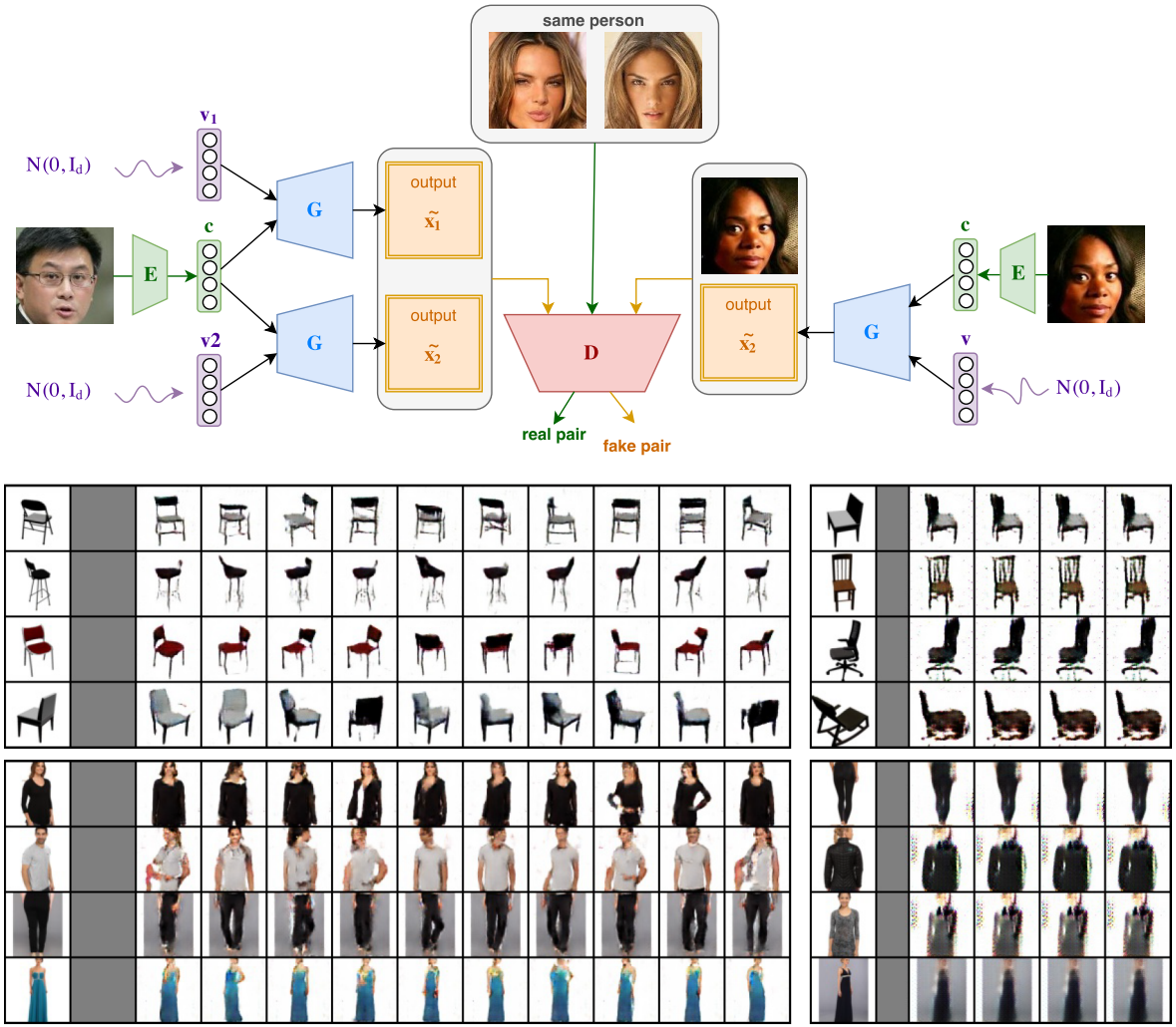
Image editing with Invertible Conditional GANs [Perarnau and al., 2016]



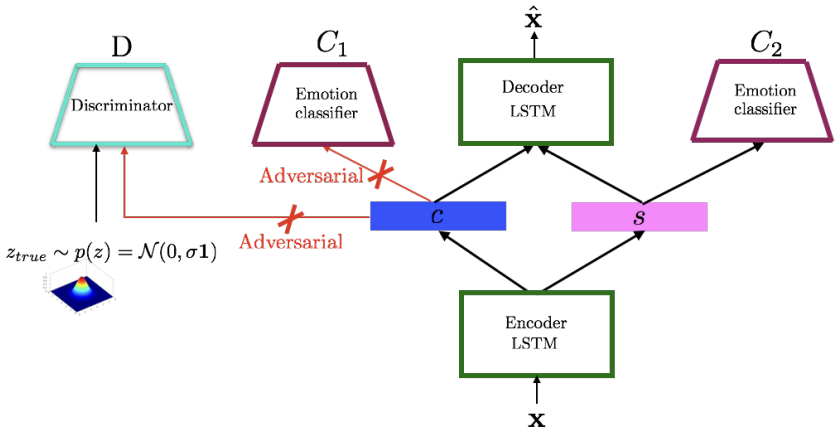


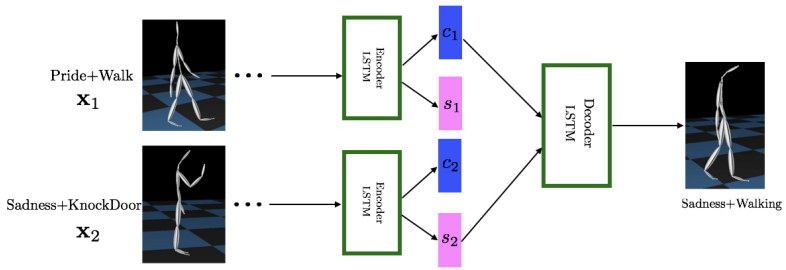
Disentangling factors of variation [Chen et al., 2018]

Transfering styles between images



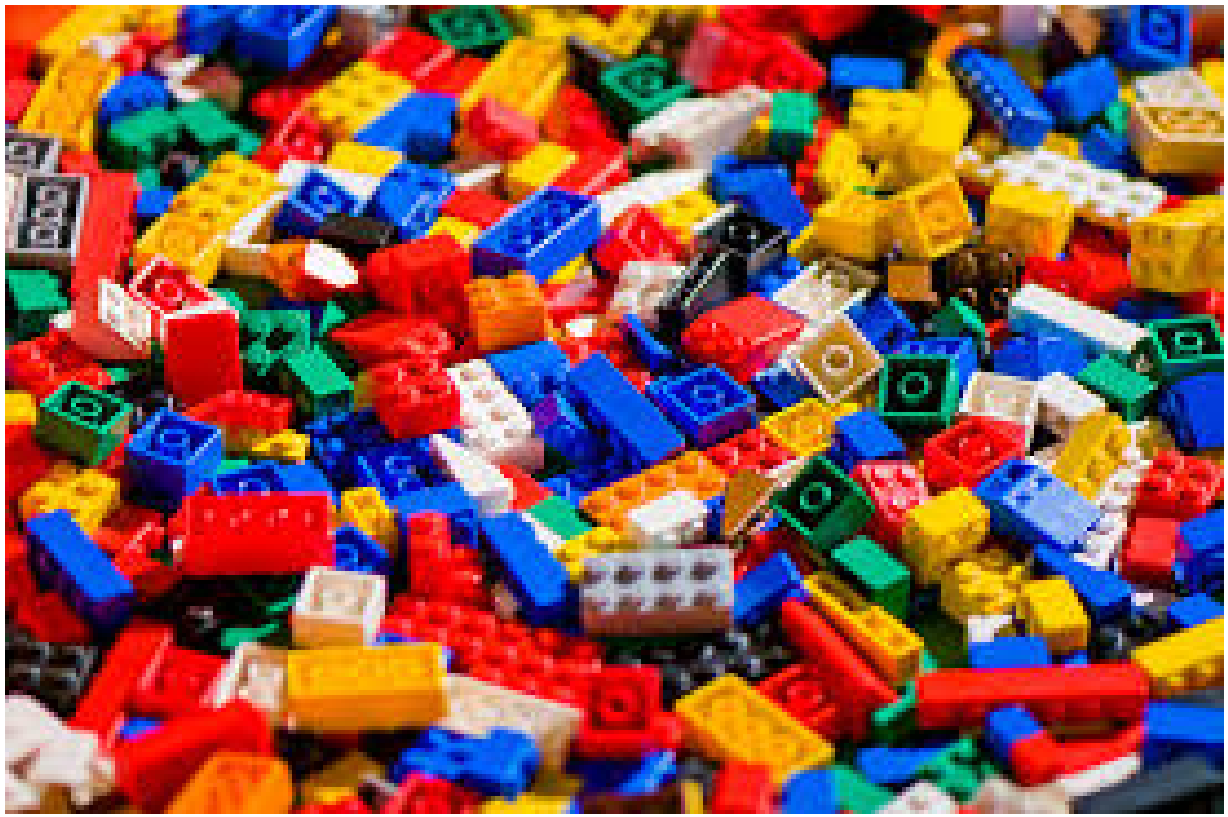
Motion capture data





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...

But

- 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 bricks: models able to compute a **universal representation** of (structured) data for:
  - Text
  - Images
  - Speech
- 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...

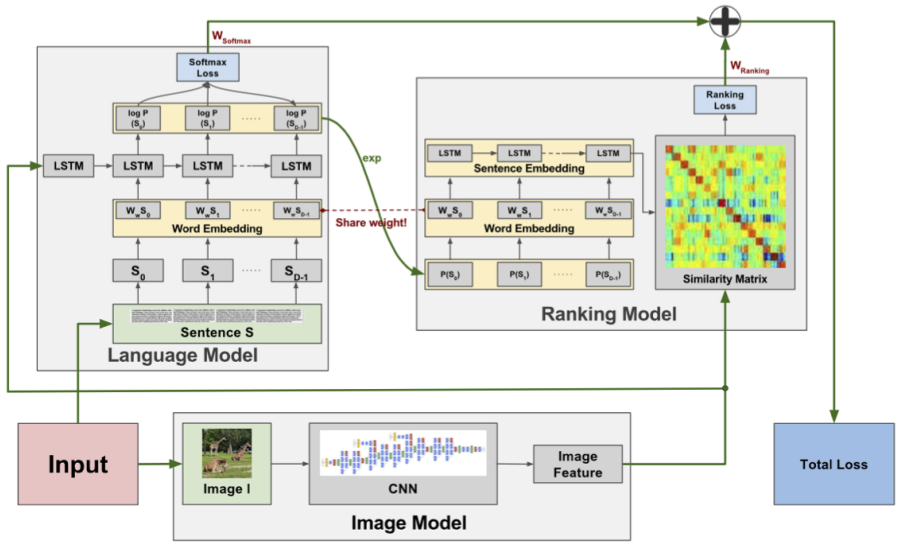
What are DNN systems at the end?

Not more than MLPs

- Feed forward propagation enabling chain rule (backpropagation)
- Stochastic Gradient Descent Optimization
- Most ideas were there in the 90's

Yet something different...

- Much more complex architectures
- End to end learning

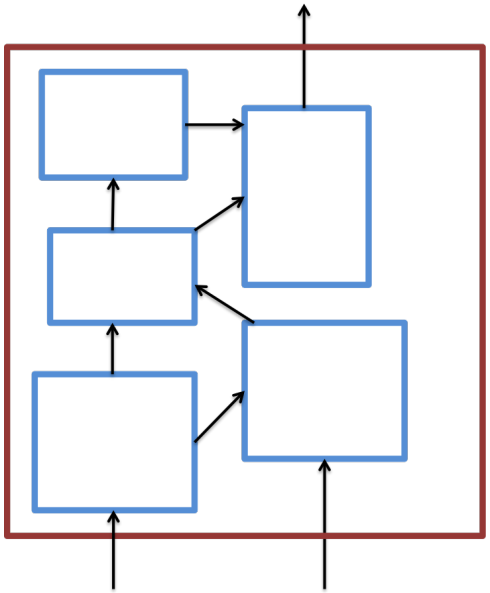


(Yeung et al., 2015)

Learning more general / modular architectures [Bottou's thesis, 1991] !

Graph of modules (better without cycles...)

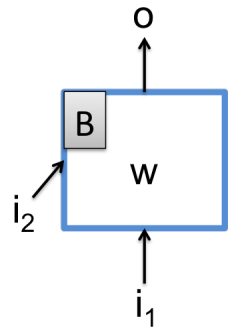
- Use automatic differentiation with computation graph



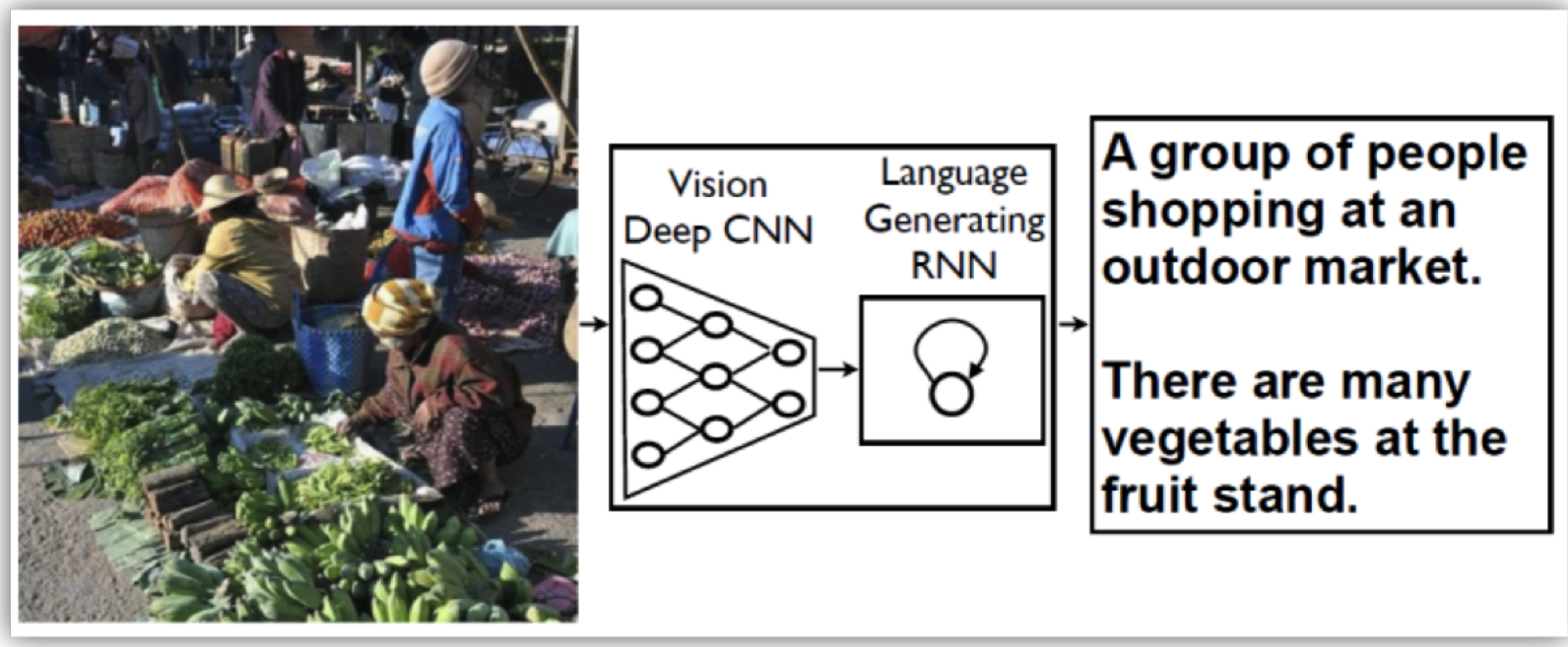
Still optimized with Gradient Descent !!

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

- 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

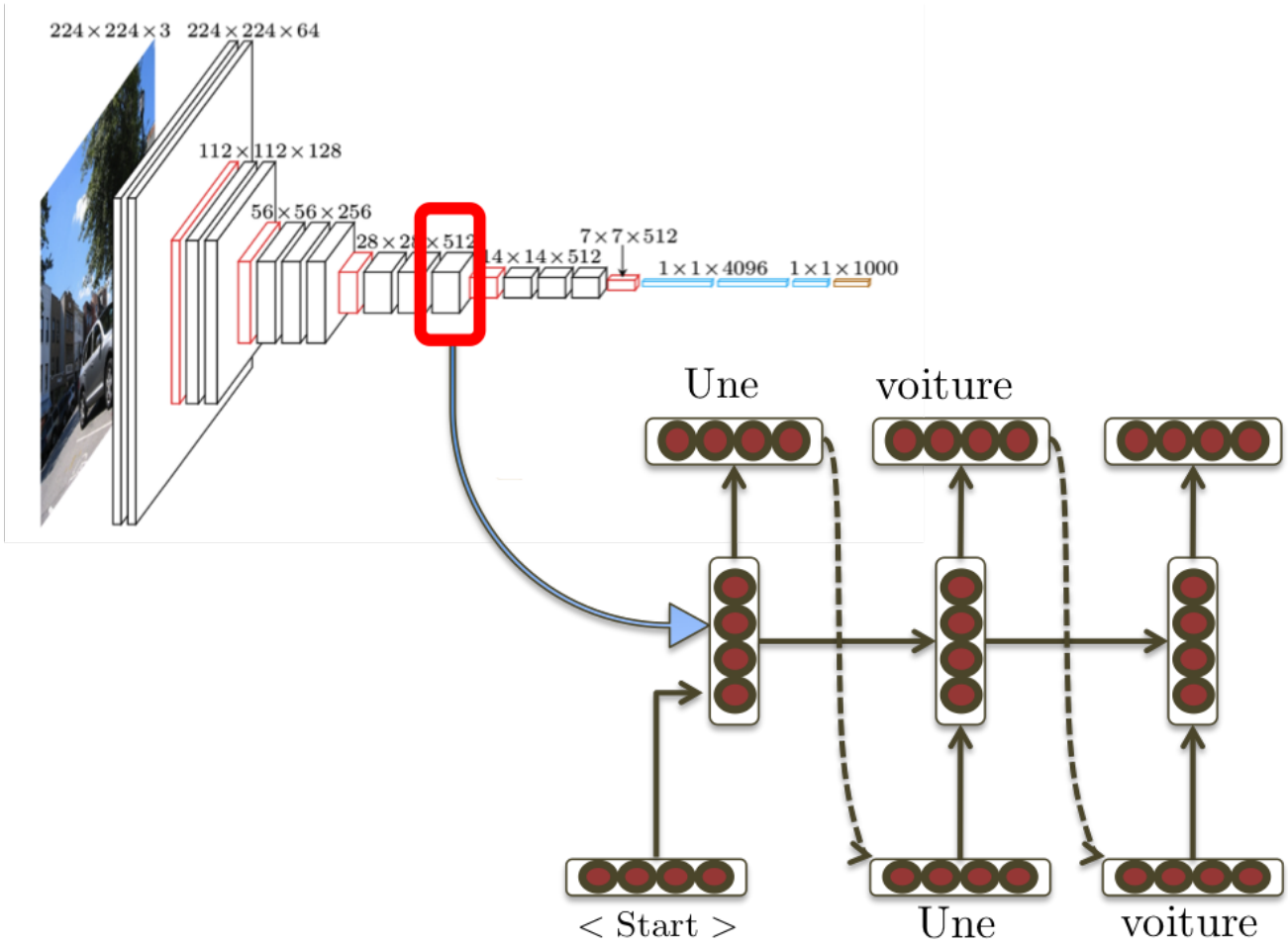


Using bricks to build complex systems

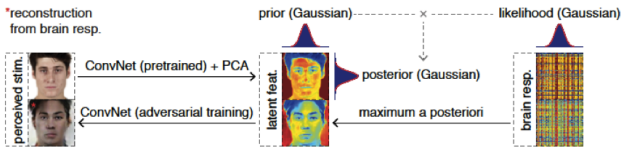


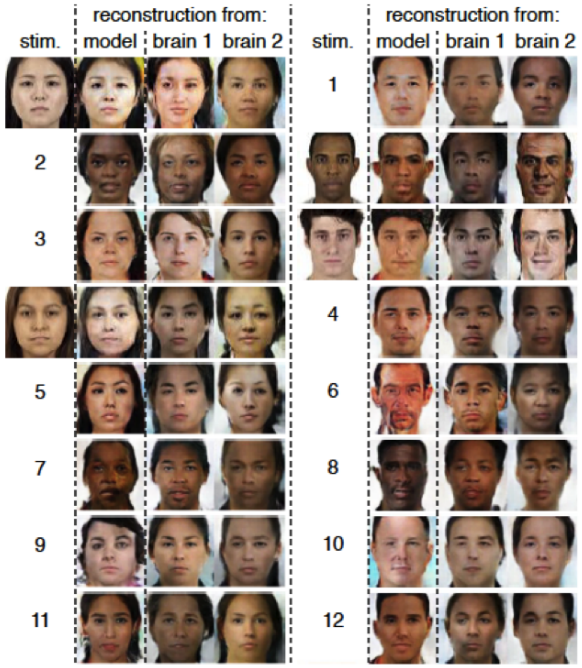
Automatic captioning [Honglak et al., 2014]]

Automatic captionning



Infering face from fMRI signal





[Guclu et al., 2017]]

Outline

Deep and HEP

Pivot adversarial learning [Louppe et al., 2017]

- Use adversarial learning to align simulated and systematic data

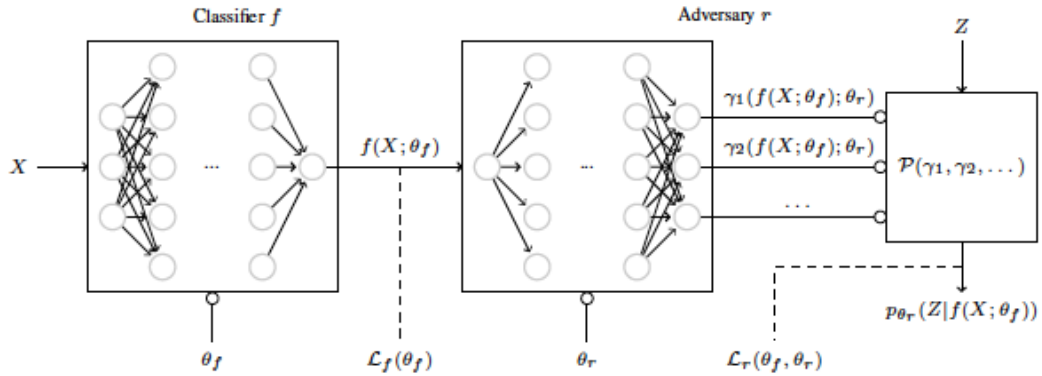
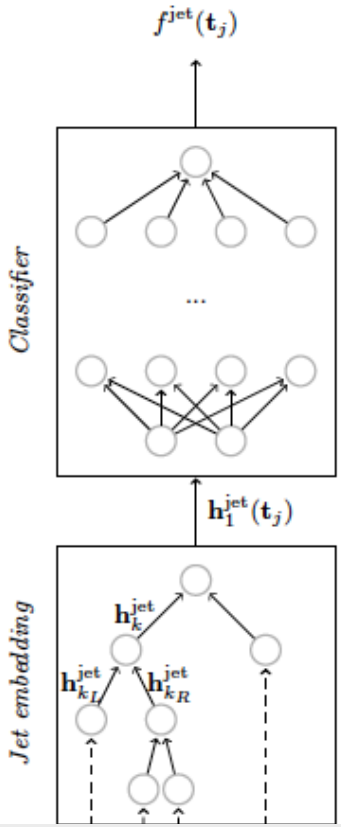


Figure 1: Architecture for the adversarial training of a binary classifier f against a nuisance parameters Z. The adversary r models the distribution $p(z|f(X;\theta_f)=s)$ of the nuisance parameters as observed only through the output $f(X;\theta_f)$ of the classifier. By maximizing the antagonistic objective $\mathcal{L}_r(\theta_f,\theta_r)$, the classifier f forces $p(z|f(X;\theta_f)=s)$ towards the prior $p(z)$, which happens when $f(X;\theta_f)$ is independent of the nuisance parameter Z and therefore pivotal.

Deep and HEP

QCD-aware Recursive Neural Networks for Jet Physic [Louppe et al., 2018]

- Learn to aggregate features for jets using a tree structure inspired from data knowledge



$$\mathbf{h}_k^{\text{jet}} = \begin{cases} \mathbf{u}_k & \text{if } k \text{ is a leaf} \\ \sigma\left(W_h \begin{bmatrix} \mathbf{h}_{k_L}^{\text{jet}} \\ \mathbf{h}_{k_R}^{\text{jet}} \\ \mathbf{u}_k \end{bmatrix} + b_h\right) & \text{otherwise} \end{cases} \tag{3.1}$$

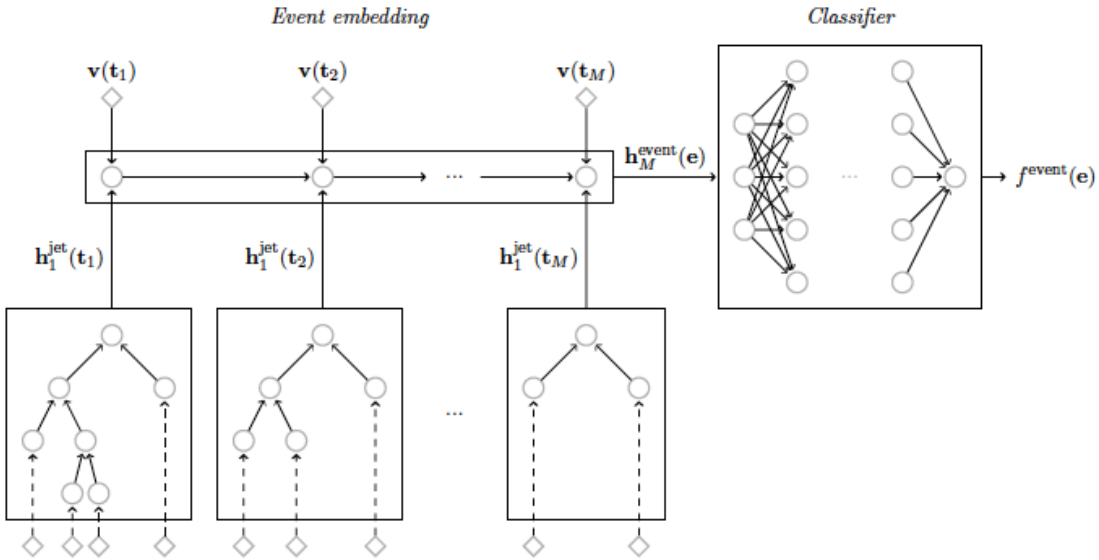
$$\mathbf{u}_k = \sigma\left(W_u g(\mathbf{o}_k) + b_u\right) \tag{3.2}$$

$$\mathbf{o}_k = \begin{cases} \mathbf{v}_{i(k)} & \text{if } k \text{ is a leaf} \\ \mathbf{o}_{k_L} + \mathbf{o}_{k_R} & \text{otherwise} \end{cases} \tag{3.3}$$

Deep and HEP

QCD-aware Recursive Neural Networks for Jet Physic [Louppe et al., 2018]

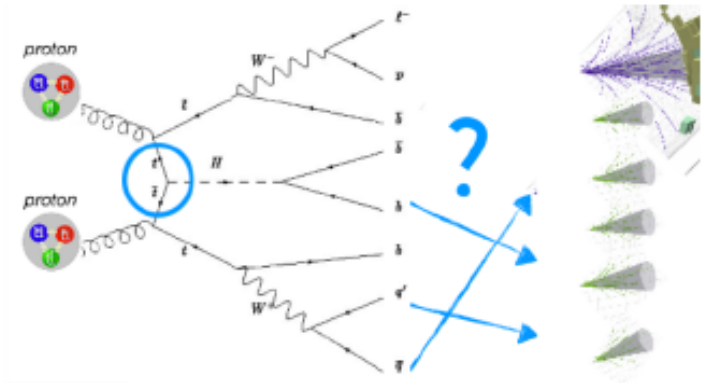
- Learn to aggregate features for jets using a tree structure inspired from data knowledge

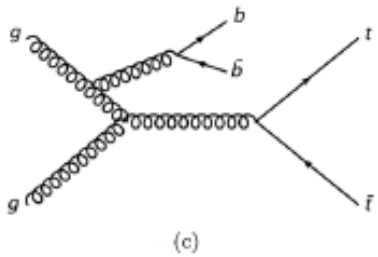


Deep and HEP

Ziyu Guo's thesis (with Y. Coadou)

- Deep learning in the search for ttH with the ATLAS experiment at the Large Hadron Collider
- 1. Replace a reconstruction BDT + classification BDT with a end to end learned joint model

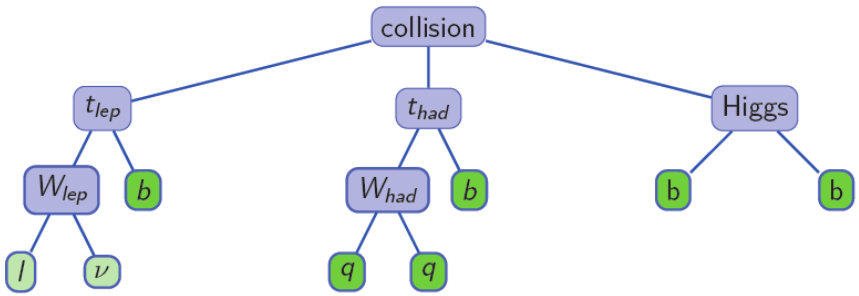


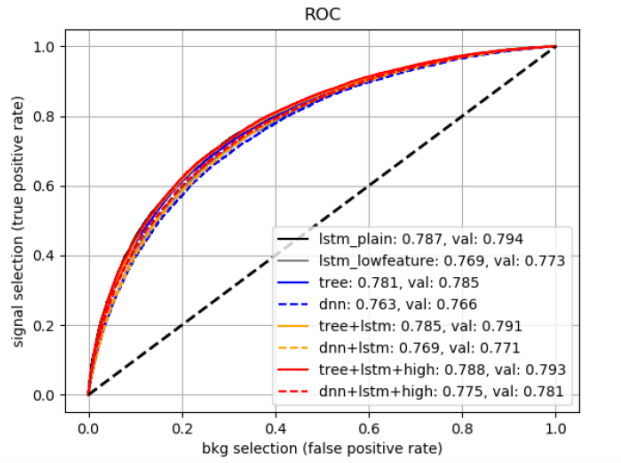


Deep and HEP

Ziyu Guo's thesis (with Y. Coadou)

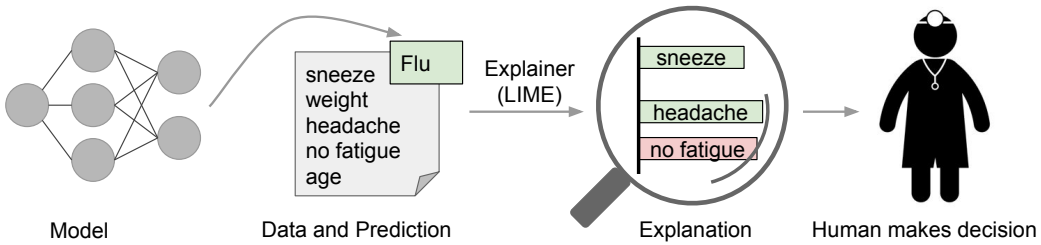
- Deep learning in the search for ttH with the ATLAS experiment at the Large Hadron Collider
- 2. Rely on the physical process to design the NN structure
- $\rightarrow$ Better results than DNN. Comparing now with BDTs





Outline

Explainabilty



[Ribeiro et al., KDD 2016]

Black box models

- Machine Learning models are black box models
- Urgent needs for trustability with the increasing performance in real-life tasks (automatic cars, army, health etc)
- Several levels of understanability / explainability: Explain a decision on an input / the whole process
- Various kind of methods: Agnostic, model-based ...

Explainabilty

Linear models

- $f(x) = w^T x = \sum_{j=1}^{d} x_j w_j$
- Learning with a regularization term : $C(W) = \sum_{i=1}^{N} l(x^i, y^i, w) + ||w||^2$
- Useless weights go to 0
- Relevance of feature j measured as $|w_j|$

Explainabilty

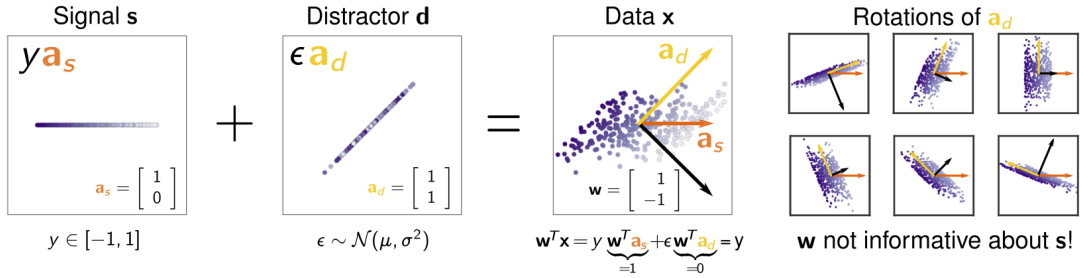
Linear vs nonlinear models

- Linear models $\rightarrow$ one weight / feature: Minimal interpretability
- Nonlinear models $\rightarrow$ take into account interdependencies between features $\rightarrow$ much more difficult to disentangle the relevance of all the features

Current methods: Gradient etc

- Popular measure: Gradient of class output wrt input feaures: $|\frac{\partial y^c}{\partial x_j}|$
- Other derived measures

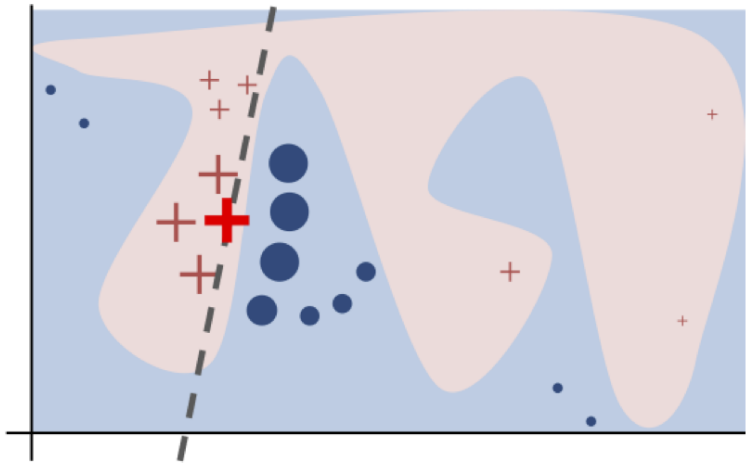
But the gradient is not a good information [Kindermans et al., 2017]



Explaining by sampling around an input [Ribeiro et al., KDD 2016]

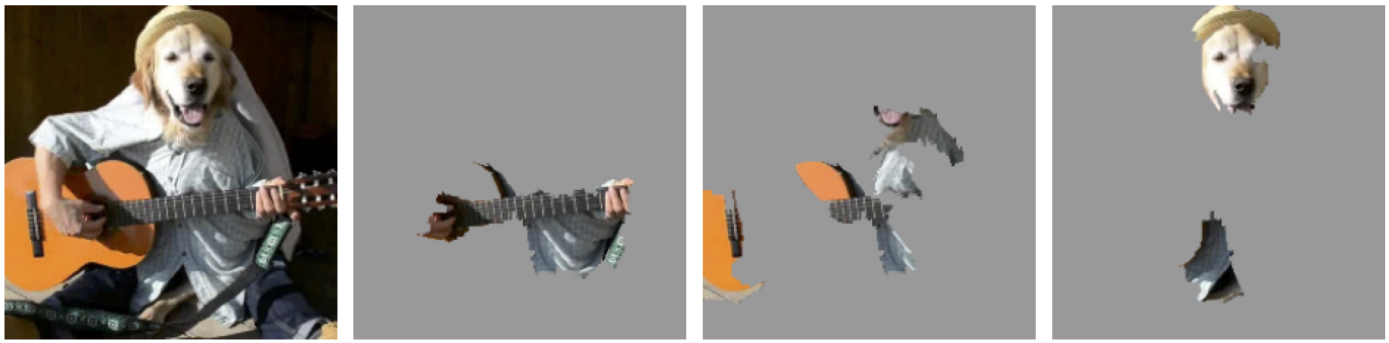
Main idea

- Decision boundary is loccaly linear (and one may then use standard techniques for linear models)



Method for explaining the decision on input x

- Sample points around a particular input x
- Fit a linear model



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Robustesse au piratage

Chatbots et apprentissage en ligne

- "A peine lancée, une intelligence artificielle de Microsoft dérape sur Twitter. L'entreprise américaine a lancé Tay, un chatbot censé discuter avec des adolescents sur les réseaux sociaux. Mais des propos racistes se sont glissés dans ces échanges." [Le Monde, Mars 2016]

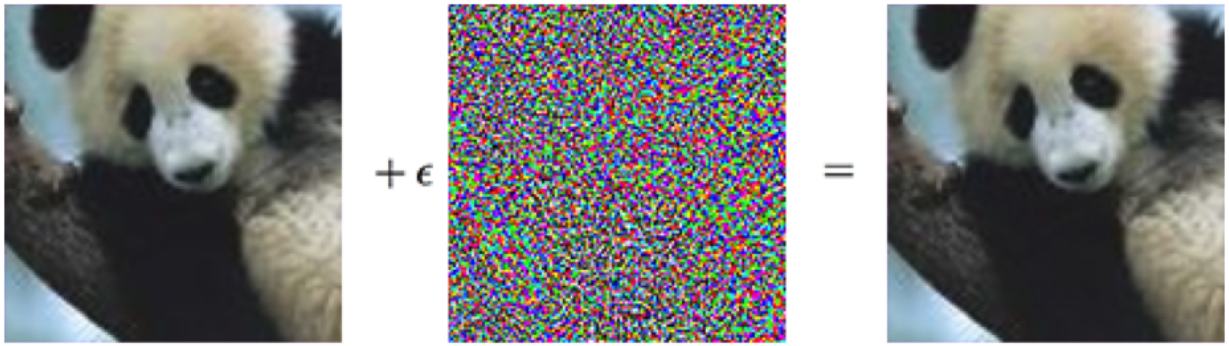
Biais de conception

- "L'application aveugle de l'apprentissage-machine risque d'amplifier les biais qui sont présents dans les données" [Tolga Bolukbasi, NIPS 2016].
- Outils de plongement lexical
  - L'homme est à la femme ce que le roi est à ... ? La reine
  - L'homme est à la femme ce que le chirurgien est à . . . ? L'infirmière
  - L'homme est à la femme ce que le programmeur informatique est à. . . ? La femme au foyer
  - L'homme est à la femme ce que l'architecte est à... ? La décoratrice
  - L'homme est à la femme ce que le commerçant est à... ? La ménagère

Adversarial examples [Goodfellow et al., 2015]

A ML/DL weakness ?

- Despite their high accuracy DNNs may be very weak, then untrustable for real life use
- Why ?
  - One may easily find adversarial noise that may fool the DNNs
  - There exist universal adversarial noise: working for any input sample
  - It may be very robust



Outline

Need for reasoning

Main idea

- Based on a query formulate a goal (information to get)
- Loop
  - Recover the information in the memory that matches the current query
  - Based on gained information and on the original query, formulate another query
- Take a decision based on the accumulated information

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

Attention mechanism

Main idea

- Most relevant element to a query?
  - Given number of memory elements u_i
  - For a (current) query q
  - Assuming queries and memory elements live un the same space
- The most relevant memory element to query q is the most similar one

$$u* = argmax_i \langle u_i, q \rangle$$

- But propagating the most similar one is not differentiable
- Use instead a smooth **argmaximum**

$$s_i = argmax_i \langle u_i, q \rangle$$
$$\alpha_i = softmax(s_i)$$
$$u* = \sum_i \alpha_i u_i$$

Attention mechanism

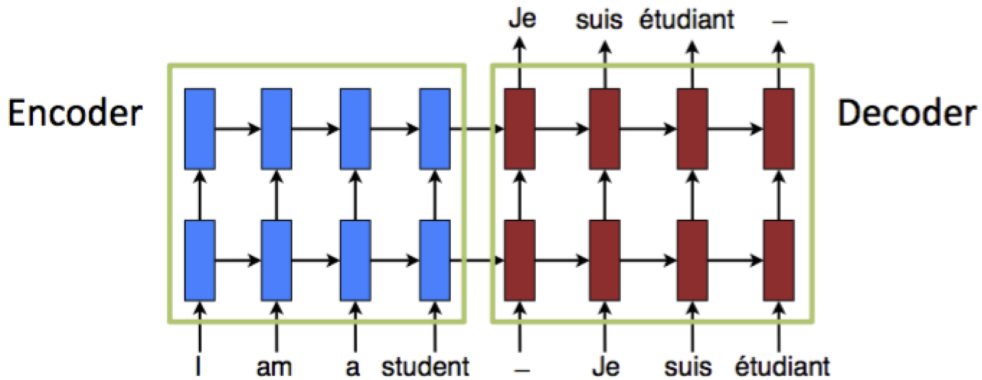
Implementation in Keras

```
inputs = Input(shape=(input_dims,))
attention_probs = Dense(input_dims, activation='softmax', name='attention_probs')(inputs)
attention_mul = merge([inputs, attention_probs], output_shape=32, name='attention_mul', mode='mul')
```

Attention mechanisms

Machine translation

- Instead of standard Seq2Seq models
- One may want to focus on one part if input sequence for producing one output word
- Attention = (fuzzy) focus on the input
- Same kind of ideas for automatic captioning

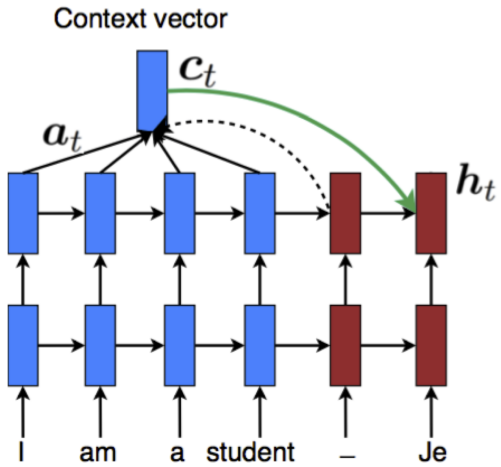


[Bahdanau and al., 2015]

Attention mechanisms

Machine translation

- Instead of standard Seq2Seq models
- One may want to focus on one part if input sequence for producing one output word
- Attention = (fuzzy) focus on the input
- Same kind of ideas for automatic captioning



[Bahdanau and al., 2015]

Attention for translation

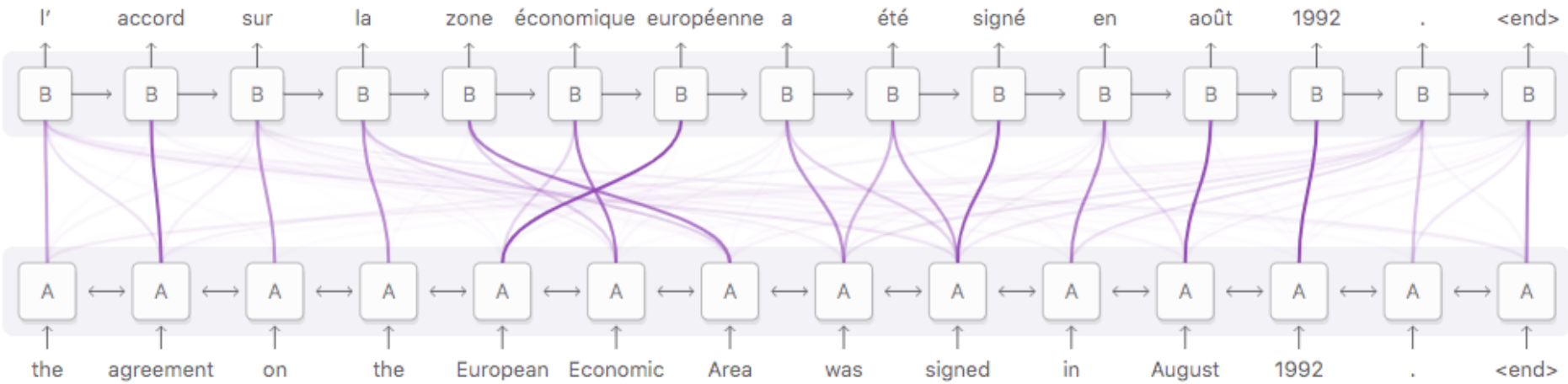


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

Attention for speech recognition

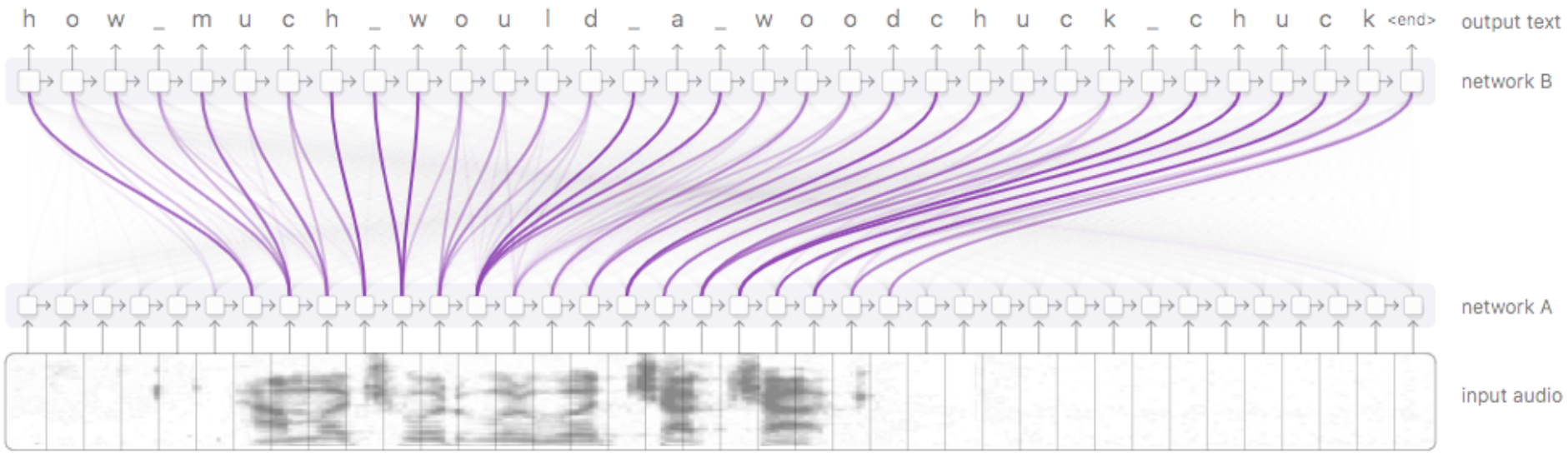


Figure derived from Chan, *et al.* 2015

Simple reasoning and baby stories

Principle Memory Networks [Weston and al., 2015]

- 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

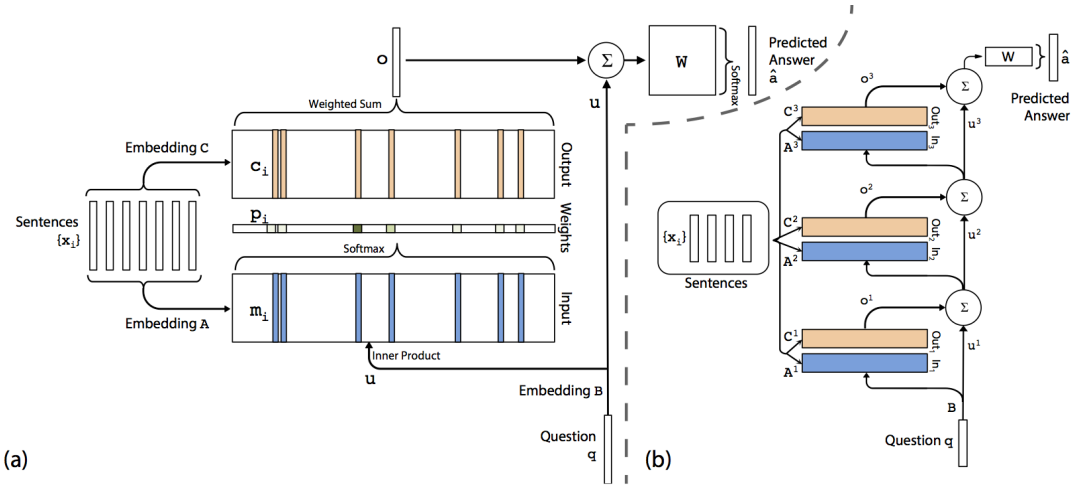


Figure 1: (a): A single layer version of our model. (b): A three layer version of our model. In practice, we can constrain several of the embedding matrices to be the same (see Section 2.2).

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

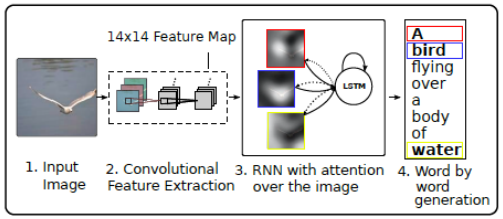
Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

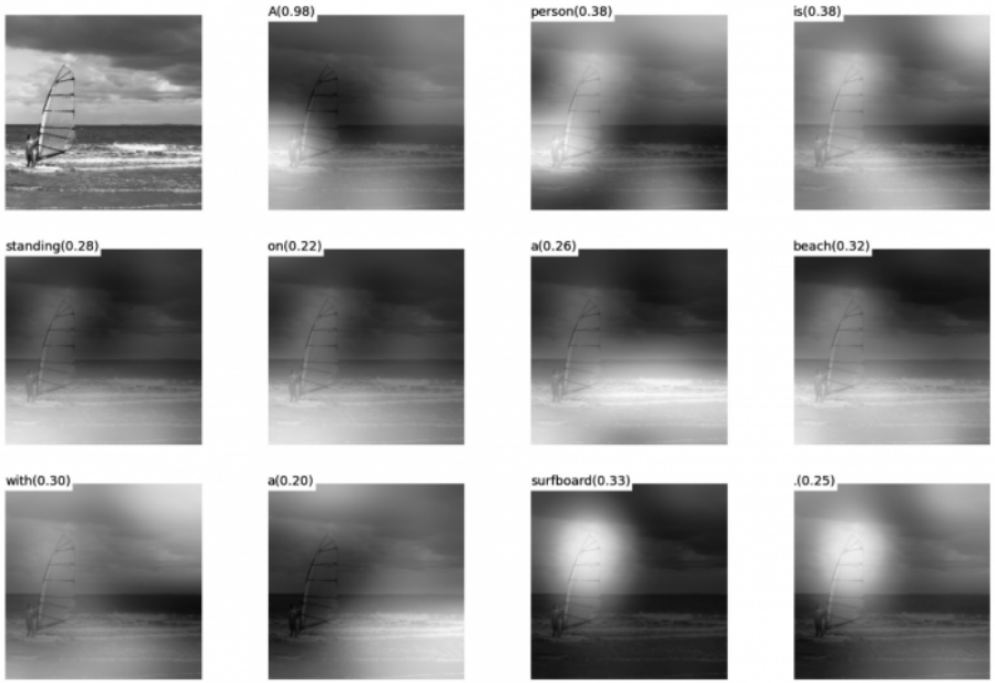
Show attend and tell

Couplage de deux espaces d'embeddings

- Texte
- Images

Figure 1. Our model learns a words/image alignment. The visualized attentional maps (3) are explained in section 3.1 & 5.4





General reasoning

More complex reasonning tasks

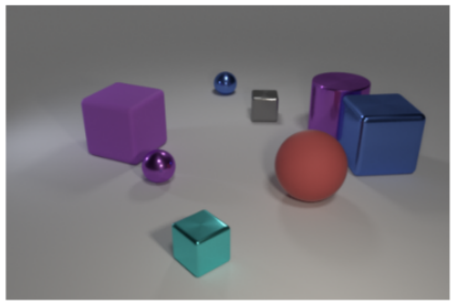


Figure 1: A sample image from the CLEVR dataset, with a question: "There is a purple cube behind a metal object left to a large ball; what material is it?"

[Hudson et al., 2018]

- Requires few steps of question answering like queries

Outline

Conclusion

- This does not happen twice in a researcher's life !
- Deep Learning is everywhere, many opportunities for designing new systems, solving new tasks
- Huge spread of Machine Learning and Deep Learning ideas: Neurosciences, Security, Health, HEP...
- On the contrary ow what many people say and think
  - It is not so easy (except if you reuse the code (= the design and the learning) from someone else)
  - Many thinks to understand
  - Many things to invent