Introduction au Deep Learning

Thierry Artières

Equipe QARMA d'apprentissage automatique
Pole Science des Données - LIS lab
Ecole Centrale Marseille

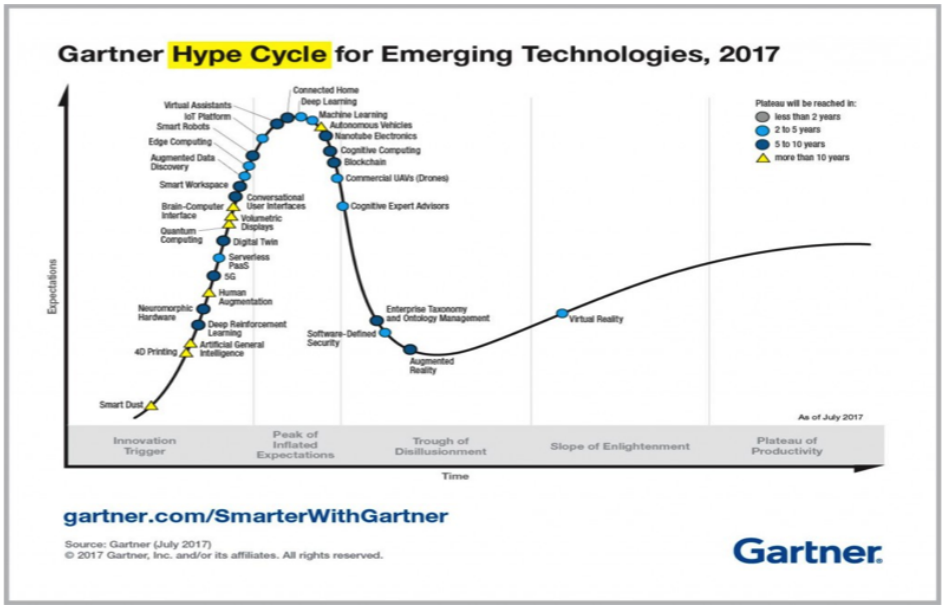
July 2, 2019

Outline

Where is AI?

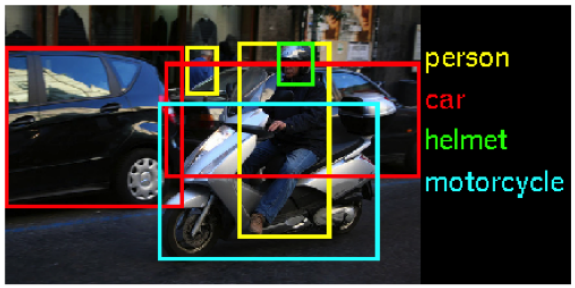
Constat

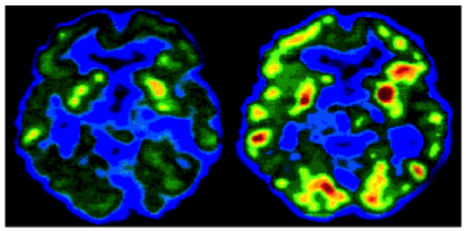
- The original AI was the General AI (IA forte)
- Today 60 years after the Dartmouth meeting
  - We have achieved some NIA results (IA faible)
  - We can start thinking more seriously about GAI
  - These are just the premises.



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)

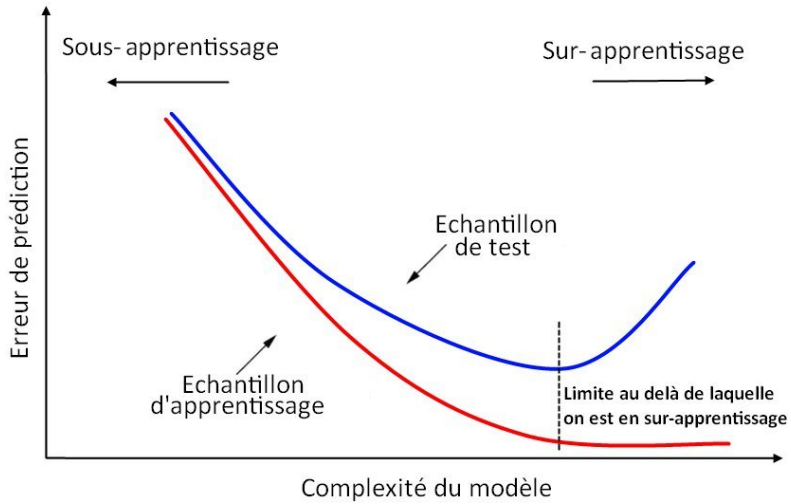
What is it?

- 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

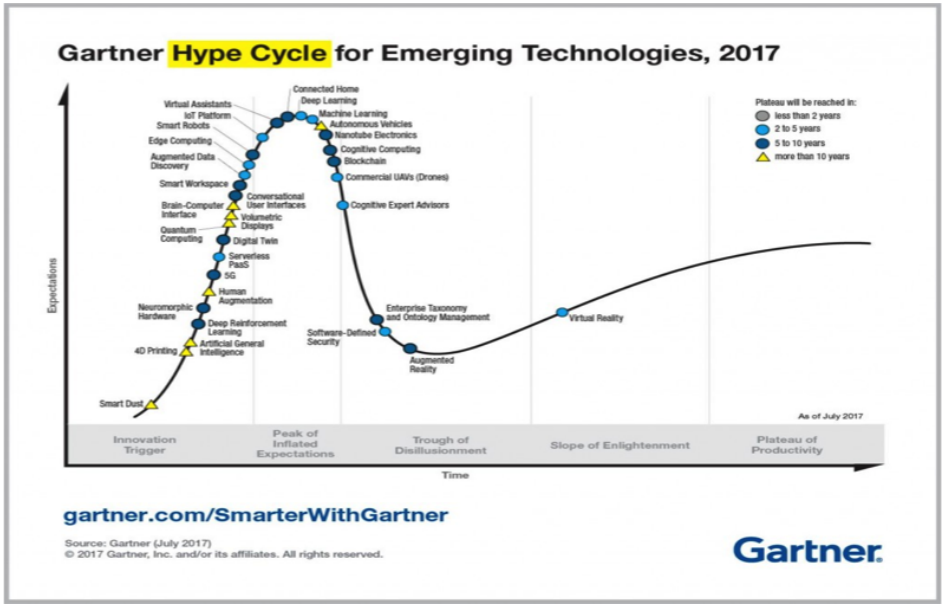
Key dates

- 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.]
- ...

AI today

Where are we?

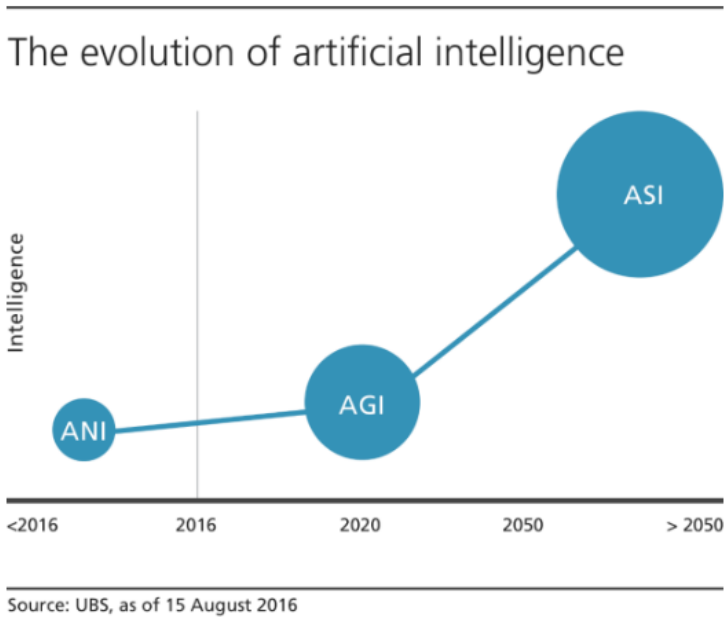
- The original AI (Dartmouth workshop) was General AI (IA forte)
- Today 60 years after Dartmouth
  - We have achieved NIA results (IA faible)
  - We can start thinking more seriously about GAI
  - These are just the premises.



AI today

Where are we?

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



Deep Learning today

Spectaculary breakthroughs - fast industrial transfer

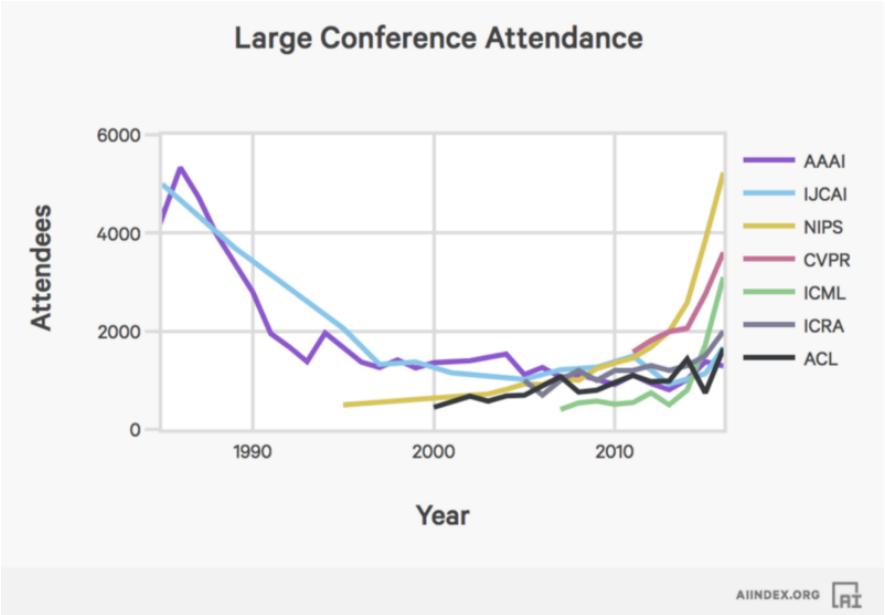
- Images, Videos, Audio, Speech, Texts
- Successful setting
  - Structured data (temporal, spatial...)
  - Huge volumes of datas
  - Huge models (millions of parameters)
  - Huge storage and computing resources (GPU, TPU)

	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

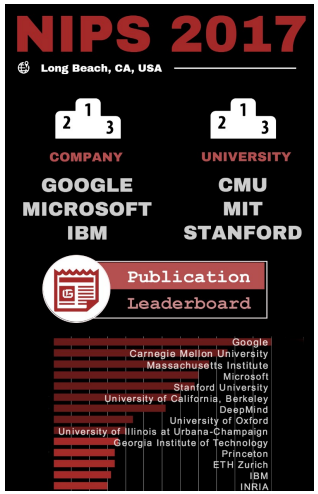


Machine Learning and Deep Learning today

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

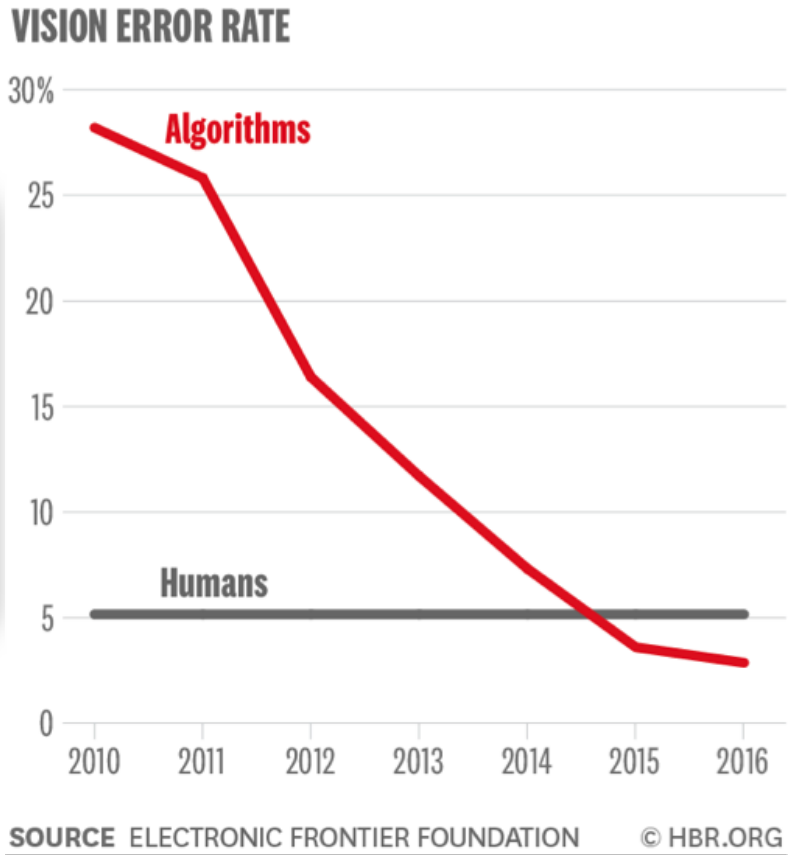
New publication mode

- 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)

Spectaculary breakthrough

Computer vision

Real time Object recognition



Spectaculary breakthrough

Speech

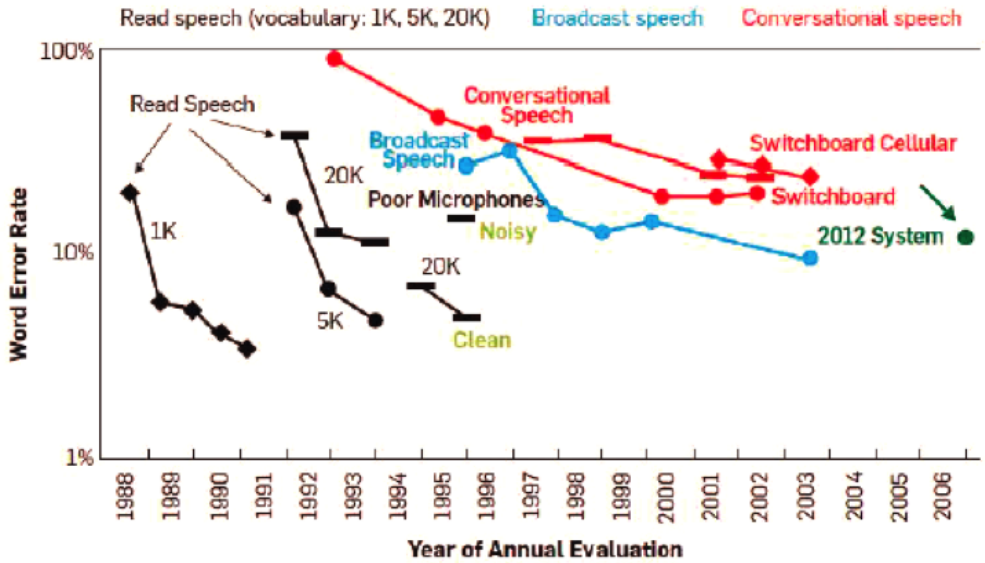


FIGURE 2.7 Historical progress on reducing the word error rate in speech recognition systems for different kinds of speech recognition tasks. Recent competency for the "difficult switchboard" task (human conversation in the wild) is marked with the green dot. SOURCE: X. Huang, J. Baker, and R. Reddy, 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

Games

- BackGammon, Chess, Go

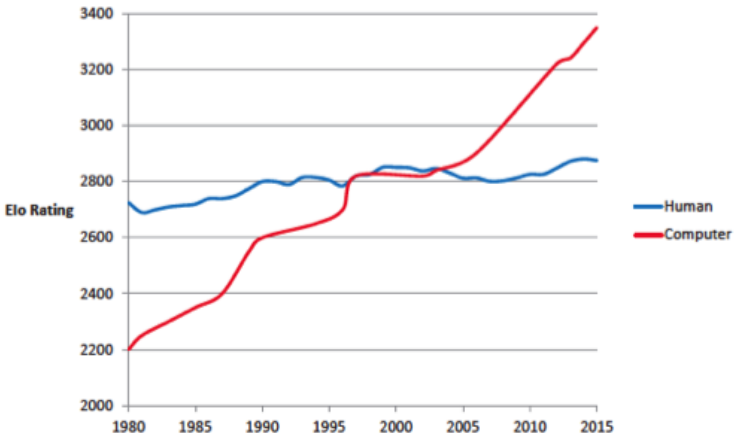
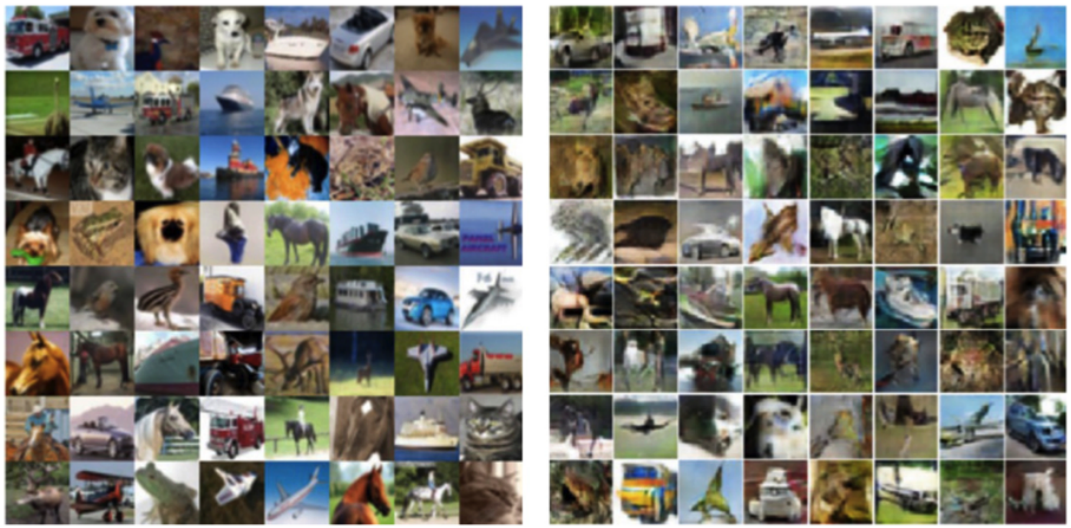


FIGURE 2.3 Elo scores—a measure of competency in competitive games—showing the chess-playing competency of humans and machines, measured over time. SOURCE: Courtesy of Murray Campbell.

Spectaculary breakthrough

Image generation

Recent Nvidia results



Training Data Samples

Spectaculary breakthrough

Should we still trust what we see?

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, LSTMs, ...)

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!)

Outline

Where DL comes from

F. Rosenblatt 1958 : The perceptron

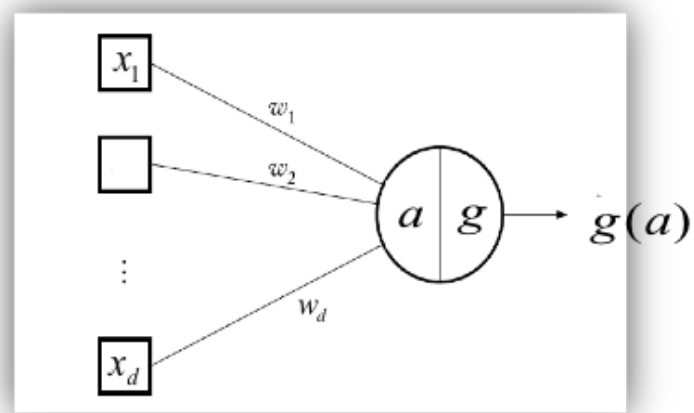
A single Neuron

One Neuron

- Elementary computation

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

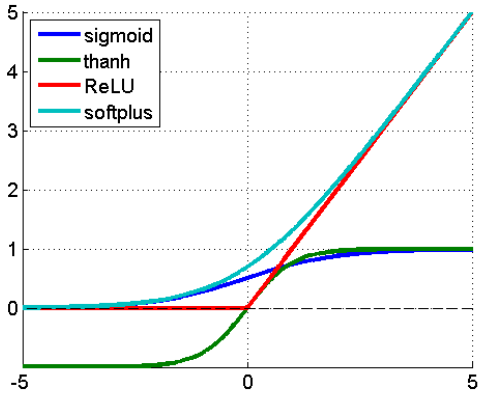
$$\text{output} = g(a(x))$$



Non linearity : g

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

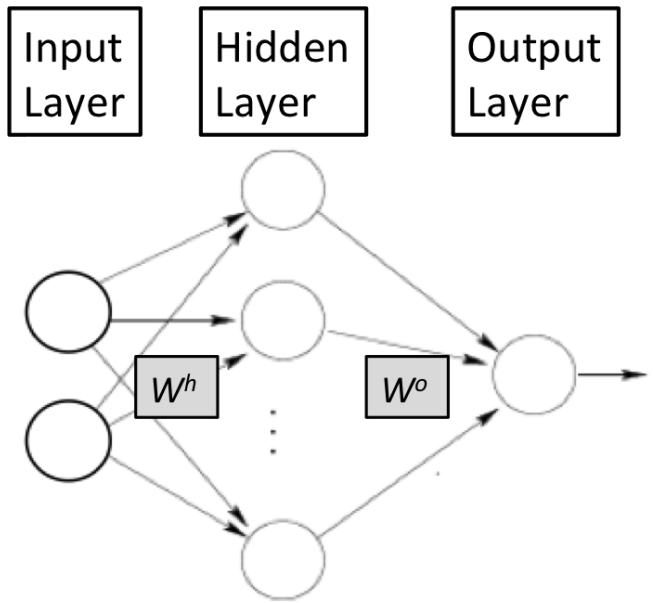
$$f(x) = 0 \text{ if } x \leq 0$$
$$= x \text{ otherwise}$$



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

Learning a MLP

Learning as an optimization problem

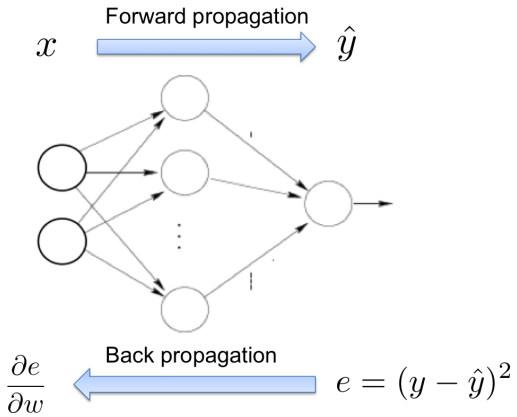
- Objective function of parameters set w for a given training set T

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

- 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



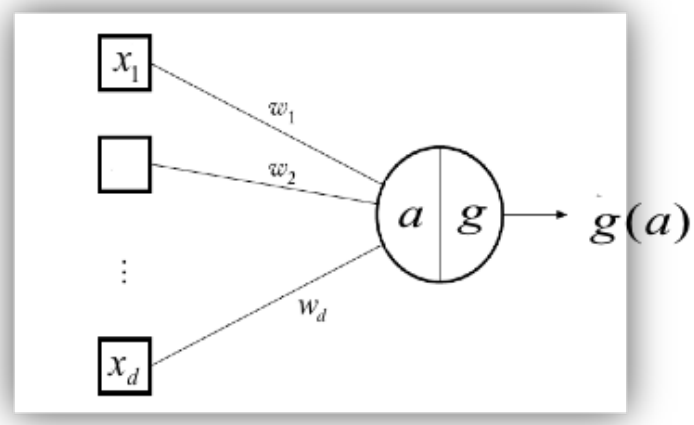
A single ReLU Neuron

One Neuron

- Elementary computation

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

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



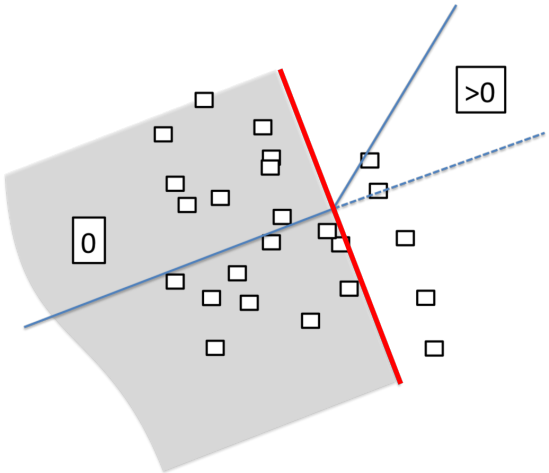
A single ReLU Neuron

One Neuron

- Elementary computation

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

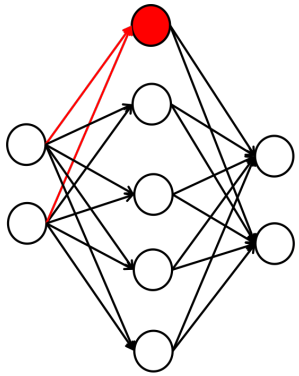
$$\text{output} = ReLU(a(x))$$

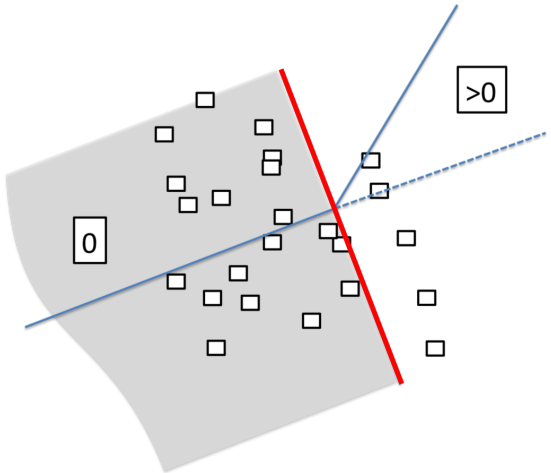


What a MLP may compute

What does a hidden neuron

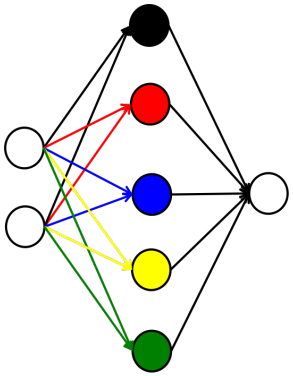
- Divides the input space in two

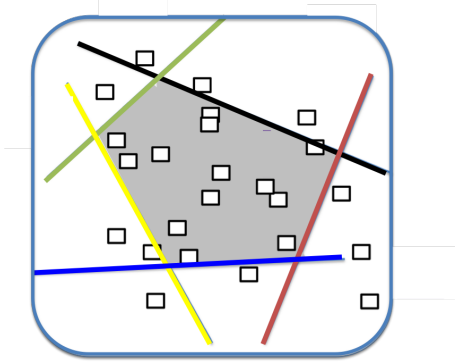




Combining multiple hidden neurons

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

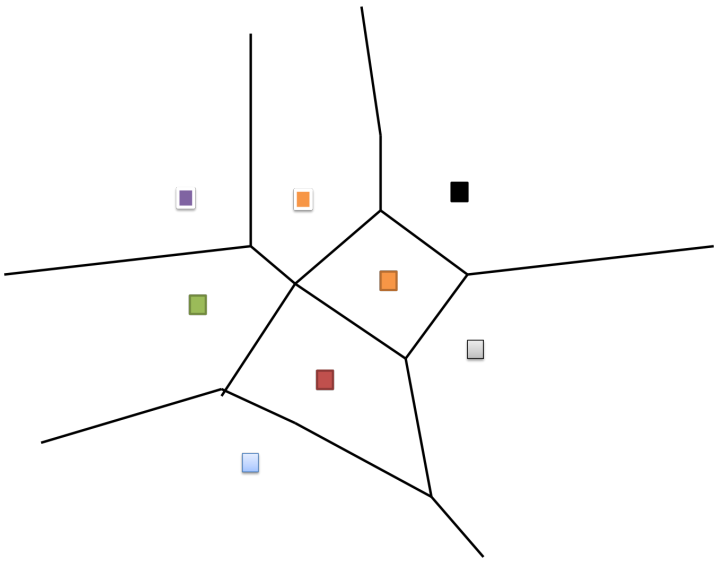


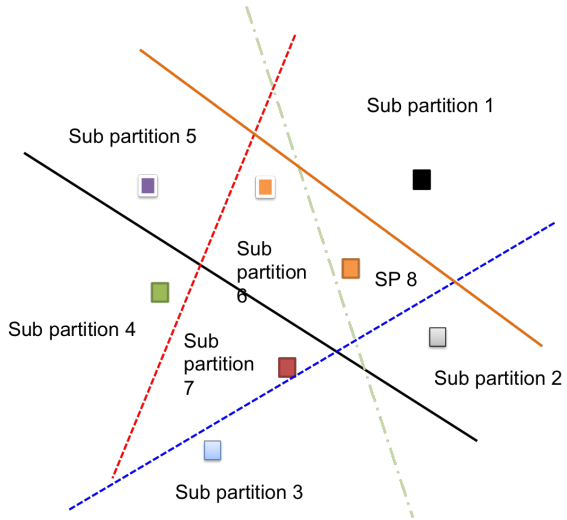


MLP compute distributed representations

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:

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

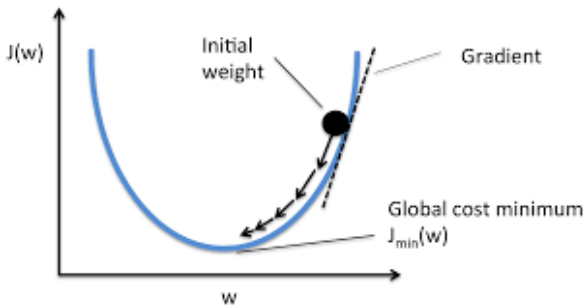
  as an approximate realization of the function f where f is independent of ϕ ; that is : $|F(x) - f(x)| < \epsilon$ for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.
- Existence theorem only
- Many reasons for not getting good results in practice

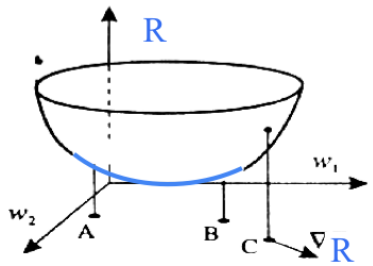
Outline

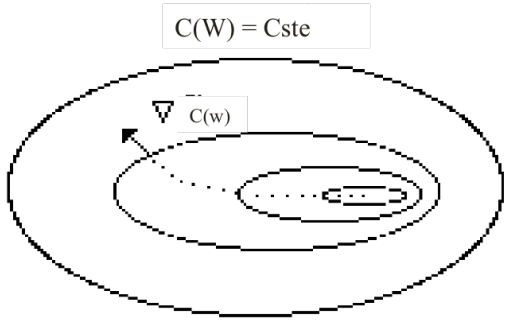
Gradient Descent Optimization

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*

Optimal learning rate and convergence speed

Second order point of view

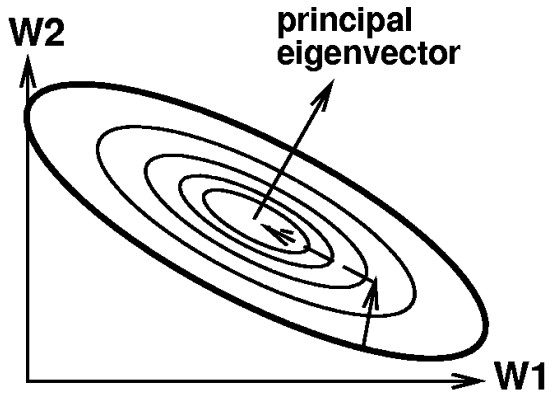
- Taylor expansion, noting $\nabla^2 C(w)$ the Hessian (a $N \times N$ matrix with N a model with parameters)

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

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

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

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



[Lecun et al, 93]

Gradient Descent: Stochastic, Batch and mini batchs

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

Batch vs Stochastic vs Minibatchs

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

Optimization routines

Many SGD variants popular in DL

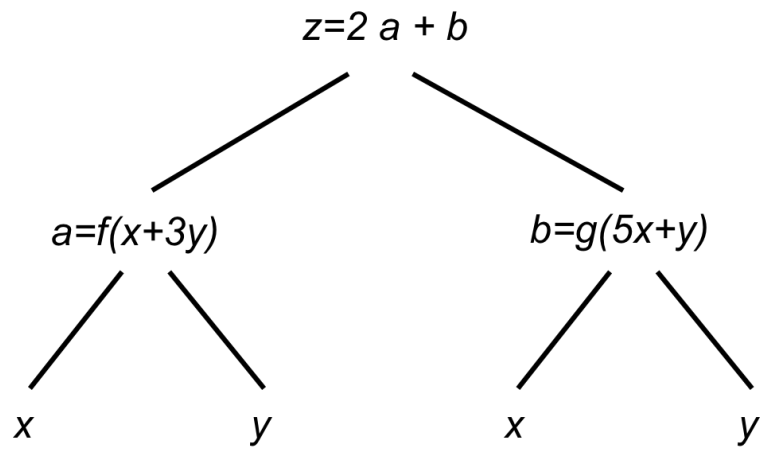
- SGD with Momentum
- Nesterov accelerated gradient
- Adragrad
- Adadelta
- RmsProp
- ...

Gradient Computation: Chain rule

Gradient of a function

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

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



Equivalent computation with the Chain rule

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

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

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

With:

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

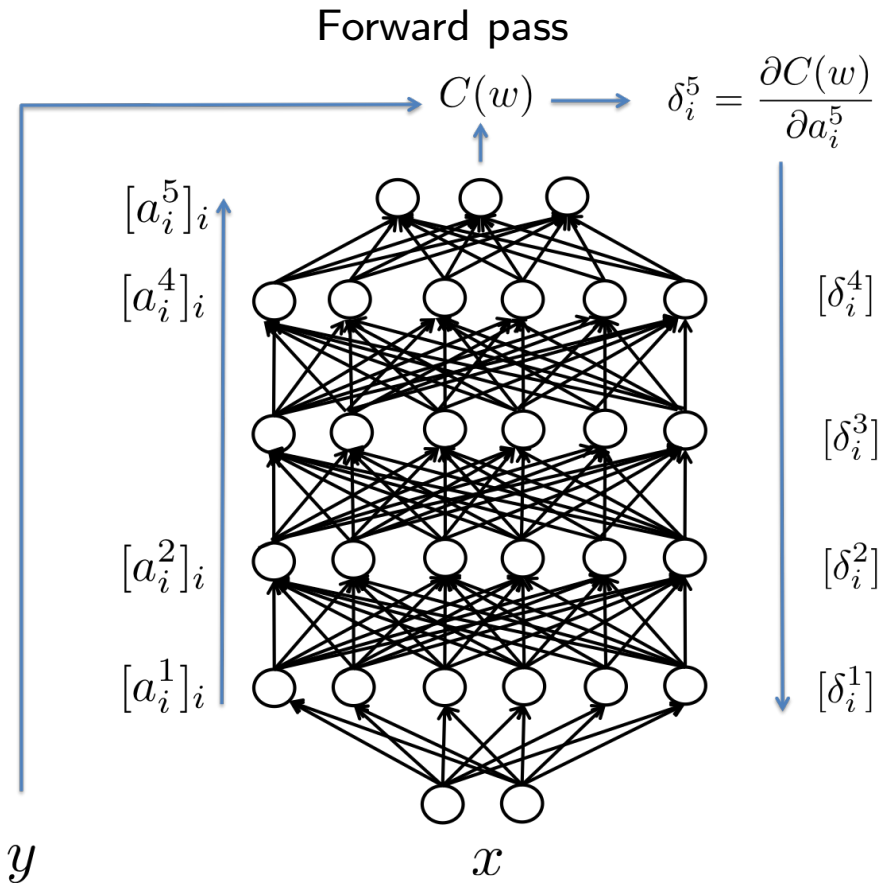
$$\frac{\partial a}{\partial x} = f'(a \times x + 3 \times y)$$

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

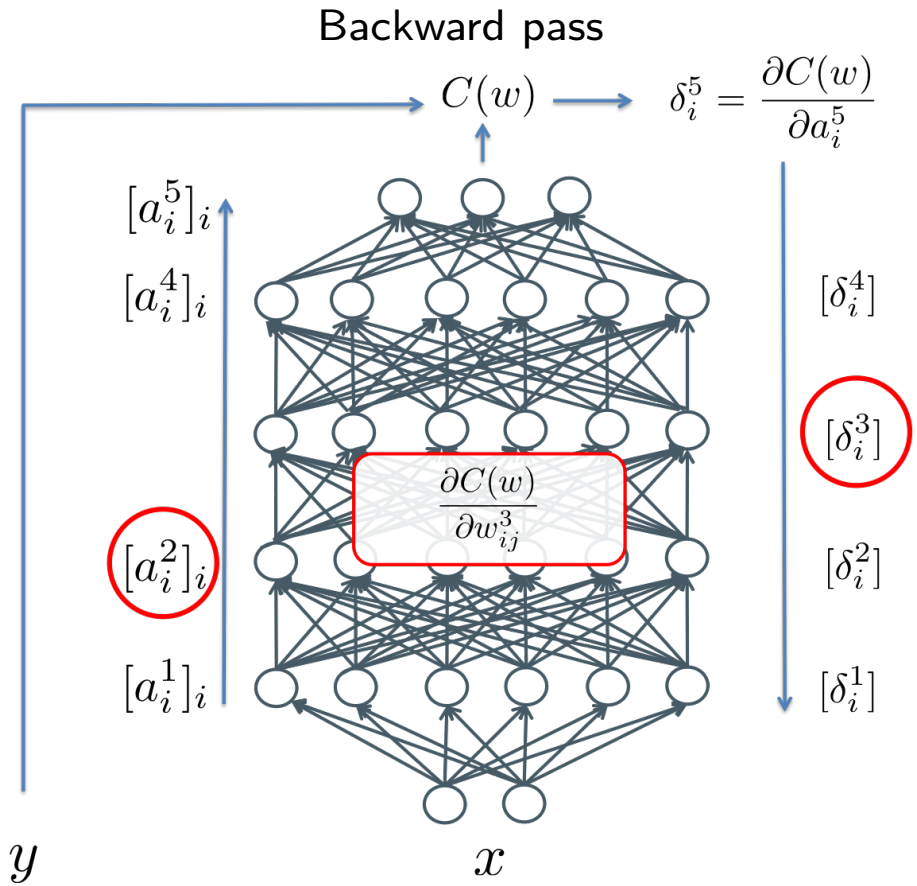
$$\frac{\partial a}{\partial y} = 3 \times f'(a \times x + 3 \times y)$$

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

BackPropagation



BackPropagation



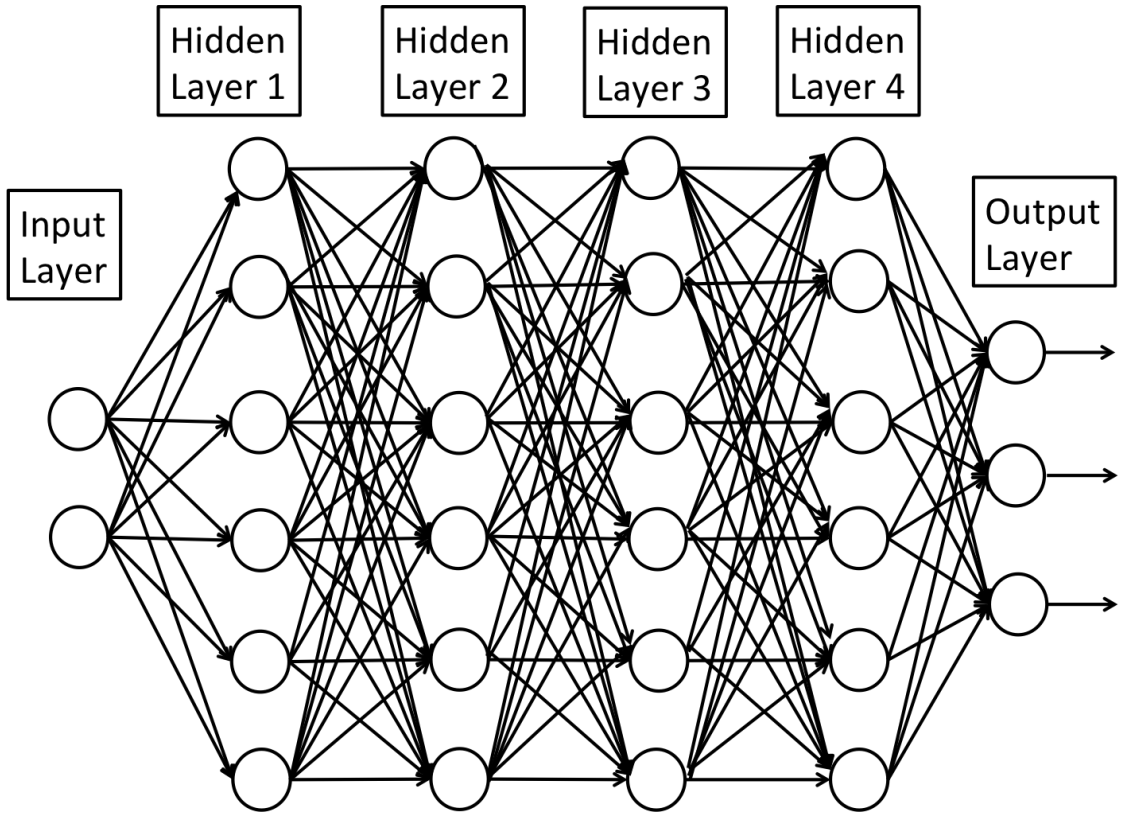
Guiding the learning through regularization

Regularization

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

Outline

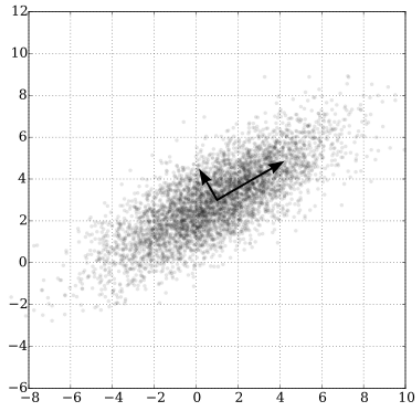
Dense architecture



Autoencoders

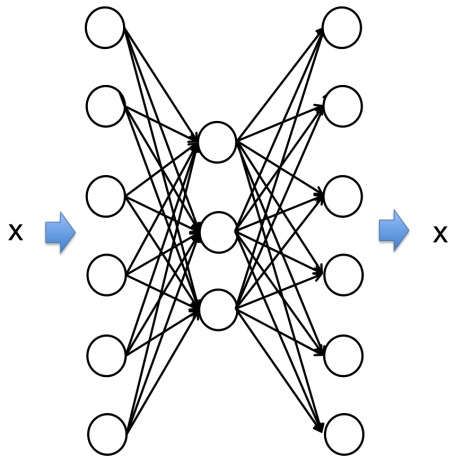
Principal Component Analysis

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



NN with Diabolo shape

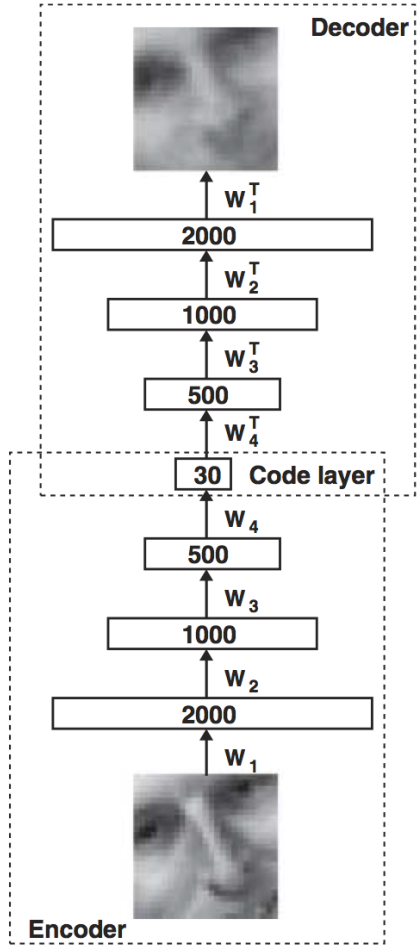
- Reconstruct the input at the output via an intermediate (small) layer
- Unsupervised learning
- Non linear projection, distributed representation
- Hidden layer may be larger than input/output layers



Deep autoencoders

Deep NN with Diabolo shape

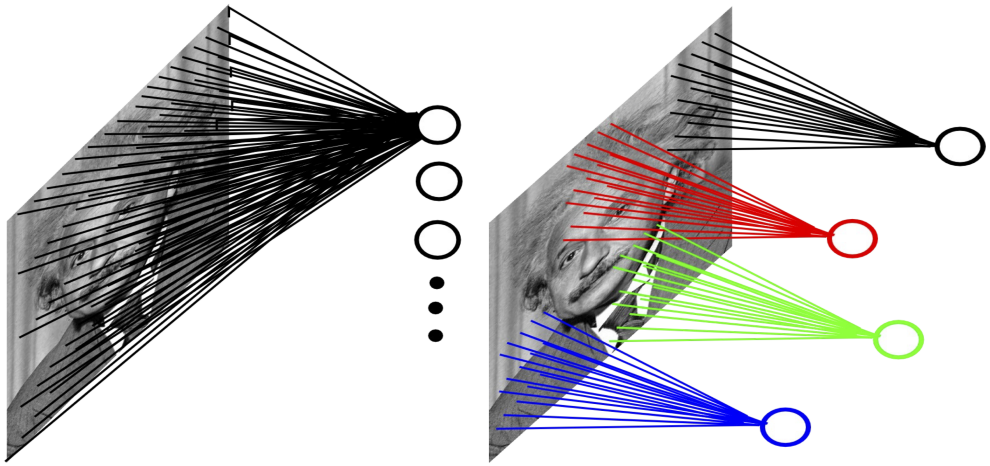
- Extension of autoencoders (figure [Hinton et al., Nature 2006])
- Pioneer work that started the Deep Learning wave



Convolutional architectures

Convolutional layers

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

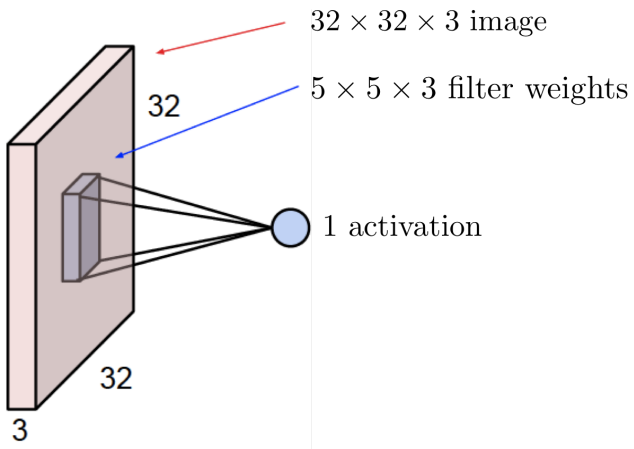


Dense vs. Locally connected

[LeCun and Ranzato Tutorial, DL, 2015]

Convolution layer

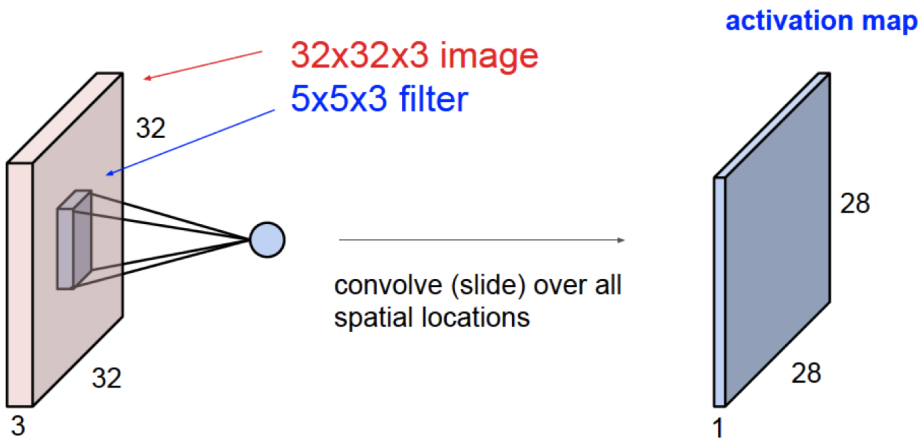
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

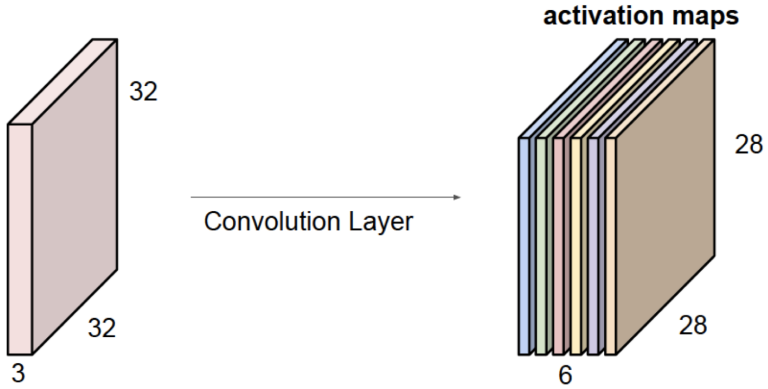
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

Convolution layer

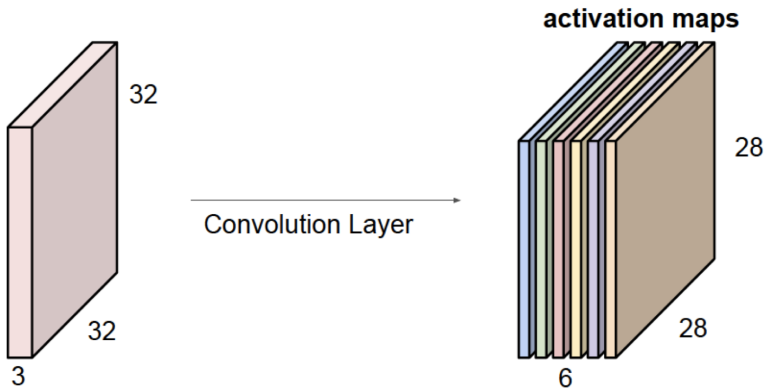
Convolution of multiple inputs with several small filter yields several activation maps



[From Fei Feil Li slides]

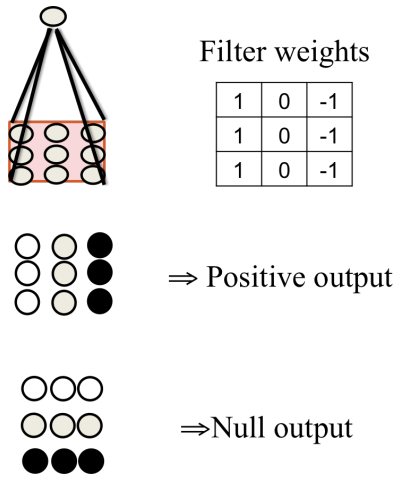
Convolution layer

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



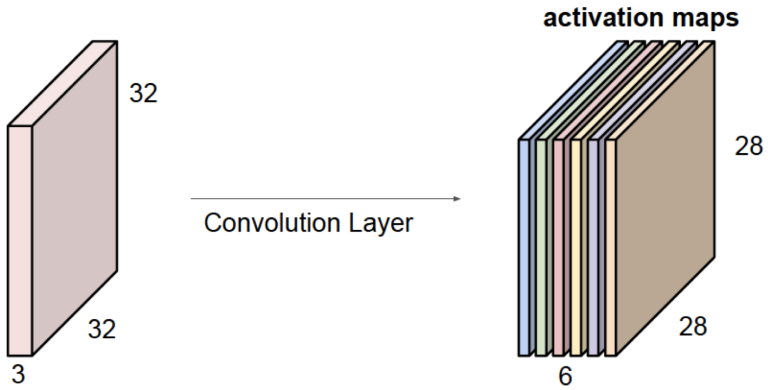
[From Fei Feil Li slides]

Example of a filter



Convolution layer

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



[From Fei Feil Li slides]

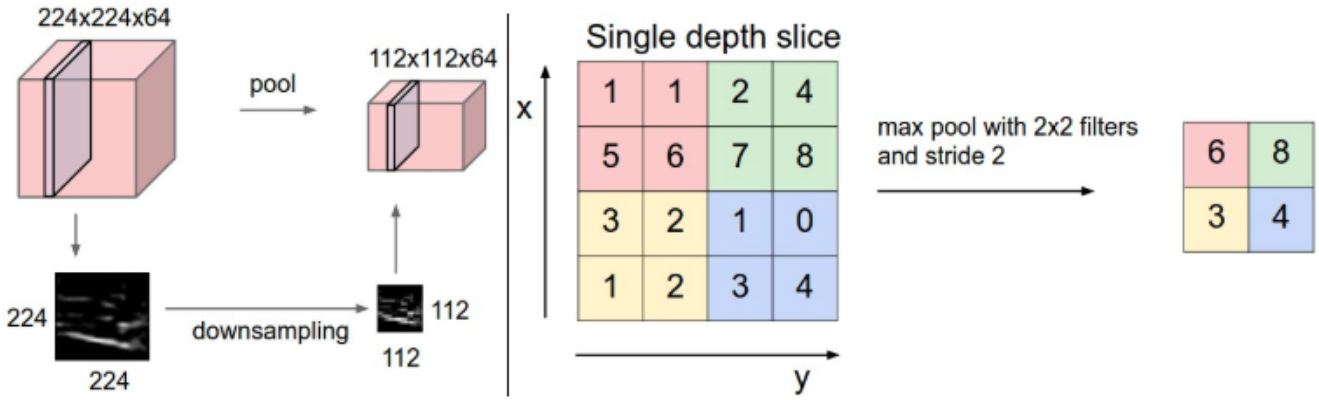
Very few free parameters but intensive computation

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

Convolution layer

Aggregation layer

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



Complexity

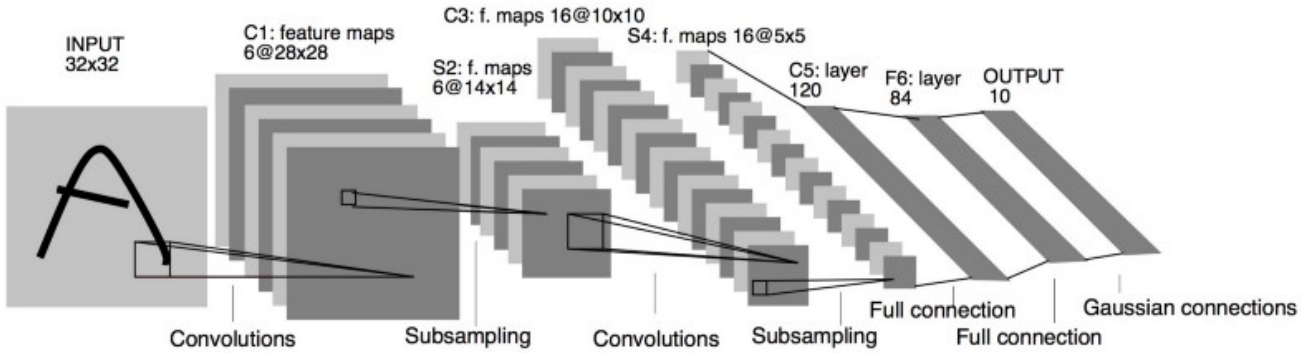
- No parameters
- Moderate computation

Old and new convolutional architectures

Convolution architectures

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

LeNet [LeCun and al., 1997]

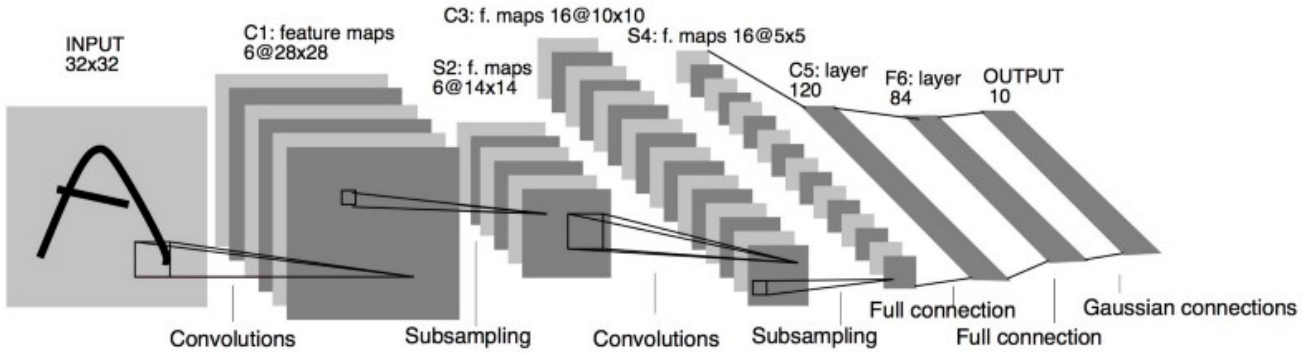


Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]

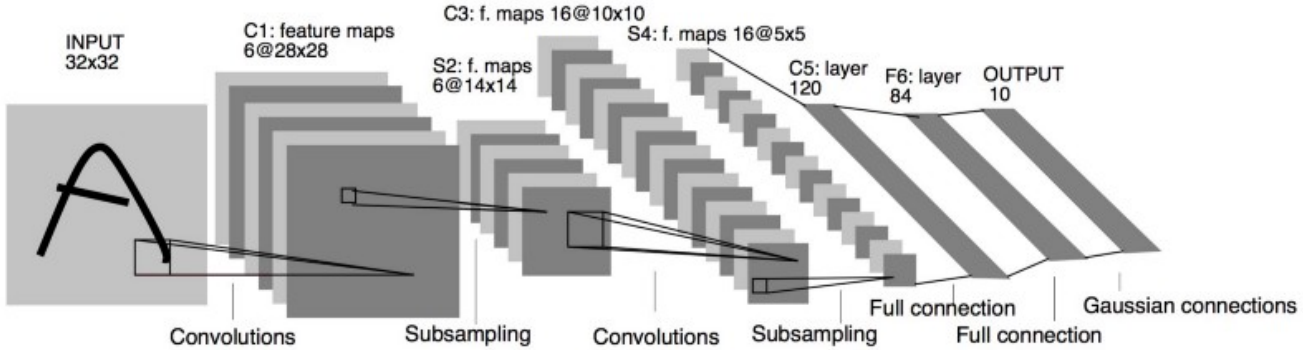


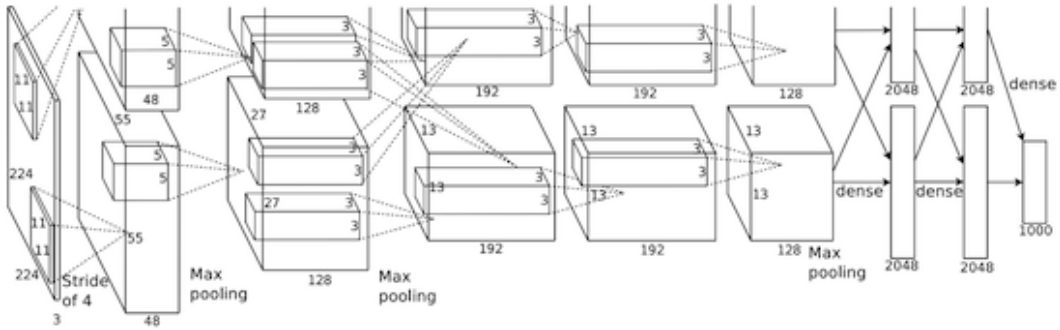
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]





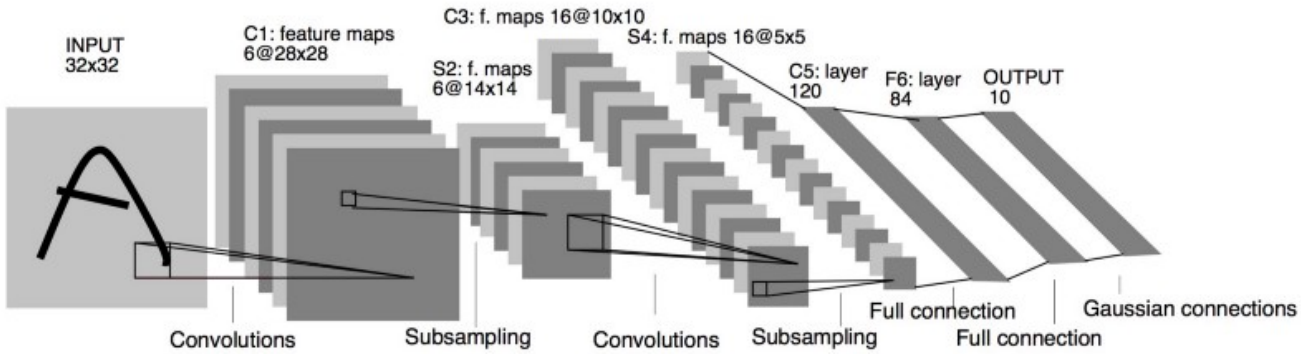
AlexNet [Krizhevsky and al., 2012]

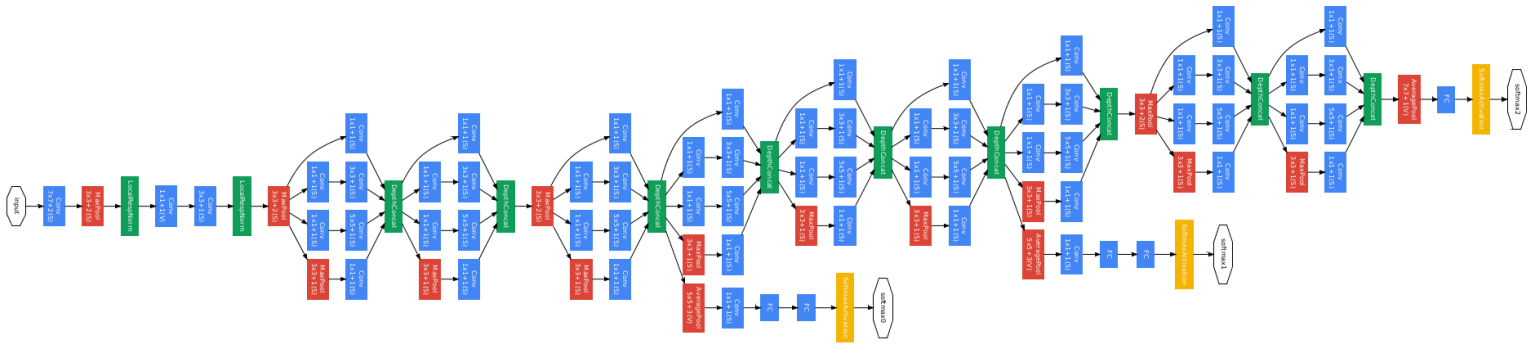
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]



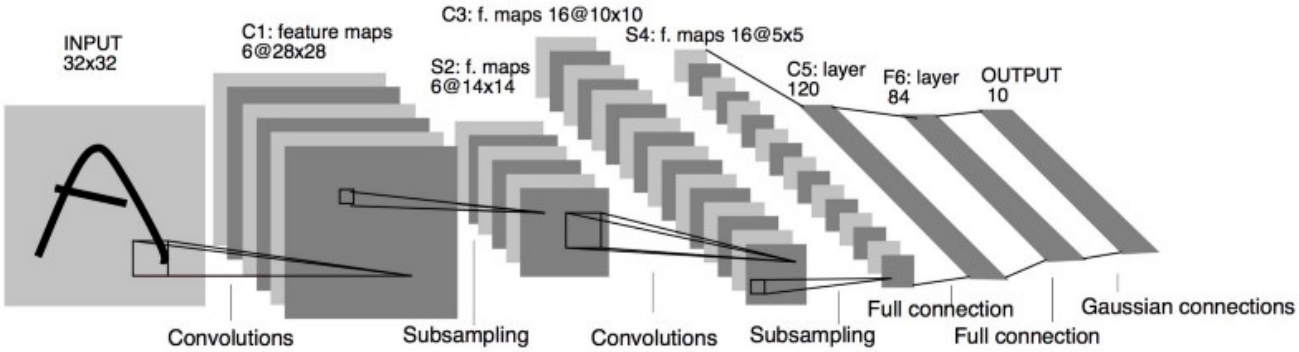


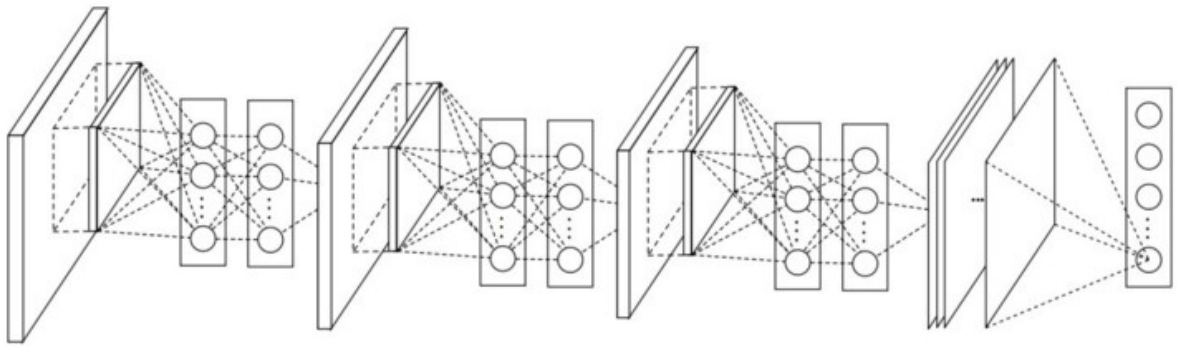
Old and new convolutional architectures

Convolution architectures

- Dit it change so much ?

LeNet [LeCun and al., 1997]



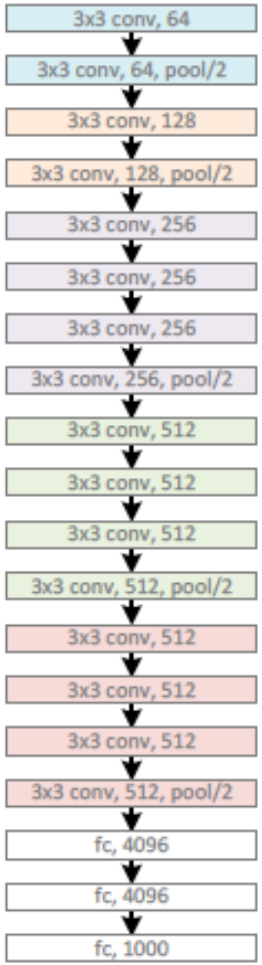


NetworkInNetwork [Lin and al.,2013]

Old and new convolutional architectures

VGG idea

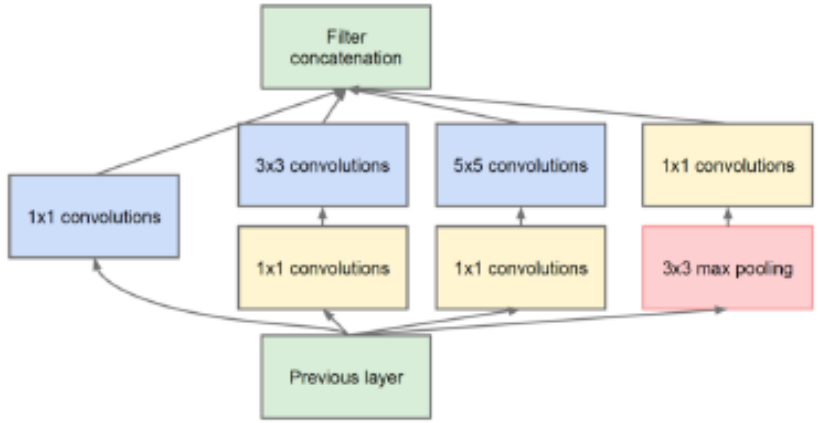
- Modular design
  - 3x3 conv as basis
  - Stack the same module
  - Same computation for each module (1/2 spatial size => 2x filters)

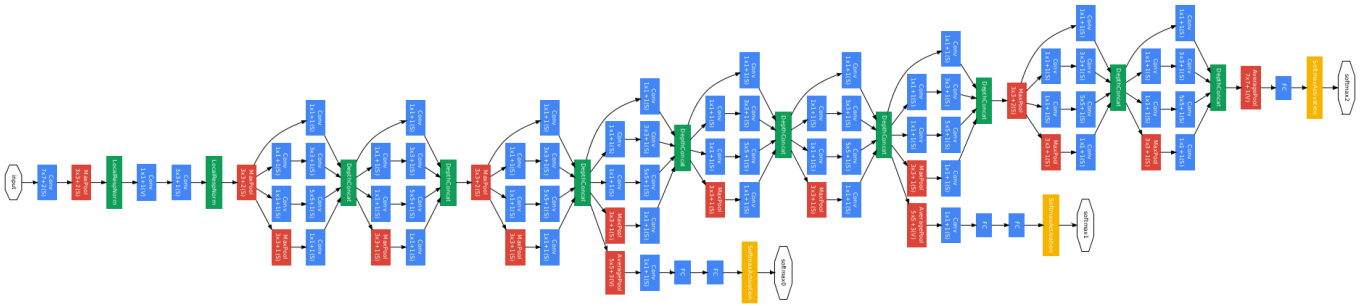


Old and new convolutional architectures

Inception idea (GoogleNet)

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

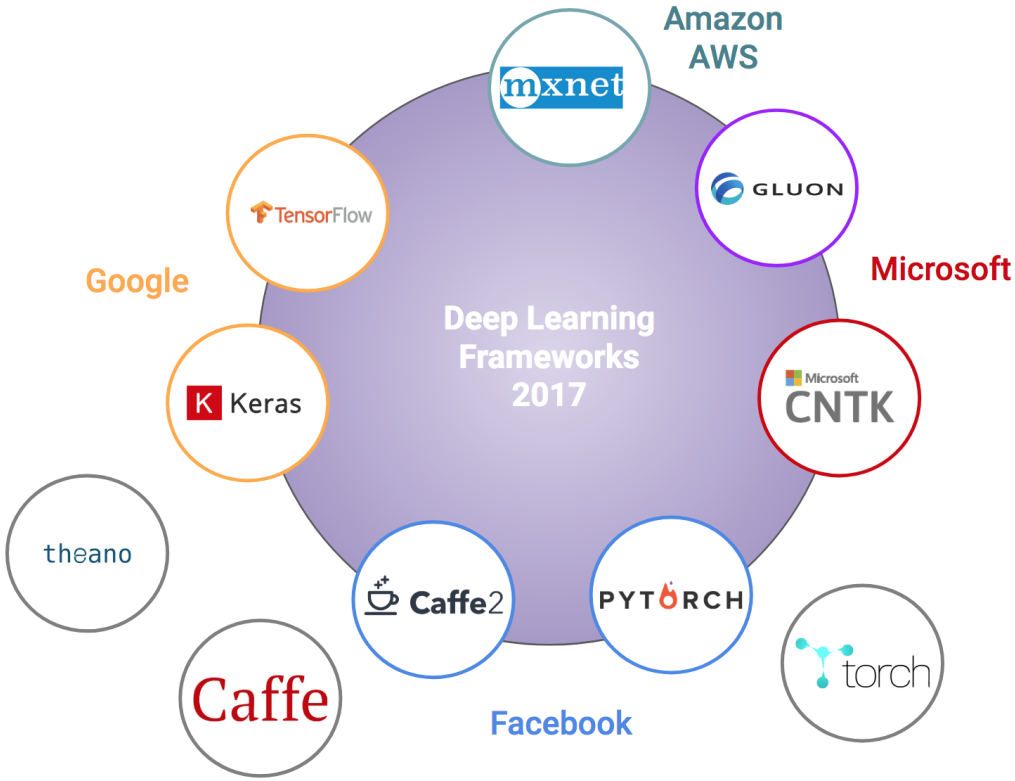




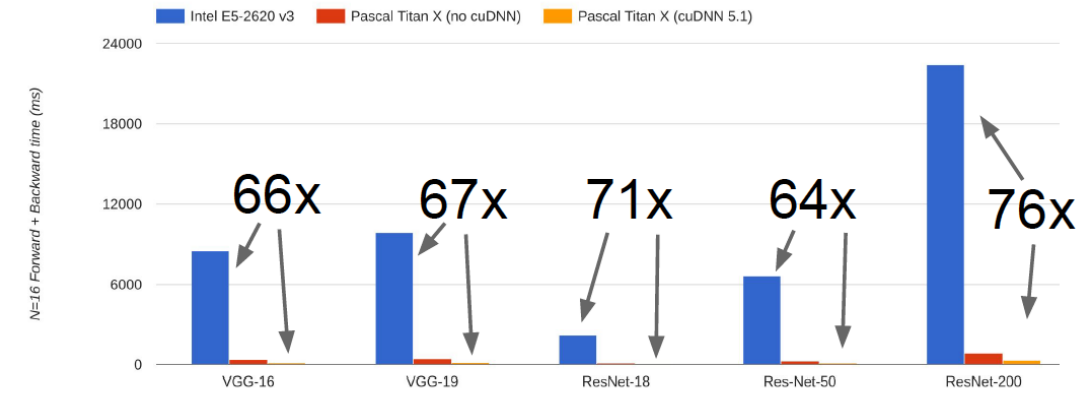
Outline

Plateforms

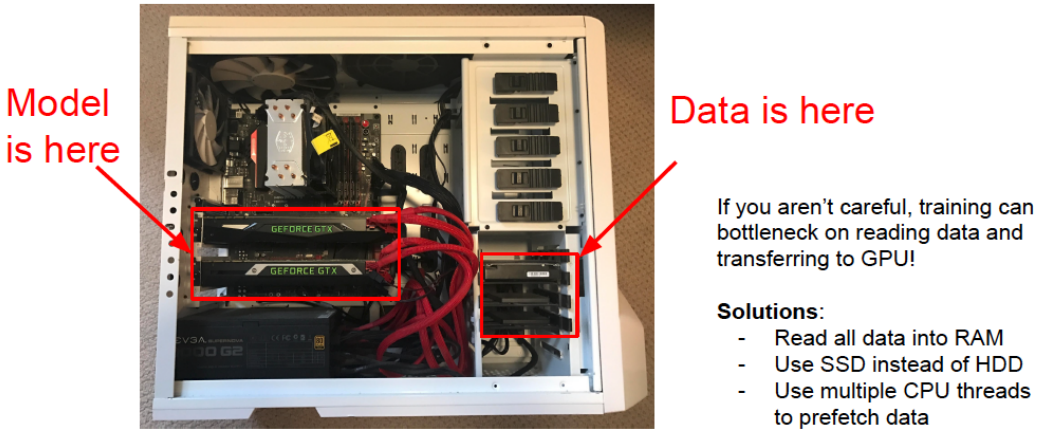
Large and active community (forums, models are available when published...) for each of these



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:

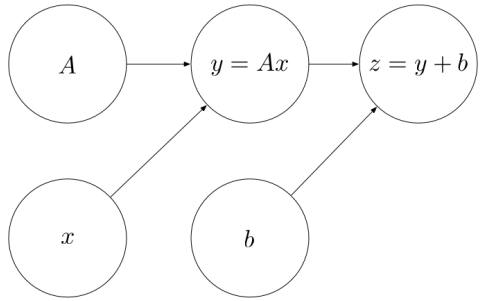
- Read all data into RAM
- Use SSD instead of HDD
- Use multiple CPU threads to prefetch data

Computation graph

Computation graph for a calculus

One may build a computation graph form the calculus definition

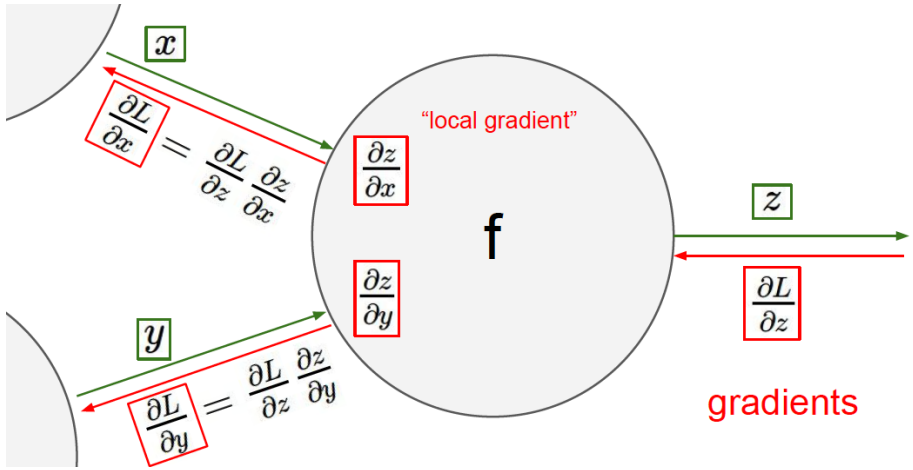
$$z = Ax + b$$



Automatic differentiation

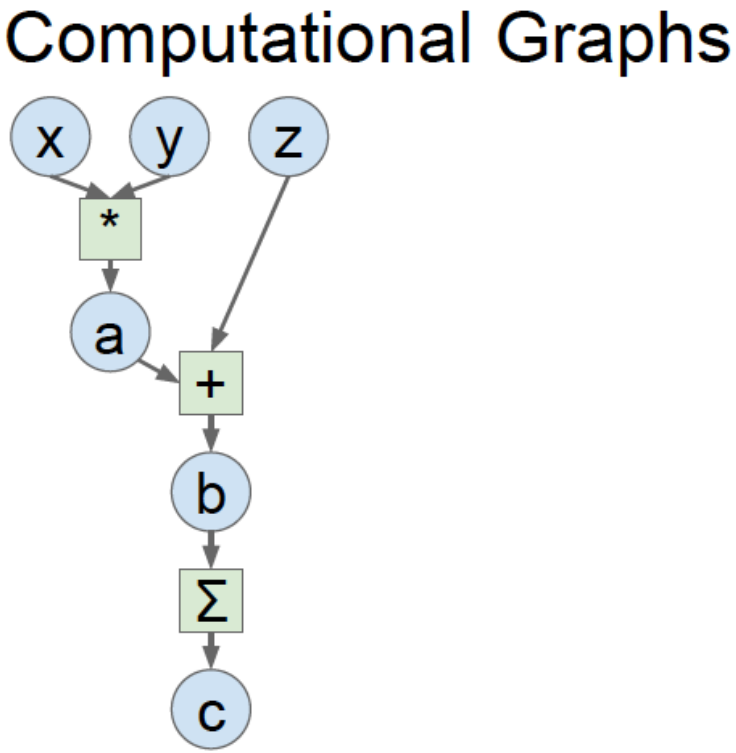
From a computation graph one may automatically compute the backward differentiation graph !

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



From Fei Fei Li slides

Computation graph and Pytorch




```
[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.],
        [2., 2., 2., 2.]], device='cuda:0')
```

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)

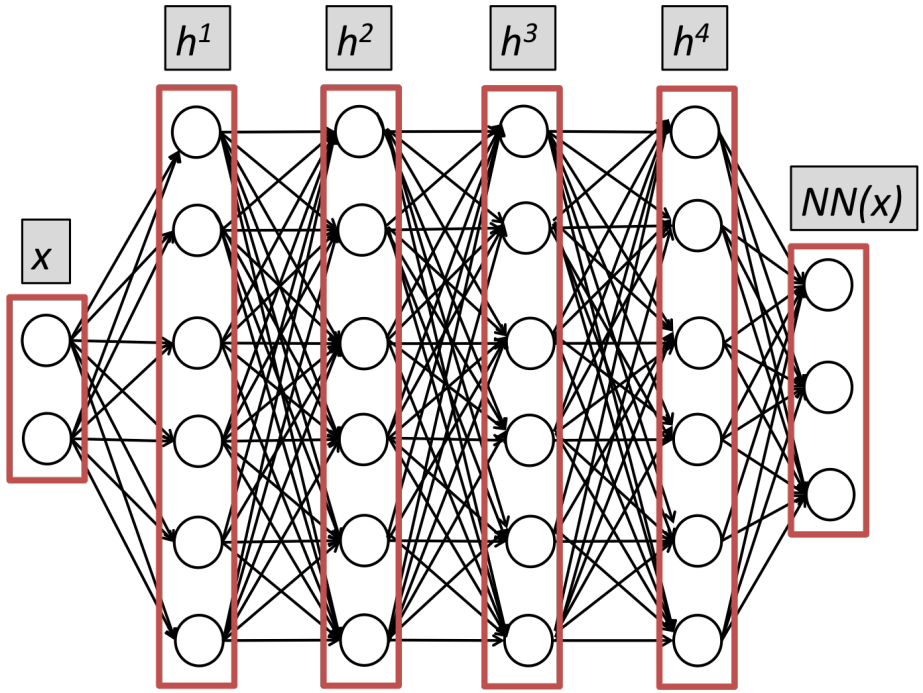
```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        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

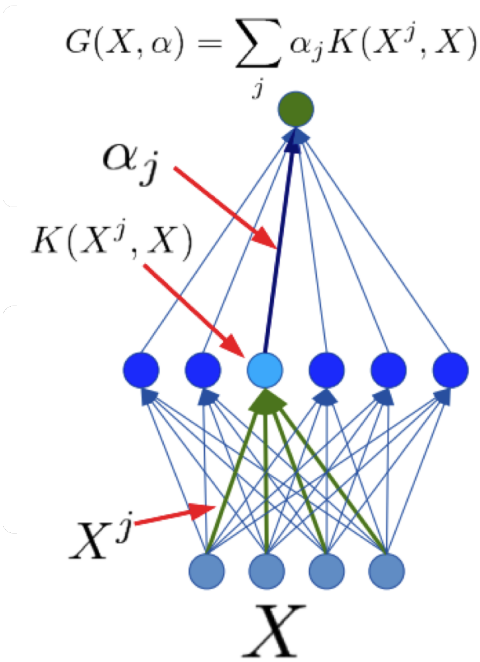
$$h^i(x) = g(W^i \times h^{i-1}(x))$$



Shallow vs deep models

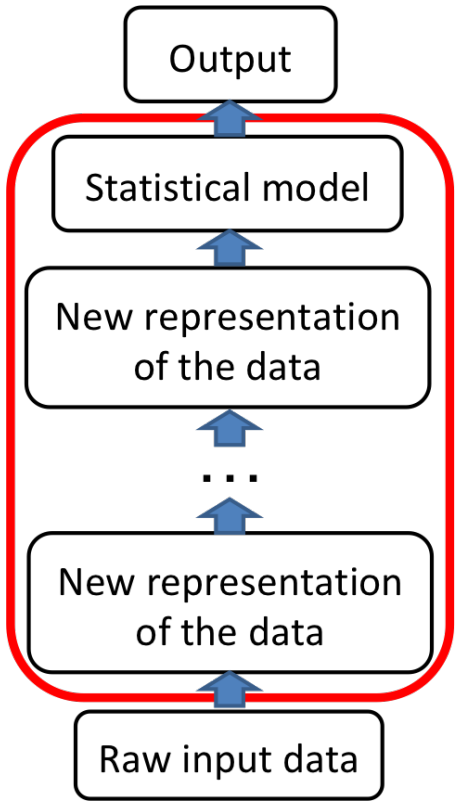
From [LeCun tutorial Statlearn]

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



DL

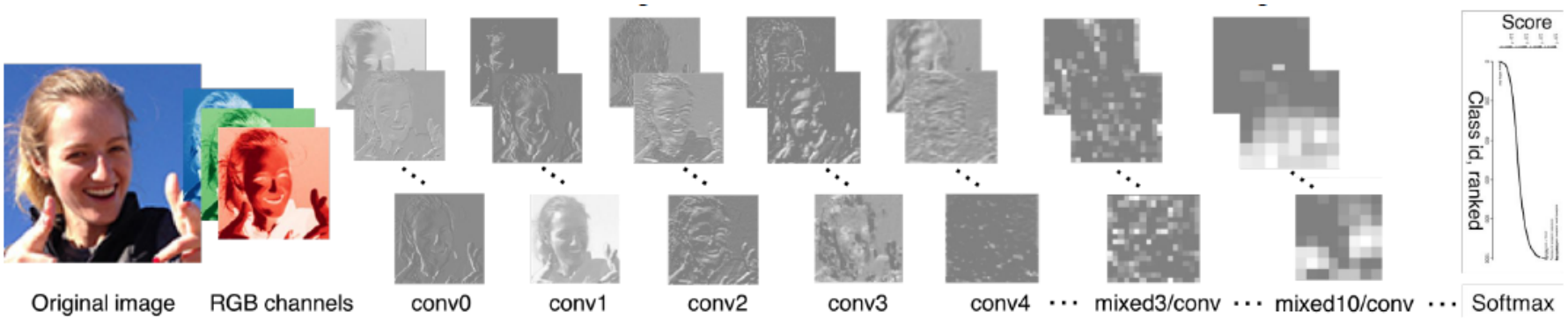
- Joint learning of a hierarchy of representations and of a prediction model



Feature hierarchy : from low to high level

What feature hierarchy means ?

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

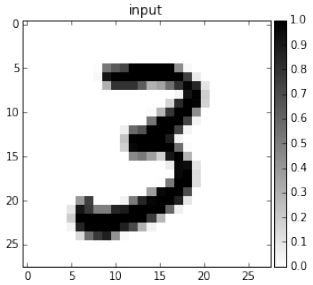


[From Taigman et al., 2014]

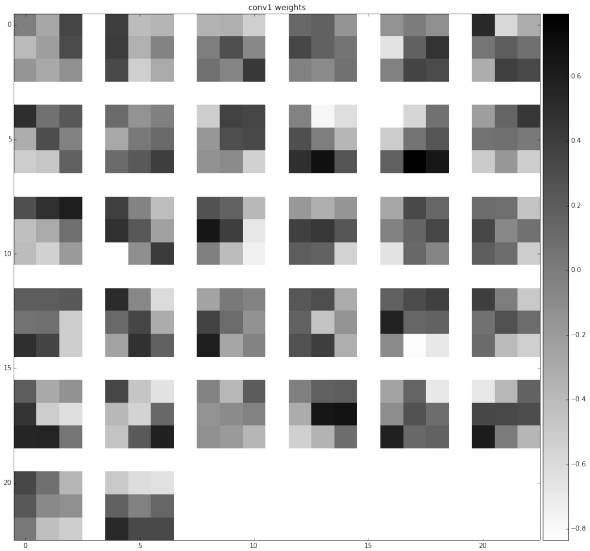
Visualizing filters and activations (primary understanding of NNs)

Mnist (toy) dataset

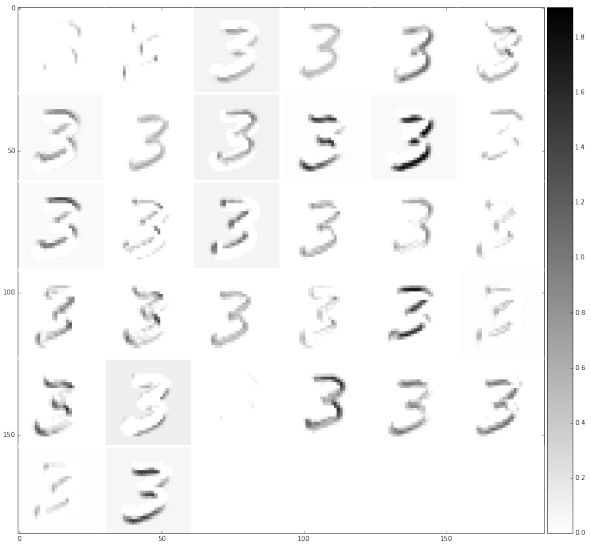
- Low resolution handwritten digit images



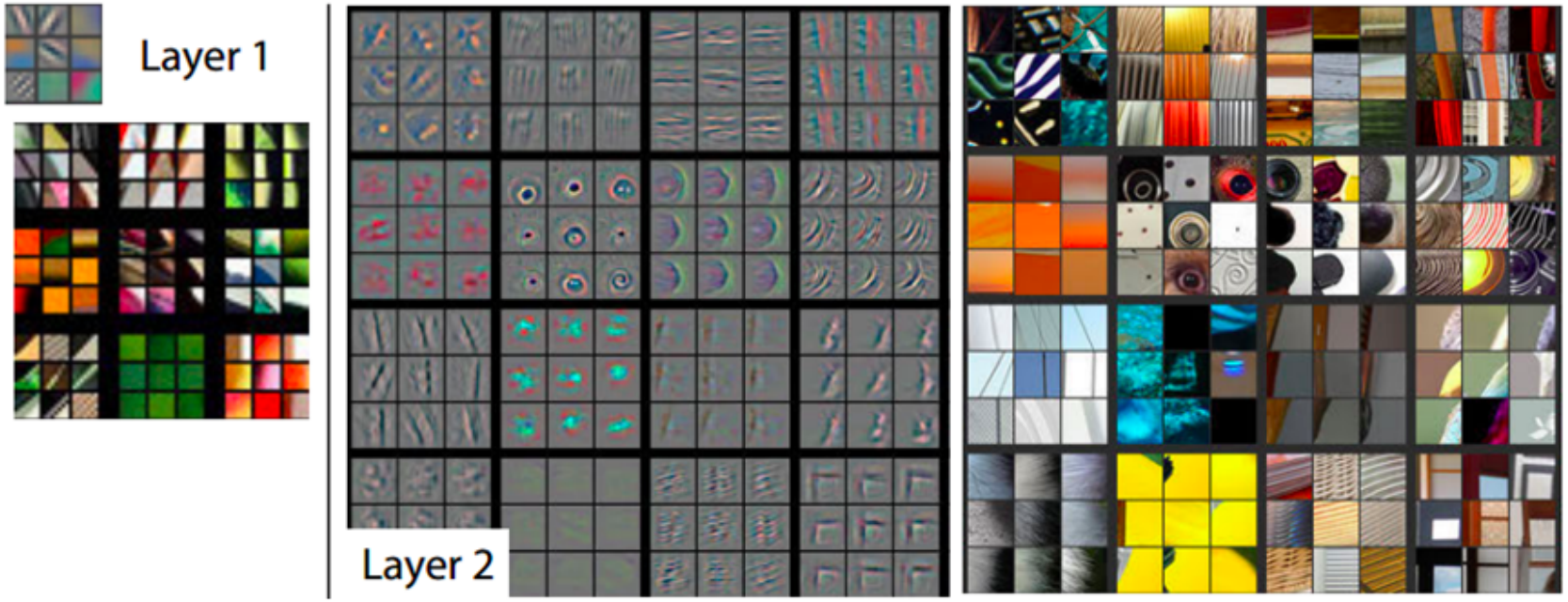
Weights of first Convolutional layer (32 maps)



Outputs of first Convolutional layer for above input



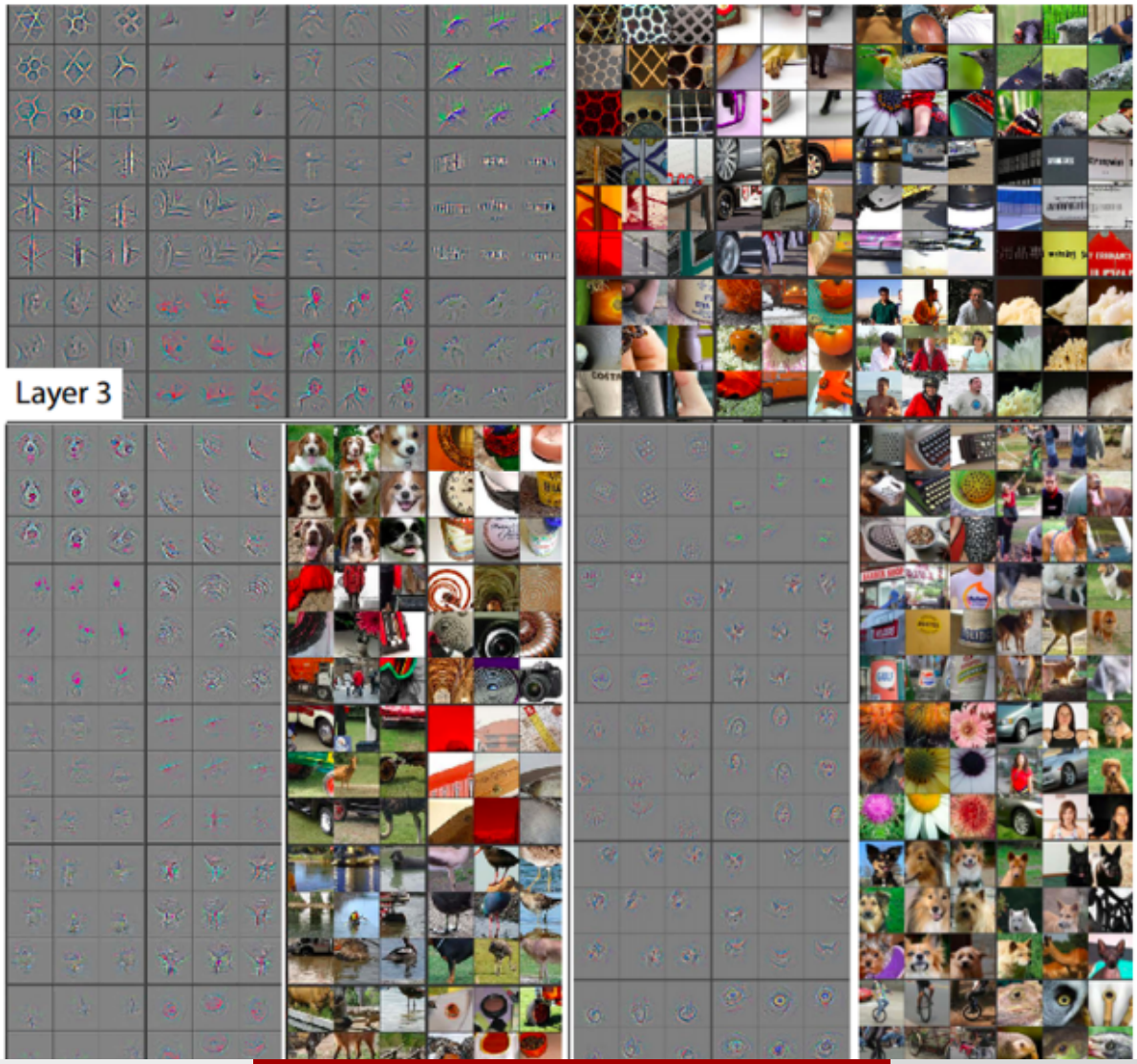
Visualizing filters and activations



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

[From Zeiler et Fergus]

Visualizing filters and activations



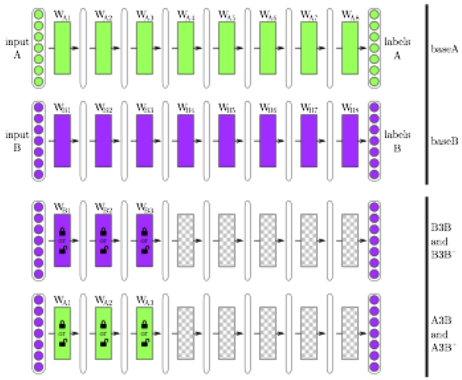
Genericity of representations [Yozinski and al., 2014]

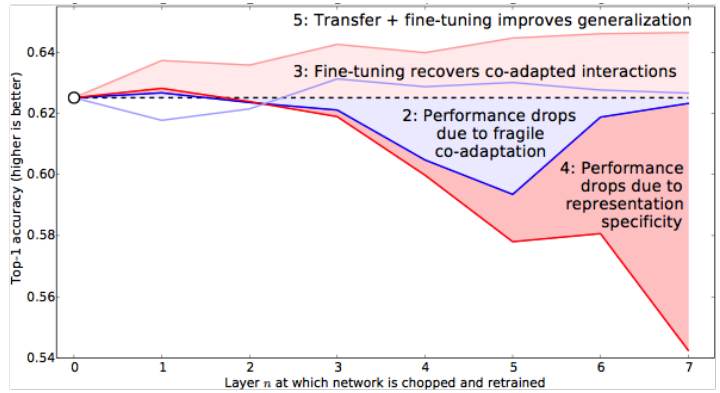
Experiments on two similar tasks

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

Main results

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

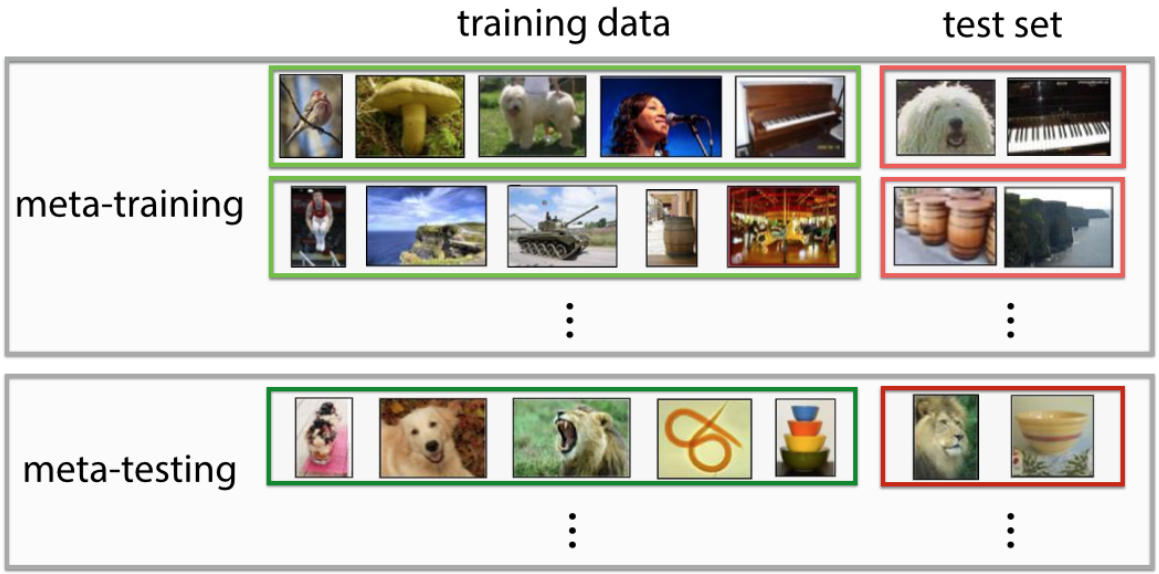




Learning with few samples

Few shot learning (and zero shot learning)

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

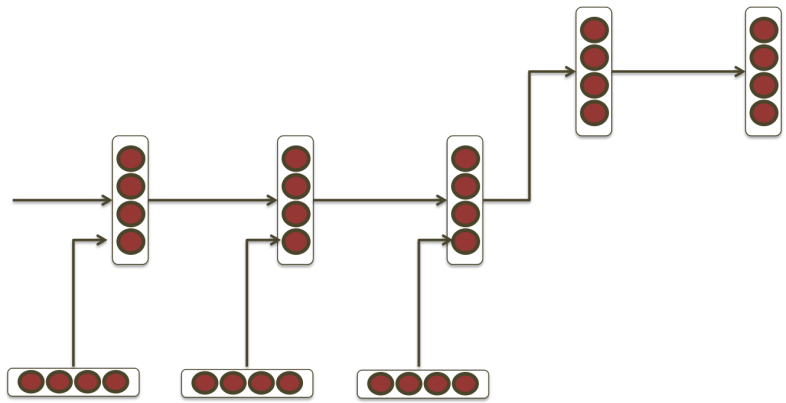


[Ravi Larochelle '17]

One goal: learning "universal" representations

Motivation : learn representations for any task

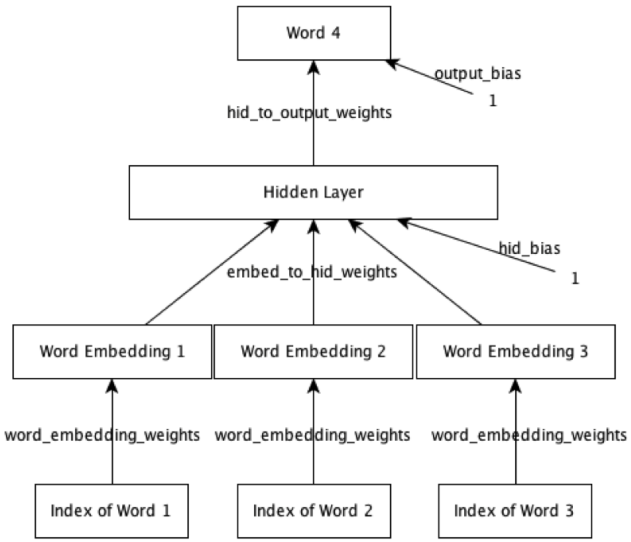
- Unsupervised or supervised
- For images, text, speech etc
  - The last layer of a CNN encodes most relevant information on the input (image)
  - The last hidden state of a RNN encodes most relevant information on the processed input sequence (e.g. sentence, signal)

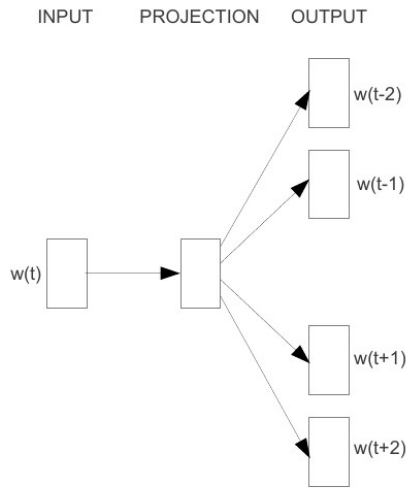


Word embeddings

Embeddings for words

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



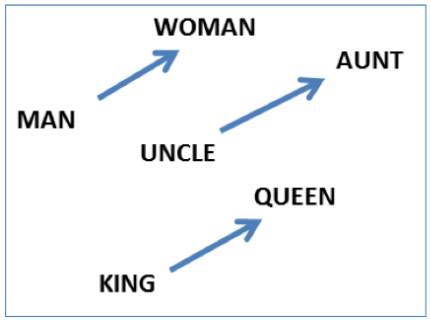


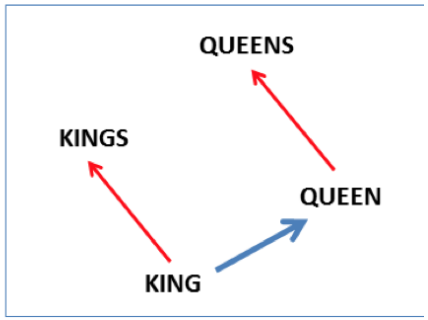
Skip-gram

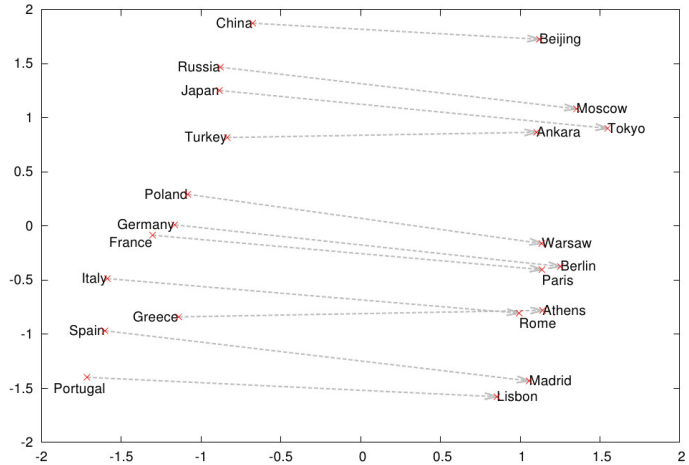
A particular interesting effect: compositionality

Idea

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

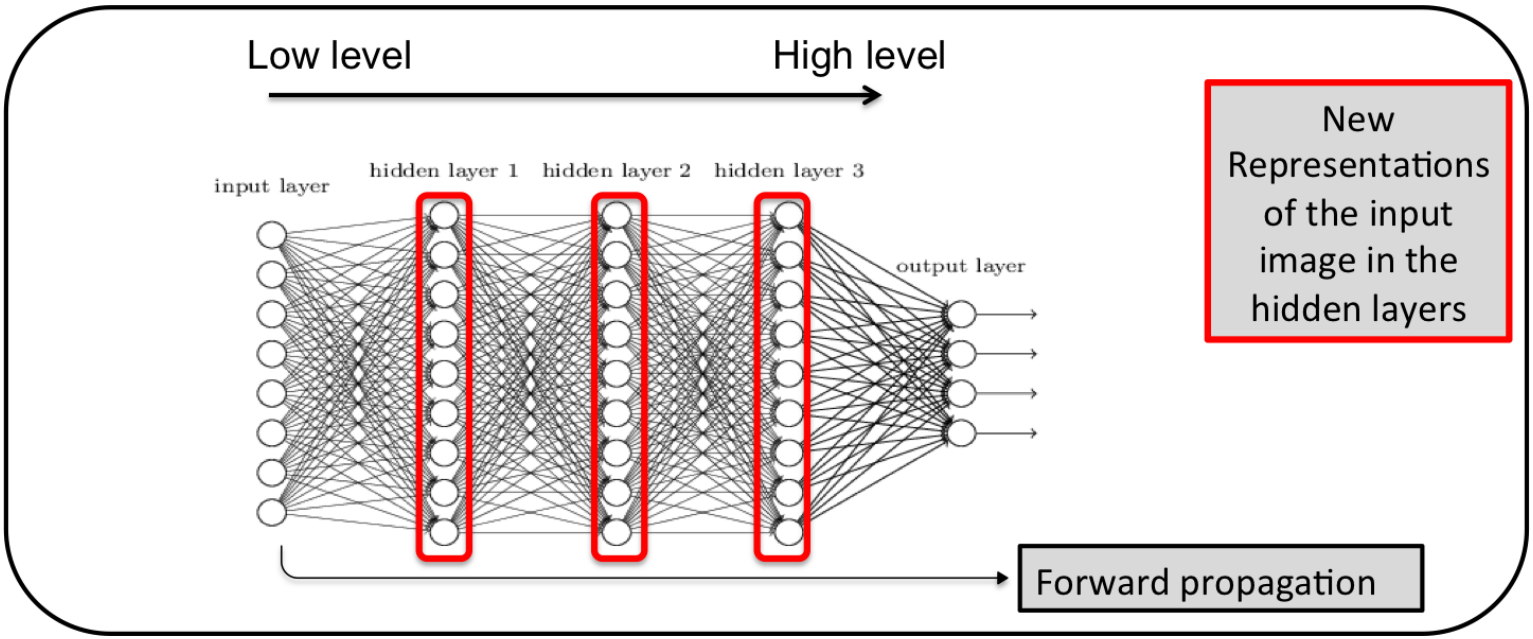




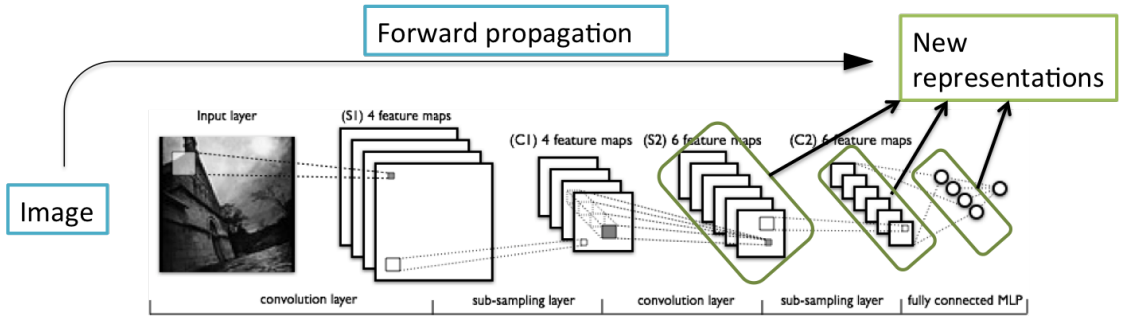


Extension of the embedding idea

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

Outline

Depth in RNNs

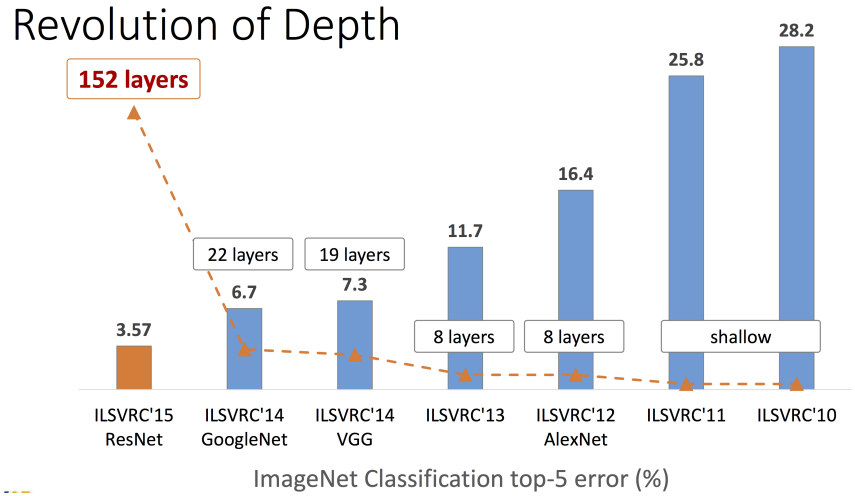
Depth in Feedforward nets

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

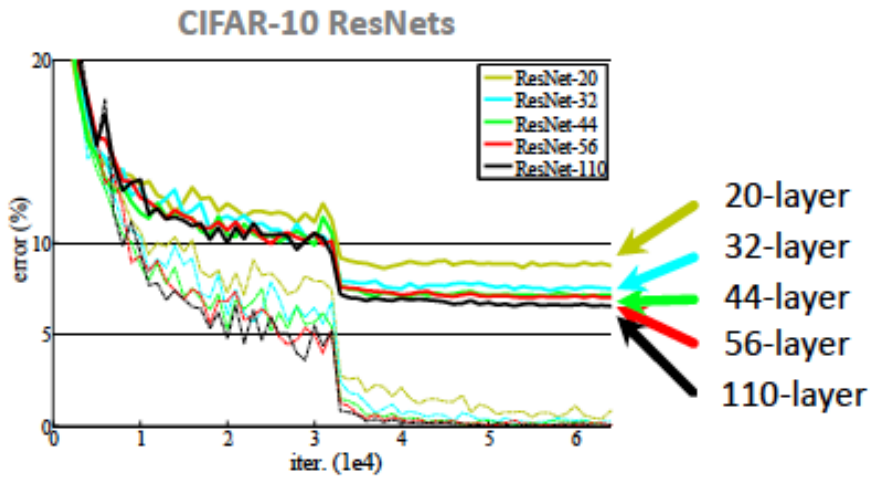
Depth in RNNs

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

Deep networks are powerful



[Kaiming He]



(Dense) Deep vs Shalow: Increased capacity

The power of depth [Eldan and Shamir, 2016]

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

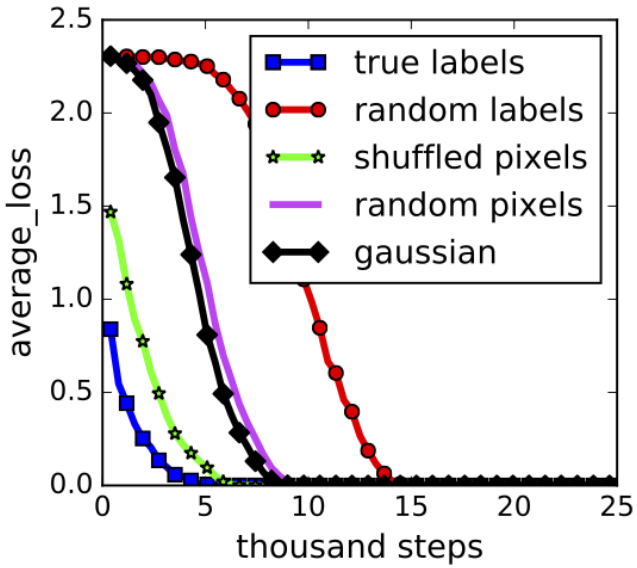
Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

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

DNNs are overparameterized

Large DNNs may even learn noise

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

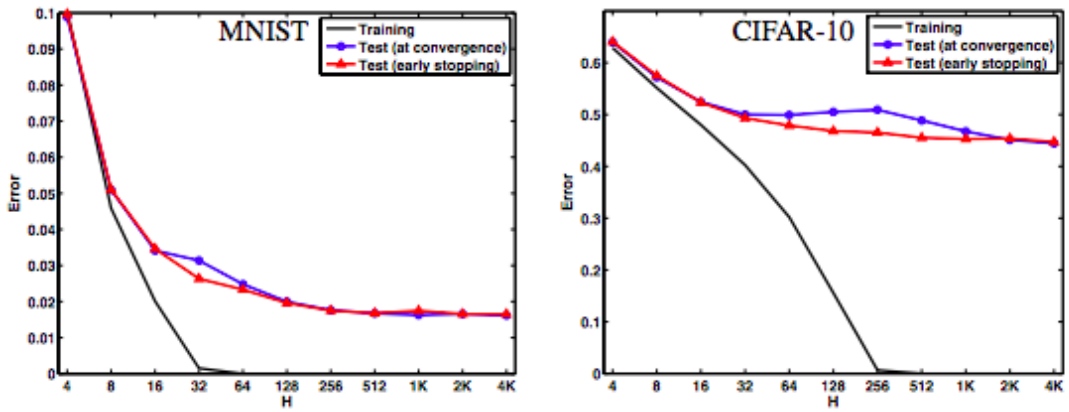


[Zhang and al., 2017]

DNNs and overfitting

Actually NNs do not easily overfit

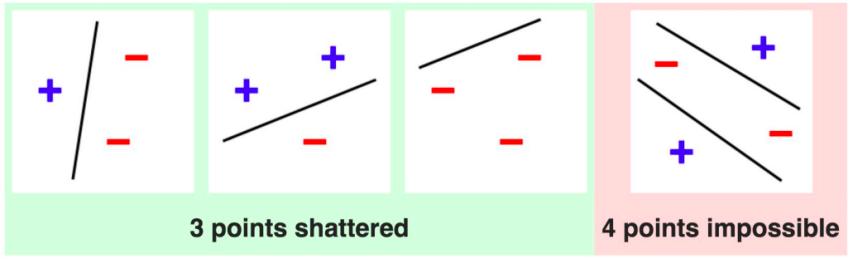
- The more you learn the better it generalizes
- Experiments on Mnist and CIFAR data (downsampled): 1 hidden layer (size H) NNs without any regularization $\rightarrow$ no overfitting observed



[Neyshabur 2017]

Capacity

Vapnik dimension



Rademacher capacity

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

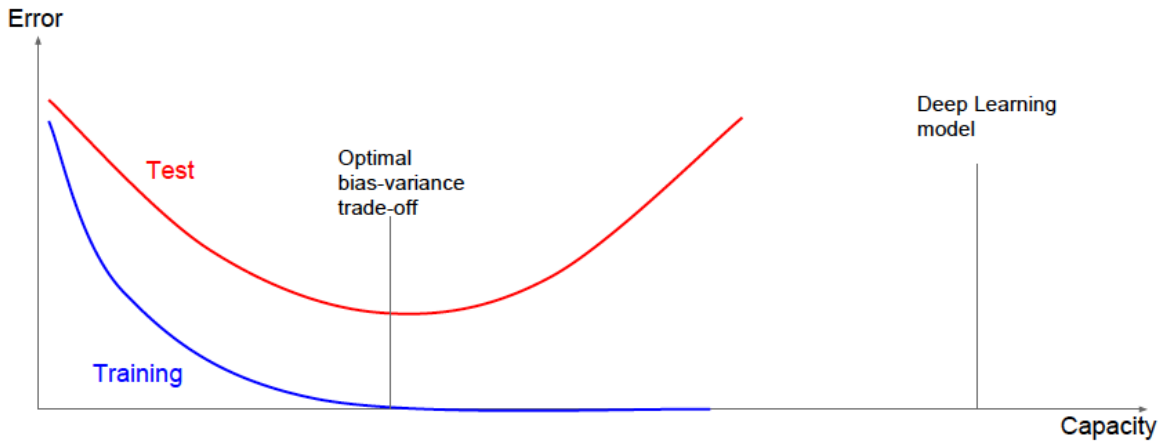
with $\sigma_i \in \{-1, 1\}$

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

DNNs' capacity

Vapnik dimension of deep NNs with ReLU

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

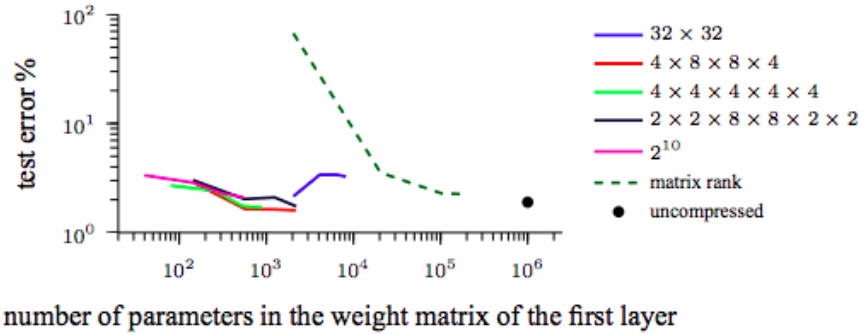


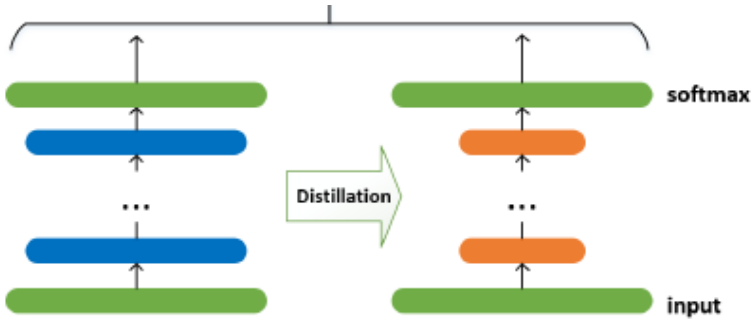
[O. Bousquet, tutorial 2017]

Deep nets do not actually need to be huge

Size helps learning but one may simplify once learned !

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

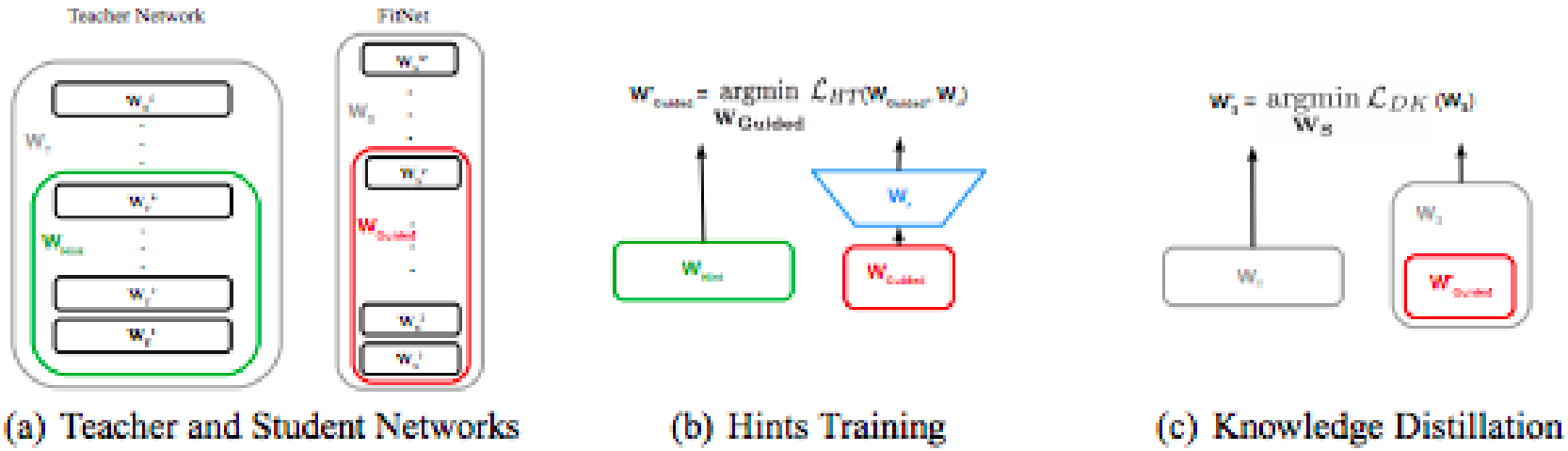




FitNets [Romero et al., 2015]

Going further in distillation with intermediate transfer

- Knowledge distillation + intermediate distillation losses



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

DL vs standard ML

Traditional Machine Learning

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

DL: mysterious phenomenon

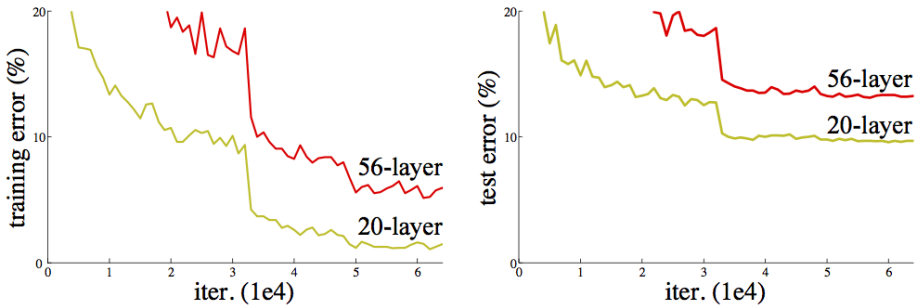
- Huge capacity without overfitting
- The size helps learning
- Overfitting idea should be revised for DNNs [Zhand and al., 2017] ?
- Regularization may slightly improve performance but is not THE answer for improving generalization
- Not clear what in the DNN may allow to predict its generalization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

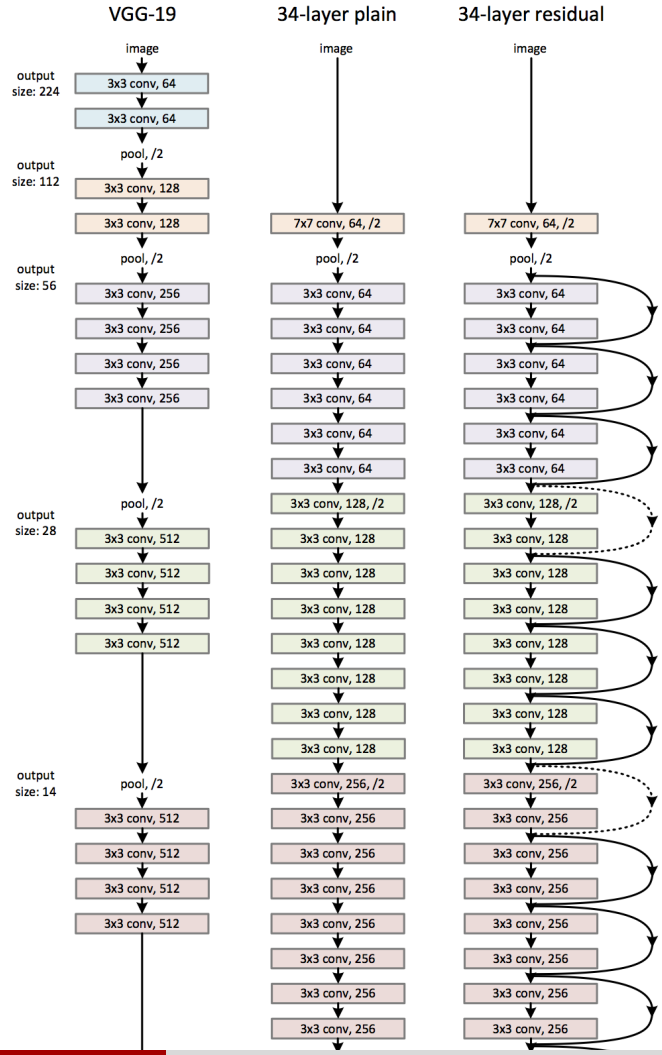
Outline

From shalow to deep

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

...





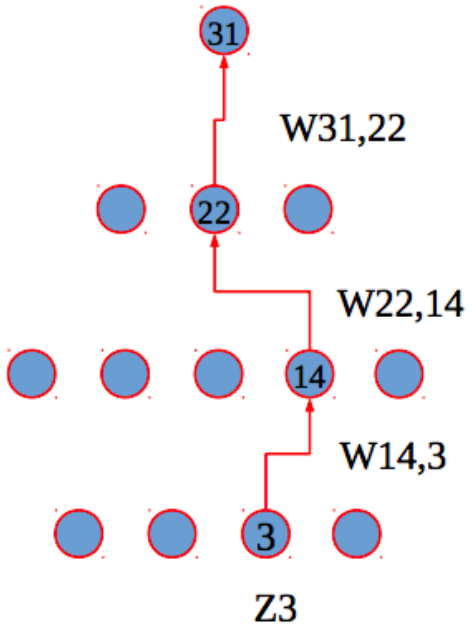
While NN optimization is reasonably easy

(Easier) analysis for ReLU DNNs [LeCun Statlearn tutorial]

- ... but expectation of generalized results to other activation functions
- With ReLu and MaxPooling operators one may formalize what happens on a path from an input to the output
- The output may be computed as :

$$\hat{y} = \sum_{P} \delta_P(W, X)(\prod_{(ij)\in P} w_{ij}) x_{j_{start}}$$

- $\delta_P(W, X)$: 1 if active path, 0 otherwise
- Implemented function is piece-wise linear



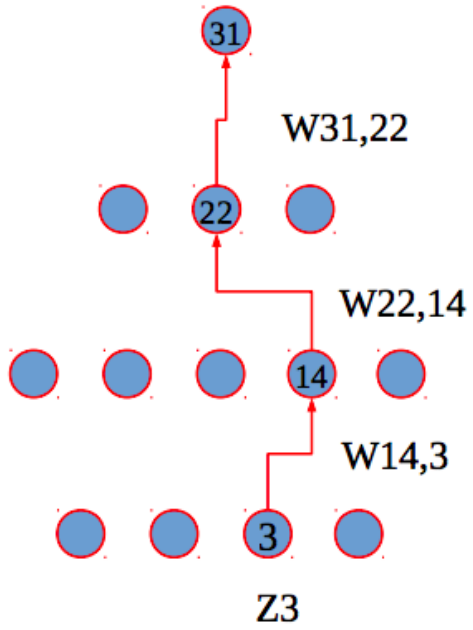
While NN optimization is reasonably easy

(Easier) analysis for ReLU DNNs [LeCun Statlearn tutorial]

- Objective function : piece wise polynomial (degree = number of hidden layers) with partially random coefficients

$$C(W) = \sum_P C_P(X, Y, W)\left(\prod_{(ij)\in P} w_{ij}\right)$$

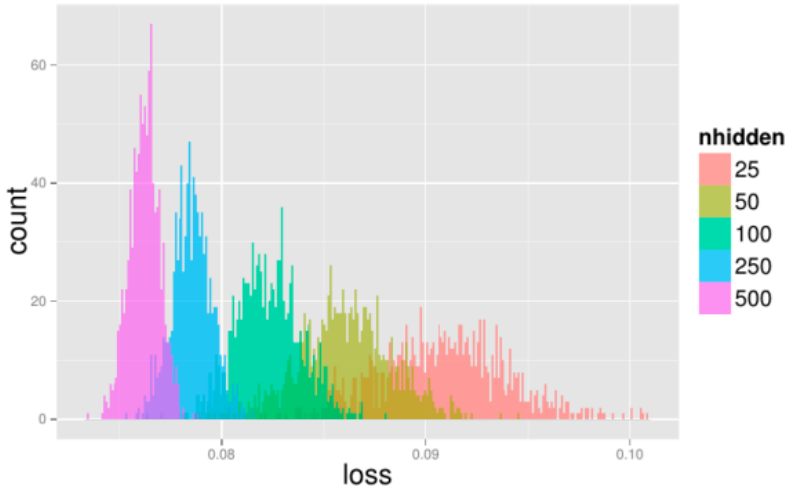
- Hint from results on distribution of critical points for polynomials with random coefficients



Deep ReLu networks

Easier analysis... [LeCun Statlearn tutorial]

- Experiments by [Choromanska and al., 2015]: Train 2-layers nets on Mnist from multiple initializations and measure loss on the test set
- Many close local minimas for large nets
- Objective function do not exhibit lots of saddle points and most local minima are good and close to globale minimas



[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]

Yet the depth alone is not enough

While SGD works well for NNs, the optimization of DNNs requires careful design and tricks

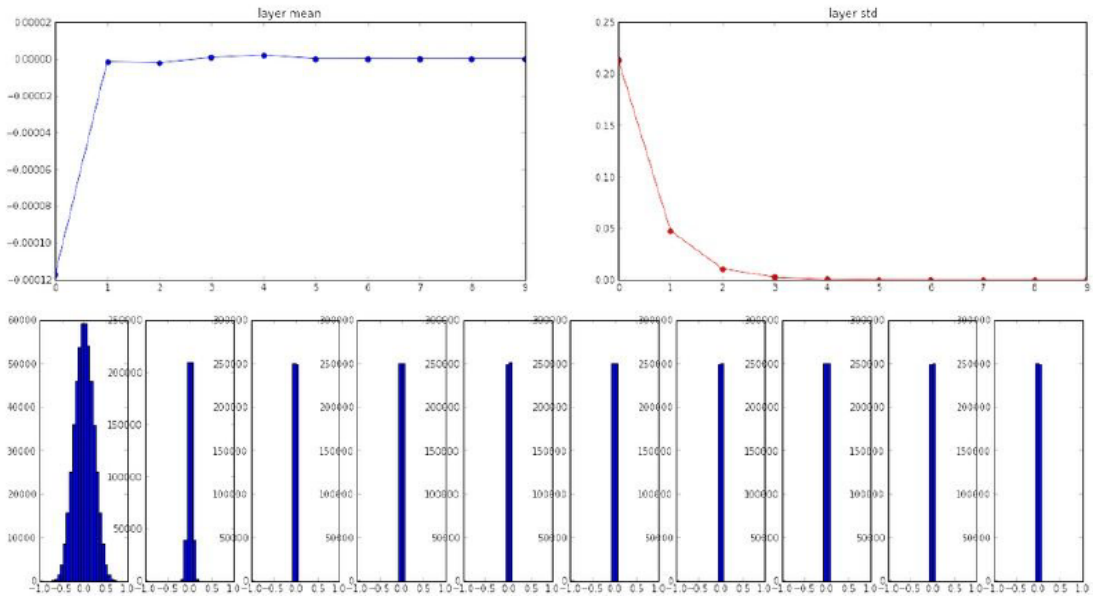
- Make the gradient flow with activation normalization (Batch Normalization)
- Make the gradient flow with structural constraints (Identity mapping)
- Regularization (Dropout)

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

Few slides from *Fei Fei Li*

Standard initialization schema for MLPs

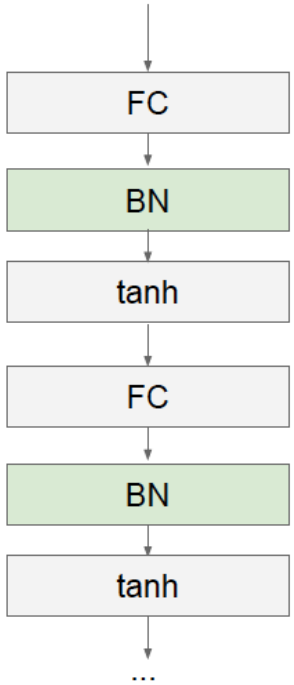
- 10 layers networks (500 neurones each, with tanh)
- Initialization : gaussian random with small (std=0.01) values (what if all null initialization?)
- All activations at 0
- What about the gradient ?



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

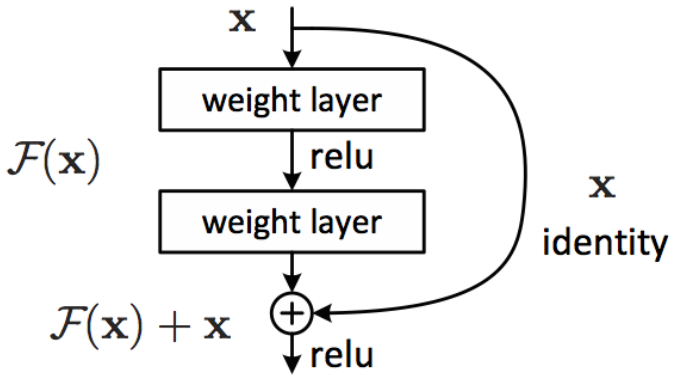


From Fei Fei Li's slides

Identity mapping in Residual Networks

Principle

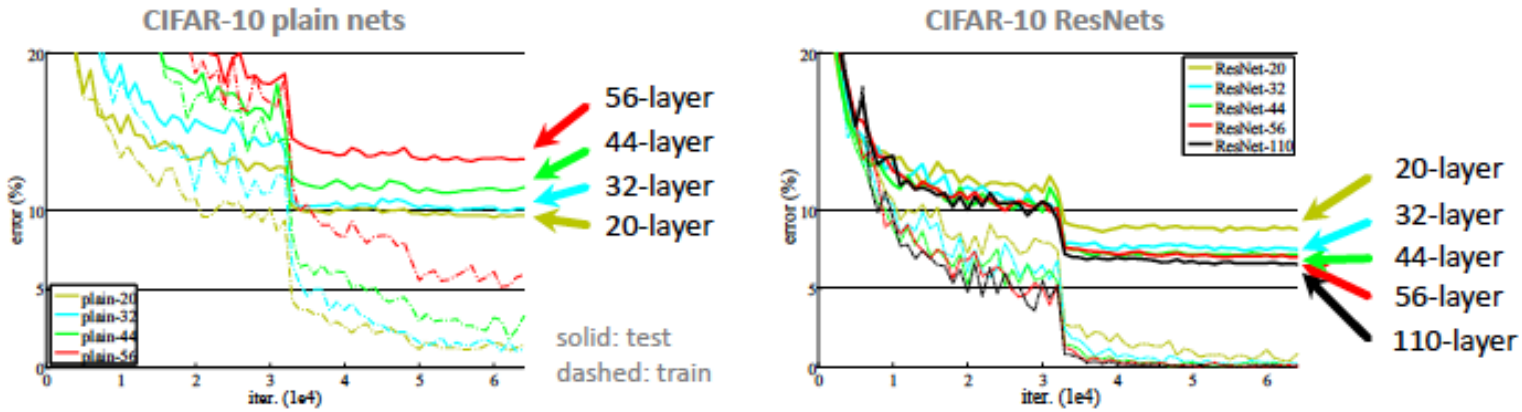
- Include identity mapping in the model
- ResNet building block [He and al., 2015]]
- Every layer becomes close to the output ($\Rightarrow$ not far in the backpropagation process)



Identity mapping in 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)



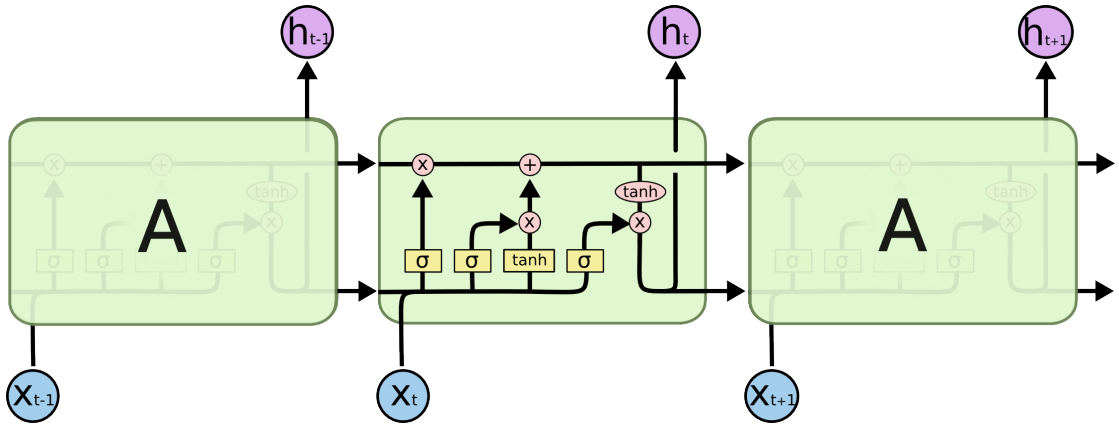
Identity mapping with LSTM units in RNNs

What it does

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

With...

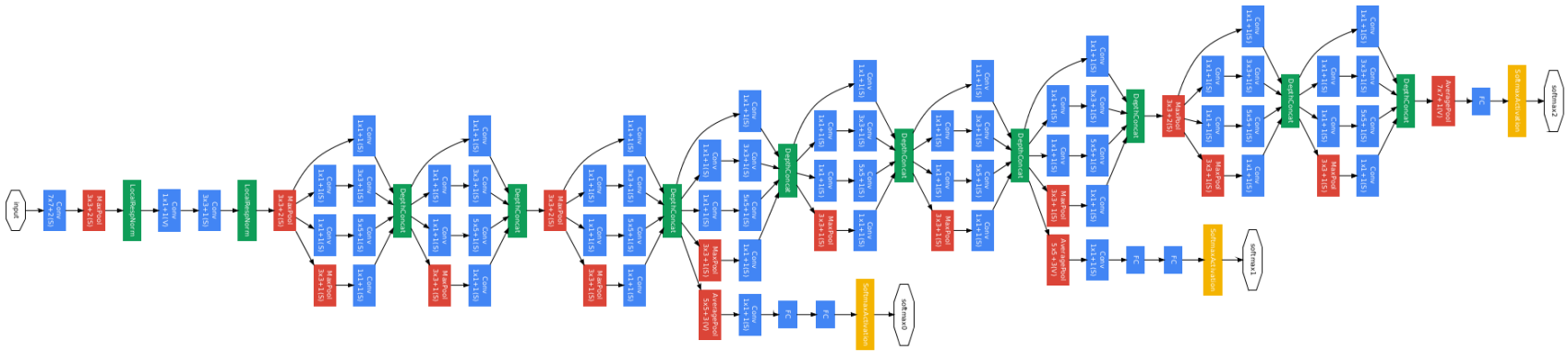
- Cell state c_t
- Forget gate f_t
- Input gate i_t
- Output o_t
- Hidden state to propagate to upper layers h_t



Auxiliary loss on intermediate layers

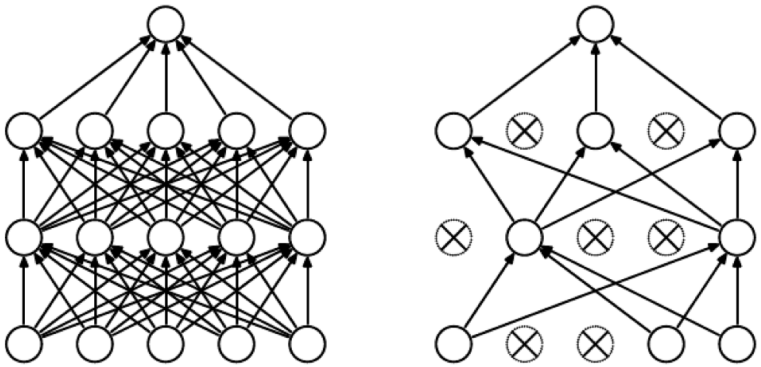
Google net

- Auxiliary loss brings some gradient to first layers



Regularizing with Dropout [Hinton 2012]

Principle

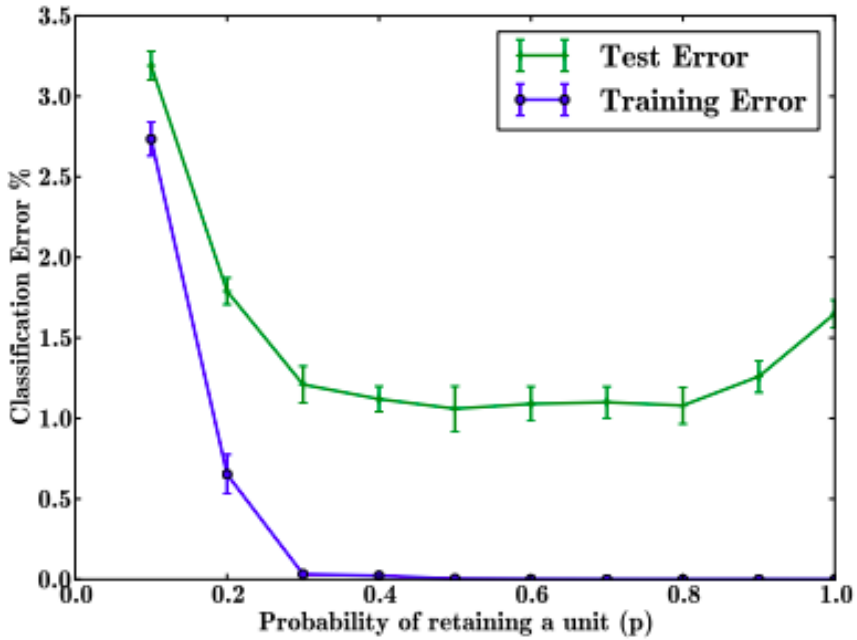


(a) Standard Neural Net (b) After applying dropout.

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

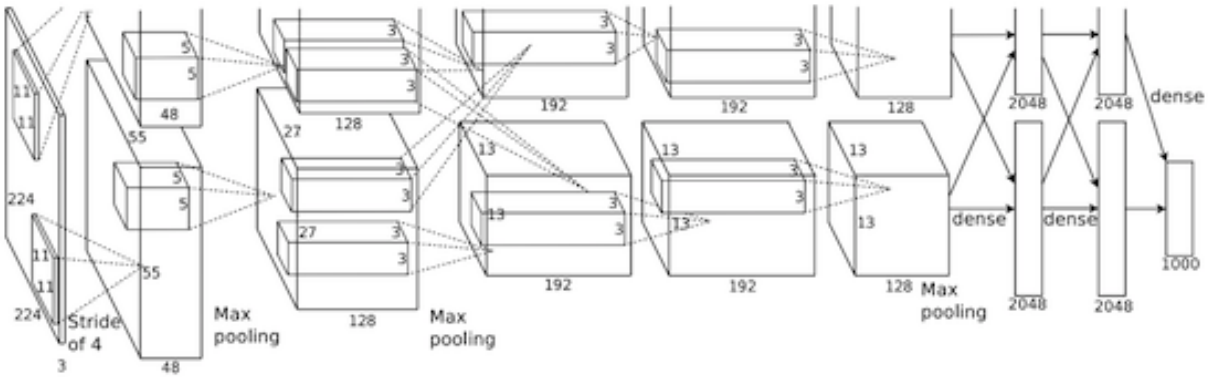
Dropout [Hinton 2012]

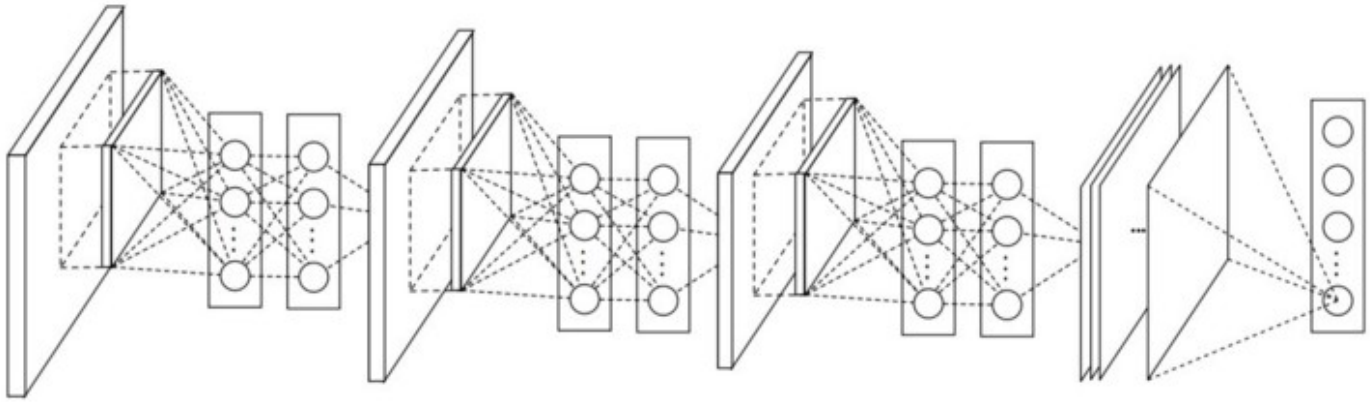
Do not ever fear overfitting !



Examples of architectures

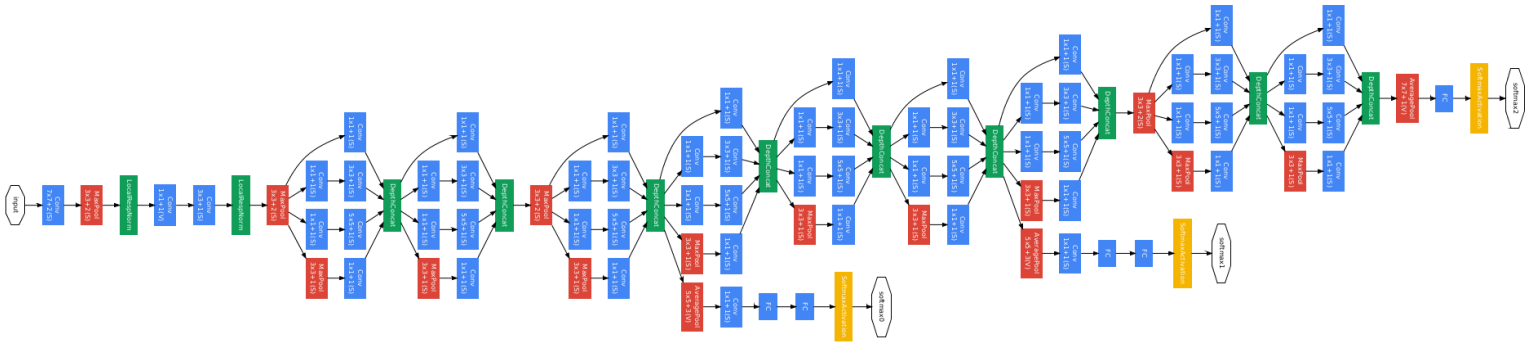
AlexNet [Krizhevsky and al., 2012] (top) and NetworkInNetwork [Lin and al.,2013] (bottom)





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, Pooling
- Dilated
- Recurrent
- Recursive
- Skip/ Residual

Criterion

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

Sizes

- Layers' sizes
- Dropout rate

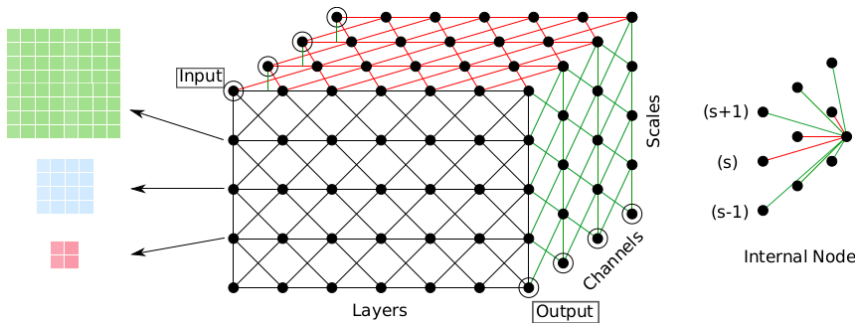
Optimization

- SGD
- Adam
- Adagrad
- Adadelta
- RMSProp
- Learning rate
- Decay
- Batch size

Looking for a good architceture

Illustration [Verbeek 2017]

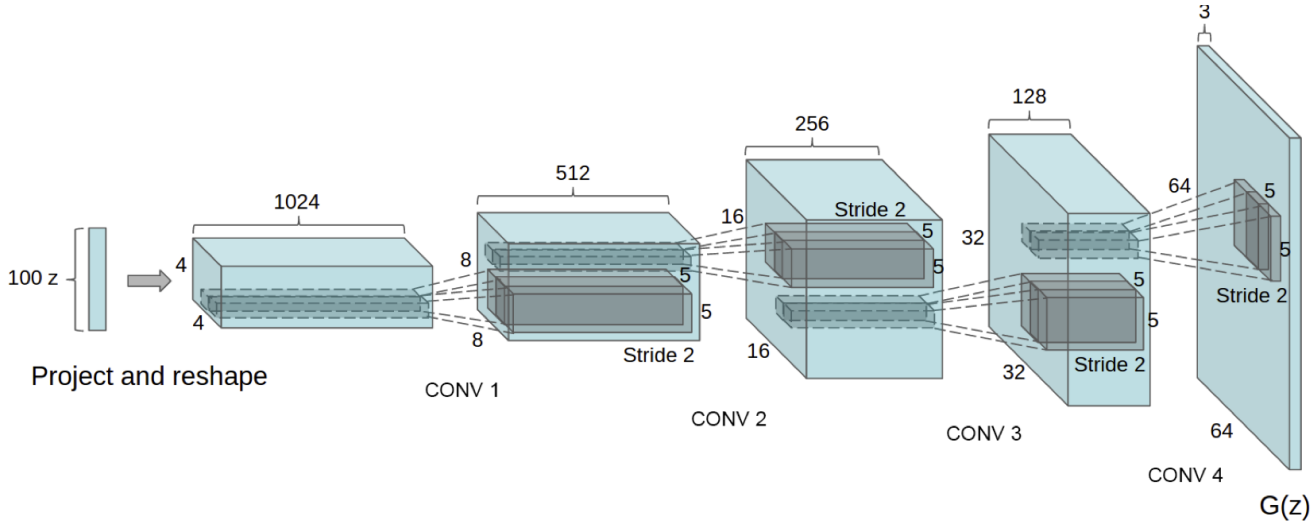
- **Simple** search (but for a large network)
  - 19 convolution layers and 5 pooling layers to set
  - Question: where to put the poooling layers? $\rightarrow$ 40 000 architectures !!
  - No question about layers' dimensions, activation function, kernels' size, pooling type etc
- Remember
  - 1 hour 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]

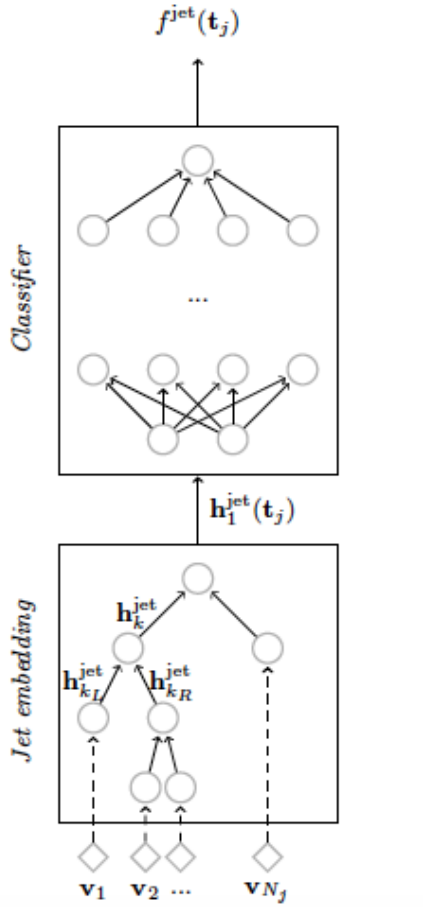
Historical attempts to scale up GANs using CNNs to model images have been unsuccessful. This motivated the authors of LAPGAN (Denton et al., 2015) to develop an alternative approach to iteratively upscale low resolution generated images which can be modeled more reliably. We also encountered difficulties 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.



Architecture design from prior knowledge (HEP example)

What if unstructured data: is DNN useful ?

- Modeling collision at CERN : features = physical features of jets during the collision (speed, energy, angle with collision axis...)
- Main approaches
  - Use non deep machine learning models
  - Represent data as images and use Deep NNs
  - Design DNN architecture from knowledge on the considered process
- Example : Learn to aggregate features of jets using a tree structure inspired from data knowledge [Louppe et al., 2018]



Architecture design from prior knowledge (HEP example)

Ziyu Guo's thesis (with Y. Coadou at CPPM)

- Deep learning in the search for ttH with the ATLAS experiment at the Large Hadron Collider
- Rely on the physical process to design the NN structure
- Better results than DNN, on par with state of the art models in HEP (BDTs)