Deep Learning

Thierry Artières

Ecole Centrale Marseille - Option DIGITALE

November 25, 2018

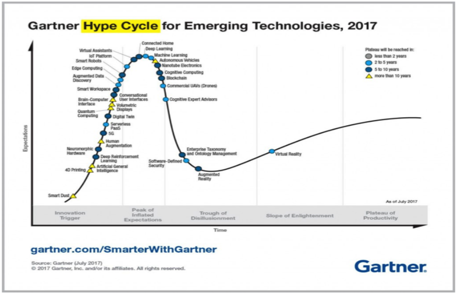
Outline

1. Introduction
2. MLPs
3. Programming

Where is AI?

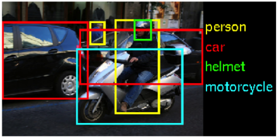
Constat

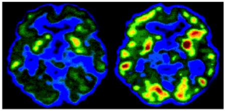
- The original AI was the General AI (IA forte)
- Today 60 years after the Dartmouth meeting
  - We have achieved some NIA results (IA faible)
  - We can start thinking more seriously about GAI
  - These are just the premises.



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)

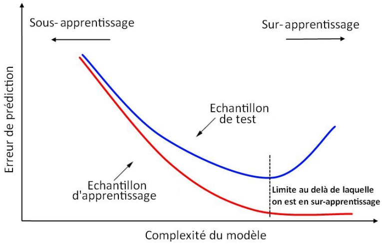
What is it?

- 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



History of Neural Networks

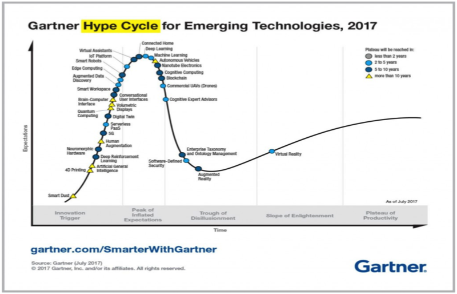
Key dates

- 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?

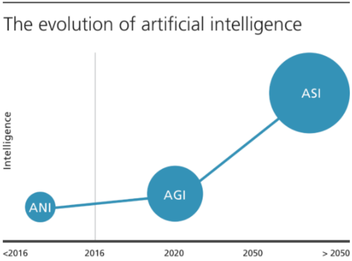
- The original AI (Dartmouth workshop) was General AI (IA forte)
- Today 60 years after Dartmouth
  - We have achieved some NIA results (IA faible)
  - We can start thinking more seriously about GAI
  - These are just the premises.



AI today

Where are we?

- The original AI (Dartmouth workshop) was General AI (IA forte)
- Today 60 years after Dartmouth
    - We have achieved some 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

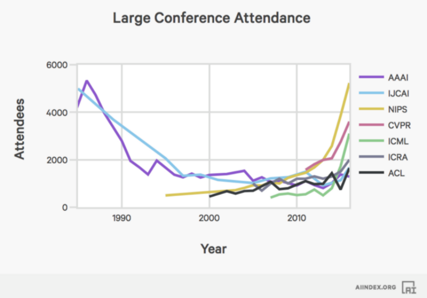
- Images, Videos, Audio, Speech, Texts
- Successful setting
  - Structured data (temporal, spatial...)
  - Huge volumes of datas
  - Huge models (millions of parameters)
  - Huge storage and computing resources (GPU, TPU)

	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

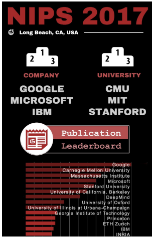


Machine Learning and Deep Learning today

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 !!

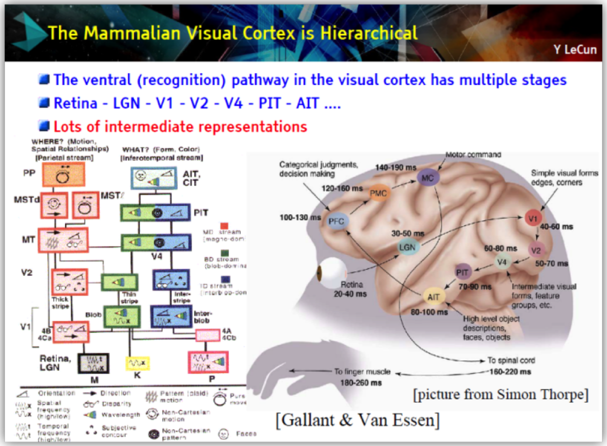
Example of an emerging topic: Generative Adversarial Networks

- 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)

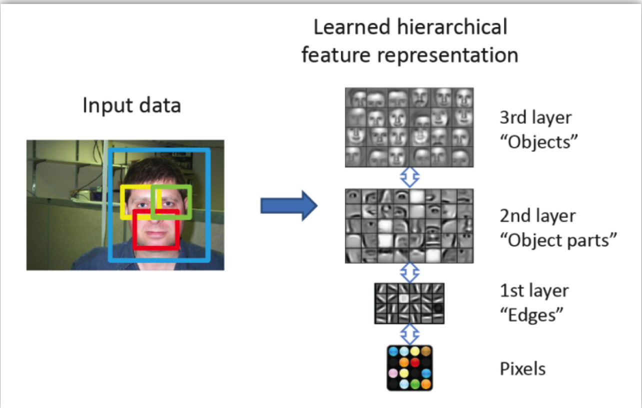
The Graal ? (but we are not there yet)



[Gallant et al., ...]

The key: features

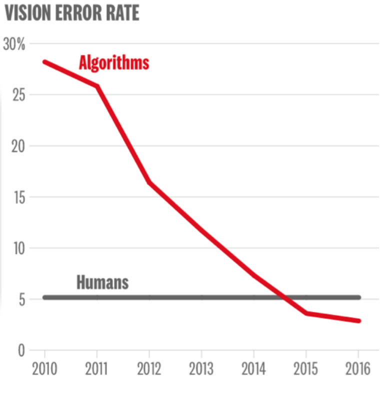
Deep learning = Representation Learning



Spectaculary breakthrough

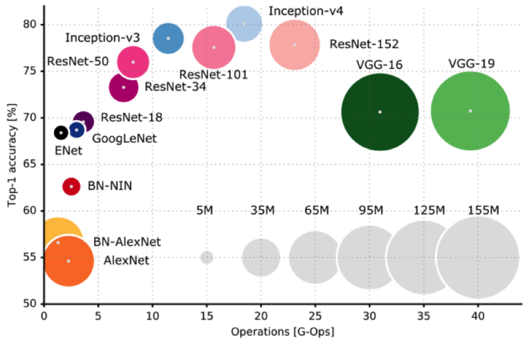
Computer vision

Real time Object recognition



Spectaculary breakthrough

Computer Vision



Spectaculary breakthrough

Speech

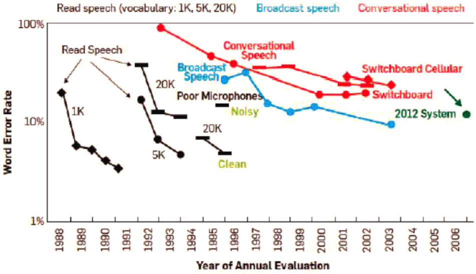
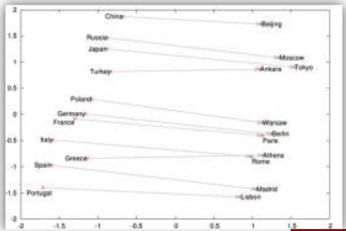


FIGURE 2.7 Historical progress on reducing the word error rate in speech recognition systems for different kinds of speech recognition tasks. Recent competency for the "difficult switchboard" task (human conversation in the wild) is marked with the green dot. SOURCE: X. Huang, J. Baker, and R. Reddy,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

Spectacular breakthrough

Games

- BackGammon, Chess, Go

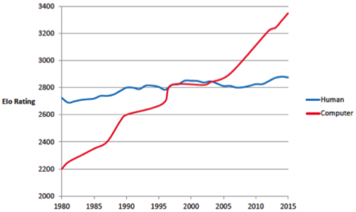


FIGURE 2.3 Elo scores—a measure of competency in competitive games—showing the chess-playing competency of humans and machines, measured over time. SOURCE: Courtesy of Murray Campbell.

Spectacular breakthrough

Image generation

Recent Nvidia results

Spectaculary breakthrough

Should we still trust what we see?

Outline

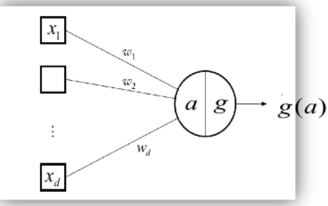
A single Neuron

One Neuron

- Elementary computation

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

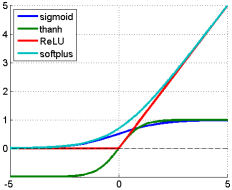
$$\text{output} = g(a(x))$$



Non linearity : g

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

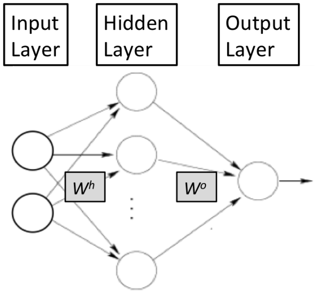
$$f(x) = 0 \text{ if } x \leq 0$$
$$= x \text{ otherwise}$$



Multi Layer Perceptron (MLP)

Structure

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



Function implemented by a MLP

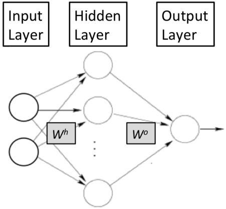
$$g(W^o.g(W^h x))$$

- 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

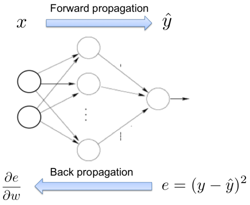
- Objective function of parameters set w for a given training set T

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

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

Backpropagation

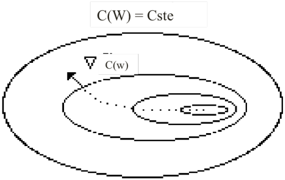
- Use chain rule for computing derivative of the loss with respect to all weights in the NN



Gradient Descent Optimization

Gradient Descent Optimization

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



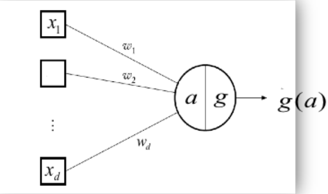
A single ReLU Neuron

One Neuron

- Elementary computation

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

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



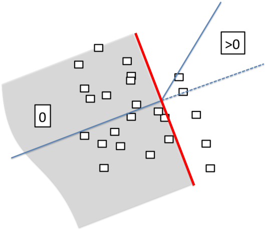
A single ReLU Neuron

One Neuron

- Elementary computation

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

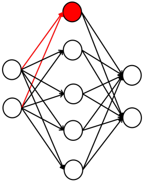
$$\text{output} = ReLU(a(x))$$

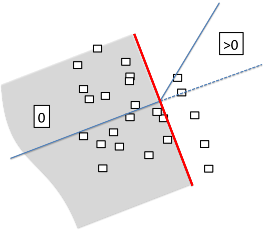


What a MLP may compute

What does a hidden neuron

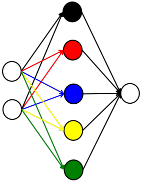
- Divides the input space in two

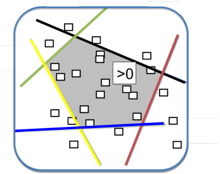




Combining multiple hidden neurons

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

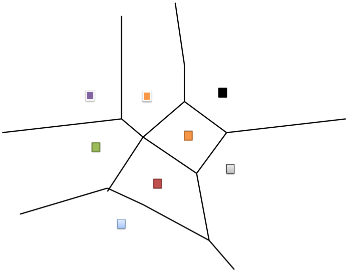


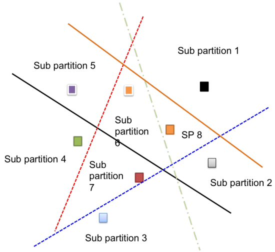


Distributed representations

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

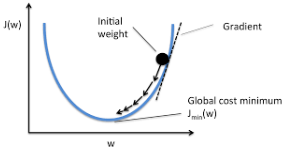
One layer is enough !

- 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

Gradient Descent Optimization

Gradient Descent Optimization

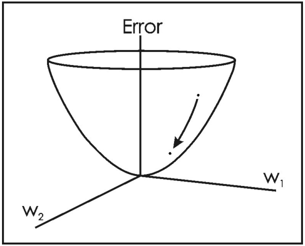
- Initialize Weights (Randomly)
- Iterate (till convergence)
  - Restimate $\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}}|_{\mathbf{w}_t}$

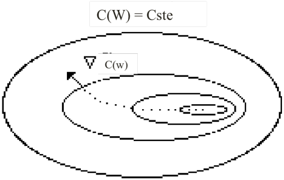


⇒ Few illustrations in these slides are taken from [LeCun et al, 1993], [Fei Fei Li lecture 6], and from *S. Ruder's blog*

Error surface

Surface error and gradient in weight space





Gradient Descent: Tuning the Learning rate

Weight trajectory for two different gradient step settings.

Two classes Classification problem

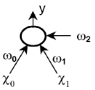
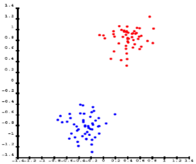
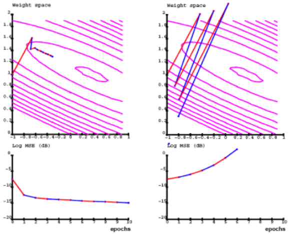


Fig. 9. Simple linear network.





(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

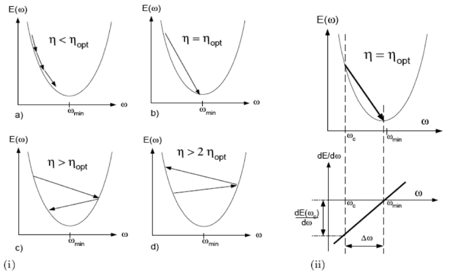


Fig. 6. Gradient descent for different learning rates.

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)

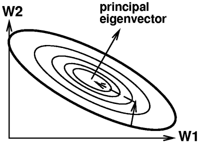
$$C(w') = C(w) + (w' - w)^T \nabla C(w) + \frac{1}{2}(w' - w)^T \nabla^2 C(w)(w' - w)$$

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

- 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. 98]

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

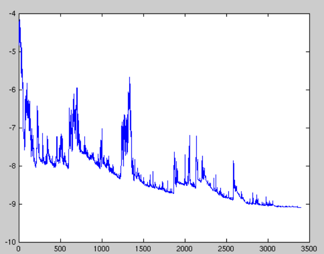
Using Momentum

SGD with Momentum

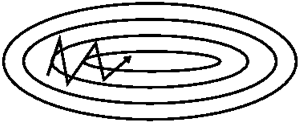
- Standard Stochastic Gradient descent : $w = w - \epsilon \frac{\partial C(w)}{\partial w}$
- SGD with Momentum:

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

$$w = w - v$$



SGD standard



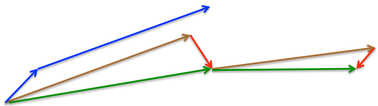
SGD avec momentum

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 $b(x, y) = g(5 \times x)$

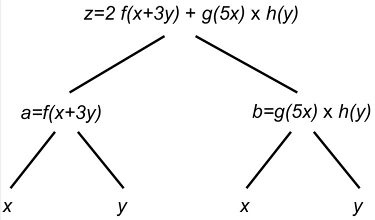
$$\Rightarrow z = 2 \times a(x) + 6 \times b(x) \times h(y)$$
$$\Rightarrow \frac{\partial z}{\partial x}|_{x,y} = \frac{\partial z}{\partial a}|_{x,y} \times \frac{\partial a}{\partial x}|_{x,y} + \frac{\partial z}{\partial b}|_{x,y} \times \frac{\partial b}{\partial x}|_{x,y}$$

With:

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

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

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



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

$$\frac{\partial C(w)}{\ldots} = \frac{\partial C(w)}{\ldots}\frac{\partial a_i^L}{\ldots} = \delta_i^L \times h^{L-1}$$

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

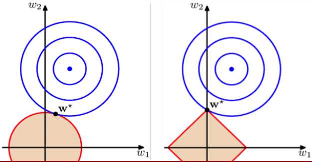
Constraints on weights

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

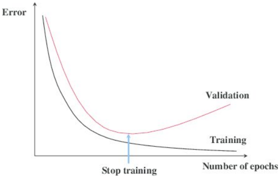
L2 and L1 norms

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



Early stopping and callbacks

Principle

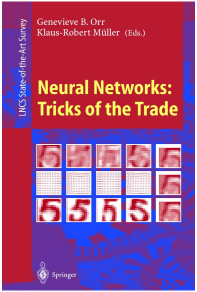


- Early stopping monitors performance (loss) on validation set
- Stopes before it reaches a plateau and starts increasing
- Related to the idea that the implemented model's capacity increases with the number of iteration
  - Think of small weights initialization and sigmoid activation
  - $\Rightarrow$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

Why now ?

Huge training resources for huge models

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

Advances in understanding optimizing NNs

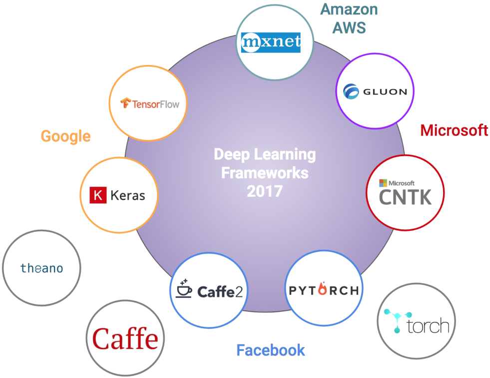
- Regularization (Dropout...)
- Making gradient flow (ResNets, LSTM, ...)

Faster diffusion than ever

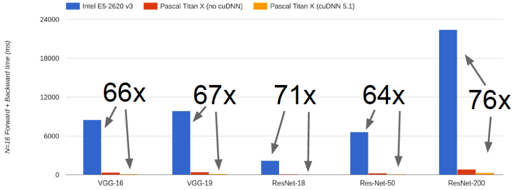
- Softwares
- Results
  - Publications (arxiv publication model) + codes
  - Architectures, weights (3 python lines for loading a state of the art computer vision model!)

Plateformes

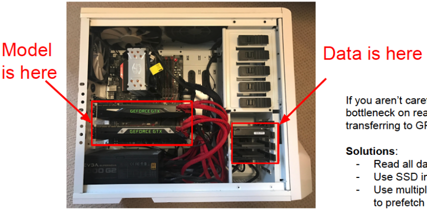
Large and active community (forums, models are available when published...) for each of these



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

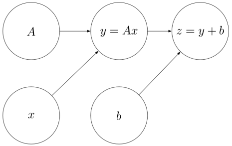
Computation graph for a calculus

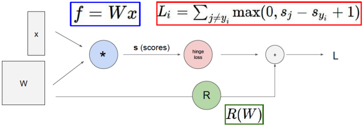
One may build a computation graph form the calculus definition

$$z = Ax + b$$

Computation graph for a calculus and a criterion

One may add a criterion (accounting for supervised learning)





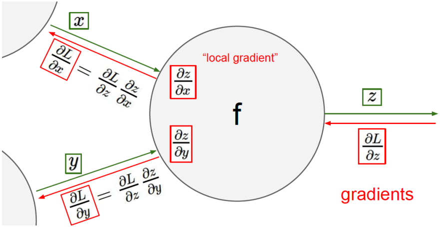
From Fei Fei Li slides

Automatic differentiation

Differentiation graph

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

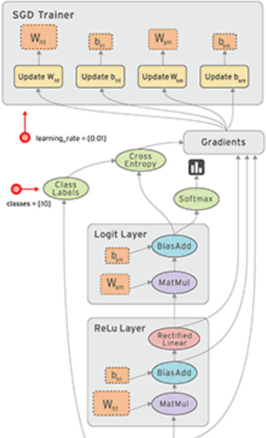
- Different rules to apply according to the operation yielding z from x and y



From Fei Fei Li slides

Computation graph and TensorFlow

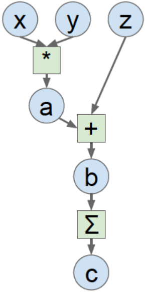
An example from [Tensorflow doc]



Computation graph and Pytorch

Another example

Computational Graphs



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()))
```