

# Introduction to Deep Learning

S. Canu

Polytech' Angers



October 10, 2019

# github.com/StephaneCanu/Deep\_learning\_lecture

https://github.com/StephaneCanu/Deep\_learning\_lecture

EDT\_Mastere PythonMatlab Traducteur Adèle Anna Anna\_fact Mastere Journal\_Rank dico\_GB\_Fr Books CNU Scholar Mood

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Deep learning lecture Edit

Manage topics

50 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

StephaneCanu	Update README.md	Latest commit 7172e37 a minute ago
jupyter_notebooks	Rename TP1_MNIST_Optim.ipynb to jupyter_notebooks/TP1_MNIST_Optim.ipynb	21 hours ago
00_Main_Deep_2018.pdf	Add files via upload	2 months ago
AFIA_AG_2018.pdf	Add files via upload	20 days ago
Intro_Deep_SJTU_2018.pdf	Add files via upload	5 minutes ago
README.md	Update README.md	a minute ago
TP_Deep_1_MNIST.py	Add files via upload	2 months ago
TP_Deep_2_webcam.py	Add files via upload	2 months ago
TP_Deep_3_fine_tuning.py	Add files via upload	2 months ago
test_cheese.zip	Add files via upload	2 months ago
train_cheese.zip	Add files via upload	2 months ago

README.md

## Deep learning: an historical perspective

### Description

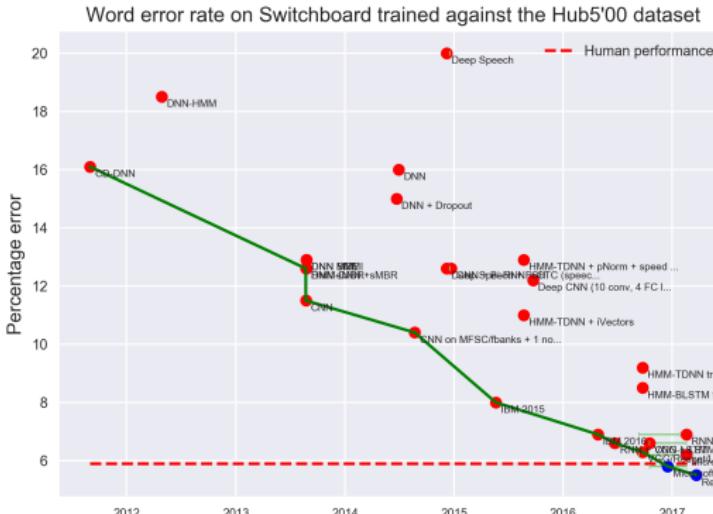
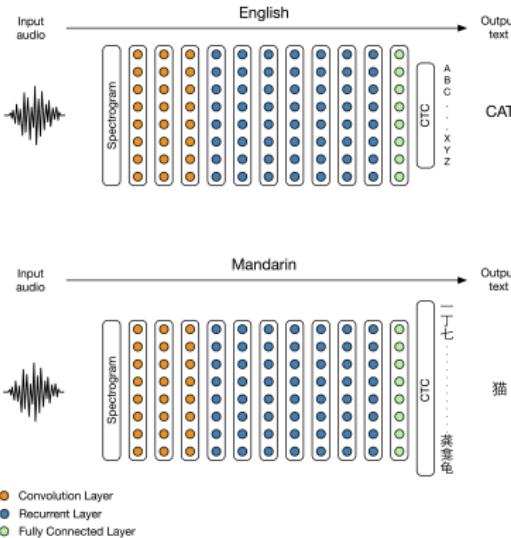
This repository contains the slides of the seminar "Deep learning: an historical perspective", given October 31 2pm-3pm at SJTU co-organized by SJTU School of Mathematical Sciences and SJTU-ParisTech Elite Institute of Technology ([Intro\\_Deep\\_SJTU\\_2018.pdf](#), 20.4 MB)

# Road map

- ① Why deep learning?
- ② The first stage: 1890 - 1969
- ③ The second stage: 1985 - 1995
- ④ The third stage: 2006 - (2012) - 2018...
- ⑤ What's new in deep learning?
  - Big is beautiful
  - Two Hot topics: data and architecture
- ⑥ Conclusion



# Deep learning for turning text into speech (and vice versa)



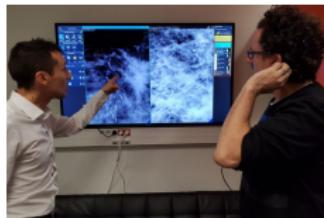
Baidu deep speech 2 (2015) and Deep voice (2017)

Trained on 9,400 hours of labeled audio with 11 million utterances.

# Deep learning for healthcare



Skin cancer classification  
130 000 training images  
validation error rate : 28 % (human 34 %)



the Digital Mammography DREAM Challenge  
640 000 mammographies (1209 participants)  
5 % less false positive

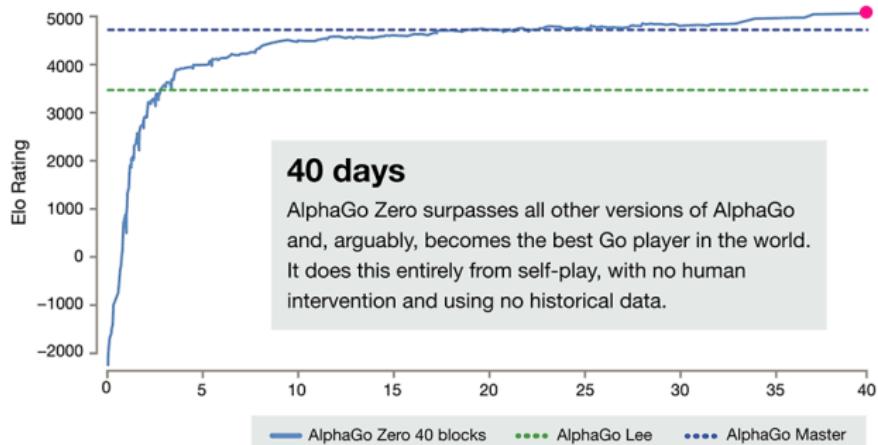


heart rate analysis  
500 000 ECG  
precision 92.6 % (humain 80.0 %) sensitivity 97 %

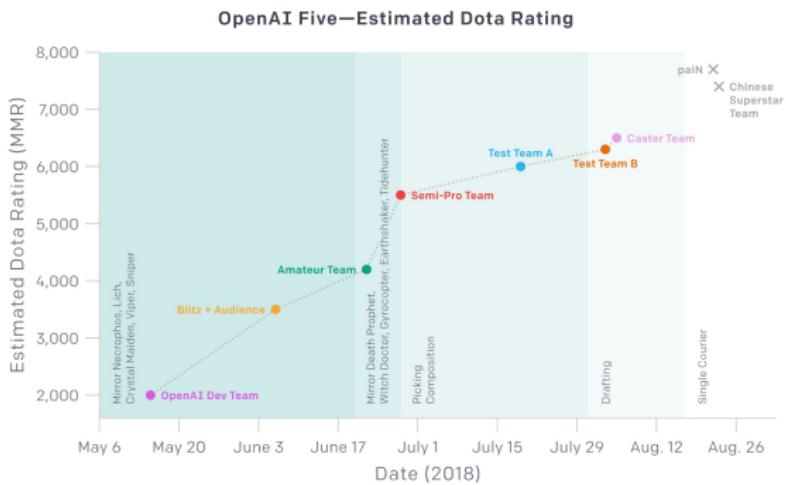
## Statistical machine learning: retrieving correlations

with deep learning end-to-end architecture  
"April showers bring May flowers"

# Deep learning success in playing GO

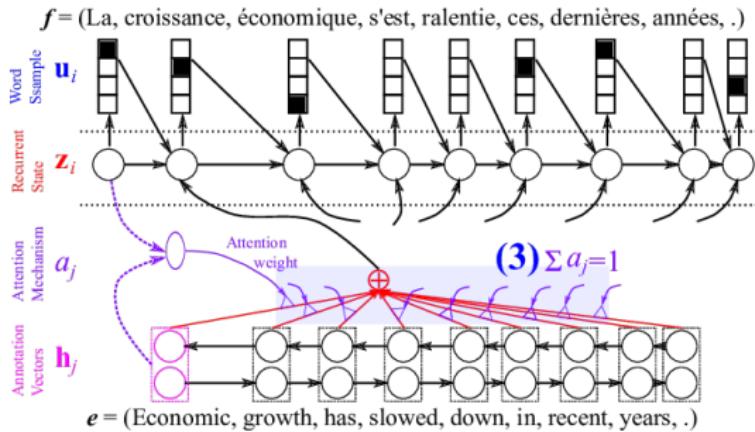


# Deep learning progresses in playing Dota 2



- separate LSTM for each hero
- 180 years/days of games against itself
- Proximal Policy Optimization
- 256 GPUs and 128,000 CPU
- the OpenAI Five is very much still a work in progress project

# Deep learning (limited) success in NLP



## Learning to translate with 36 million sentences

- Near Human-Level Performance in Grammatical Error Correction
- Achieving Human Parity on Automatic News Translation

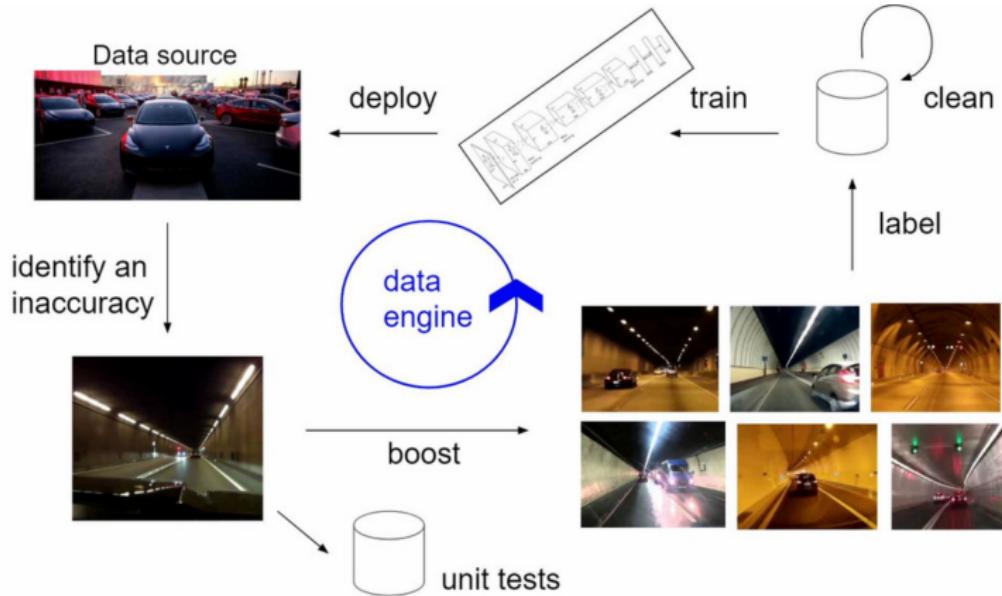
# Deep learning to drive: the Rouen autonomous lab



Driving Video Database = 100.000 videos – 120 million images

- When It Comes to Safety, Autonomous Cars Are Still "Teen Drivers"
- companies are developing different modes of automation

# Tesla improving: iterative process



- data
- multi task learning
- fleet learning
- + shadow mode for more training data

# So far, so good

- Deep learning performance breakthrough
  - ▶ Low level perception tasks: speech, image and video processing, natural language processing, games...
  - ▶ ...and specific tasks in health care, astronomy...
- It requires
  - ▶ Big data
  - ▶ Big computers
  - ▶ Specific tasks
- Yet to be solved
  - ▶ Complex games
  - ▶ Translation
  - ▶ Virtual Assistant
  - ▶ Autonomous vehicle

The image shows a digital representation of a Fortune magazine cover. At the top, there's a black bar with the word 'FORTUNE' in white capital letters next to a red three-line menu icon. To the right of the title is a 'SUBSCRIBE' button. Below this, the author's name 'By Roger Parloff' is displayed, followed by 'ILLUSTRATION BY JUSTIN METZ' and the date 'SEPTEMBER 28, 2016, 5:00 PM EDT'. The main title 'WHY DEEP LEARNING IS SUDDENLY CHANGING YOUR LIFE' is prominently displayed in large, white, sans-serif capital letters against a red rectangular background.

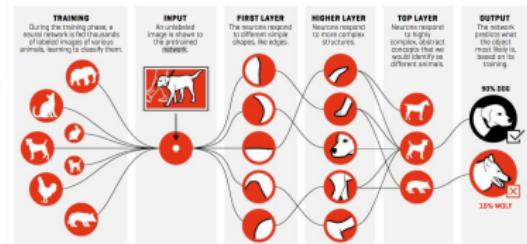
Decades-old discoveries are now electrifying the computing industry and will soon transform corporate America.

Over the past four years, readers have doubtlessly noticed quantum leaps in the quality of a wide range of everyday technologies.

Most obviously, the speech-recognition functions on our smartphones work much better than they used to. When we use a voice command to call our spouses, we reach them now,

# Road map

- 1 Why deep learning?
- 2 The first stage: 1890 - 1969
- 3 The second stage: 1985 - 1995
- 4 The third stage: 2006 - (2012) - 2018...
- 5 What's new in deep learning?
  - Big is beautiful
  - Two Hot topics: data and architecture
- 6 Conclusion



## The neural networks time line

- The first stage: 1890 - 1969

~1890 Ramón y Cajal: the biological neuron

1943 McCulloch & Pitts formal neuron

1949 Hebb's rule

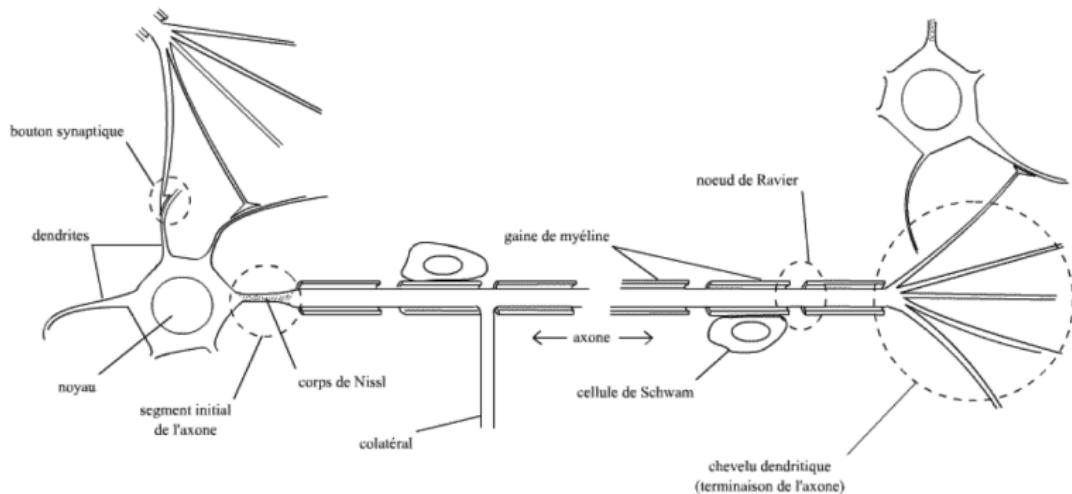
1958 Rosenblatt's Perceptron: learning with stochastic gradient

1969 Minsky & Papert: stop – the 1st NN winter

- The second stage: 1985 - 1995

- The third stage: 2006 - (2012) - 2018...

# The biological neuron



# The neural networks time line

- The first stage: 1890 - 1969

~1890 Ramón y Cajal: the biological neuron

1943 McCulloch & Pitts formal neuron

1949 Hebb's rule

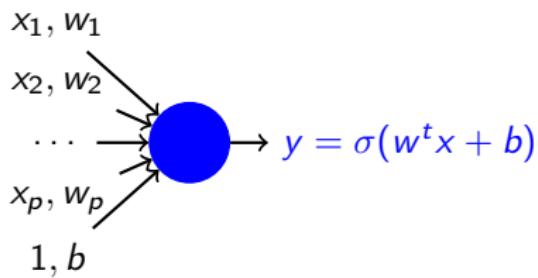
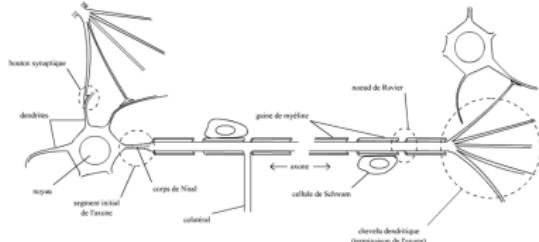
1958 Rosenblatt's Perceptron: learning with stochastic gradient

1969 Minsky & Papert: stop – the 1st NN winter

- The second stage: 1985 - 1995

- The third stage: 2006 - (2012) - 2018...

# McCulloch & Pitts formal neuron 1943



$x$  input  $\in \mathbb{R}^p$

$w$  weight,  $b$  bias

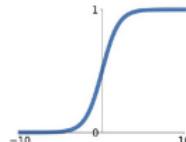
$\sigma$  activation function

$y$  output  $\in \mathbb{R}$

# Activation functions

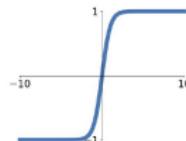
## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



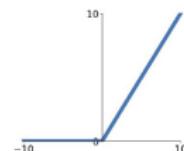
## tanh

$$\tanh(x)$$



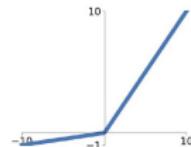
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$

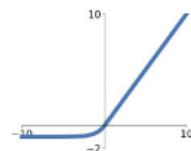


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

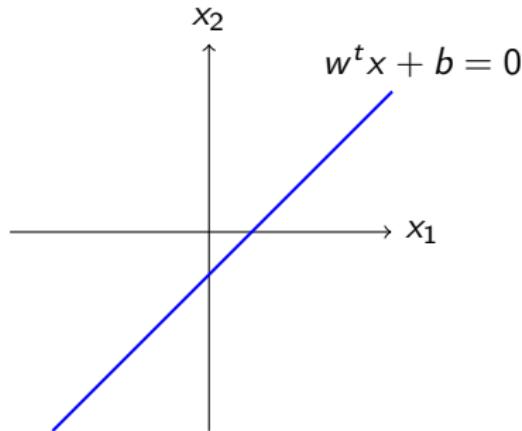
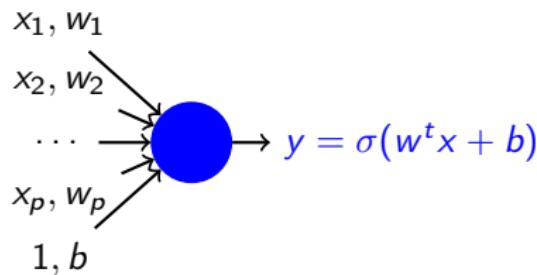


- non linear
- computationally efficient
- differentiable
- non zero

## Softmax

$$\sigma_M(x) = \frac{\exp^x}{\sum_k \exp^{x_k}}$$

# The artificial neuron as a linear threshold unit



$x$  input  $\in \mathbb{R}^p$

$\sigma$  activation function (non linear)

$w$  weight,  $b$  bias

$\mathbb{R} \mapsto \mathbb{R}$

$a$  activation,  $a = w^t x + b$

$a \rightarrow y = \sigma(a)$

$\sigma$  activation function

$\Phi$  transfer function

$\Phi$  transfer function

$y$  output  $\in \mathbb{R}$

$\mathbb{R}^p \mapsto \mathbb{R}$

$x \rightarrow y = \Phi(x) = \sigma(w^t x + b)$

# The neural networks time line

- The first stage: 1890 - 1969

~1890 Ramón y Cajal: the biological neuron

1943 McCulloch & Pitts formal neuron

1949 Hebb's rule

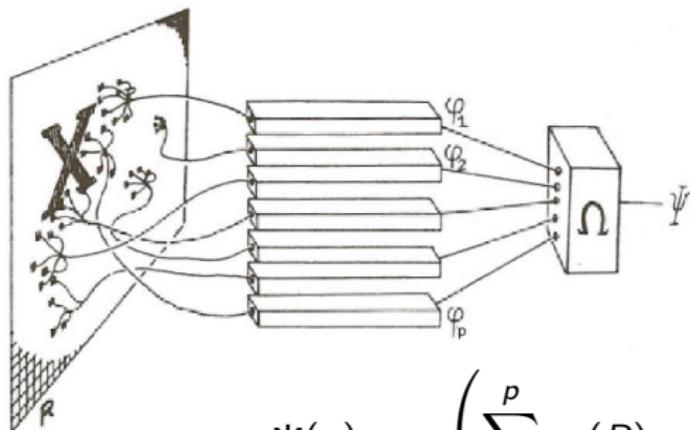
1958 Rosenblatt's Perceptron: learning with stochastic gradient

1969 Minsky & Papert: stop – the 1st NN winter

- The second stage: 1985 - 1995

- The third stage: 2006 - (2012) - 2018...

## The formal neuron as a learning machine: fit the $w$



$$\Psi(x) = \sigma \left( \sum_{j=1}^p \varphi_j(R) w_j + b \right)$$

Rosenblatt's Perceptron, 1958 (Widrow & Hoff's Adaline, 1960)

given  $n$  pairs of input-output data  $x_i = \varphi_j(R_i), t_i, i = 1, n$

find  $w$  such that

$$\underbrace{\sigma(w^t x_i)}_{\text{prediction of the model}} = \underbrace{t_i}_{\text{ground truth}}$$

# Cost minimization (energy-based model)

Minimize a loss       $\min_{w \in \mathbb{R}^{p+1}} \sum_{i=1}^n \text{loss}(w) \quad \text{loss}(w) = (\sigma(w^t x_i) - t_i)^2$

Gradient descent       $w \leftarrow w - \rho d$        $d = \sum_{i=1}^n \nabla_w \text{loss}(w)$

Stochastic gradient       $d = \nabla_w \text{loss}(w)$

---

## Algorithm 1 Gradient epoch

**Data:**  $w$  initialization,  $\rho$  stepsize

**Result:**  $w$

**for**  $i=1, n$  **do**

$x_i, t_i \leftarrow$  pick a point  $i$

$d \leftarrow d + \nabla_w \text{loss}(w, x_i, t_i)$

**end**

$w \leftarrow w - \rho d$

---

## Algorithm 2 Stochastic gradient

**Data:**  $w$  initialization,  $\rho$  stepsize

**Result:**  $w$

**for**  $i=1, n$  **do**

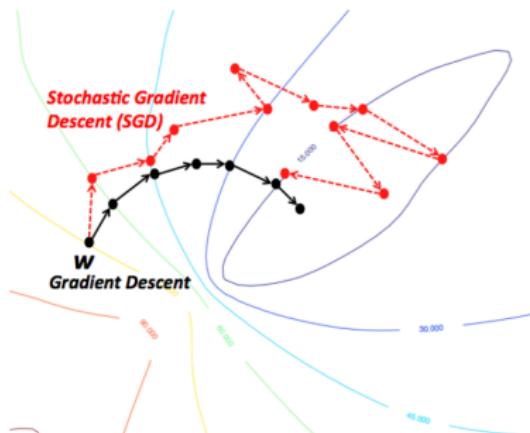
$x_i, t_i \leftarrow$  pick a point  $i$

$d \leftarrow \nabla_w \text{loss}(w, x_i, t_i)$

$w \leftarrow w - \rho d$

**end**

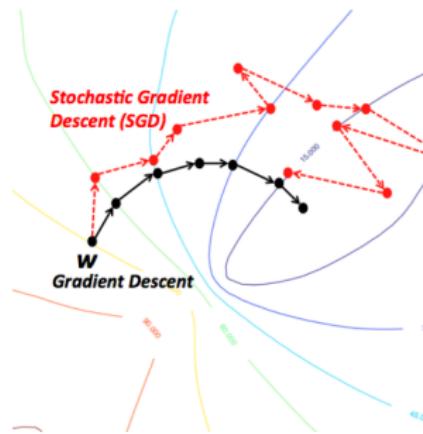
# Gradient vs. stochastic gradient



- trading precision for computational efficiency
- minimizing the generalization error (early stopping)
- slower to optimize but faster to learn
- a large scale learning perspective:  $\mathcal{O}(n)$

# Accelerating the stochastic gradient

- stochastic average (mini batch)
  - ▶ parameters (Polyak and Juditsky, 1992)
  - ▶ gradients SAG-A, (Le Roux et al 2012)
  - ▶ variance reduction (Johnson, Zhang, 2013)
- convergence acceleration
  - ▶ Nesterov's method (1983)
  - ▶ momentum (heuristic)
- acceleration and averaging
  - ▶ (Dieuleveut, Flammarion & Bach, 2016)
- stepsize adaptation
  - ▶ RMSprop (Tieleman & Hinton, 2012)
  - ▶ Adaptive Moment Estimation – ADAM (Kingma & Ba, 2015)
  - ▶ AMSGRAD (Reddi et al, BPA ICRL 2018 )



## Averaging through examples (mini batch)

For mini batch of size  $m$  (typically  $m \in \{32, 256\}$ , up to 64k!)

$$\mathbf{d} = \sum_{i=1}^{m \ll n} \nabla_{\mathbf{w}} \text{loss}(\mathbf{w})$$

---

### Algorithm 3 Mini batch gradient epoch

---

**Data:**  $\mathbf{w}$  initialization,  $\rho$  stepsize

**Result:**  $\mathbf{w}$

**for**  $i=1, n$  **do**

$\mathbf{x}_i, t_i \leftarrow$  pick a point  $i$

$\mathbf{d} \leftarrow \mathbf{d} + \nabla_{\mathbf{w}} \text{loss}(\mathbf{w}, \mathbf{x}_i, t_i)$

**if**  $n \bmod m == 0$  **then**

$\mathbf{w} \leftarrow \mathbf{w} - \rho \mathbf{d}$

$d \leftarrow 0$

**end**

**end**

---

## Averaging through examples (mini batch)

For mini batch of size  $m$  (typically  $m \in \{32, 256\}$ , up to 64k!)

$$\mathbf{d} = \sum_{i=1}^{m \ll n} \nabla_{\mathbf{w}} \text{loss}(\mathbf{w})$$

---

### Algorithm 4 Mini batch gradient epoch

---

**Data:**  $\mathbf{w}$  initialization,  $\rho$  stepsize

**Result:**  $\mathbf{w}$

**for**  $i=1, n$  **do**

$\mathbf{x}_i, t_i \leftarrow$  pick a point  $i$

$\mathbf{d} \leftarrow \mathbf{d} + \nabla_{\mathbf{w}} \text{loss}(\mathbf{w}, \mathbf{x}_i, t_i)$

**if**  $n \bmod m == 0$  **then**

$\mathbf{w} \leftarrow \mathbf{w} - \rho \mathbf{d}$

$\mathbf{d} \leftarrow 0$

**end**

**end**

---

with momentum:  $\mathbf{d} \leftarrow \mu \mathbf{d}$

# Accelerating stochastic gradient

- convergence acceleration: use velocity =  $w_t - w_{t-1}$ 
  - ▶ stochastic gradient with momentum (MOM)  $\mu = .9$ :

$$d = \mu d + \nabla_w L(w)$$

$$w = w - \rho d$$

- ▶ Nesterov accelerated gradient (NAG, 1982)  $\mu = .5$ :

$$g_t = w - \rho \nabla_w L(w)$$

$$w = g_t - \mu(g_t - g_{t-1})$$

- stepsize adaptation

- ▶ Adaptive stepsize gradient  $\varepsilon = 10^{-7}$

$$\rho = \frac{\rho_0}{\|d\| + \varepsilon}$$

- ▶ RMSprop (Tieleman and Hinton, 2012)  $\alpha = 0.9$

$$g = \alpha g + (1 - \alpha) d^\top d$$

$$\rho = \frac{\rho_0}{\sqrt{g} + \varepsilon}$$

- ▶ Adam (Kingma and Ba, 2015)  $\alpha_1 = 0.9, \alpha_2 = 0.999$

$$h = \alpha_1 h + (1 - \alpha) d$$

$$g = \alpha_2 g + (1 - \alpha) d^\top d$$

$$w = w - \frac{\rho_0}{\sqrt{g} + \varepsilon} h$$

# Optimizers available with Keras

Layer wrappers

Writing your own Keras layers

Preprocessing

Sequence Preprocessing

Text Preprocessing

Image Preprocessing

Losses

Metrics

Optimizers

Usage of optimizers

Parameters common to all Keras optimizers

SGD

RMSprop

Adagrad

Adadelta

Adam

Adamax

Nadam

TFOptimizer

Activations

Callbacks

Datasets

Applications

Backend

Initializers

Regularizers

Constraints

Visualization

Scikit-learn API

Utils

## Parameters common to all Keras optimizers

The parameters `clipnorm` and `clipvalue` can be used with all optimizers to control gradient clipping:

```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)
```

### SGD

[\[source\]](#)

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

Stochastic gradient descent optimizer.

Includes support for momentum, learning rate decay, and Nesterov momentum.

### Arguments

- lr: float  $\geq 0$ . Learning rate.
- momentum: float  $\geq 0$ . Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- decay: float  $\geq 0$ . Learning rate decay over each update.
- nesterov: boolean. Whether to apply Nesterov momentum.

### RMSprop

[\[source\]](#)

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

## loss functions and gradients

stochastic gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \rho \mathbf{d}$$

$$\mathbf{d} = \sum_{i=1}^m \nabla_{\mathbf{w}} \text{loss}(\mathbf{w})$$

$t$  target values (expected output)

$y$  output of the perceptron

Square error:

$$\text{loss}(\mathbf{y}, t) = \frac{1}{2}(y - t)^2$$

$$\nabla_{\mathbf{y}} \text{loss}(\mathbf{w}) = (y - t)$$

Cross entropy:  $t \in \{0, 1\}$      $\text{loss}(\mathbf{w}) = -t \log y - (1 - t) \log(1 - y)$

$$\nabla_{\mathbf{y}} \text{loss}(\mathbf{w}) = \frac{y - t}{y(1 - y)}$$

# loss functions available with Keras

Core Layers  
Convolutional Layers  
Pooling Layers  
Locally-connected Layers  
Recurrent Layers  
Embedding Layers  
Merge Layers  
Advanced Activations Layers  
Normalization Layers  
Noise layers  
Layer wrappers  
Writing your own Keras layers  
Preprocessing  
Sequence Preprocessing  
Text Preprocessing  
Image Preprocessing

## Losses

Usage of loss functions  
Available loss functions  
mean\_squared\_error  
mean\_absolute\_error  
mean\_absolute\_percentage\_error  
mean\_squared\_logarithmic\_error  
squared\_hinge  
hinge  
categorical\_hinge  
logcosh  
categorical\_crossentropy  
sparse\_categorical\_crossentropy  
binary\_crossentropy  
kullback\_leibler\_divergence  
poisson  
cosine\_proximity

## Metrics

## Optimizers

## Activations

```
from keras import losses
model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

You can either pass the name of an existing loss function, or pass a TensorFlow/Theano symbolic function that returns a scalar for each data-point and takes the following two arguments:

- \* `y_true`: True labels. TensorFlow/Theano tensor.
- \* `y_pred`: Predictions. TensorFlow/Theano tensor of the same shape as `y_true`.

The actual optimized objective is the mean of the output array across all datapoints.

For a few examples of such functions, check out the losses source.

## Available loss functions

### mean\_squared\_error

```
mean_squared_error(y_true, y_pred)
```

### mean\_absolute\_error

```
mean_absolute_error(y_true, y_pred)
```

### mean\_absolute\_percentage\_error

```
mean_absolute_percentage_error(y_true, y_pred)
```

### mean\_squared\_logarithmic\_error

```
mean_squared_logarithmic_error(y_true, y_pred)
```

### squared\_hinge

```
squared_hinge(y_true, y_pred)
```

### hinge

```
hinge(y_true, y_pred)
```

## Linear neuron and the gradient

- propagation

input  $x \in \mathbb{R}^P$

activation  $a = w^\top x + b \in \mathbb{R}$

output  $y = \sigma(a)$

loss  $J(x) = L(y, t)$

- gradient update:  $w = w - \rho \nabla_w L$

► chain rule:  $\nabla_w J(x) = \frac{\partial L(y, t)}{\partial y} \frac{\partial y}{\partial a} \nabla_w a = \text{back propagation}$

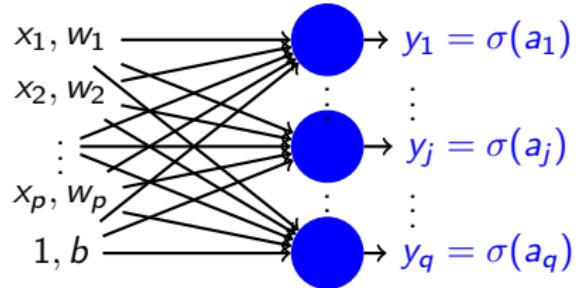
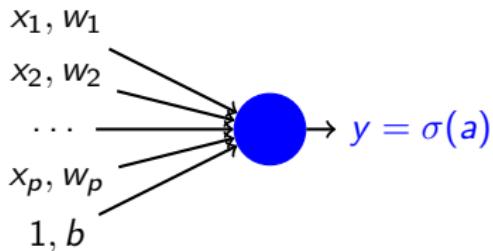
$$\frac{\partial L(y, t)}{\partial w_i} = \frac{\partial L(y, t)}{\partial y} \frac{\partial y}{\partial a} \frac{\partial a}{\partial w_i}$$

- ★  $\partial_y L$
- ★  $\partial_a y = \sigma'(a)$
- ★  $\nabla_w a = x$

► for the square loss:  $w = w - \rho(y - t)\sigma'(a)x$

- gradient with respect to inputs:  $\nabla_x J(x) = (y - t)\sigma'(a)w$

## From one neuron to many ( $q$ )



input  $\mathbf{x} \in \mathbb{R}^p$

activation  $a = \mathbf{w}^\top \mathbf{x} + b \in \mathbb{R}$

weights  $\mathbf{w} \in \mathbb{R}^p$

output  $y = \sigma(a)$

loss  $J(\mathbf{x}) = L(\mathbf{y}, \mathbf{t})$

input  $\mathbf{x} \in \mathbb{R}^p$

activation  $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^q$

weights  $\mathbf{W} \in \mathbb{R}^{q \times p}$

output  $\mathbf{y} = \sigma(\mathbf{a})$

$q$  output

loss  $J(\mathbf{x}) = L(\mathbf{y}, \mathbf{t})$

$L(\mathbf{y}, \mathbf{t}) = \sum_{j=1}^q L(y_j, t_j)$

# Linear neurons and the gradient

- propagation

input  $x \in \mathbb{R}^p$

activation  $a = Wx + b \in \mathbb{R}^q$

weights  $W \in \mathbb{R}^{q \times p}$

output  $y = \sigma(a)$

loss  $J(x) = L(y, t)$

$q$  neurons

a matrix of synaptic weights

$q$  output

$$L(y, t) = \sum_{j=1}^q L(y_j, t_j)$$

- gradient update:  $W = W - \rho \nabla_w L$

► chain rule:  $\nabla_w J(x) = \sum_{j=1}^q \frac{\partial L(y_j, t_j)}{\partial y_j} \frac{\partial y_j}{\partial a_j} \nabla_w a_j = \text{back propagation}$

$$\frac{\partial L(y, t)}{\partial w_{ij}} = \frac{\partial L(y_j, t_j)}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

► for the square loss:

$$W = W - \rho \underbrace{\mathbf{v} x^\top}_{\text{outer product}}$$

with  $\mathbf{v} = (y - t) \cdot \sigma'(a)$

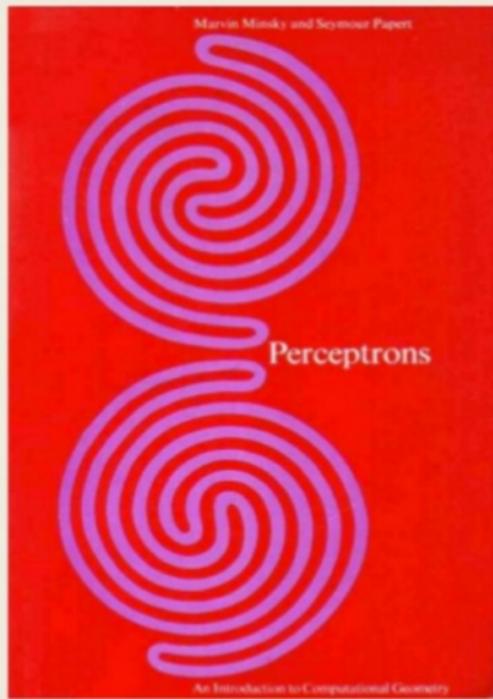
- gradient with respect to inputs :  $\nabla_x J(x) = W^\top \mathbf{v}$

# The neural networks time line

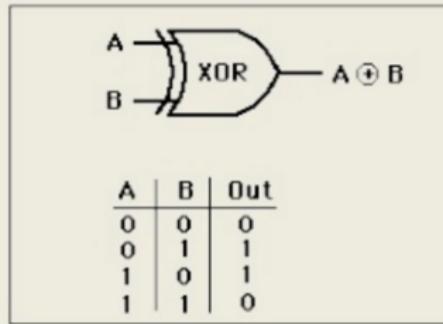
- The first stage: 1890 - 1969
  - ~1890 Ramón y Cajal: the biological neuron
  - 1943 McCulloch & Pitts formal neuron
  - 1949 Hebb's rule
  - 1958 Rosenblatt's Perceptron: learning with stochastic gradient
  - 1969 Minsky & Papert: stop – the 1st NN winter
- The second stage: 1985 - 1995
- The third stage: 2006 - (2012) - 2018...

However, linear neurons are linear

## 1969: Perceptrons can't do XOR!



<http://www.i-programmer.info/images/stories/BabBag/AI/book.jpg>



<http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/ietron/xor.gif>



Minsky & Papert

<https://constructingkids.files.wordpress.com/2013/05/minsky-papert-71-csolomon-x640.jpg>

## Limitations of the Perceptron

- Linear threshold units as Boolean gates
- Circuit theory is poorly known
- Learning deep circuits means solving the credit assignment pb
- Linearly separable problems are few
- Elementary problems need complex circuits. (parity, x-or...)
- But have simple algorithmic solutions  
→programming versus learning

Abandon perceptrons and other analog computers

Develop symbolic computers and symbolic AI techniques.

## The neural networks time line

- The first stage: 1890 - 1969
- The second stage: 1985 - 1995

1985 Rumelhart, Hinton & Williams; Le Cun: go - backpropagation

1989 Universal Approximation Cybenko-Hornik-Funahashi Theorem

1989 Y. Le Cun's convolutional neural networks

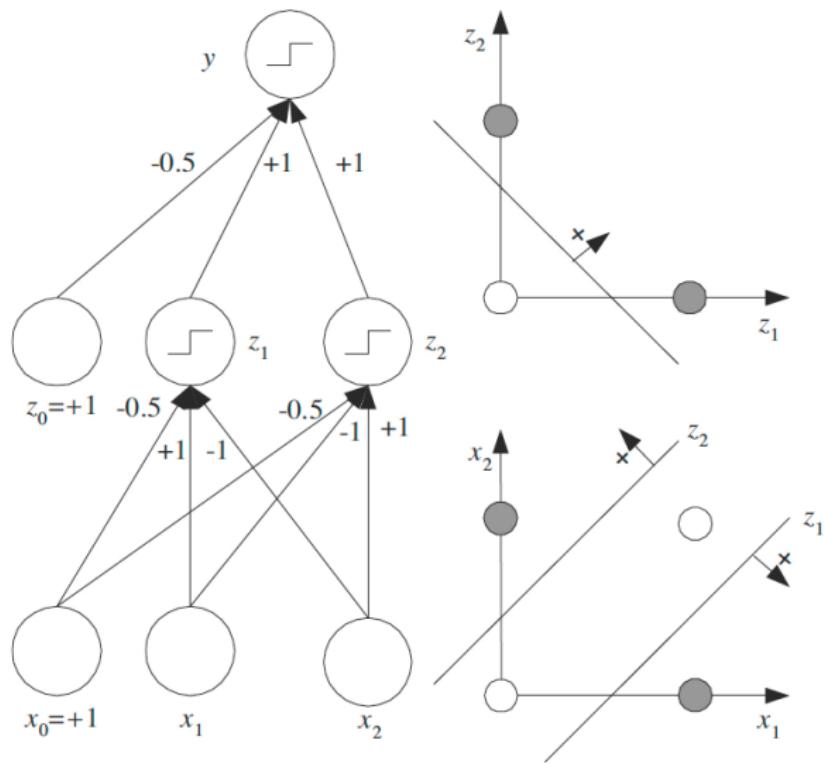
1995 Recurrent neural networks, LSTM

1995 SVM

2004 Caltech 101: the 2nd NN winter

- The third stage: 2006 - (2012) - 2018...

# Non linearity combining linear neurons: the Xor case



# Neural networks

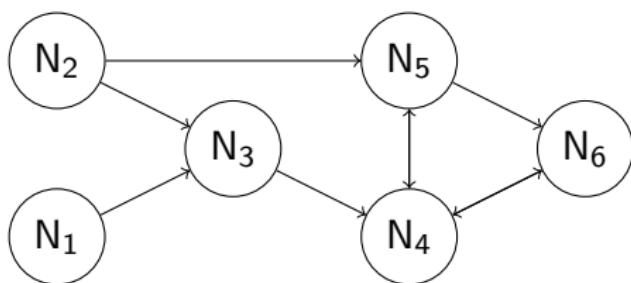
Definition: Neural network

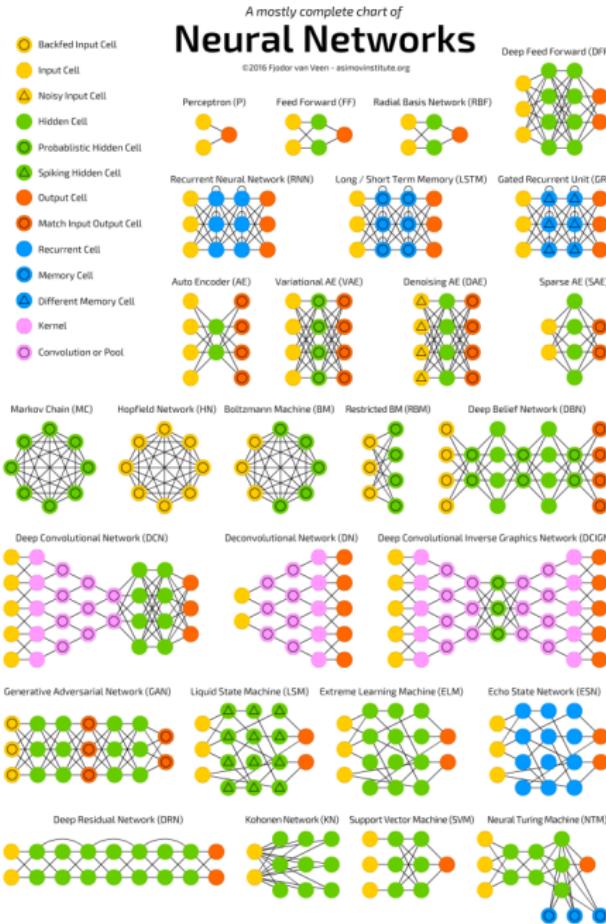
A neural network is an oriented graph of formal neurons

When two neurons are connected (linked by an oriented edge of the graph), the output of the head neuron is used as an input by the tail neuron. It can be seen as a weighted directed graph.

3 different neurons are considered:

- input neurons (connected with the input)
- output neurons
- hidden neurons





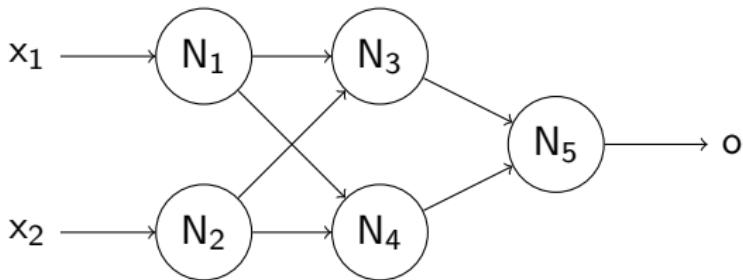
# The Multilayer perceptron (MLP)

## Definition: Multilayer perceptron

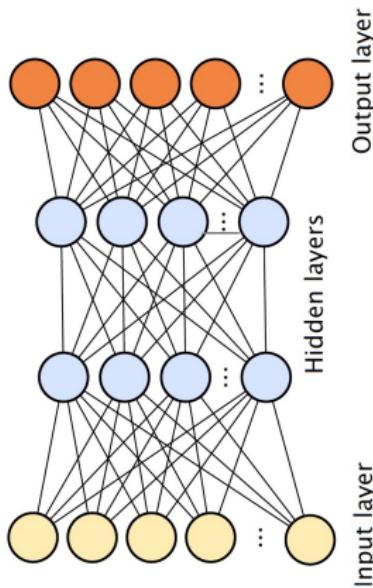
A Multilayer perceptron is an acyclic neural network,

where the neurons are structured in successive layers, beginning by an input layer and finishing with an output layer.

Example: The X-or neural network is a MLP with a single hidden unit with 2 hidden neurons.



# MLP training with back propagation (and SGD)



$$y = \sigma(W_3 h^{(2)})$$

$$\nabla_{W_3} J = (y - y_a) \sigma'(W_3 h^{(2)}) h^{(2)}$$

↑

↓

$$h^{(2)} = \sigma(W_2 h^{(1)})$$

$$\nabla_{W_2} J = \nabla_{h^{(2)}} J \sigma'(W_2 h^{(1)}) h^{(1)}$$

↑

↓

$$h^{(1)} = \sigma(W_1 x)$$

$$\nabla_{W_1} J = \nabla_{h^{(1)}} J \sigma'(W_1 x) x^\top$$

↑

x

$$y = \sigma(W_3 \sigma(W_2 \sigma(W_1 x)))$$

backpropagation = chain rule (autodiff)

Used to learn internal representation  $W_1, W_2, W_3$

# Back propagation in python

$$y = W_2 \sigma(W_1 x)$$

```
#           PROPAGATION
x = x_train[ind,:]
a = W1@np.r_[x.T, np.ones((1,batch_size))]
h = MyReLU.forward(a)
y = W2@np.r_[h, np.ones((1,batch_size))]

#           COST
err, exps = MyCrossEntropy.Cost(y,Y_train[ind,:].T)

#           BACK PROPAGATION
dJ_dy = exps - Y_train[ind,:].T
gradW2 = dJ_dy@np.r_[h, np.ones((1,batch_size))].T + la*W2
dJ_da = W2[:,0:n1].T@dJ_dy * MyReLU.backward(a1)
gradW1 = dJ_da@np.r_[x.T, np.ones((1,batch_size))].T + la*W1

#           GRADIENT UPDATE
W1 = W1 - stepsize*gradW1
W2 = W2 - stepsize*gradW2
```

# Back propagation is differential learning



Yann LeCun

5 janvier · 🌎

OK, Deep Learning has outlived its usefulness as a buzz-phrase.  
Deep Learning est mort. Vive Differentiable Programming!

## Numpy

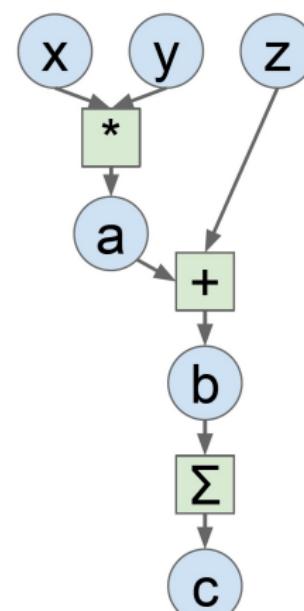
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

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

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
```



## The neural networks time line

- The first stage: 1890 - 1969
- The second stage: 1985 - 1995

1985 Rumelhart, Hinton & Williams; Le Cun: go - backpropagation

1989 Universal Approximation Cybenko-Hornik-Funahashi Theorem

1989 Y. Le Cun's convolutional neural networks

1995 Recurrent neural networks, LSTM

1995 SVM

2004 Caltech 101: the 2nd NN winter

- The third stage: 2006 - (2012) - 2018...

## Two theoretical results about MLP

### Universal approximation theorem for one hidden layer MLP

- given any  $\varepsilon > 0$
- for any continuous function  $f$  on compact subsets of  $\mathbb{R}^P$
- for any admissible activation function  $\sigma$  (not a polynomial)
- there exists  $h$ ,  $W_1 \in \mathbb{R}^{P \times h}$ ,  $b \in \mathbb{R}^h$ ,  $c \in \mathbb{R}$  and  $w_2 \in \mathbb{R}^h$  such that

$$\|f(x) - w_2\sigma(W_1x + b) + c\|_\infty \leq \varepsilon$$

Approximation theory of the MLP model in neural networks, A Pinkus - Acta Numerica, 1999

SVM, Boosting and Random Forest also are universal approximators

### Why two hidden layers can be better than one?

There exists a function on  $\mathbb{R}^P$ , expressible by a small two hidden layer MLP, which cannot be approximated by any two hidden layer MLP, to more than a certain constant accuracy, unless its width is exponential in the  $p$ .

# The neural networks time line

- The first stage: 1890 - 1969
- The second stage: 1985 - 1995

1985 Rumelhart, Hinton & Williams; Le Cun: go - backpropagation

1989 Universal Approximation Cybenko-Hornik-Funahashi Theorem

1989 Y. Le Cun's convolutional neural networks

1995 Recurrent neural networks, LSTM

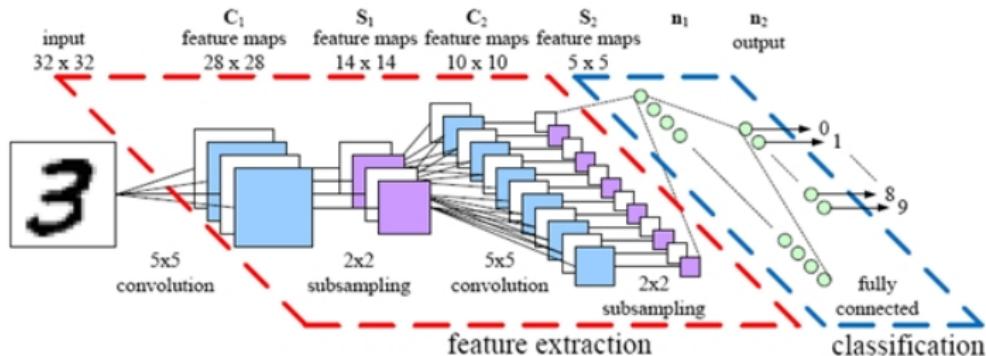
1995 SVM

2004 Caltech 101: the 2nd NN winter

- The third stage: 2006 - (2012) - 2018...

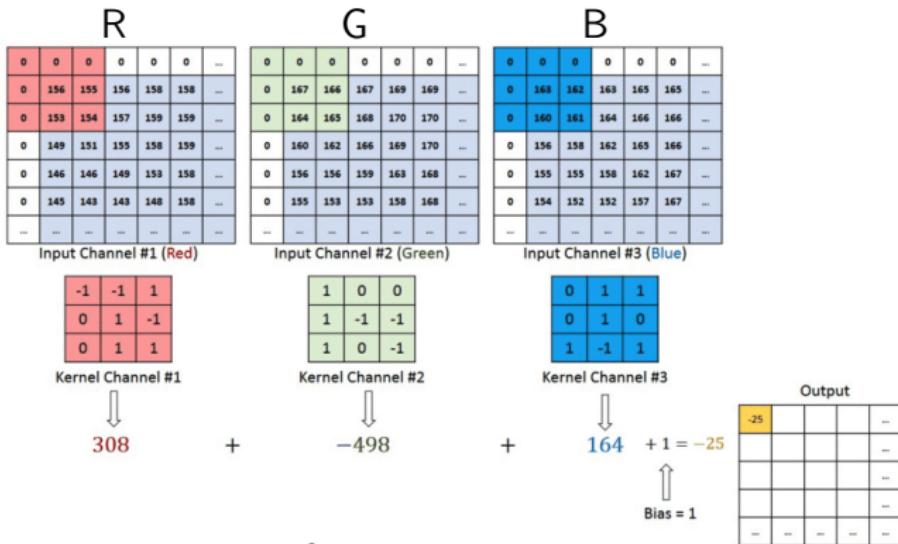
# OCR: MNIST database (LeCun, 1989)

3	8	6	9	6	4	5	3	8	4	5	2	3	8	4	8
1	5	0	5	9	7	4	1	0	3	0	6	2	9	9	4
1	3	6	8	0	7	7	6	8	9	0	3	8	3	7	7
8	4	4	1	2	9	8	1	1	0	6	6	5	0	1	1
7	2	7	3	1	4	0	5	0	6	8	7	6	8	9	9
4	0	6	1	9	2	6	3	9	4	4	5	6	6	1	7
2	8	6	9	7	0	9	1	6	2	8	3	6	4	9	5
8	6	8	7	8	8	6	9	1	7	6	0	9	6	7	0



use convolution layers

# A convolution layer in a neural network



$$O_{ij}^1 = \sum_{k,\ell,m=1}^3 w_{k,\ell,m}^1 l_{2i+k, 2j+\ell, m} + b_1$$

Convolution reduces the number of parameters

# CNN results on MNIST in 1998 (historical debate)

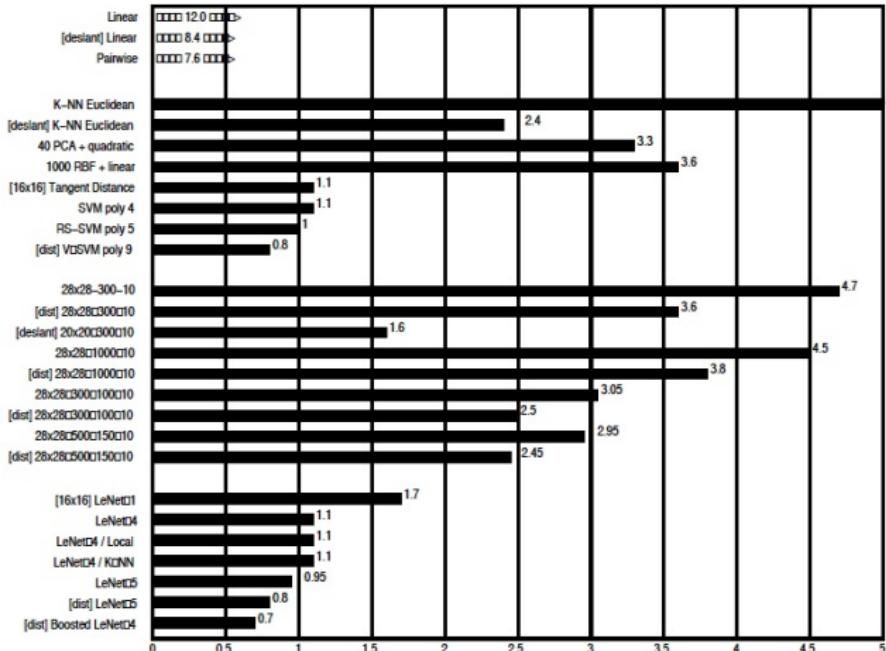


Fig. 9. Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system used the 16x16 pixel images. The uncertainty in the quoted error rates is about 0.1%.

## The neural networks time line

- The first stage: 1890 - 1969
- The second stage: 1985 - 1995

1985 Rumelhart, Hinton & Williams; Le Cun: go - backpropagation

1989 Universal Approximation Cybenko-Hornik-Funahashi Theorem

1989 Y. Le Cun's convolutional neural networks

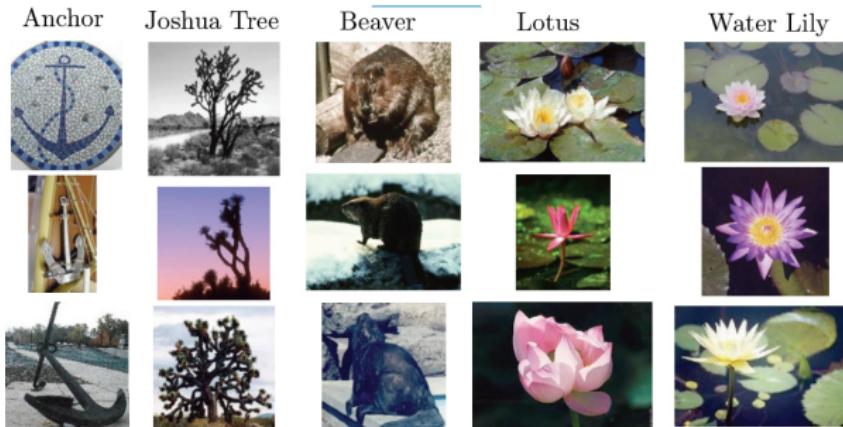
1995 Recurrent neural networks, LSTM

1995 SVM

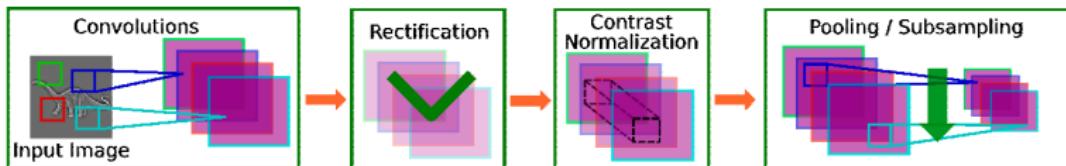
2004 Caltech 101: the NN winter

- The third stage: 2006 - (2012) - 2018...

# The caltech 101 database (2004)



- 101 classes,
- 30 training images per category
- ...and the winner is NOT a deep network
  - ▶ dataset is too small



use convolution + Rectification + Normalization + Pooling

## The neural networks time line

- The first stage: 1890 - 1969
  - The second stage: 1985 - 1995
  - The third stage: 2006 - (2012) - 2018...
- 2006 Deep learning: Bengio's, Hinton's RBM, Y LeCun's proposals  
2010 Andrew Ng's GPU for Deep GPU  
2011 Deep frameworks, tools (theano, torch, cuda-convnet... )  
2012 ImageNet – AlexNet  
2013 M. Zuckerberg at NIPS the deep fashion  
2014 Representation learning fine tuning  
2015 Deep learning in the industry: speech, traduction, image...  
2016 Goodfellow's generative adversarial networks (GAN)  
2017 Reinforcement learning: Deep win's GO  
2018 Automatic design, adversarial defense, green learning, theory...

# The ImageNet database (Deng et al., 2012)



ImageNet = 15 million high-resolution images of 22,000 categories.  
Large-Scale Visual Recognition Challenge (a subset of ImageNet)

- 1000 categories.
- 1.2 million training images,
- 50,000 validation images,
- 150,000 testing images.

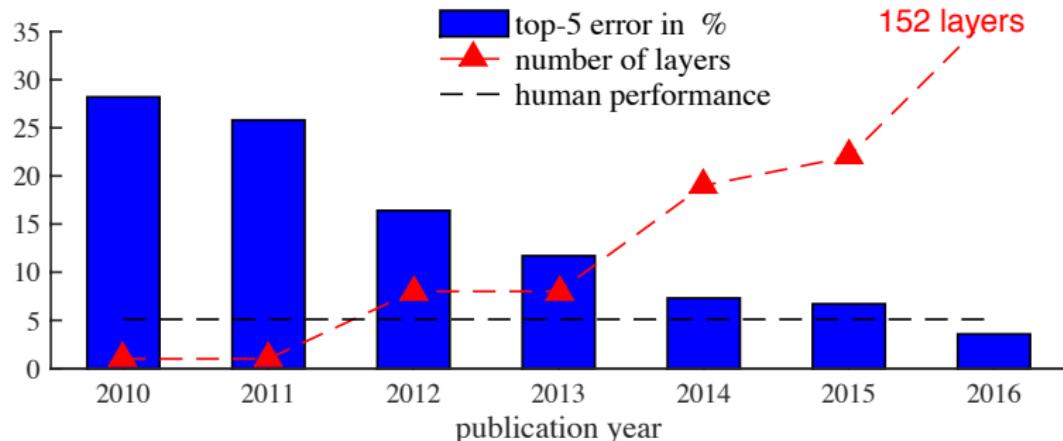
# A new fashion in image processing

2012 Teams	%error	2013 Teams	%error	2014 Teams	%error
Supervision (Toronto)	15.3	Clarifai (NYU spinoff)	11.7	GoogLeNet	6.6
ISI (Tokyo)	26.1	NUS (singapore)	12.9	VGG (Oxford)	7.3
VGG (Oxford)	26.9	Zeiler-Fergus (NYU)	13.5	MSRA	8.0
XRCE/INRIA	27.0	A. Howard	13.5	A. Howard	8.1
UvA (Amsterdam)	29.6	OverFeat (NYU)	14.1	DeeperVision	9.5
INRIA/LEAR	33.4	UvA (Amsterdam)	14.2	NUS-BST	9.7
		Adobe	15.2	TTIC-ECP	10.2
		VGG (Oxford)	15.2	XYZ	11.2
		VGG (Oxford)	23.0	UvA	12.1

shallow approaches

deep learning

# ImageNet results



2012 Alex Net

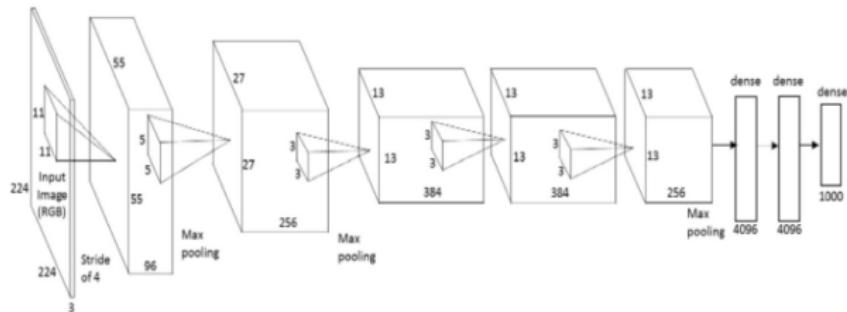
2013 ZFNet

2014 VGG

2015 GoogLeNet / Inception

2016 Residual Network

# Deep architecture for ImageNet (15%)



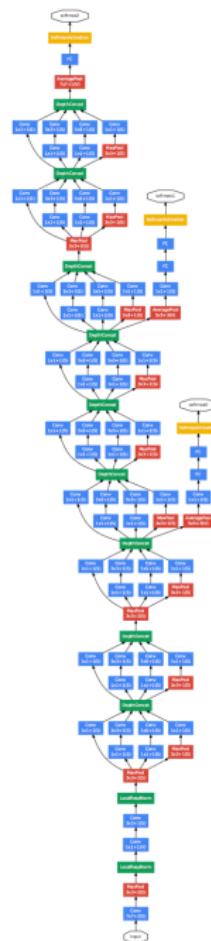
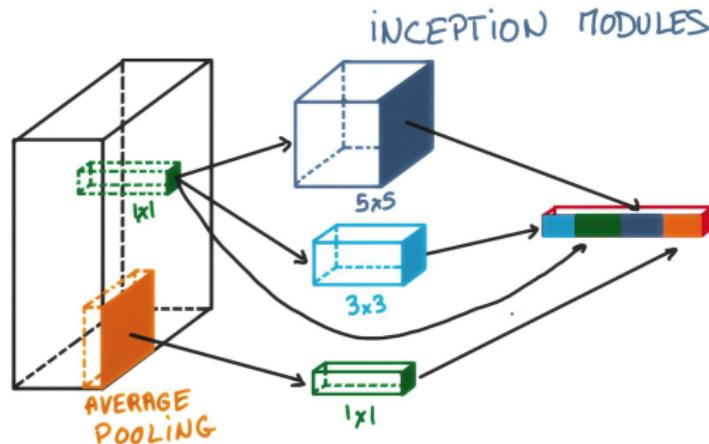
The *AlexNet* architecture [Krizhevsky, Sutskever, Hinton, 2012]

Convolution + Rectification (ReLU) + Normalization + Pooling

- 60 million parameters
- using 2 GPU – 6 days
- regularization
  - ▶ data augmentation
  - ▶ dropout
  - ▶ weight decay



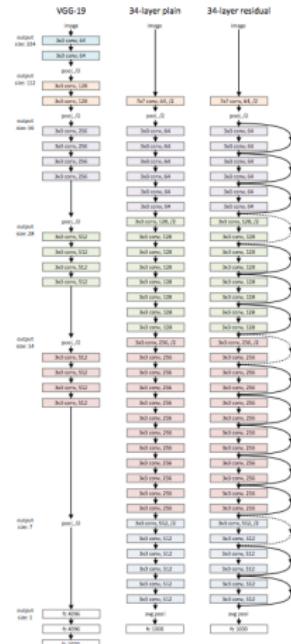
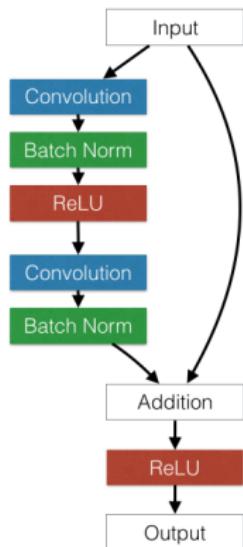
# From 15% to 7%: Inceptionism



Network in a network (deep learning lecture at Udacity)

Christian Szegedy et. al. Going deeper with convolutions. CVPR 2015.

# From 7% to 3%: Residual Nets



Beating the gradient vanishing effect

K. He et al., 2016

## The neural networks time line

- The first stage: 1890 - 1969
  - The second stage: 1985 - 1995
  - The third stage: 2006 - (2012) - 2018...
- 2006 Deep learning: Bengio's, Hinton's RBM, Y LeCun's proposals  
2010 Andrew Ng's GPU for Deep GPU  
2011 Deep frameworks, tools (theano, torch... )  
2012 ImageNet – AlexNet  
2013 M. Zuckerberg at NIPS: the deep fashion  
2014 Representation learning fine tuning  
2015 Deep learning in the industry: speech, traduction, image...  
2016 Goodfellow's generative adversarial networks (GAN)  
2017 Reinforcement learning: Deep win's GO  
2018 Automatic design, adversarial defense, green learning, theory...

# Deep learning, AI and the industry



- data science, **artificial intelligence** and deep learning
- the GAFAM – BATX vision
  - ▶ they got the infrastructure (hard+software)
  - ▶ they got the data
  - ▶ deep learning bridges the gap between applications and ML

In 2016, Google Chief Executive Officer (CEO) Sundar Pichai said, Machine learning [a subfield of AI] is a core, transformative way by which we're rethinking how we're doing everything.

## So far so good

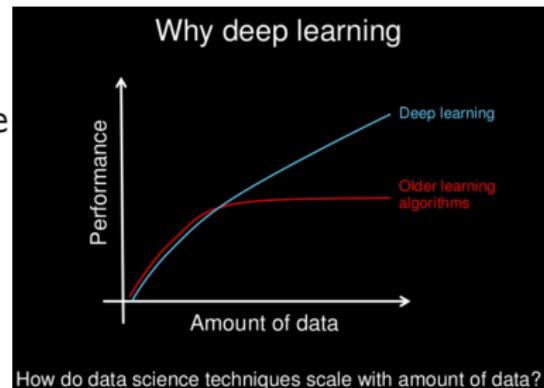
- from the formal neuron to deep learning
  - ▶ one neuron is a linear perceptron
  - ▶ many layered neurons are non linear multilayer perceptrons
  - ▶ deep networks is a new name for multilayer perceptrons
- deep learning breakthrough starts with ImageNet
  - ▶ better than human performances
  - ▶ on many perception tasks
- deep learning could transform almost any industry
  - ▶ the AI revolution

Neural networks+backpropagation exist since 1985

→ what's new?

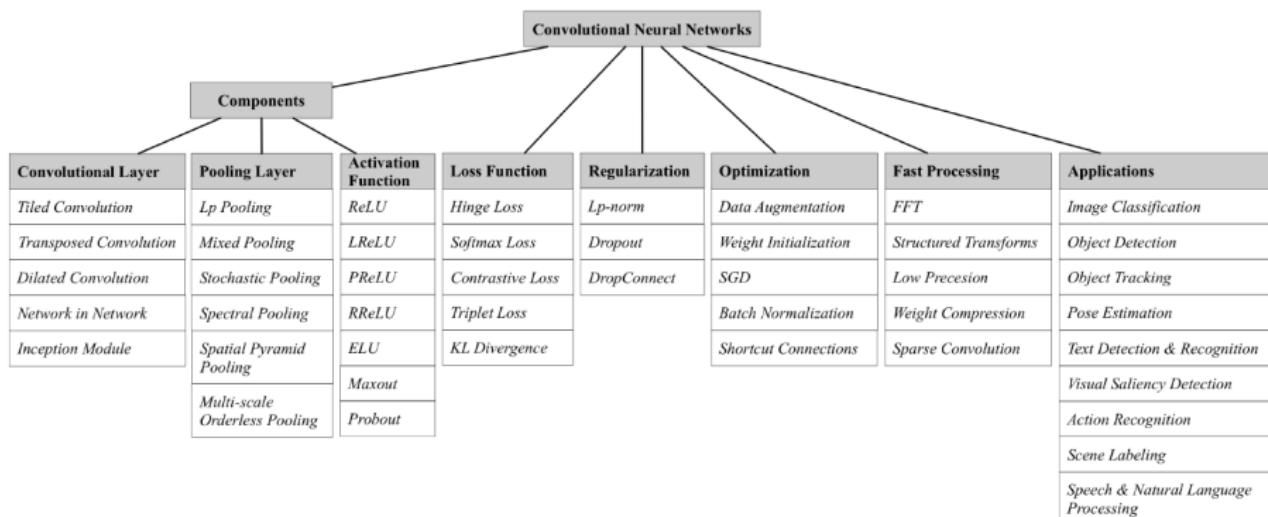
# Road map

- 1 Why deep learning?
- 2 The first stage: 1890 - 1969
- 3 The second stage: 1985 - 1995
- 4 The third stage: 2006 - (2012) - 2018...
- 5 What's new in deep learning?
  - Big is beautiful
  - Two Hot topics: data and architecture
- 6 Conclusion



# What's new with deep learning

- a lot of **data** (big data)
- big computing resources (**hardware & software**),
- big **model** (deep vs. shallow)
  - new architectures
  - new learning tricks



## Big data: a lot of available training data



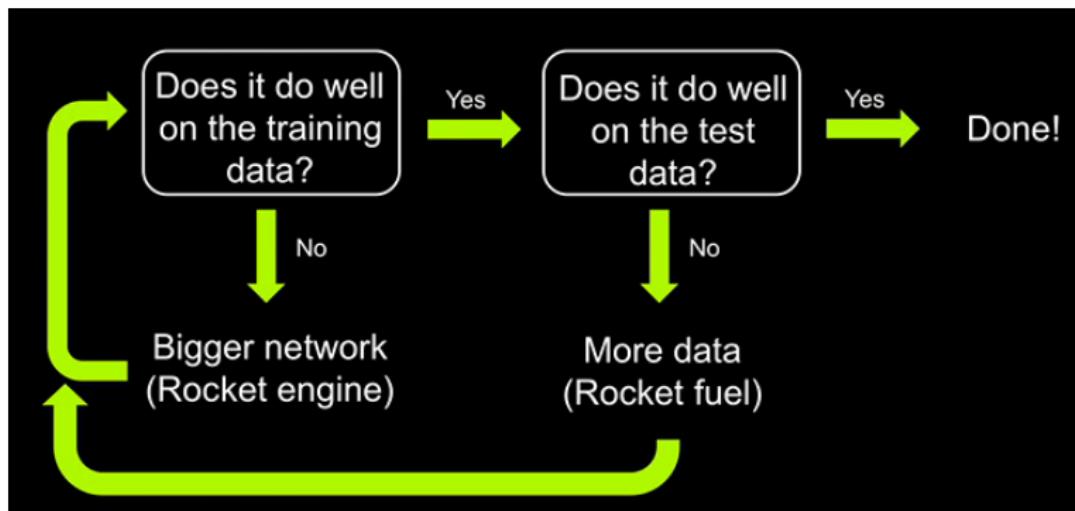
- ImageNet: 1,200,000x256x256x3 (about 200GB) block of pixels
- MS COCO for supervised learning
  - ▶ Multiple objects per image
  - ▶ More than 300,000 images
  - ▶ More than 2 Million instances
  - ▶ 80 object categories
  - ▶ 5 captions per image
- YFCC100M for unsupervised learning
- Google Open Images, 9 million URLs to images annotated over 6000 categories
- Visual genome: data + knowledge <http://visualgenome.org/>

# Andrew Ng basic recipe for machine learning

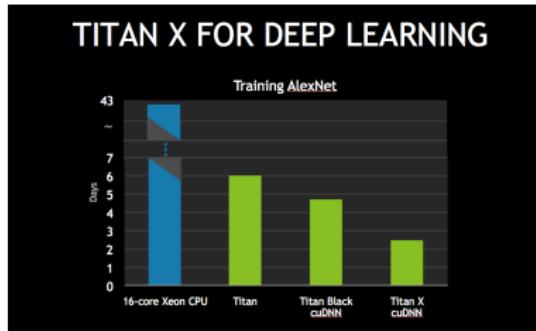
## Why is Deep Learning taking off?

**fuel = data**

engine = model (a deep network)



# Big computers: GPU needed



Now 2 hours with Nvidia DGX-1, and enough Memory

**Table 1 : Training time and top-1 1-crop validation accuracy with ImageNet/ResNet-50**

	Batch Size	Processor	DL Library	Time	Accuracy
He et al. [7]	256	Tesla P100 x8	Caffe	29 hours	75.3%
Goyal et al. [1]	8K	Tesla P100 x256	Caffe2	1 hour	76.3%
Smith et al. [4]	8K→16K	full TPU Pod	TensorFlow	30 mins	76.1%
Akiba et al. [5]	32K	Tesla P100 x1024	Chainer	15 mins	74.9%
Jia et al. [6]	64K	Tesla P40 x2048	TensorFlow	6.6 mins	75.8%
<b>This work</b>	<b>34K→68K</b>	<b>Tesla V100 x2176</b>	<b>NNL</b>	<b>224 secs</b>	<b>75.03%</b>

ImageNet/ResNet-50 Training in 224 Seconds, 2018

# big software: deep learning frameworks

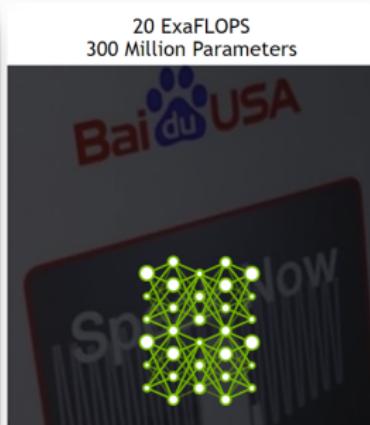
	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Tensorflow (Google) is the most popular with Keras. Pytorch is a challenger.

# Big architectures



2015 - Microsoft ResNet  
Superhuman Image Recognition



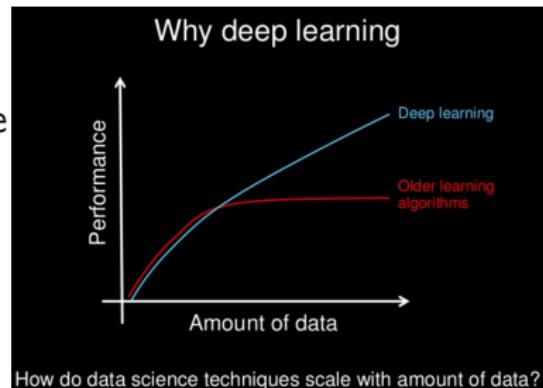
2016 - Baidu Deep Speech 2  
Superhuman Voice Recognition



2017 - Google Neural Machine Translation  
Near Human Language Translation

# Road map

- ① Why deep learning?
- ② The first stage: 1890 - 1969
- ③ The second stage: 1985 - 1995
- ④ The third stage: 2006 - (2012) - 2018...
- ⑤ What's new in deep learning?
  - Big is beautiful
  - Two Hot topics: data and architecture
- ⑥ Conclusion



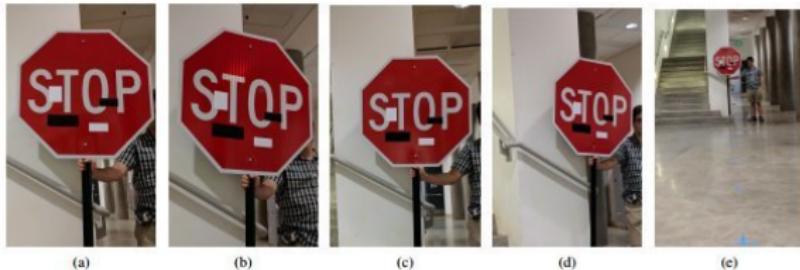
## Deep neural networks are easily fooled (1/2)

$$\begin{array}{ccc} \text{Image of a panda} & + .007 \times & \text{Image of a gibbon} \\ x & \text{sign}(\nabla_x J(\theta, x, y)) & x + \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \\ \text{"panda"} & \text{"nematode"} & \text{"gibbon"} \\ 57.7\% \text{ confidence} & 8.2\% \text{ confidence} & 99.3 \% \text{ confidence} \end{array}$$

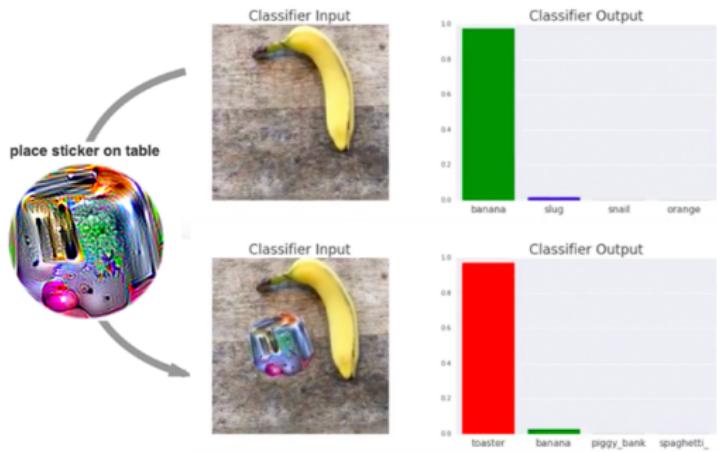
Explaining and Harnessing Adversarial Examples, Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, 2015

<https://arxiv.org/abs/1412.6572>

## Adversarial examples (2/2)



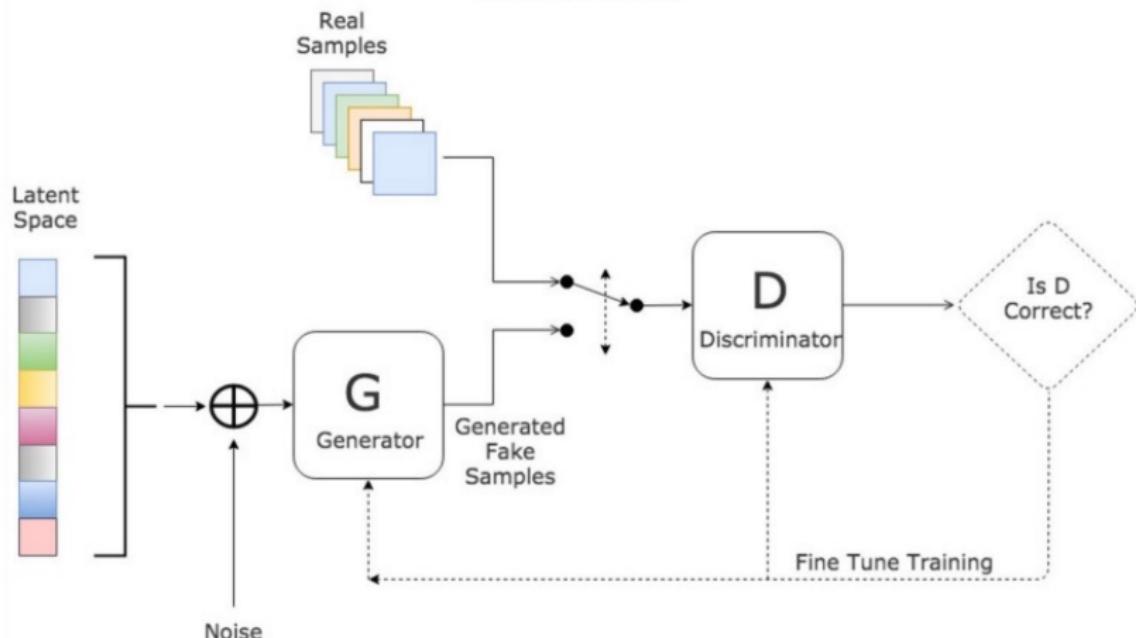
Adversarial Examples for Evaluating Reading Comprehension Systems, Robin Jia, Percy Liang, 2017



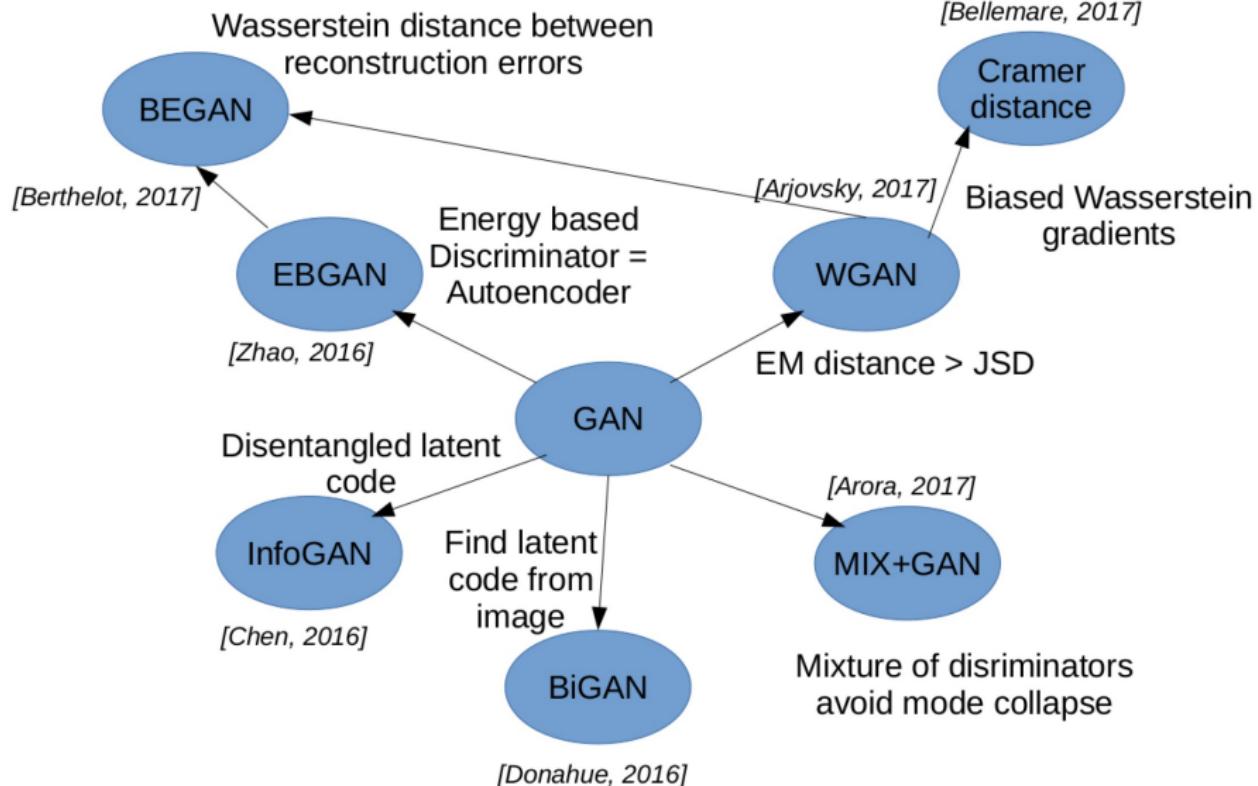
Adversarial Patch Tom B. Brown, Dandelion Mané, Aurko Roy, Martin Abadi, Justin Gilmer, 2017

## Generative models

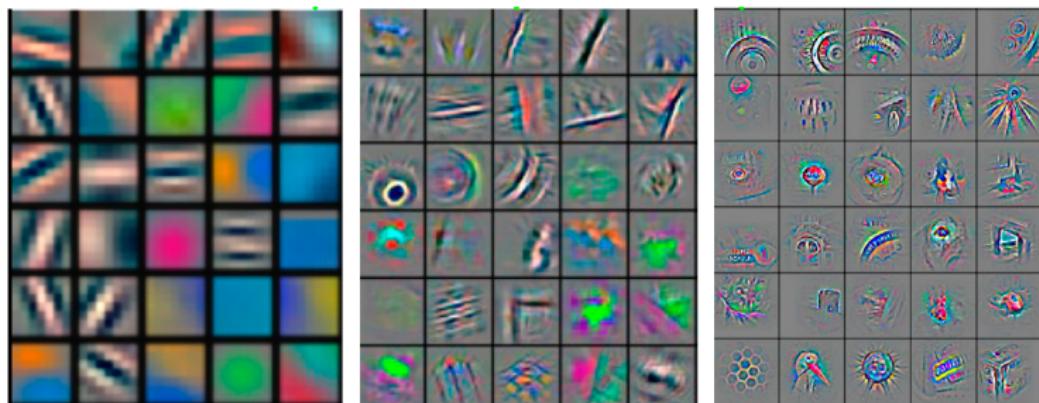
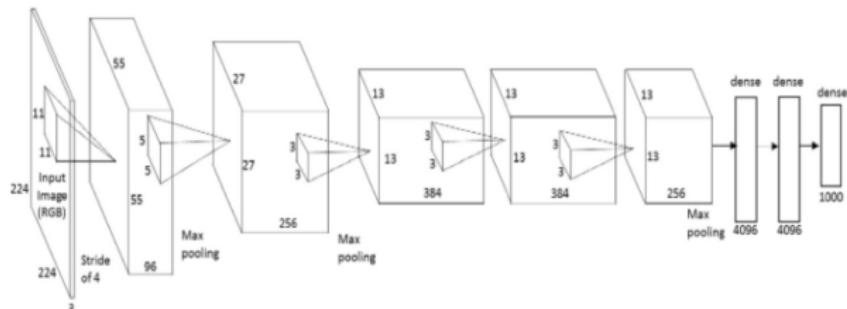
# Generative Adversarial Network



# Other Generative architectures

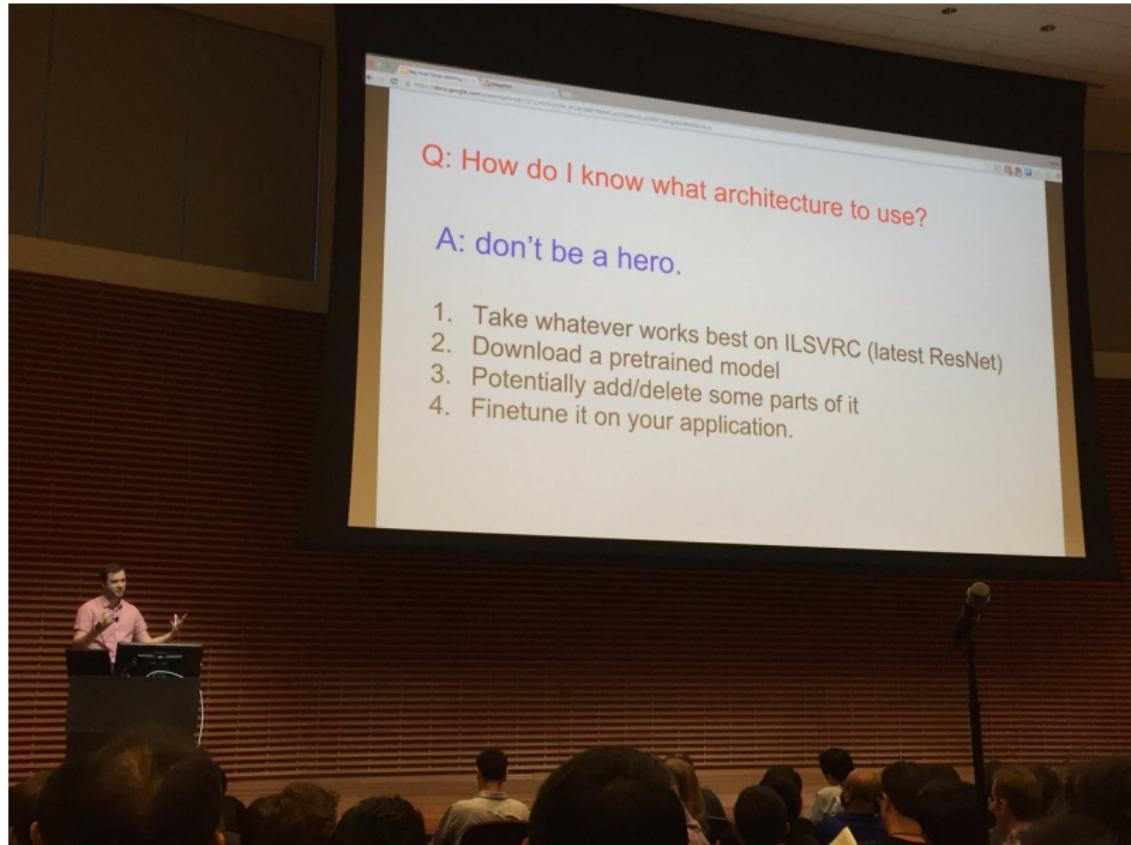


# AlexNet works because of learning internal representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

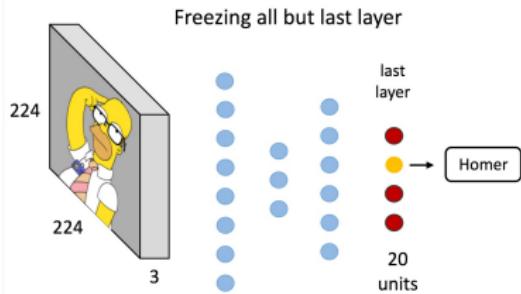
# How to start with deep learning?



# The art of using pre-trained models

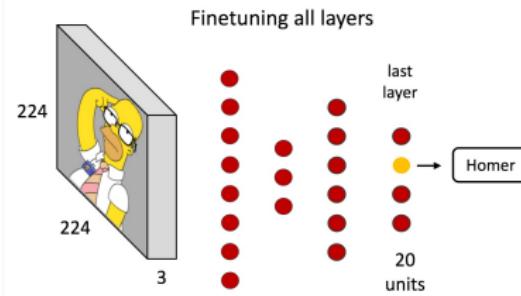
- Transfer learning:

- 1 download a pre-trained deep architecture (e.g. AlexNet for image processing)
- 2 propagate new data through the network **without its last(s) layer(s)**
- 3 use the output of the network as new feature



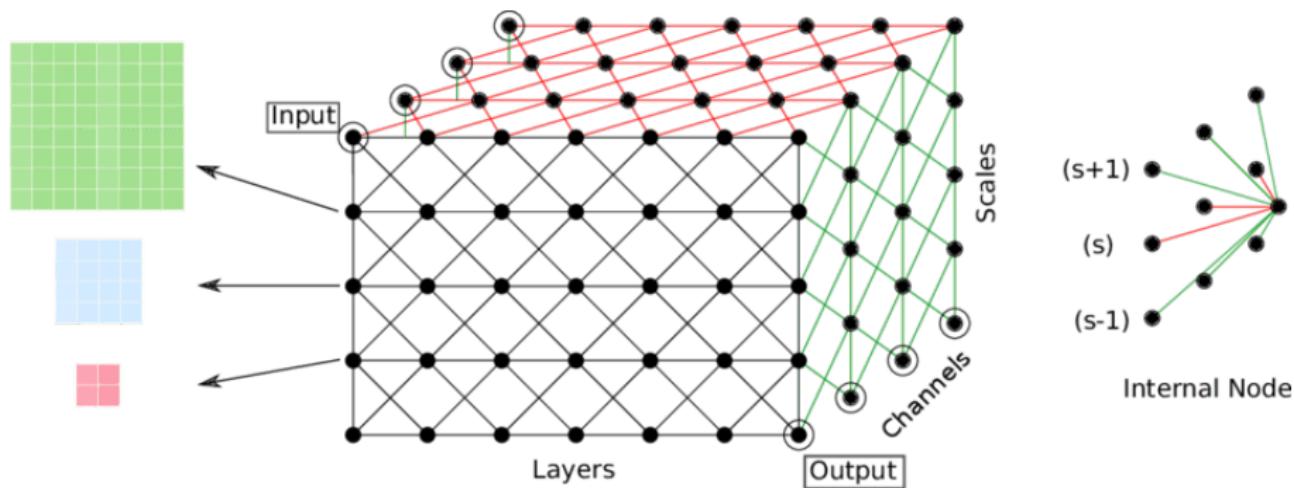
- Fine tuning

- 1 download a pre-trained deep architecture (AlexNet)
- 2 adapt the output layer to your problem
- 3 train the deep architecture with your data using the pre-trained model as a starting point

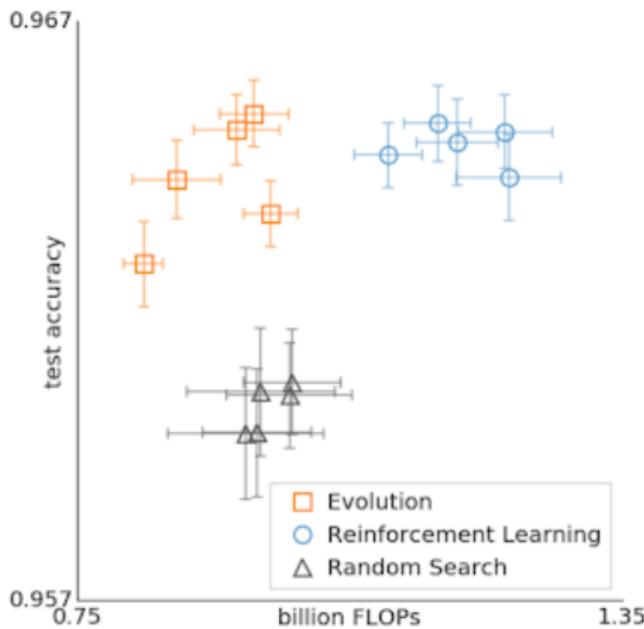
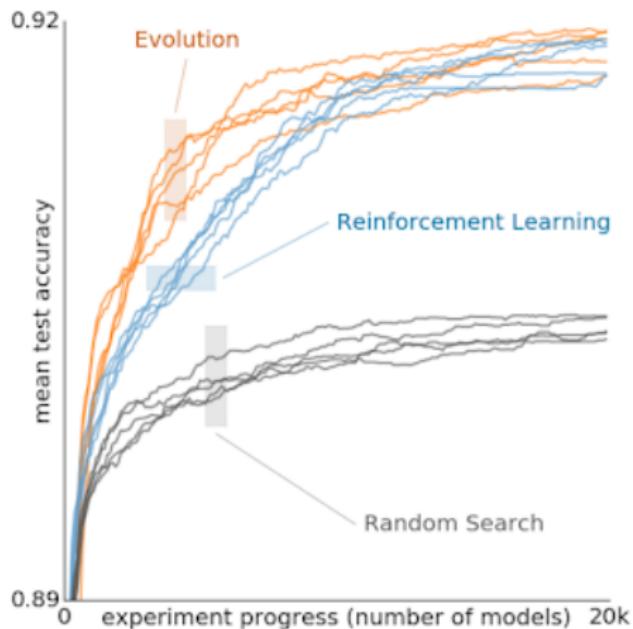


# Convolutional Neural Fabrics

- problem: how to find the most relevant architecture
- todays solution: try and test
- A new solution: learn the architecture



# Neural Architecture Search

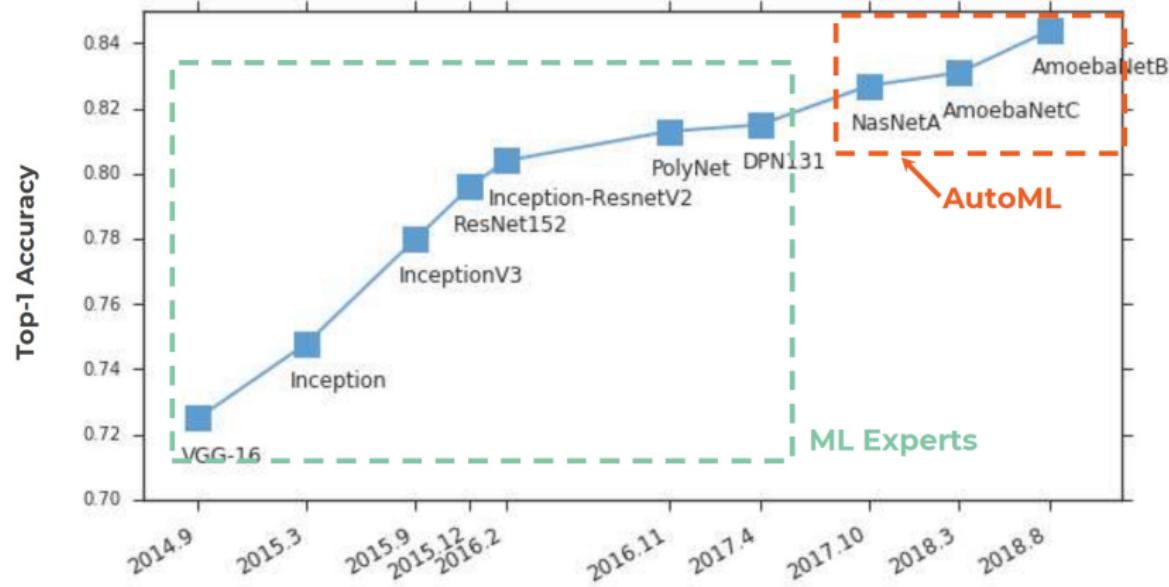


Regularized Evolution for Image Classifier Architecture Search, E. Real et al, 2018

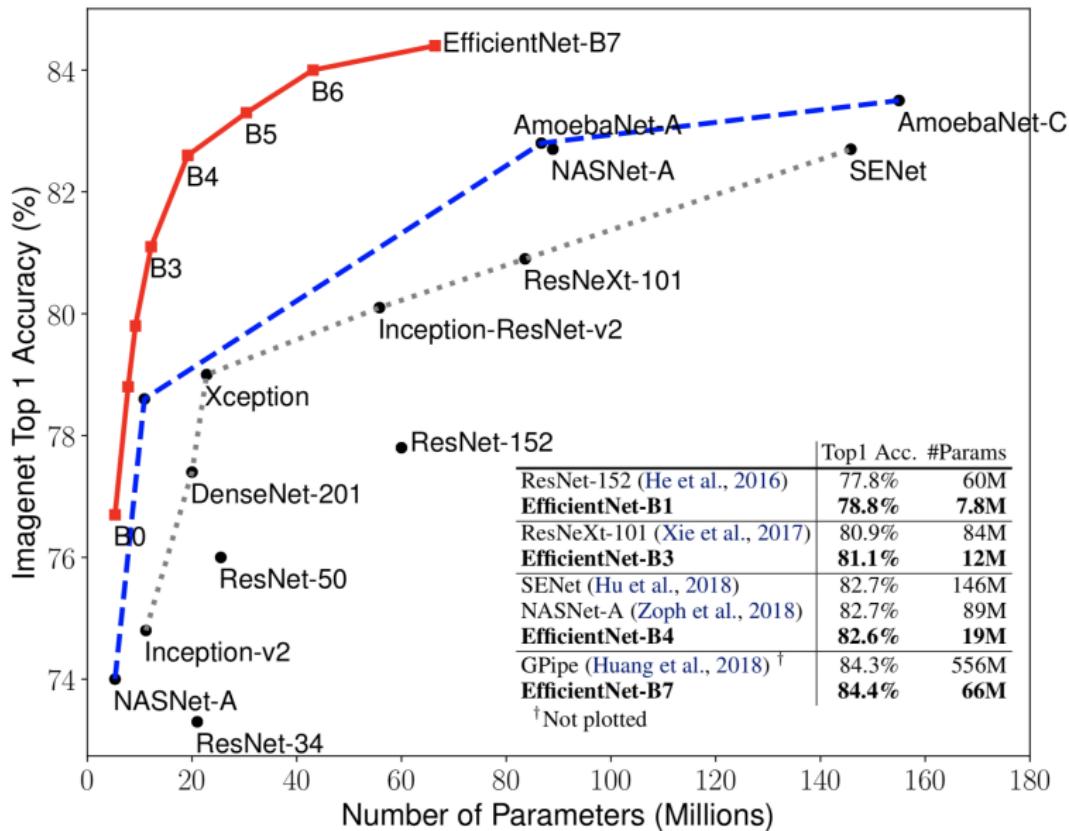
<https://chinagdg.org/2018/03/using-evolutionary-automl-to-discover-neural-network-architectures/>

# Experts vs AutoML

## ImageNet

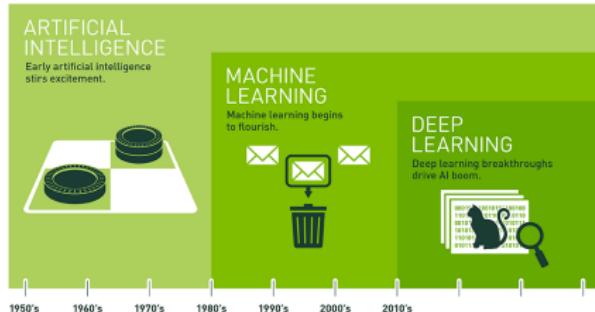


# Neural Architecture Search



# Road map

- 1 Why deep learning?
- 2 The first stage: 1890 - 1969
- 3 The second stage: 1985 - 1995
- 4 The third stage: 2006 - (2012) - 2018...
- 5 What's new in deep learning?
  - Big is beautiful
  - Two Hot topics: data and architecture
- 6 Conclusion



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

## The deep learning time line

- The first stage: 1890 - 1969
  - ▶ learning is optimization with **stochastic gradient** (to scale)
- The second stage: 1985 - 1995
  - ▶ NN are universal approximator **differentiable graphs** (that scales)
- The third stage: 2006 - (2012) - 2018...
  - ▶ scale with **big** data+computers+architecture (deep)
- Open issues
  - ▶ provide guaranties: adversarial examples and representation learning
  - ▶ architecture design (autoML)
  - ▶ theory needed
  - ▶ do more with less: green learning
  - ▶ the future of deep learning depends on trust

# Open issues in Deep learning : A critical appraisal

arXiv.org > cs > arXiv:1801.00631

Search or Art

(Help | Advanced)

Computer Science > Artificial Intelligence

## Deep Learning: A Critical Appraisal

Gary Marcus

(Submitted on 2 Jan 2018)

Although deep learning has historical roots going back decades, neither the term "deep learning" nor the approach was popular just over five years ago, when the field was reigned by papers such as Krizhevsky, Sutskever and Hinton's now classic (2012) deep network model of Imagenet. What has the field discovered in the five subsequent years? Against a background of considerable progress in areas such as speech recognition, image recognition, and game playing, and considerable enthusiasm in the popular press, I present ten concerns for deep learning, and suggest that deep learning must be supplemented by other techniques if we are to reach artificial general intelligence.

Comments: 1 figure

Subjects: Artificial Intelligence (cs.AI); Machine Learning (cs.LG); Machine Learning (stat.ML)

MSC classes: 97R40

ACM classes: I.2.0; I.2.6

Cite as: arXiv:1801.00631 [cs.AI]

(or arXiv:1801.00631v1 [cs.AI] for this version)

### Bibliographic data

[Enable Bibex (What is Bibex?)]

### Submission history

From: Gary Marcus [view email]

[v1] Tue, 2 Jan 2018 12:49:35 GMT (258kb)

For most problems where deep learning has enabled transformationally better solutions (vision, speech), we've entered diminishing returns territory in 2016-2017.

Francois Chollet, Google, author of Keras neural network library Dec. 2017

## 10 Limits on the scope of deep learning

- It is data hungry
- specialized training (no learning)
- it has no natural way to deal with hierarchical structure
- it has struggled with open-ended inference
- it is not sufficiently transparent
- not well integrated with prior knowledge (NLP)
- cannot distinguish causation from correlation
- assume stationarity
- can be fooled (it is not robust)
- is difficult to engineer with

# 10 Limits on the scope of deep learning

## GREEN LEARNNING

- It is data hungry
- specialized training (no learning)
- it has no natural way to deal with hierarchical structure
- it has struggled with open-ended inference

## THEORY NEEDED

- it is not sufficiently transparent

- not well integrated with prior knowledge (NLP)

- cannot distinguish causation from correlation

- assume stationarity

## ADVERSARIAL TRAINING

- can be fooled (it is not robust)

- is difficult to engineer with

## AUTO ML

# To go further

- books
  - ▶ I. Goodfellow, Y. Bengio & A. Courville, *Deep Learning*, MIT Press book, 2016  
<http://www.deeplearningbook.org/>
  - ▶ Gitbook [leonardoaraujosantos.gitbooks.io/artificial-intelligence/](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/)
- conferences
  - ▶ NIPS, ICLR, xCML, AIStats,
- Journals
  - ▶ JMLR, Machine Learning, Foundations and Trends in Machine Learning, machine learning survey <http://www.ml-surveys.com/>
- lectures
  - ▶ Deep Learning: Course by Yann LeCun at Collège de France in 2016  
[college-de-france.fr/site/en-yann-lecun/inaugural-lecture-2016-02-04-18h00.htm](http://college-de-france.fr/site/en-yann-lecun/inaugural-lecture-2016-02-04-18h00.htm)
  - ▶ Convolutional Neural Networks for Visual Recognition (Stanford)
  - ▶ deep mind (<https://deepmind.com/blog/>)
  - ▶ CS 229: Machine Learning at stanford Andrew Ng
- Blogs
  - ▶ Andrej Karpathy blog (<http://karpathy.github.io/>)
  - ▶ <http://deeplearning.net/blog/>
  - ▶ <https://computervisionblog.wordpress.com/category/computer-vision/>