

# Deep Learning

---

Romain Tavenard (Université de Rennes)

A course @EDHEC

# Administrative details

---

- 24 hours  
2 full days (mixed setting) + 4 half days (100% remote)
- Instructor: Romain Tavenard  
[romain.tavenard@univ-rennes2.fr](mailto:romain.tavenard@univ-rennes2.fr)
- Webpage:  
[rtavenar.github.io/teaching/deep\\_edhec/](http://rtavenar.github.io/teaching/deep_edhec/)
- Evaluation (April 14th, morning)
  - Multiple Choice Questionnaire (1h)

# Contents

---

- Intro to deep learning
- Fully-connected models
- Images & ConvNets
- Sequences
- (Generative models)

# Some slides are more important than others...

---

- Slides marked with this symbol:



Are considered basic knowledge required to pass the exams

# An introduction to deep learning

---

Romain Tavenard (Université de Rennes)

# What can deep learning do?

- Skin cancer image classification  
130 000 images  
Error rate : 28 % (human expert 34 %)



- ECG signal classification  
500 000 ECG  
Precision 92.6 %  
(human expert 80.0 %)



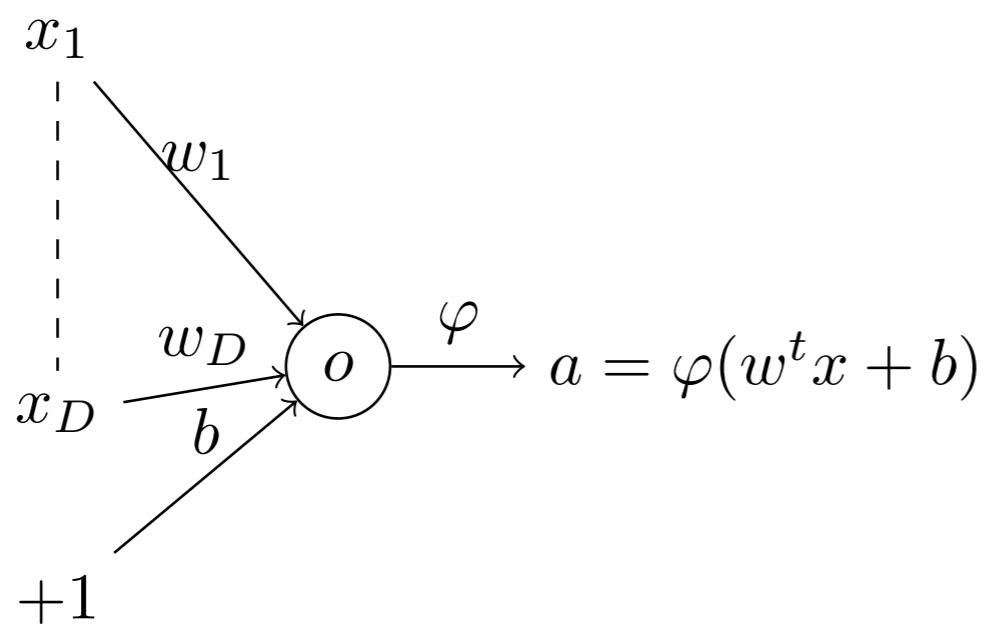
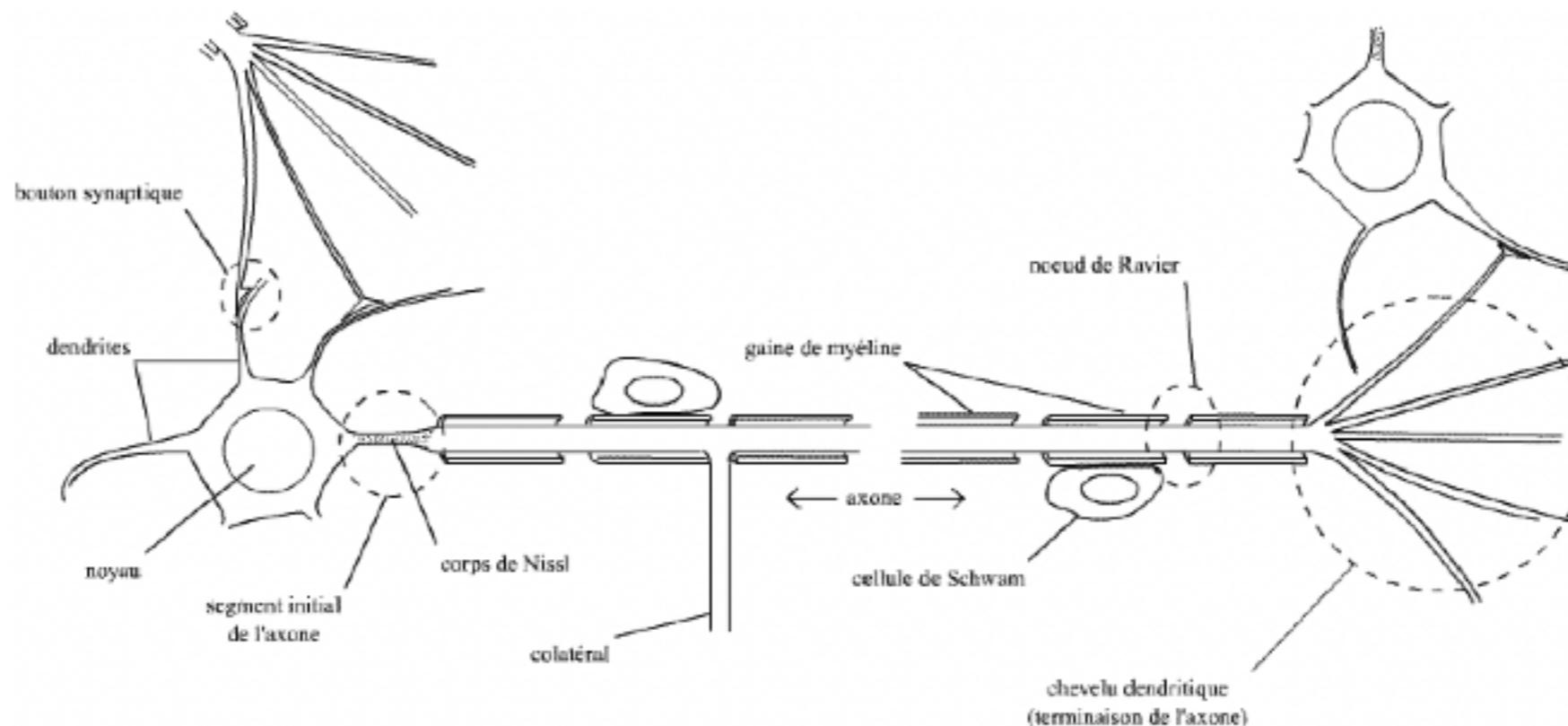
# A short history of deep learning

---

- Early stage: 1943 - 1969
  - 1943: a formal model for the neuron
  - 1947: perceptron as a learning machine
  - 1969: perceptrons can't do XOR
- Back in the game: 1985 - 1995
- A *de facto* standard in computer vision: 2009 - ?

# Formal neuron

(McCulloch & Pitts, 1943)



$\varphi$  activation function

$a$  neuron response

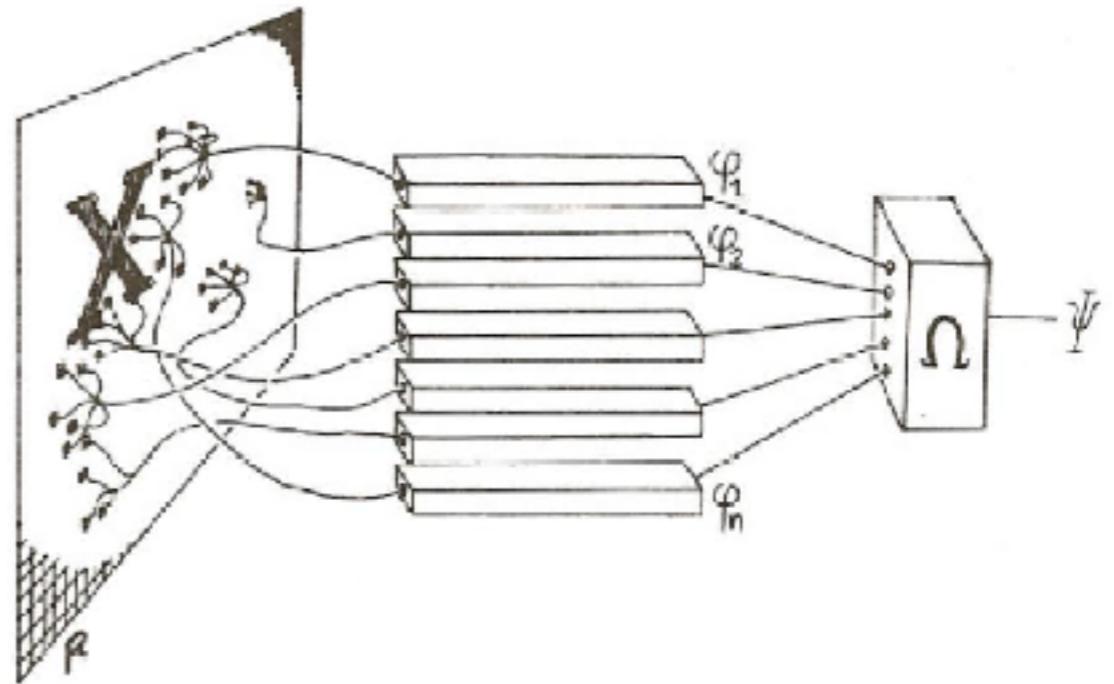
$w, b$  weight, bias

# Learning with the perceptron

(Rosenblatt, 1957)

---

- Problem statement
  - Given pairs of input-output data  $x_i, y_i$
  - Find  $w$  such that:
$$\forall i, \varphi(w^t x_i) \approx y_i$$



Source : (Minsky & Papert, 1969)

- To do so:
  - Gradient descent

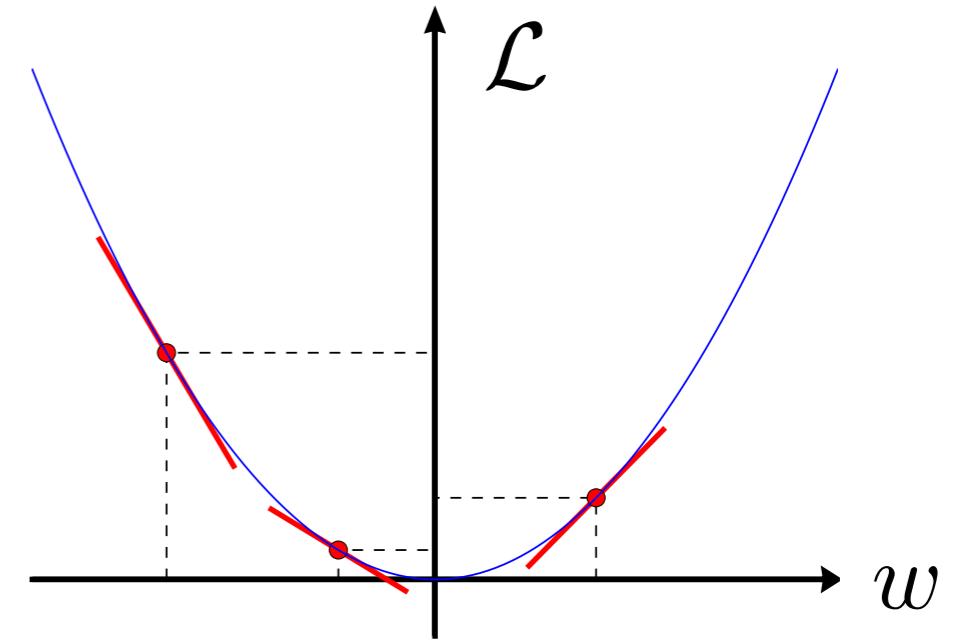


# Gradient descent

1. Pick a (differentiable) loss function to be minimized

$$\begin{aligned}\text{eg. } \mathcal{L}(w, \{x_i, y_i\}) &= \frac{1}{n} \sum_{i=1}^n \mathcal{L}_i(w, x_i, y_i) \\ &= \frac{1}{n} \sum_{i=1}^n (w^t x_i - y_i)^2\end{aligned}$$

2. Use gradient descent



---

**Algorithm 1:** (Batch) Gradient Descent

**Data:**  $\mathcal{D}$ : a dataset  
Initialize weights  
**for**  $e = 1..E$  **do**  
    // e is called an epoch  
    **for**  $(x_i, y_i) \in \mathcal{D}$  **do**  
        Compute prediction  $\hat{y}_i = h(x_i)$   
        Compute gradient  $\nabla_w \mathcal{L}_i$   
    **end**  
    Compute overall gradient  $\nabla_w \mathcal{L} = \frac{1}{n} \sum_i \nabla_w \mathcal{L}_i$   
    Update parameter  $w$  using  $\nabla_w \mathcal{L}$   
**end**

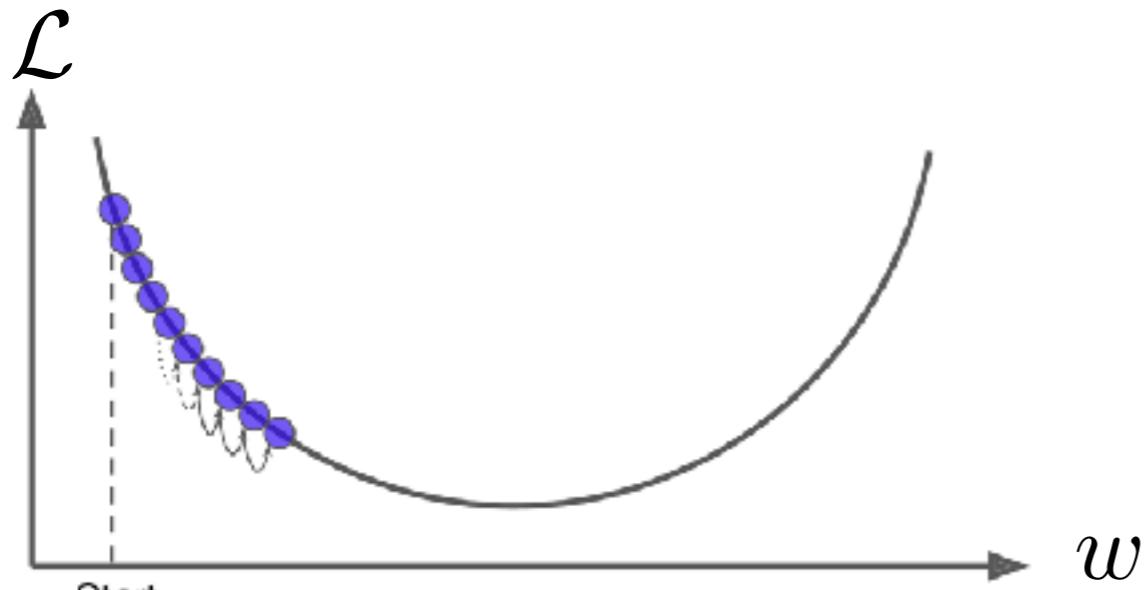
---

**Algorithm 2:** (Naive) Stochastic Gradient Descent

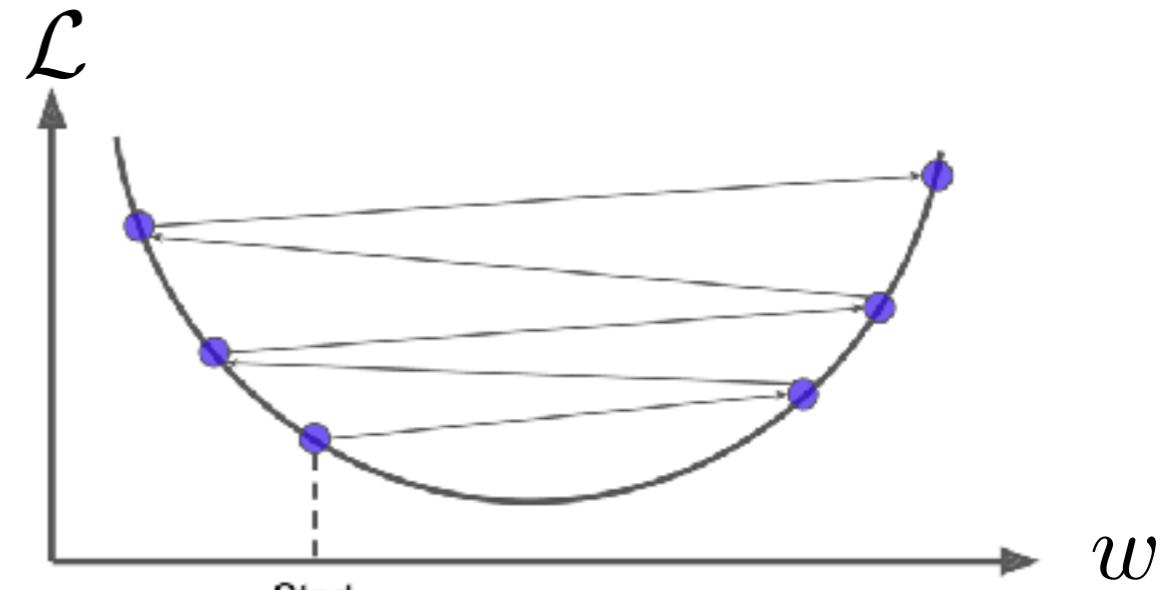
**Data:**  $\mathcal{D}$ : a dataset  
Initialize weights  
**for**  $t = 1..T$  **do**  
    // t is called an iteration  
    Draw  $(x_i, y_i)$  with replacement from  $\mathcal{D}$   
    Compute prediction  $\hat{y}_i = h(x_i)$   
    Compute gradient  $\nabla_w \mathcal{L}_i$   
    Update parameter  $w$  using  $\nabla_w \mathcal{L}_i$   
**end**



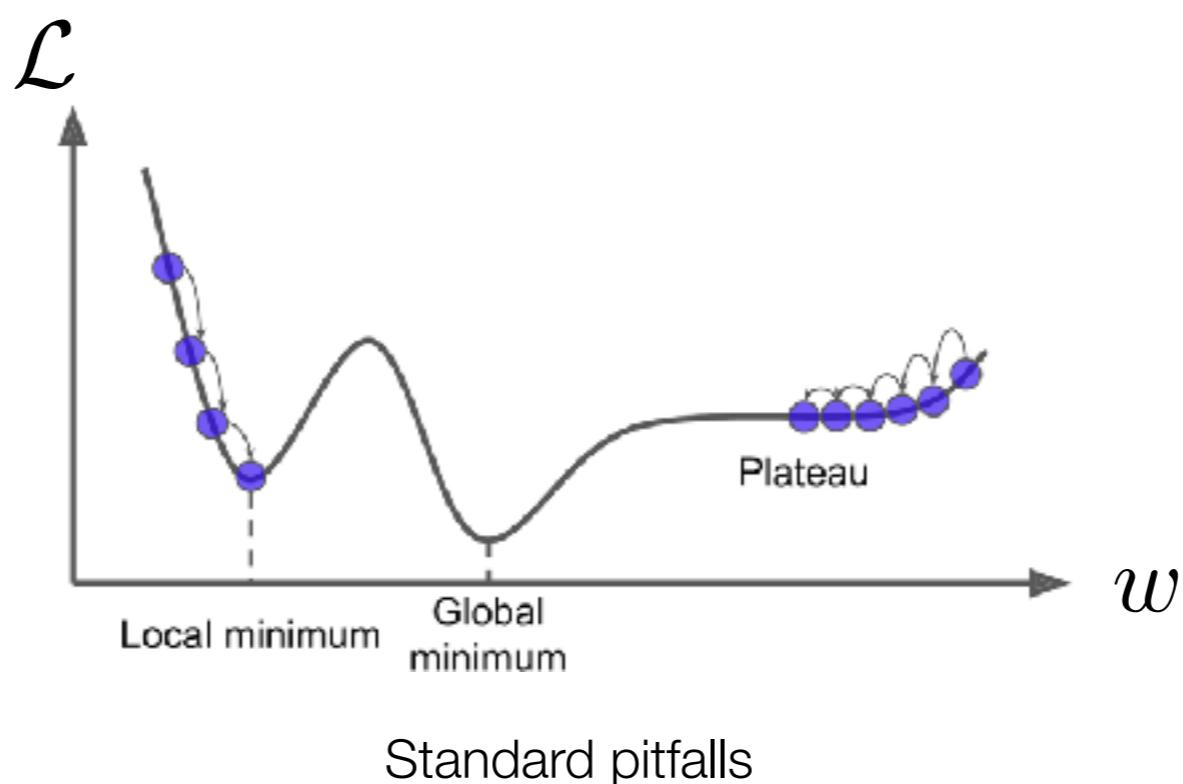
# Gradient descent in Real Life



Learning rate is too small



Learning rate is too large



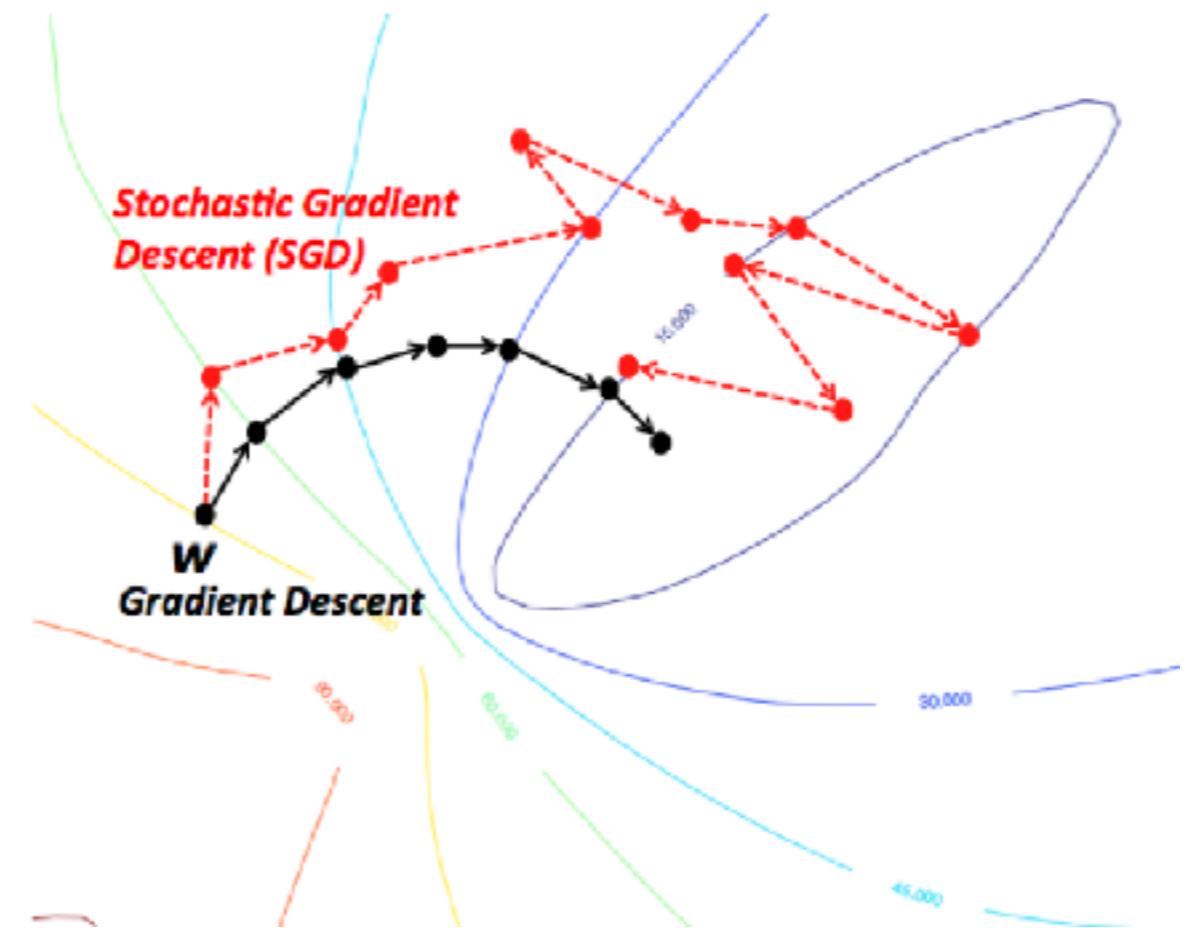
Standard pitfalls

Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow", A. Géron



# Stochastic Gradient Descent

- Cons
  - Subject to high variance
- Pros
  - Faster weight update (each sample, or each mini batch)
  - Escape local minima in non-convex settings



Source: [wikidocs.net/3413](http://wikidocs.net/3413)

# Stochastic Gradient Descent and its variants

---

**Algorithm 1:** (Batch) Gradient Descent

---

**Data:**  $\mathcal{D}$ : a dataset  
Initialize weights  
**for**  $e = 1..E$  **do**  
    // e is called an epoch  
    **for**  $(x_i, y_i) \in \mathcal{D}$  **do**  
        Compute prediction  $\hat{y}_i = h(x_i)$   
        Compute gradient  $\nabla_w \mathcal{L}_i$   
    **end**  
    Compute overall gradient  $\nabla_w \mathcal{L} = \frac{1}{n} \sum_i \nabla_w \mathcal{L}_i$   
    Update parameter  $w$  using  $\nabla_w \mathcal{L}$   
**end**

---

---

**Algorithm 2:** (Naive) Stochastic Gradient Descent

---

**Data:**  $\mathcal{D}$ : a dataset  
Initialize weights  
**for**  $t = 1..T$  **do**  
    // t is called an iteration  
    Draw  $(x_i, y_i)$  with replacement from  $\mathcal{D}$   
    Compute prediction  $\hat{y}_i = h(x_i)$   
    Compute gradient  $\nabla_w \mathcal{L}_i$   
    Update parameter  $w$  using  $\nabla_w \mathcal{L}_i$   
**end**

---

---

**Algorithm 3:** Mini-Batch Gradient Descent

---

**Data:**  $\mathcal{D}$ : a dataset  
Initialize weights  
**for**  $e = 1..E$  **do**  
    // e is called an epoch  
    **for**  $t = 1..n_b$  **do**  
        // t is called an iteration  
        **for**  $i = 1..m$  **do**  
            Draw  $(x_i, y_i)$  without replacement from  $t$ -th minibatch of  $\mathcal{D}$   
            Compute prediction  $\hat{y}_i = h(x_i)$   
            Compute gradient  $\nabla_w \mathcal{L}_i$   
        **end**  
        Compute gradient for the  $t$ -th minibatch  $\nabla_w \mathcal{L}_{(t)} = \frac{1}{m} \nabla_w \mathcal{L}_i$   
        Update parameter  $w$  using  $\nabla_w \mathcal{L}_{(t)}$   
    **end**  
**end**

---

# Improvements to stochastic gradient descent

---

- Averaging updates (mini-batches)
  - Polyak and Juditsky, 1992
  - SAGA (2014)
- Convergence acceleration
  - Momentum (Polyak, 1964)
  - Nesterov (1983)
- Step size adaptation
  - RMSProp (2012)
  - Adam (2015)

# SGD variants: a focus on Adam

---

- Adam uses ideas from
  - Momentum
  - AdaGrad

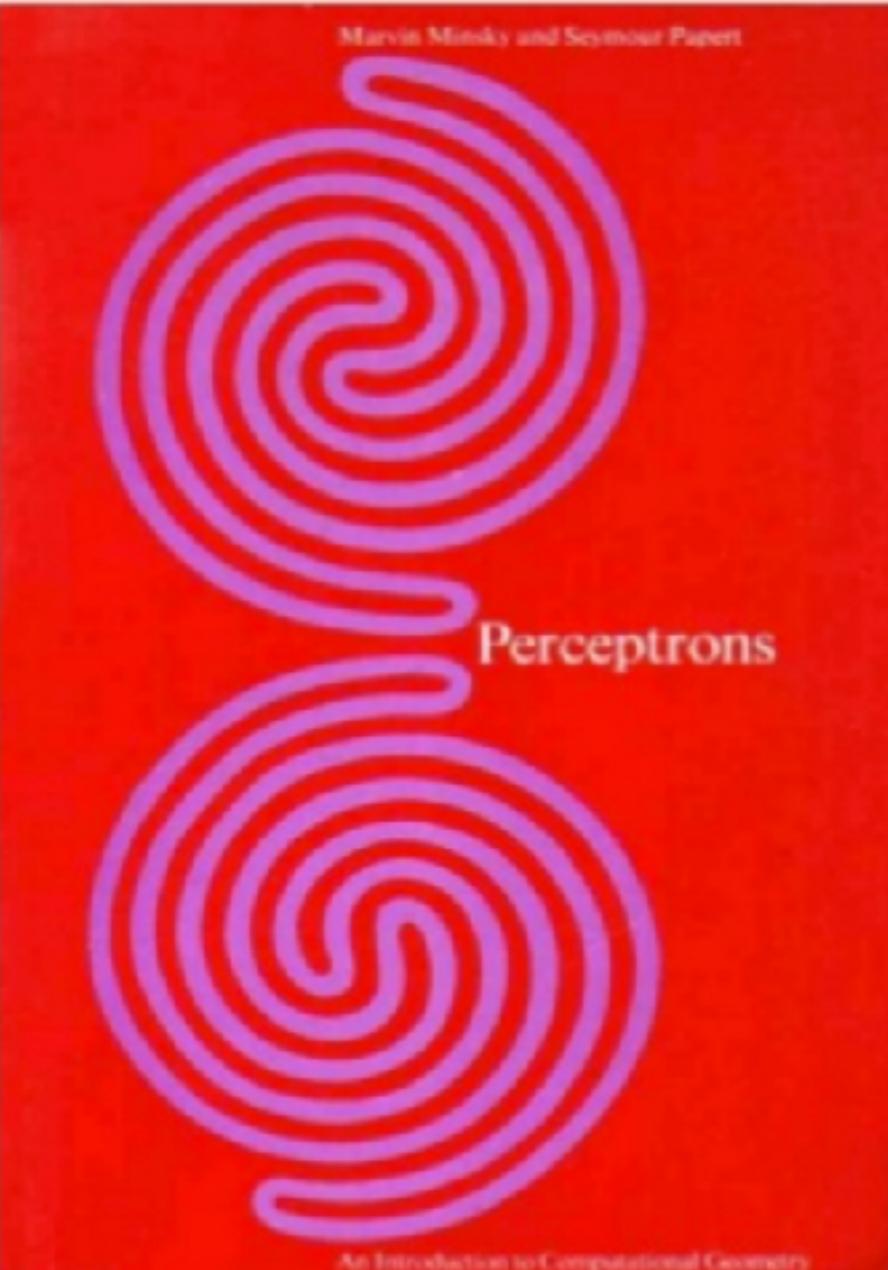
$$\mathbf{m}^{(t+1)} \propto \beta_1 \mathbf{m}^{(t)} + (1 - \beta_1) \nabla_w \mathcal{L}$$

$$\mathbf{s}^{(t+1)} \propto \beta_2 \mathbf{s}^{(t)} + (1 - \beta_2) \nabla_w \mathcal{L} \otimes \nabla_w \mathcal{L}$$

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \rho \mathbf{m}^{(t+1)} \oslash \sqrt{\mathbf{s}^{(t+1)} + \epsilon}$$

# First NN winter (Minsky & Papert, 1969)

1969: Perceptrons can't do XOR!

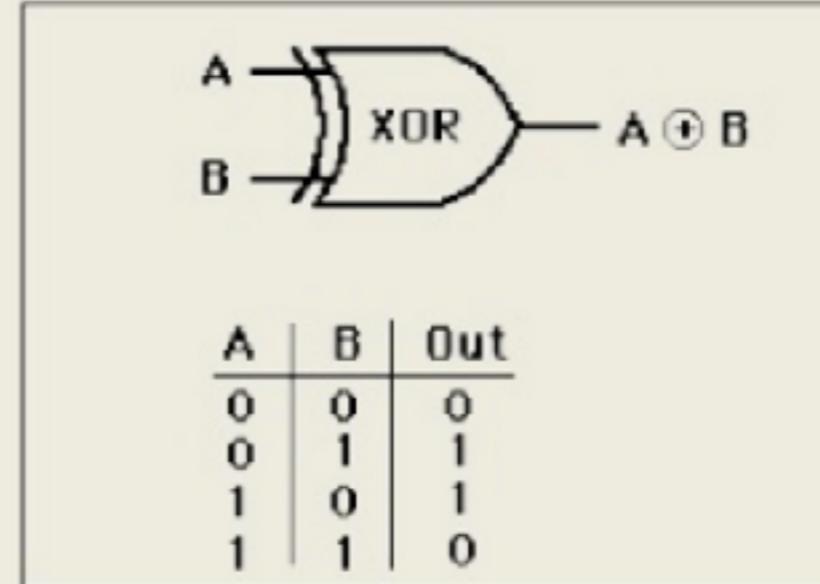


Marvin Minsky and Seymour Papert

Perceptrons

An Introduction to Computational Geometry

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



A logic gate diagram for a XOR operation. It has two inputs, A and B, entering a single-gate symbol labeled "XOR". The output is labeled  $A \oplus B$ .

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

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



Minsky & Papert

# A short history of deep learning

---

- Early stage: 1943 - 1969
- Back in the game: 1985 - 1995
  - 1985: Multilayer perceptron
  - 1989: Universal approximation theorem
  - 1989: LeCun's Convolutional Neural Networks
  - 1995: Recurrent Neural Networks
- A *de facto* standard in computer vision: 2009 - ?

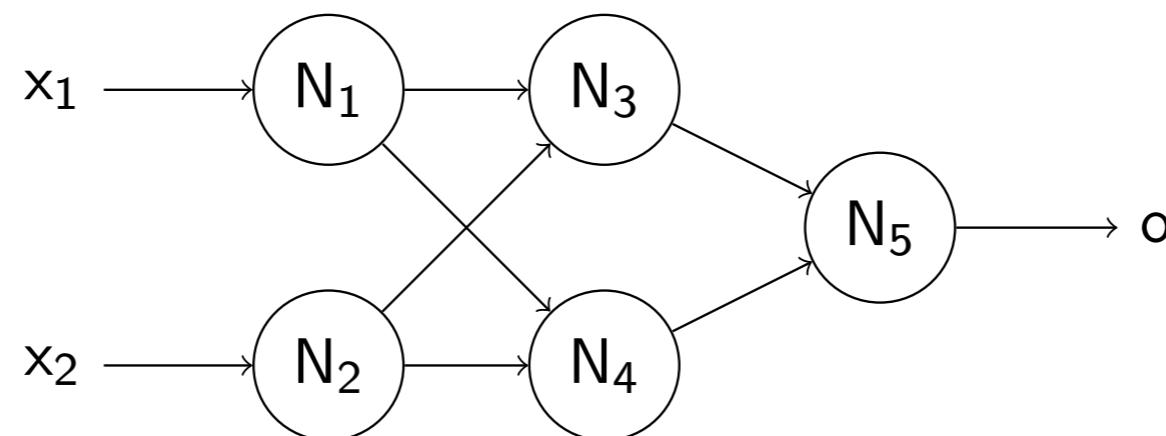
# Multilayer perceptron

(Rumelhart, Hinton & Williams, 1985)



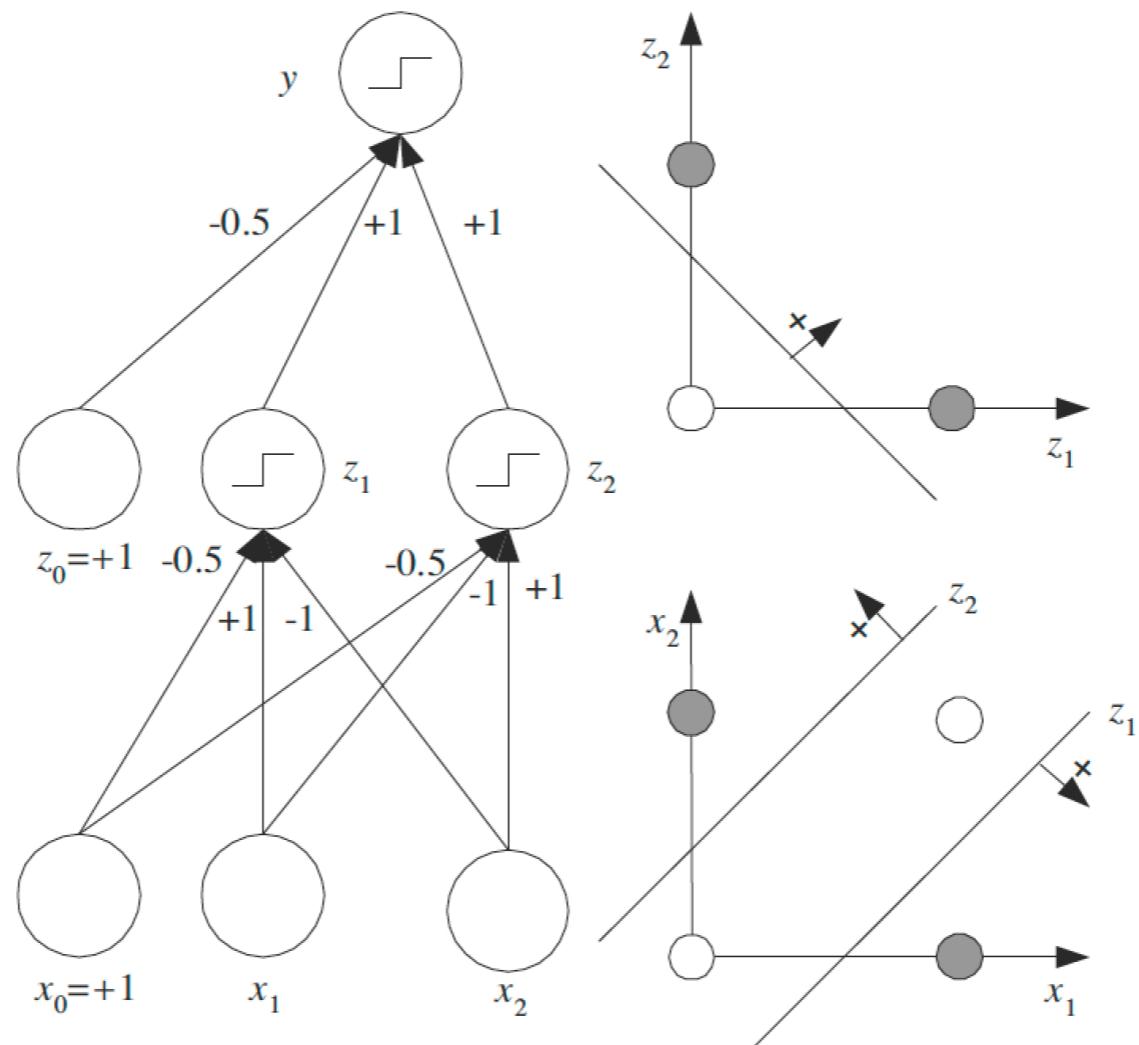
## Definition

A Multilayer perceptron is an acyclic graph of neurons, where neurons are structured in successive layers, beginning by an input layer and finishing with an output layer.



# Back to XOR

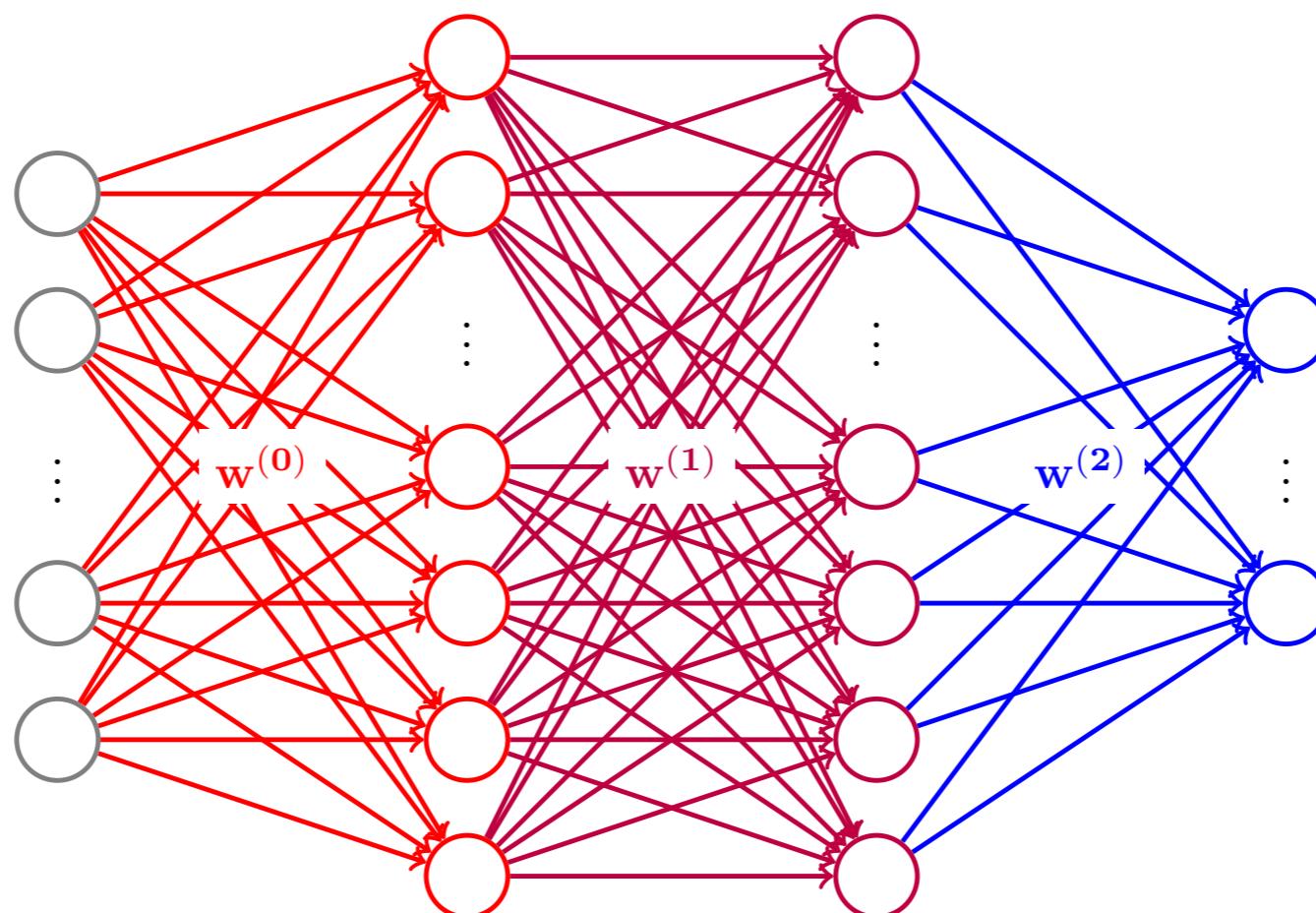
---



Interested in playing with MLPs ?  
<http://playground.tensorflow.org/>

# Optimizing multi-layer perceptron parameters

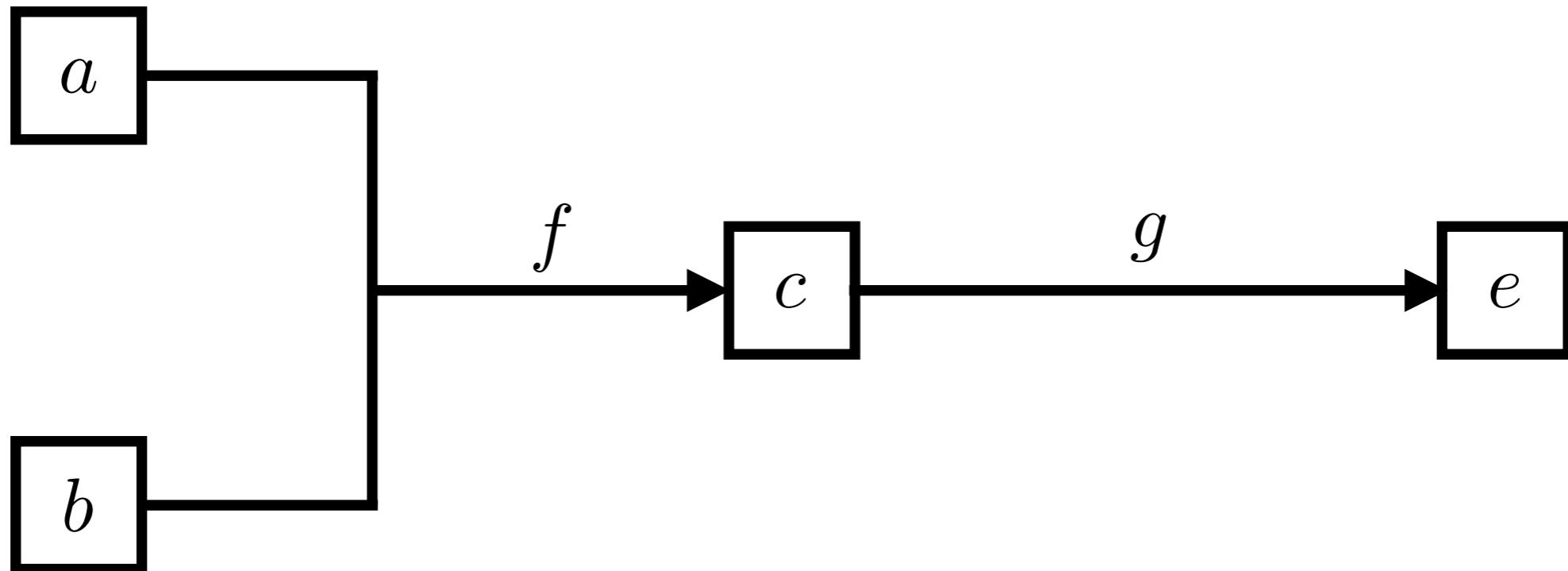
- Who wants to compute gradients by hand for such networks (and deeper ones)?



$$\hat{\mathbf{y}} = \varphi \left[ \mathbf{w}^{(2)} \varphi \left( \mathbf{w}^{(1)} \varphi(\mathbf{w}^{(0)} \mathbf{x} + b^{(0)}) + b^{(1)} \right) + b^{(2)} \right]$$

# Optimizing multi-layer perceptron parameters

## Automatic differentiation to the rescue!

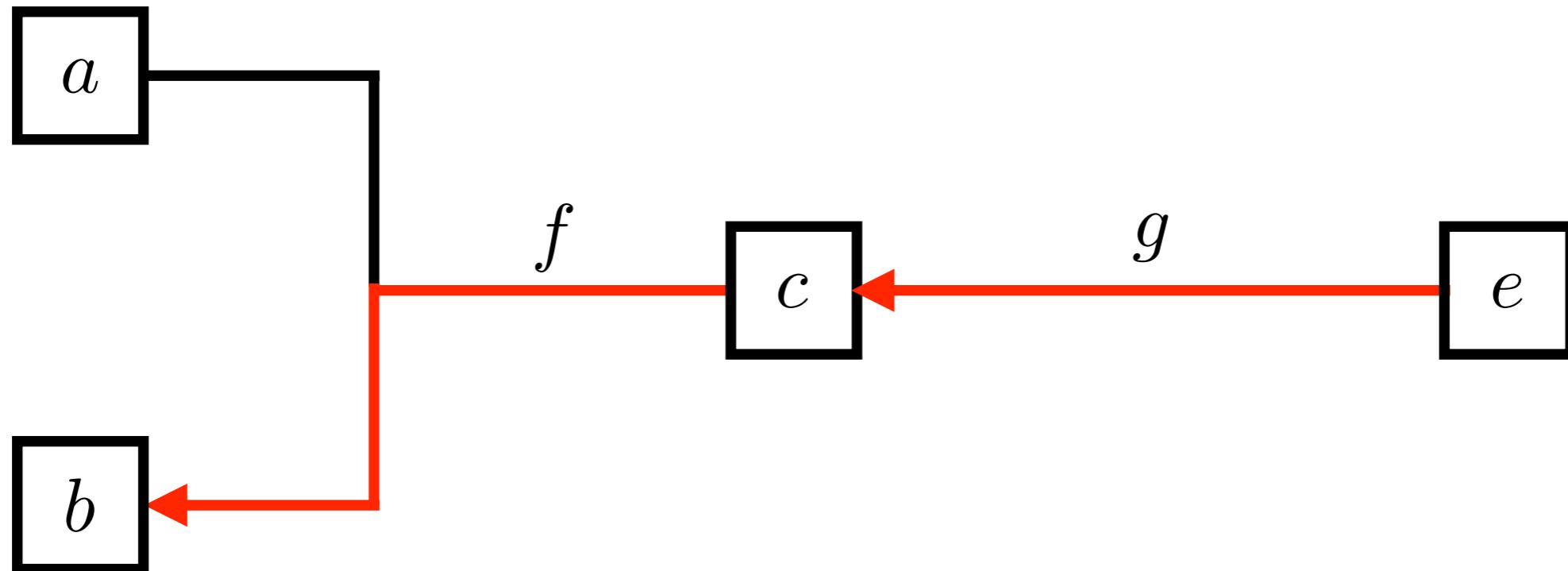


$$c = f(a, b)$$

$$e = g(c)$$

# Optimizing multi-layer perceptron parameters

## Automatic differentiation to the rescue!



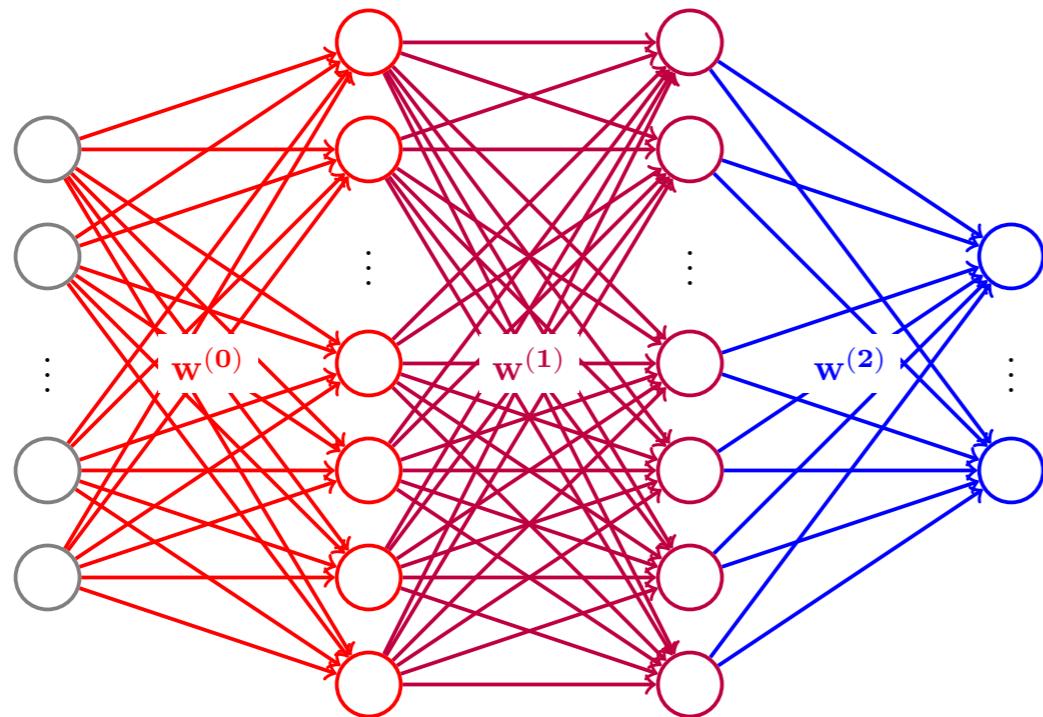
$$c = f(a, b)$$

$$e = g(c)$$

$$\frac{\partial e}{\partial b} = \underbrace{\left. \frac{\partial e}{\partial c} \right|_{c=c_0}}_{g'(c_0)} \cdot \left. \frac{\partial c}{\partial b} \right|_{b=b_0}$$



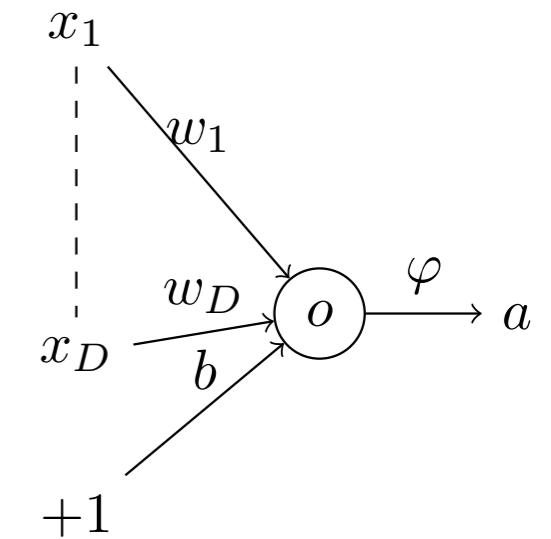
# Neural networks and back-propagation



$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial w^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial w^{(1)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$



$$\frac{\partial a^{(l)}}{\partial o^{(l)}} = \varphi'(o^{(l)})$$

$$\frac{\partial o^{(l)}}{\partial a^{(l-1)}} = w^{(l-1)}$$

# Neural networks and back-propagation: losses



$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$

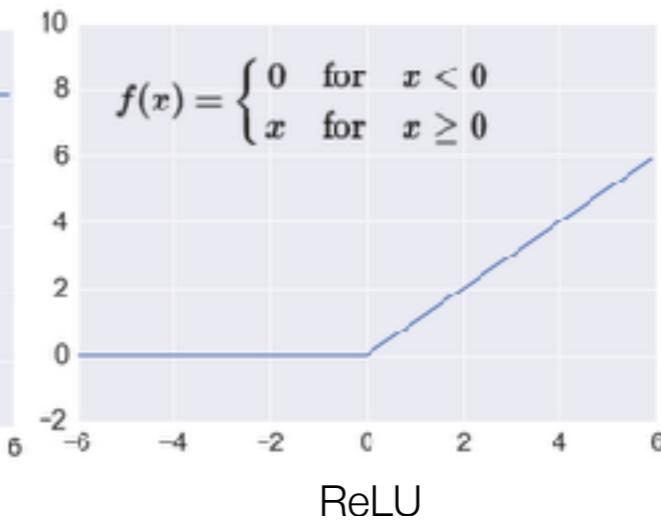
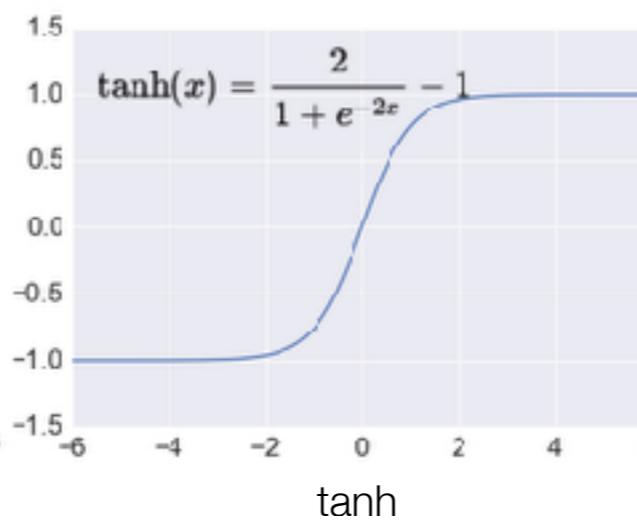
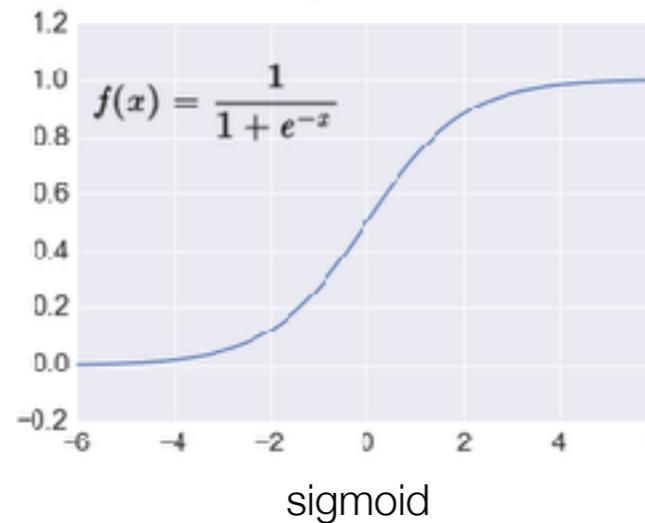
- Requirement
  - $\mathcal{L}$  should be differentiable wrt. to the net's output
- Standard losses
  - Mean Squared Error (MSE) for regression
$$\mathcal{L}(x_i, y_i; \theta) = (f_\theta(x_i) - y_i)^2$$
  - Cross-entropy for classification
$$\mathcal{L}(x_i, y_i; \theta) = -\log P_\theta(y = y_i | x_i)$$

# Neural networks and back-propagation: activation functions



$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$
$$\frac{\partial a^{(l)}}{\partial o^{(l)}} = \varphi'(o^{(l)})$$

- Important features (more on that later)
  - $\varphi$  should be differentiable almost everywhere
  - Non-linearities
  - Some linear regime
- Examples

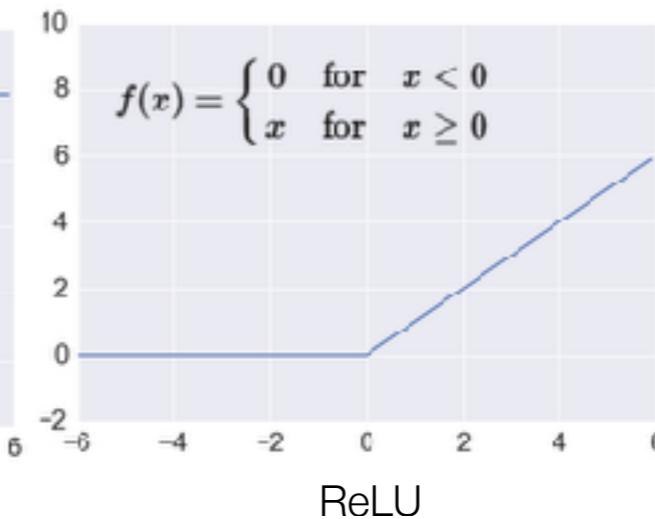
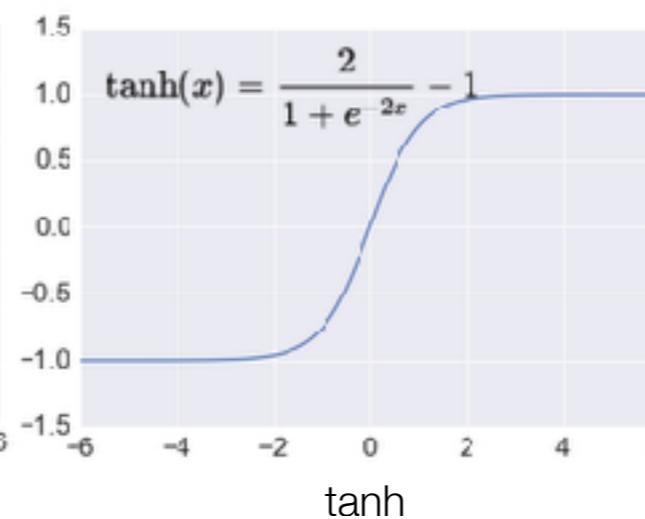
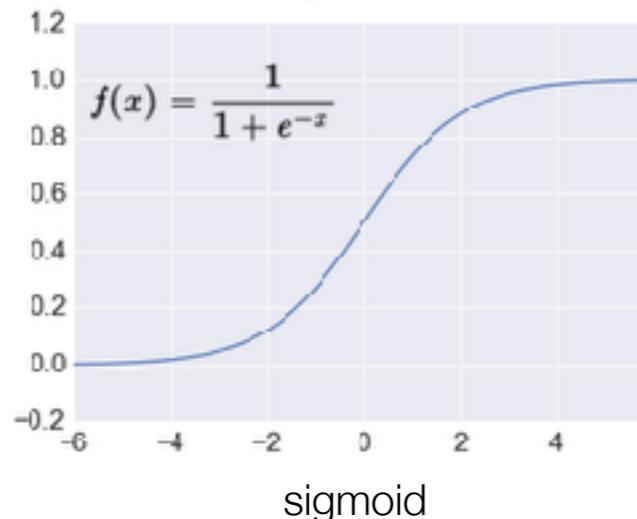


# Neural networks and back-propagation: activation functions: the reign of ReLU



$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$
$$\frac{\partial a^{(l)}}{\partial o^{(l)}} = \varphi'(o^{(l)})$$

- ReLU has become the default choice over time
- 2 main reasons:
  - cheap to compute (both ReLU and its derivative)
  - vanishing gradients phenomenon



# Neural networks and back-propagation: link with keras implementation

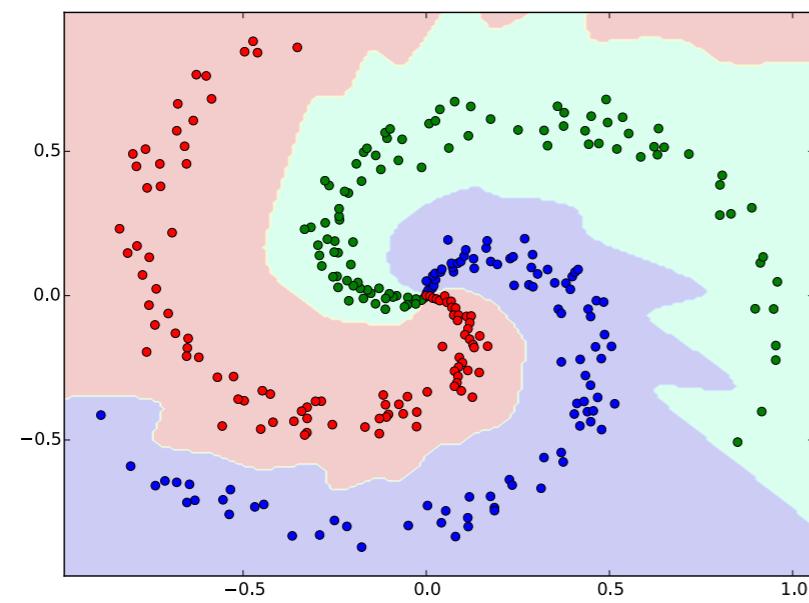


- In keras, these considerations have practical impact:
  - Model structure:
    - Input layer dimension is the number of features in the dataset
    - Output layer has as many units as columns in y
  - Output layer activation:
    - Binary classification: "sigmoid"
    - Multiclass classification: "softmax"
  - Loss function:
    - Binary classification: "binary\_crossentropy"
    - Multiclass classification: "categorical\_crossentropy"
    - Regression: "mse"

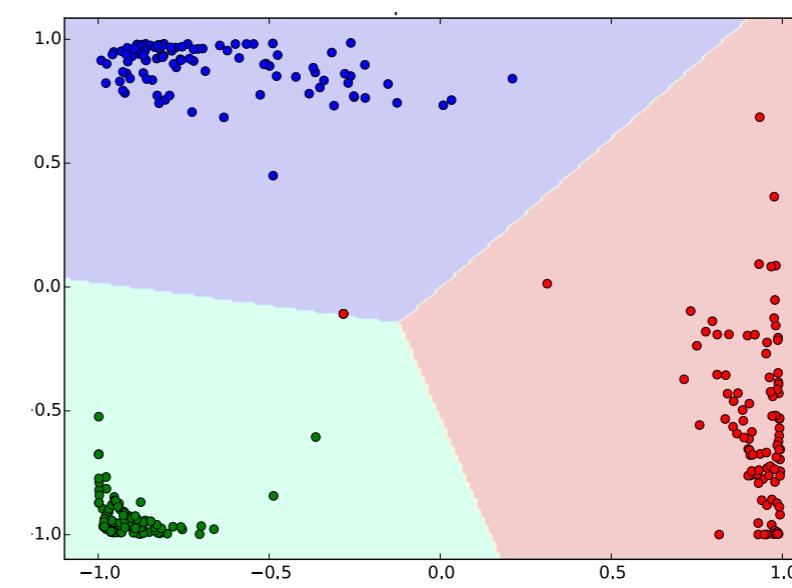
# End-to-end learning

---

- Classification using MLP
  - Hidden layers: non-linear transformations
  - Last layer: logistic regression
- Example with a 3-hidden-layer net (last layer with 2 units)



Input space



Last hidden layer

# Universal approximation theorem (Cybenko, 1989)

---

- Under reasonable assumptions on the activation function to be used\*
- For any continuous function on a compact  $g$  and any precision threshold  $\epsilon$
- There exists a 1-hidden-layer MLP with a finite number of neurons that can approximate  $g$  at level  $\epsilon$

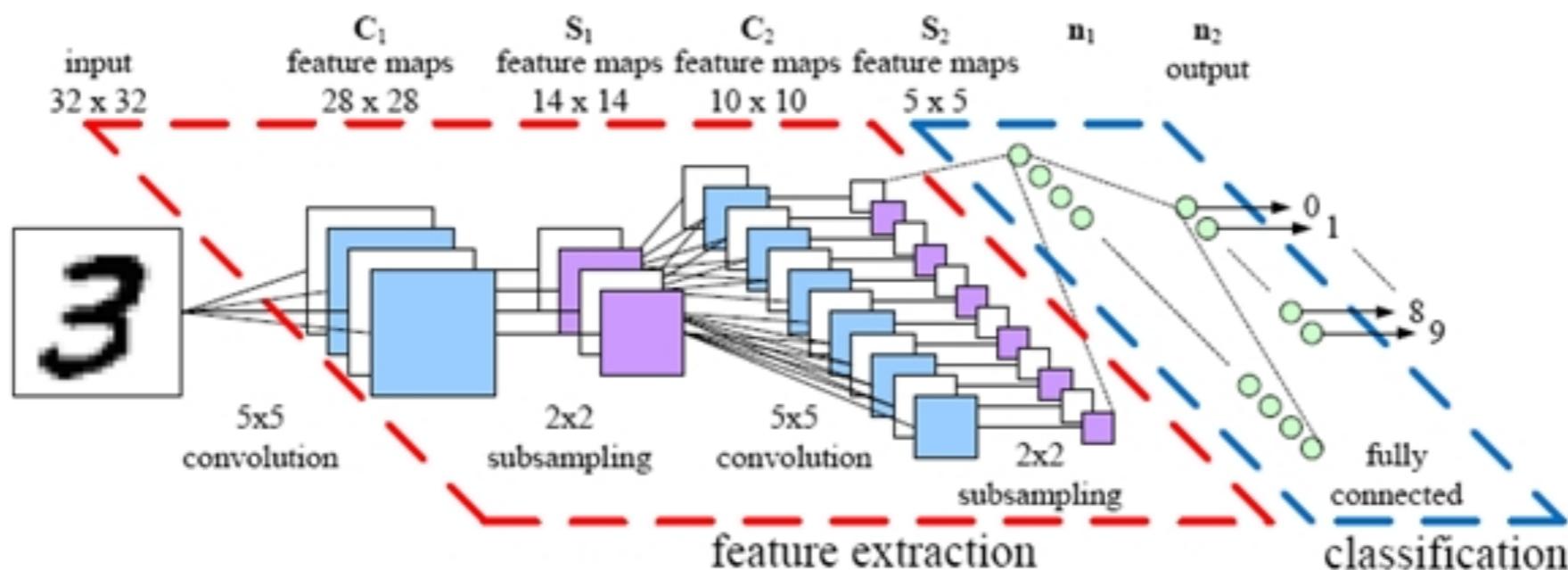
\*Non-constant, bounded, monotonically increasing, continuous  
also holds for some unbounded functions like ReLU, cf. [Somoda & Murata, 2015]

# Convolutional neural networks (CNN)

(LeCun, 1989)

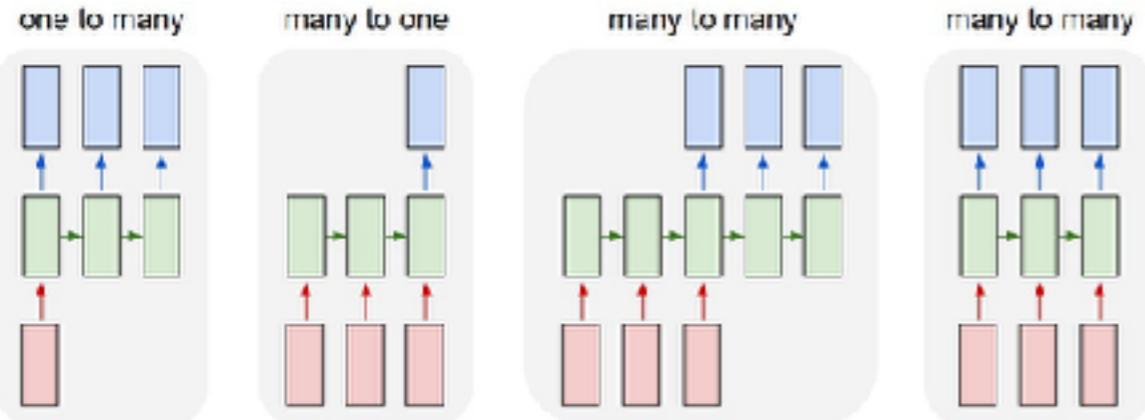
3	8	6	9	6	4	5	3	8	4	5	2	3	8	4	8
1	5	0	5	9	7	4	1	0	3	0	6	2	9	9	4
1	3	6	8	0	7	1	6	8	9	0	3	8	3	7	7
8	4	4	1	2	9	4	1	1	0	6	4	5	0	1	1
7	2	7	3	1	4	0	5	0	6	8	7	6	8	9	9
4	0	6	1	9	2	1	3	9	4	4	5	6	6	1	7
2	8	6	9	7	0	9	1	6	2	8	3	6	4	9	5
8	6	8	7	8	8	6	9	1	7	6	0	9	6	7	0

MNIST dataset  
10 classes  
60,000 images



# Neural Nets & sequences: Recurrent Neural Networks (RNN)

Warning: these examples are more recent than 1995!



### PANDARUS:

Alas, I think he shall be come approached and the day  
When little strain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

### Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

### DUKE VINCENTIO:

Well, your wit is in the care of side and that.

### Second Lord:

They would be ruled after this chamber, and  
my fair nues begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Sample text generated by a RNN  
trained on Shakespeare words

For  $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\text{Sch}_{fppf}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section ?? and the fact that any  $U$  affine, see Morphisms, Lemma ???. Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $\text{Sh}(G)$  such that  $\text{Spec}(R) \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,x}$  is a scheme where  $x, x', s'' \in S'$  such that  $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}|_U$  is a covering of  $X'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X/S}$  for  $i > 0$  and  $\mathcal{F}_p$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $C$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\widetilde{M}^{\bullet} = \mathcal{I}^{\bullet} \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)^{opp}_{fppf}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \hookrightarrow (U, \text{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result for prove any open covering follows from the less of Example ???. It may replace  $S$  by  $X_{\text{spacess,\'etale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{\text{Zar}}$ , see Descent, Lemma ???. Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

Sample LaTeX generated by a RNN  
trained on a book of algebraic geometry

# Caltech 101: the second winter (2004)

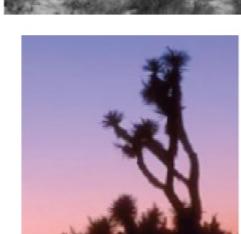
---

- 101 classes
- 30 training images per class
- NN are bad competitors here
  - Dataset is too small

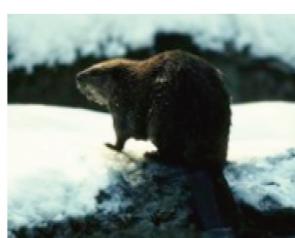
Anchor



Joshua Tree



Beaver



Lotus



Water Lily



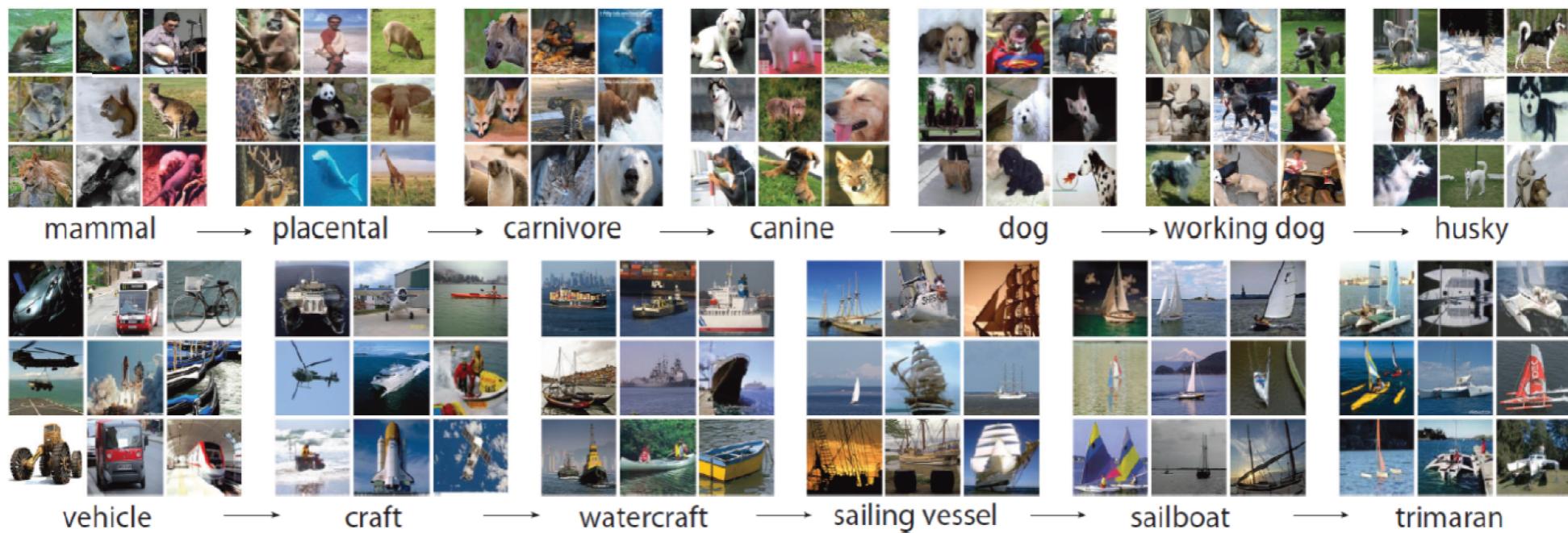
# A short history of deep learning

---

- Early stage: 1943 - 1969
- Back in the game: 1985 - 1995
- A *de facto* standard in computer vision: 2009 - ?
  - 2009: GPUs for deep learning
  - 2012: ImageNet & AlexNet
  - 2014: Inception
  - 2016: Residual Networks
  - ...

# ImageNet & LSVRC (2012)

- ImageNet
  - 15M images
  - 22k classes
- LSVRC
  - Subset of ImageNet (1.2M images, 1k classes)



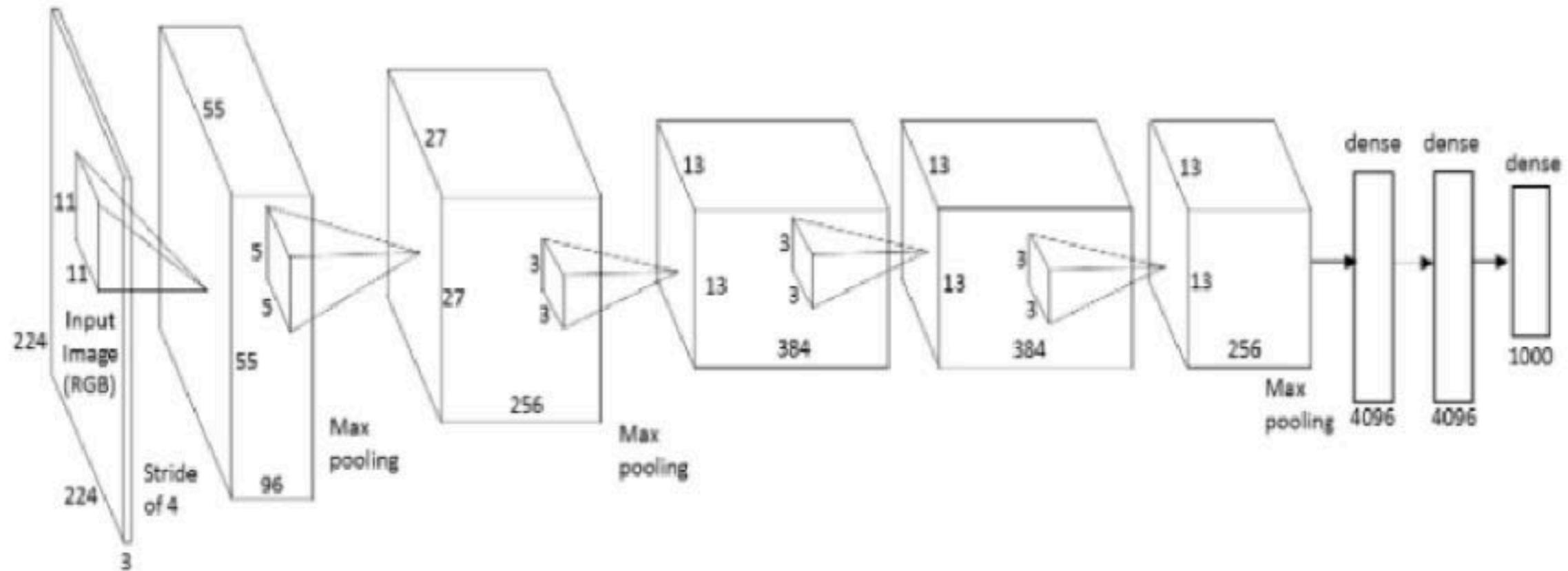
# A drastic improvement on performances (LSVRC)

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

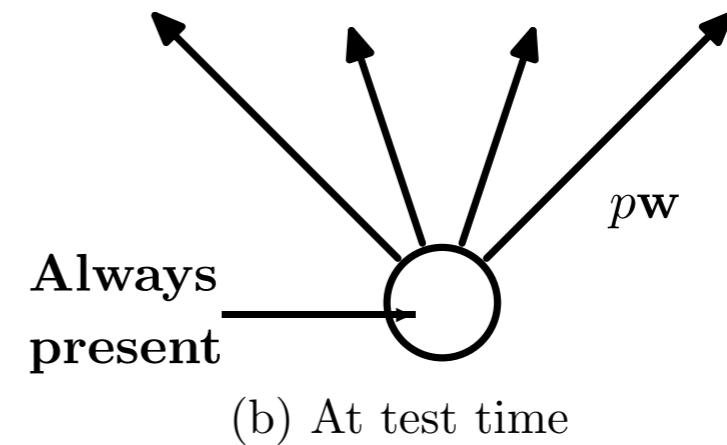
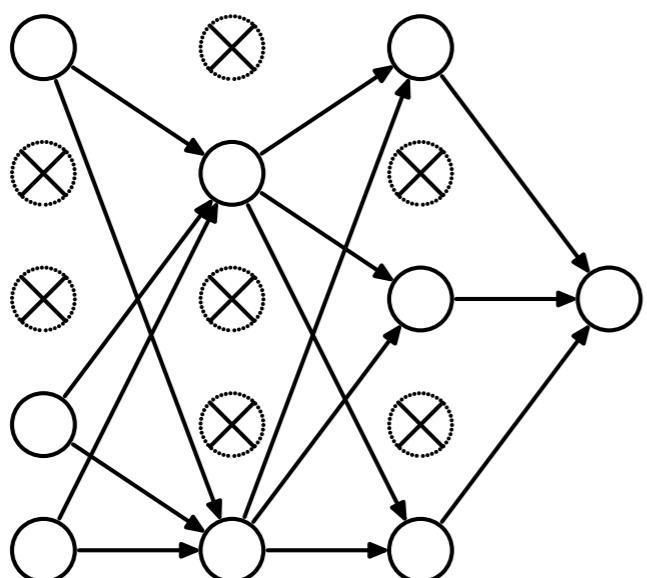
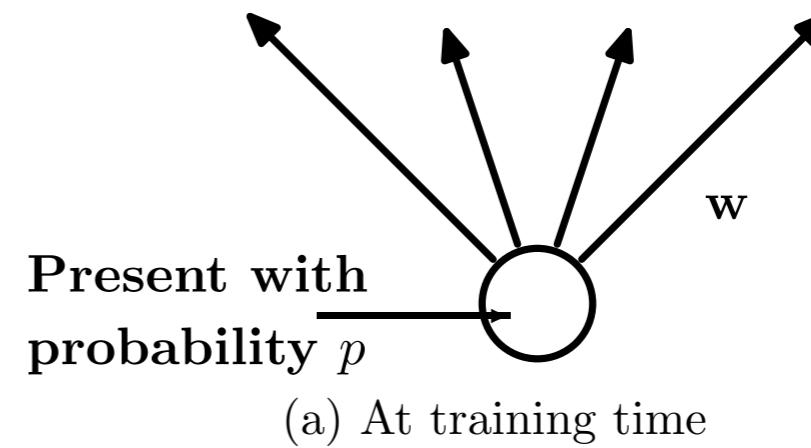
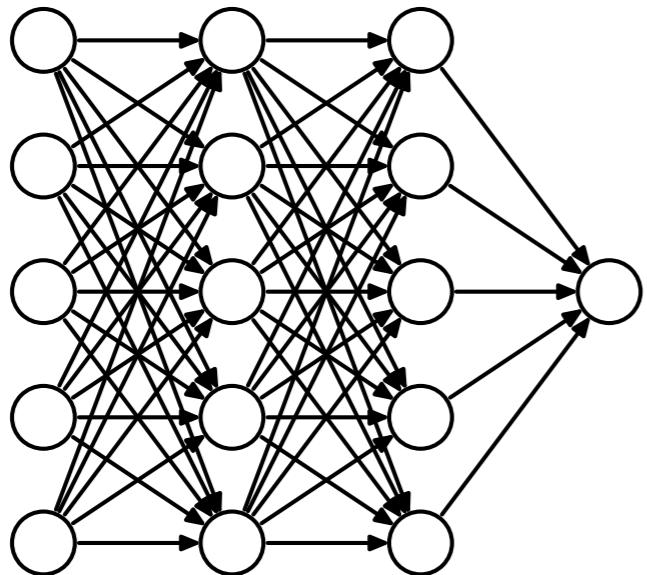
deep learning

# AlexNet (2012)



- Error rate : 15%
- 60M parameters
- 2 GPUs – 6 days
- Regularization
  - Data augmentation
  - Dropout
  - L2

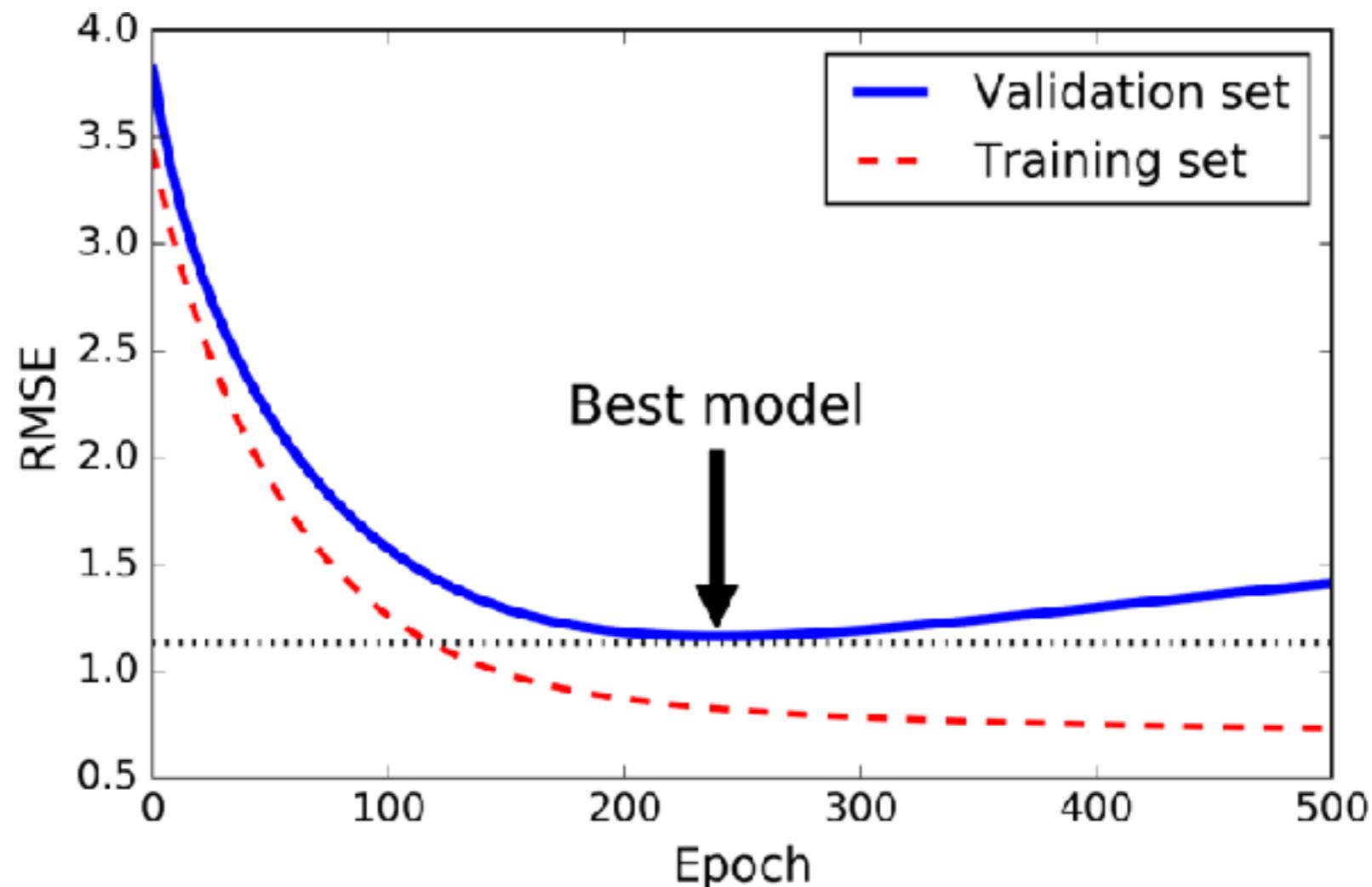
# Regularization: Dropout



Images from [Srivastava et al., 2014]



# Regularization: Early Stopping



# Model selection: How to test hyper-parameter configs?

---



- Many hyper-parameters
  - structure: #layers, #units, etc.
  - estimation: optimizer, learning rate
- Good practice
  - pick a "reasonable" optimizer (eg. Adam)
  - pick structure based on hold-out data
    - advanced strategies exist  
(eg. HyperBand [Li *et al.* 2018])

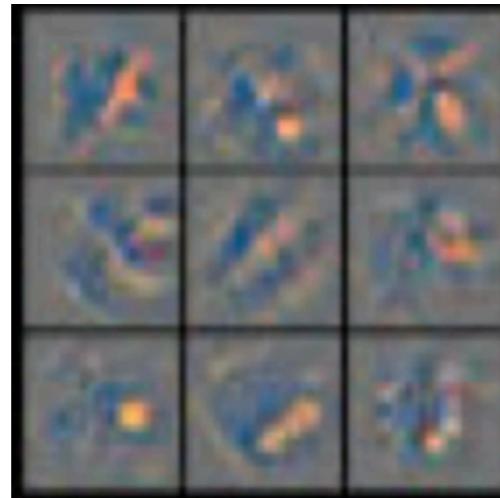
# What does AlexNet learn?

---

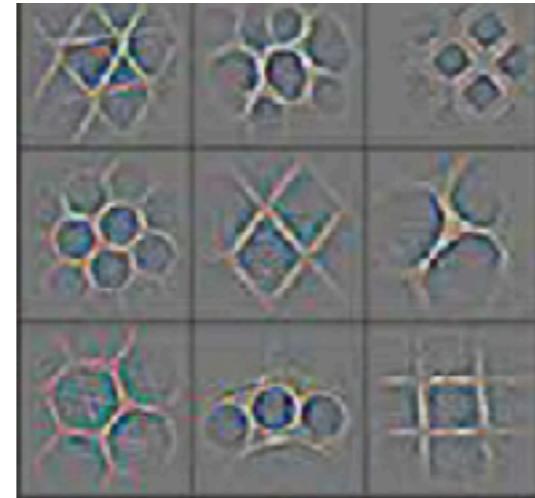
Sample convolution filters learned:



Layer 1



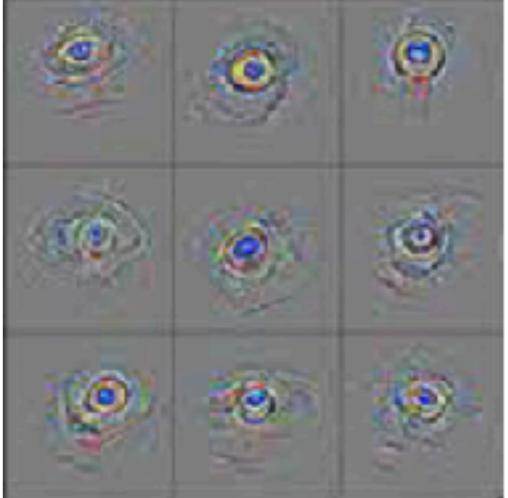
Layer 2



Layer 3



Layer 4

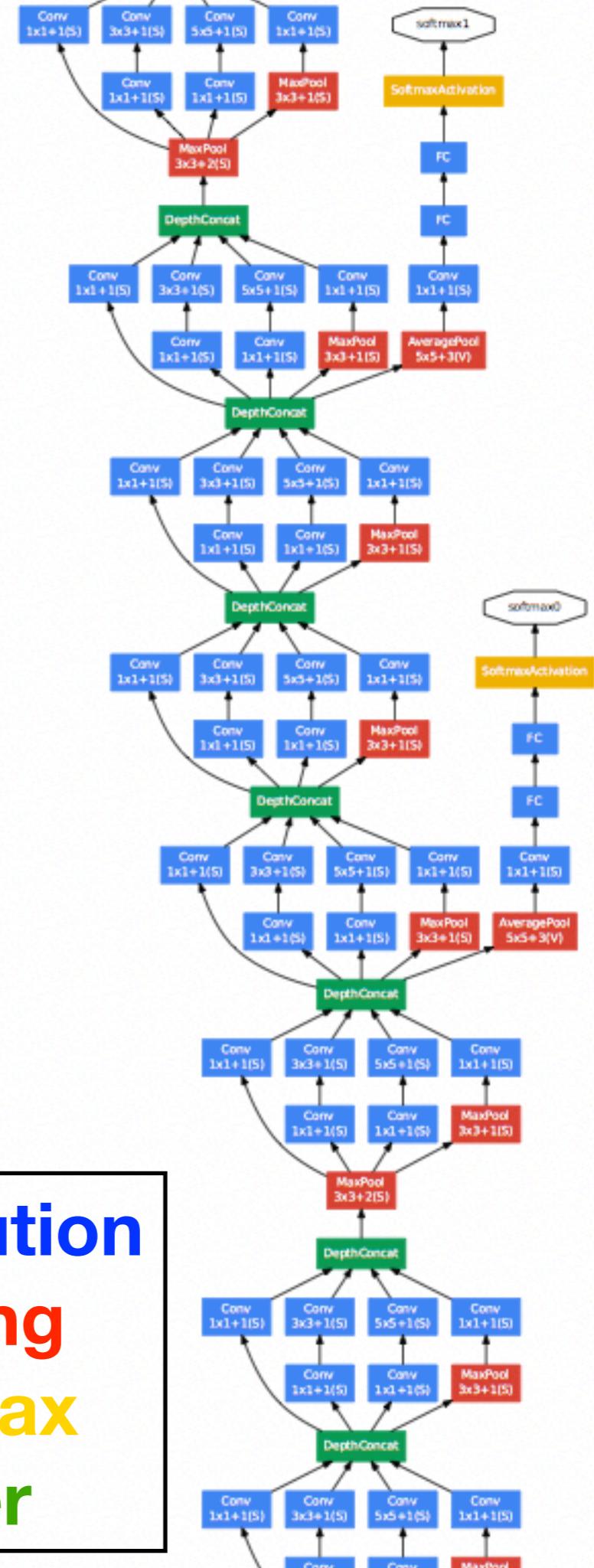


Layer 5

# From 15% to 7% : Inception (2014)

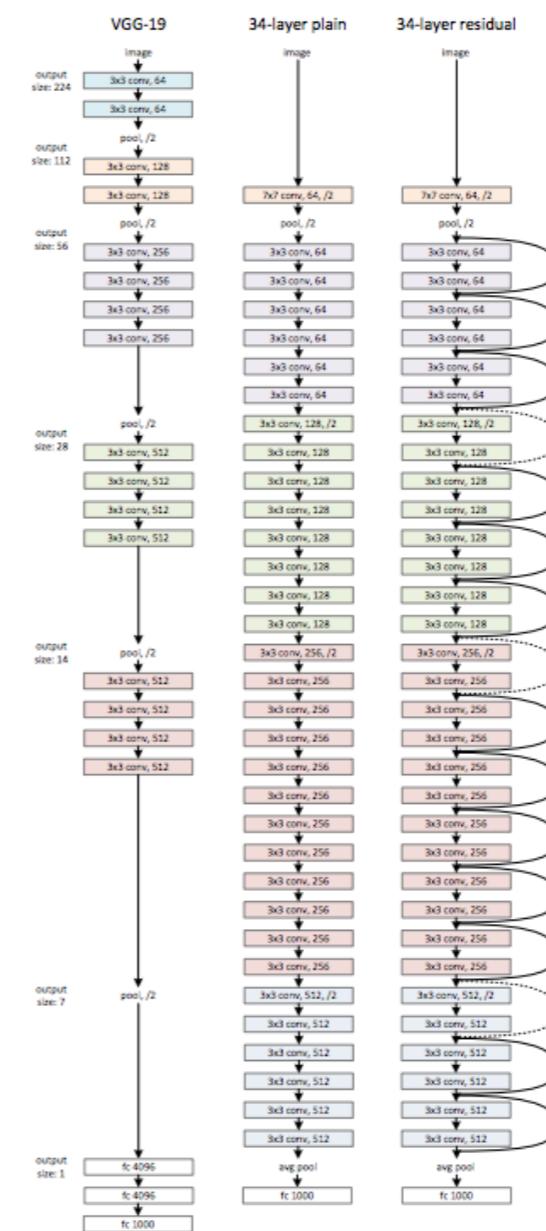
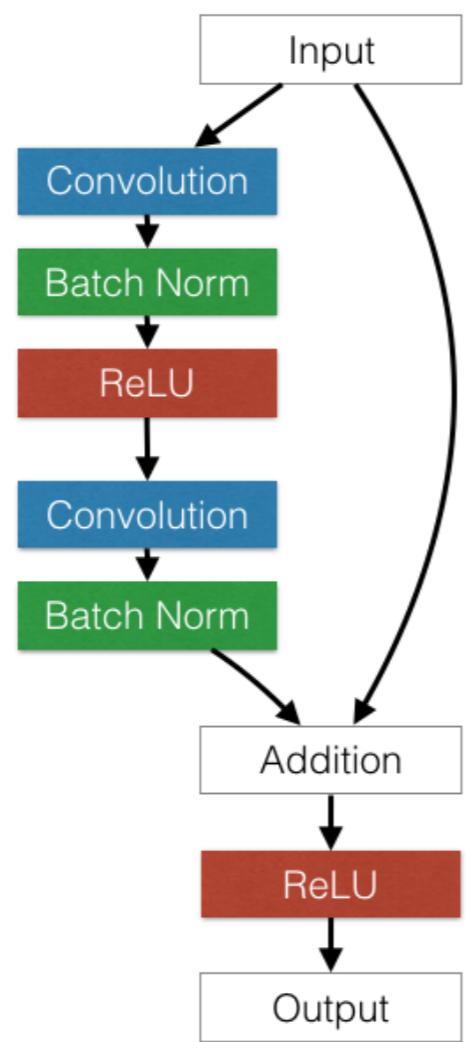
- *Network of networks*
- ~100 blocks, 22 layers
  - Several convolutions per layer
- 5 million parameters
- *Intermediate classification outputs*

Convolution  
Pooling  
Softmax  
Other



# From 7% to 3% : Residual Networks (aka ResNets)

- Aims at facing the *vanishing gradient* effect



# Why such sudden changes?

---

- Big data (ImageNet & co)
- Big infrastructures (GPU)
- Optimization
  - Algorithms
  - *Tricks* (initialization, regularization, fighting vanishing gradients)
- Automatic differentiation libraries (tensorflow, pytorch, ...)

# A few hot topics

## A - Adversarial examples

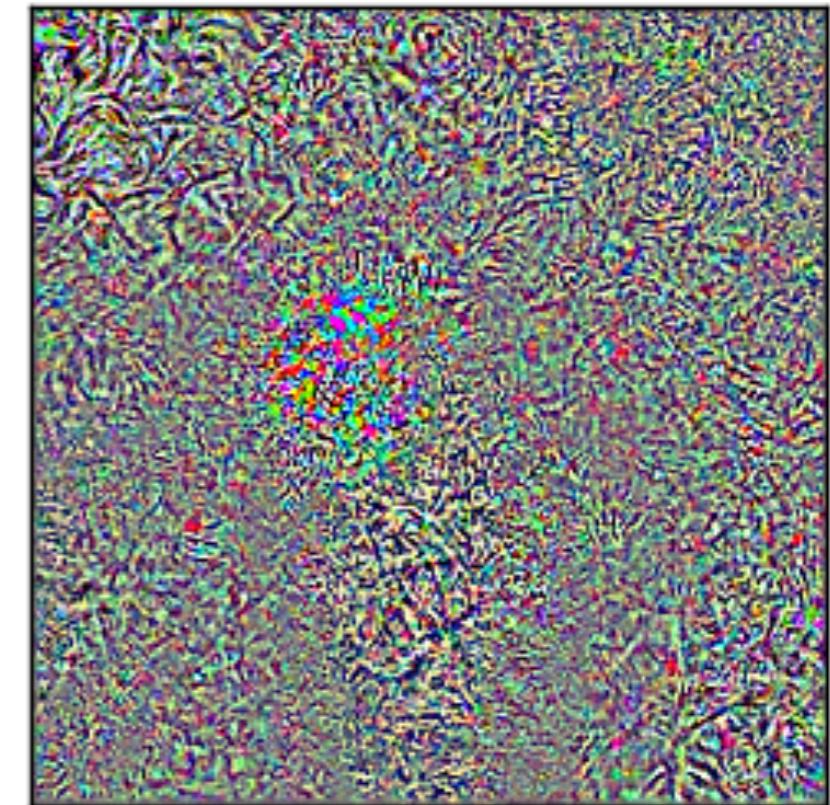
---



Original image:  
Ara : 97%



Transformed image:  
Ara : 0%  
Bookshelf : 99%

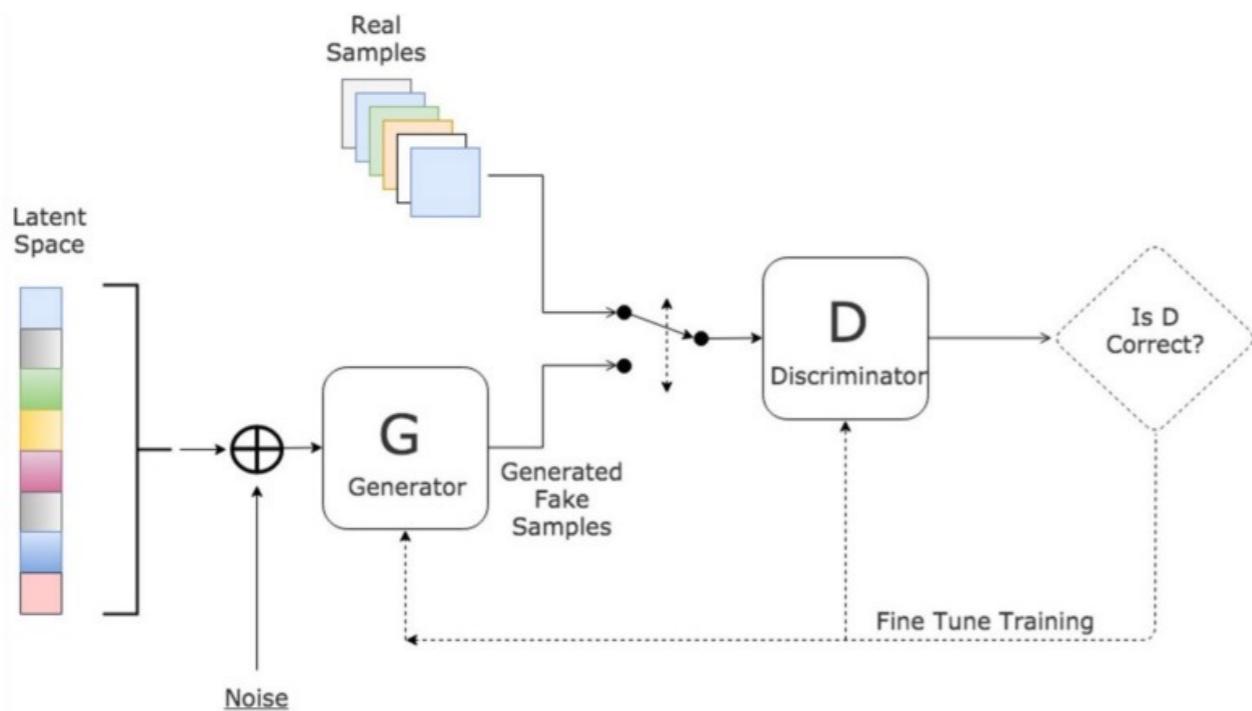


Amplified noise

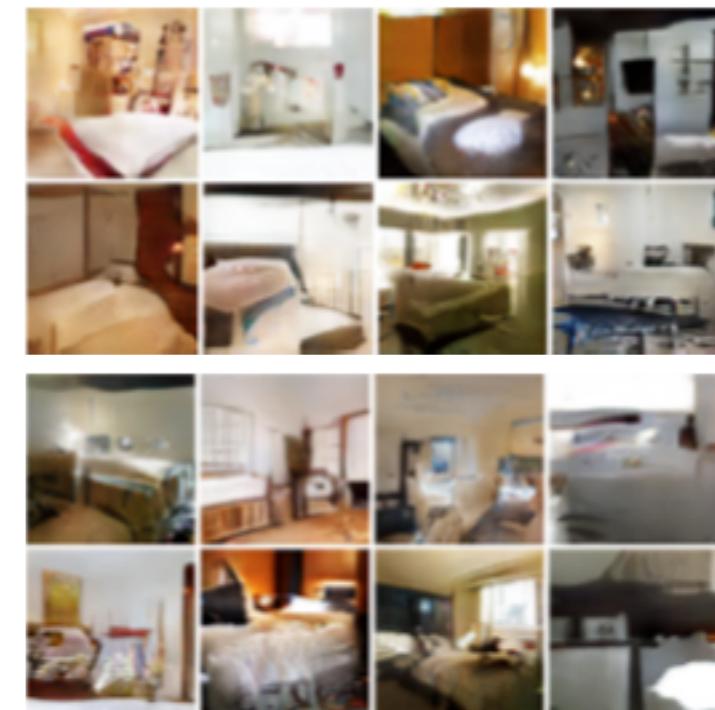
Source : [github.com/Hvass-Labs/TensorFlow-Tutorials](https://github.com/Hvass-Labs/TensorFlow-Tutorials)

# A few hot topics

## B - Generative Adversarial Networks (GAN)



Source : <https://www.datasciencecentral.com/profiles/blogs/generative-adversarial-networks-gans-engine-and-applications>



Sample images generated by a WGAN

# A few hot topics

## B - Generative Adversarial Networks (GAN)

---



**Which sample is a real one?**  
Image from [Brock et al., 2018]

# A few hot topics

## C - Learning from little supervised data

---



We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state of the art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization —

Twitter status, @openai, Feb 14th, 2019

---

# Conclusion

---

- Early stage: 1943 - 1969
  - learning with stochastic gradient descent
- Back in the game: 1985 - 1995
  - NN are universal approximators
- A *de facto* standard in computer vision: 2009 - ?
  - deep nets can leverage on big data + high perf. computers
- Open issues
  - learning on structured data (eg. graphs)
  - vulnerability to adversarial attacks
  - architecture design
  - sample-efficient learning
  - more theory needed