Deep learning for image analysis quick introduction

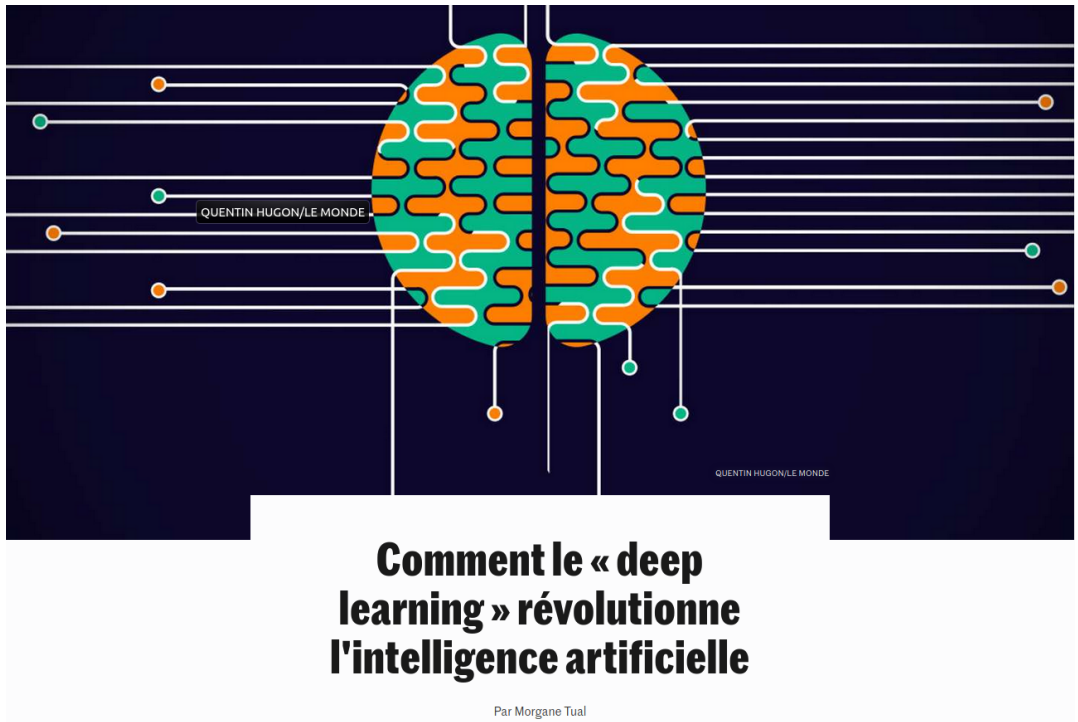
E. Decencière

MINES ParisTech
PSL Research University
Center for Mathematical Morphology

Contents

Contents

The rise of deep learning



Le Monde, juillet 2015

The rise of deep learning

Nature, 2016

The rise of deep learning

Le prix Turing récompense trois pionniers de l'intelligence artificielle (IA)

L'association américaine ACM a remis son prestigieux prix aux chercheurs français, canadien et britannique : Yann LeCun, Yoshua Bengio et Geoffrey Hinton.

Par David Larousserie • Publié le 27 mars 2019 à 11h01 - Mis à jour le 29 mars 2019 à 12h11

Le Monde, mars 2019

The rise of deep learning

Pour Elon Musk, l'intelligence artificielle pourrait menacer la civilisation

L'entrepreneur américain, qui a fondé Tesla, a alerté les politiques américains sur la nécessité de réguler l'intelligence artificielle.

Par **Le Figaro**
Publié le 18/07/2017 à 06:00, mis à jour le 18/07/2017 à 11:25

Le Figaro, juillet 2017

Contents

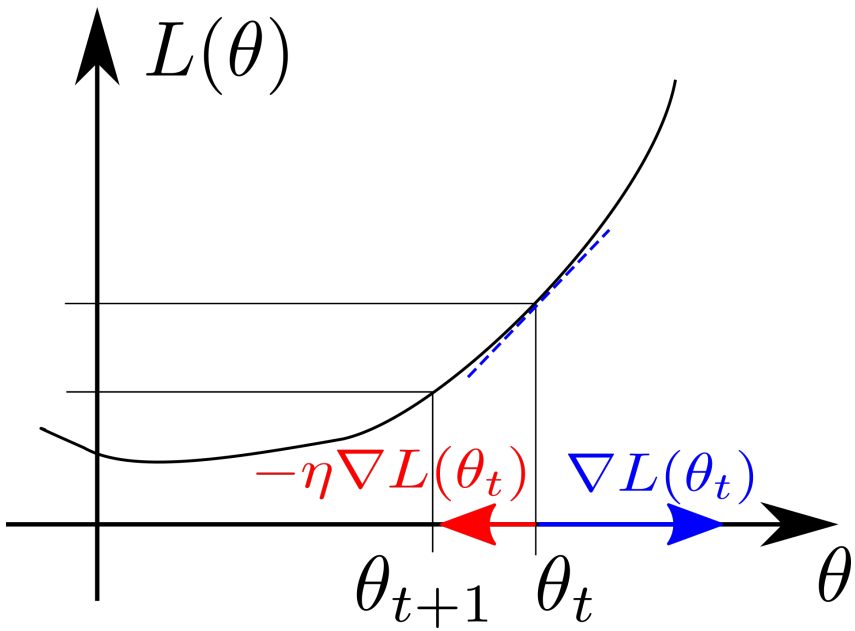
Differentiable programming

Definition

Differentiable programming is an algorithmic framework where differentiable operators are combined together to build complex systems. The parameters of the system can then be optimized via gradient descent thanks to back-propagation.

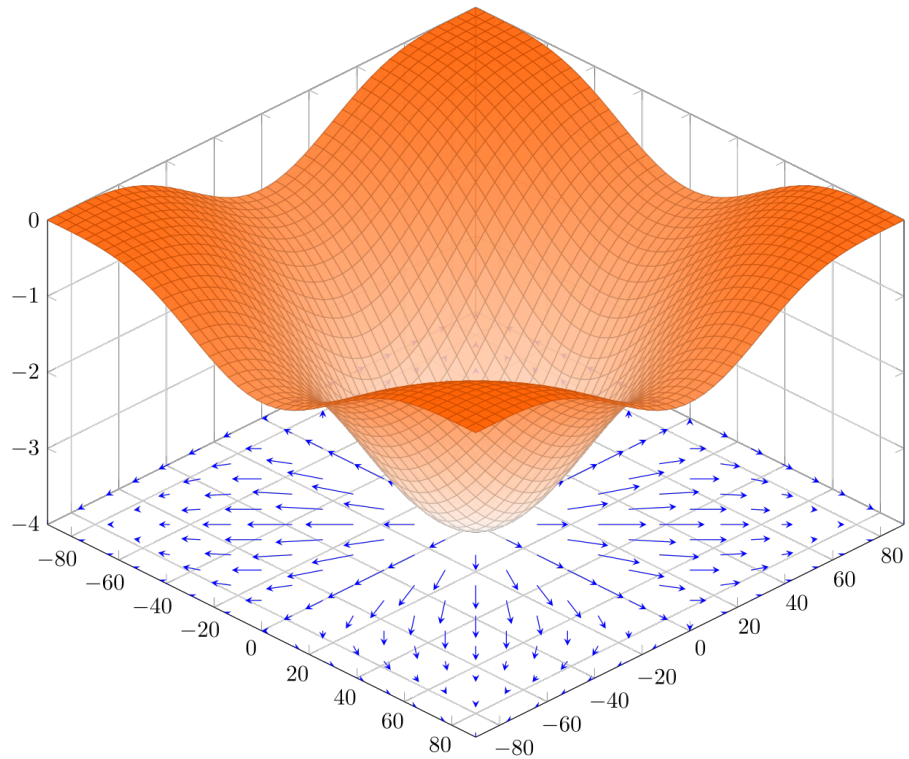
- In our case, computational graphs will be used to compute a loss function L, which depends on the input of the system $\mathbf{X}$ and its parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_q\}$.

Gradient descent in the scalar case



$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla L(\boldsymbol{\theta}_t)$$

How to minimize a function?



Definition: gradient

Let L be a derivable function from $\mathbb{R}^n$ into $\mathbb{R}$. Its gradient ∇L is:

$$\nabla L(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial L}{\partial \boldsymbol{\theta}_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial L}{\partial \boldsymbol{\theta}_n}(\boldsymbol{\theta}) \end{pmatrix}$$

Credits: By MartinThoma, CC0, https://commons.wikimedia.org/

Computational graph

Definition

A computational graph is a direct acyclic graph such that:

- A node is a mathematical operator
- To each edge is associated a value (scalar, vector, matrix, tensor, ...)
- Each node can compute the values of its output edges from the values of its input edges

Computing a *forward pass* through the graph means choosing its input values, and then progressively computing the values of all edges.

Computational graph example

Computational graph of:

$$\sigma(w_1 x + w_2 y + b)$$

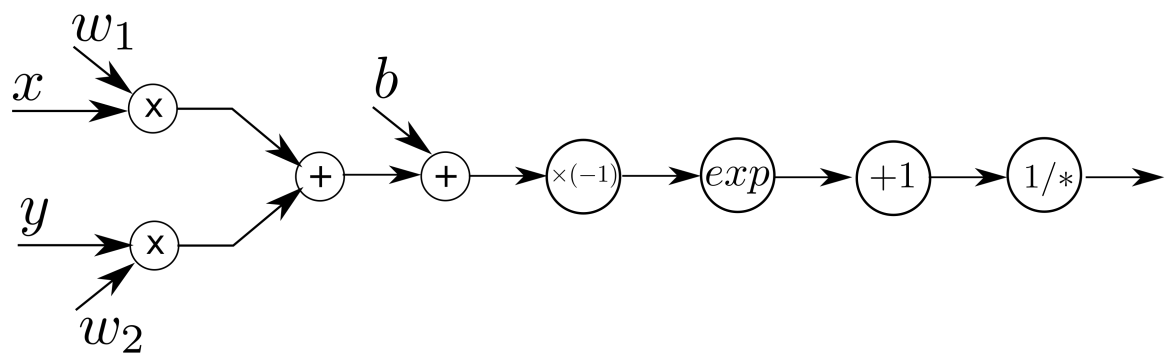
where σ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

Computational graph example

Computational graph of:

$$\sigma(w_1 x + w_2 y + b)$$

where σ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

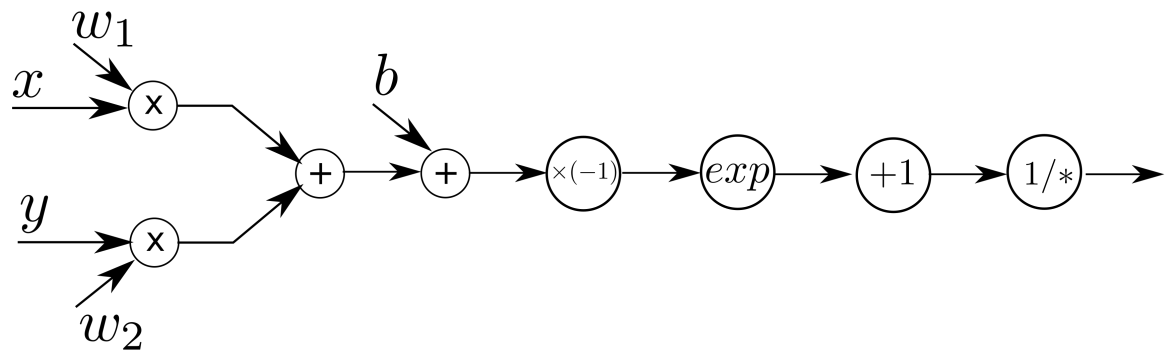


Computational graph example

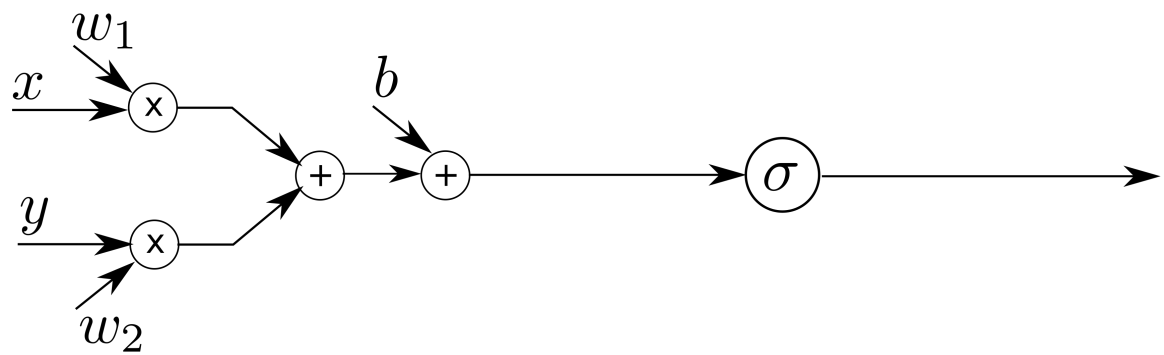
Computational graph of:

$$\sigma(w_1 x + w_2 y + b)$$

where σ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



The graph can be represented at different levels of detail:



Gradient descent applied to computational graphs

In the case of computational graphs, the loss L depends on each parameter $\boldsymbol{\theta}_i$ via the composition of several simple functions. In order to compute the gradient $\nabla_{\boldsymbol{\theta}} L$ we will make extensive use of the chain rule theorem.

Chain rule theorem

Let f_1 and f_2 be two derivable real functions ($\mathbb{R} \to \mathbb{R}$). Then for all x in $\mathbb{R}$: :

$$(f_2 \circ f_1)'(x) = f_2'(f_1(x)).f_1'(x)$$

Leibniz notation

Let us introduce variables x, y and z:

$$x \xrightarrow{f_1} y \xrightarrow{f_2} z$$

Then:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y} \cdot \frac{\mathrm{d}y}{\mathrm{d}x}$$

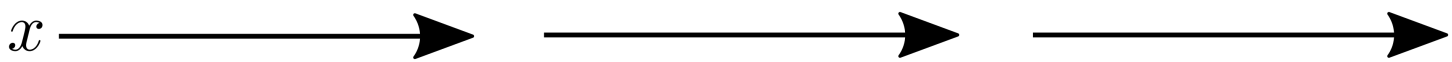
The backpropagation algorithm

- The backpropagation algorithm is used in a computational graph to efficiently compute the partial derivatives of the loss with respect to each parameter of the network.
- One can trace the origins of the method to the sixties
- It was first applied to NN in the eighties
  [Werbos, 1982, LeCun, 1985]

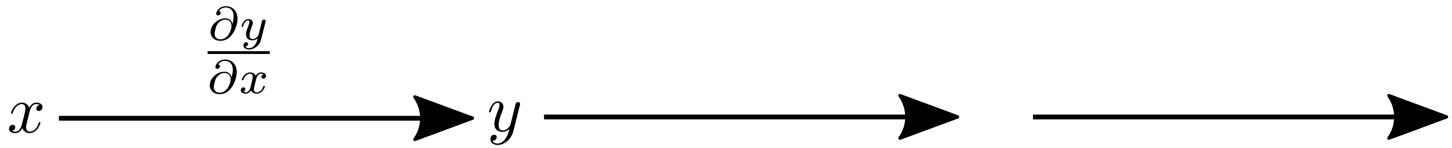
The backpropagation algorithm: intuition

- Given a computational graph, the main idea is to compute the local partial derivatives during a forward pass
- Then, during a backward pass, the partial derivatives of the loss with respect to each parameter is computed

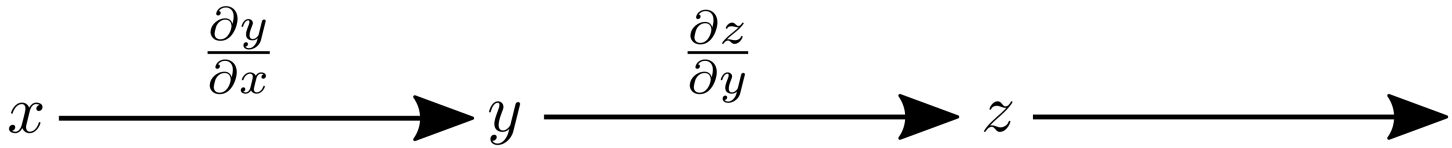
Simple backpropagation example



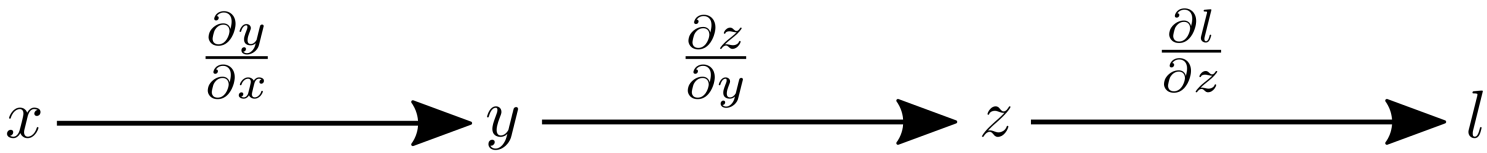
Simple backpropagation example



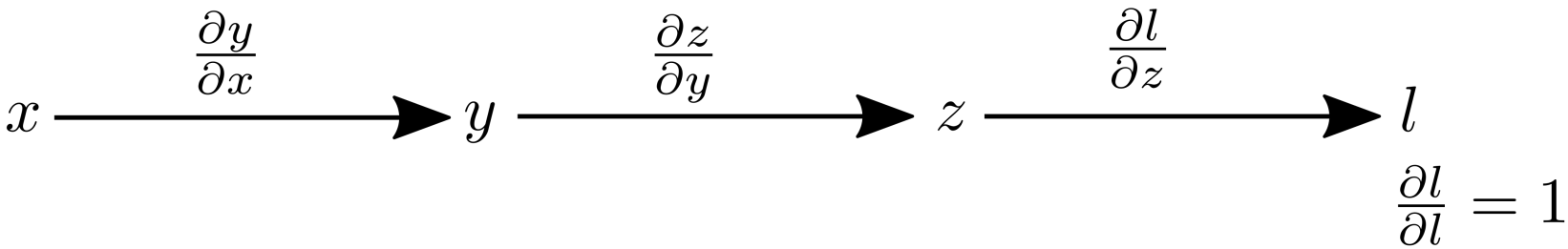
Simple backpropagation example



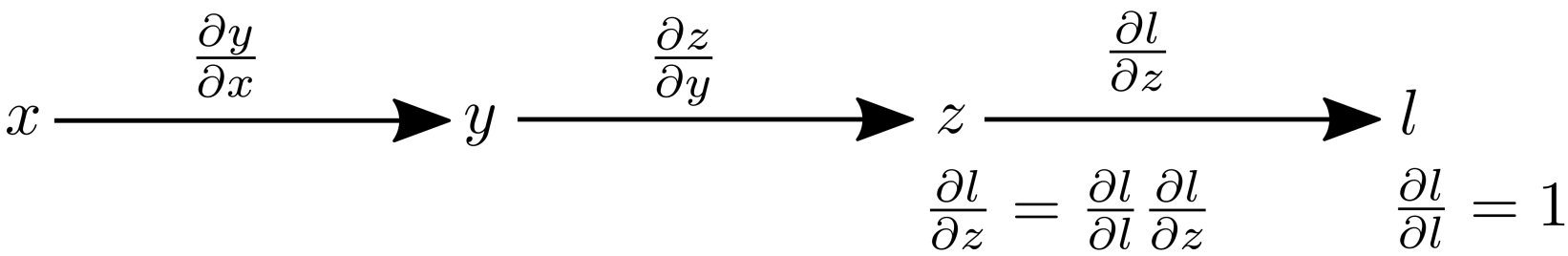
Simple backpropagation example



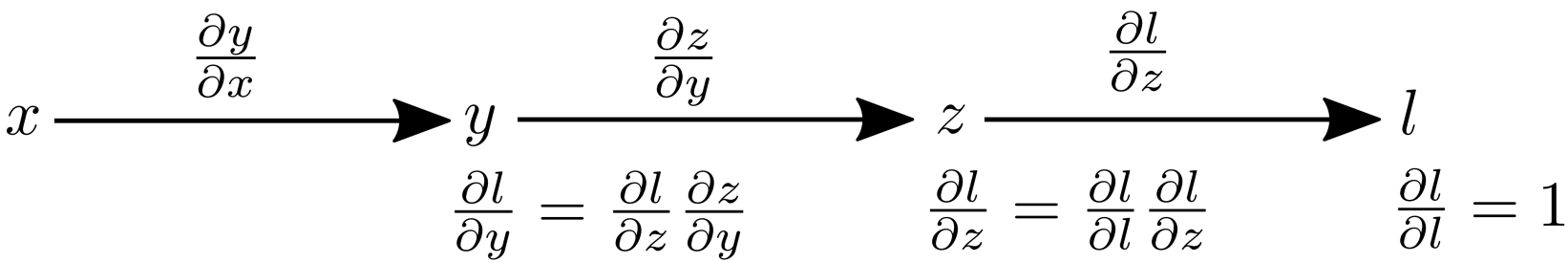
Simple backpropagation example



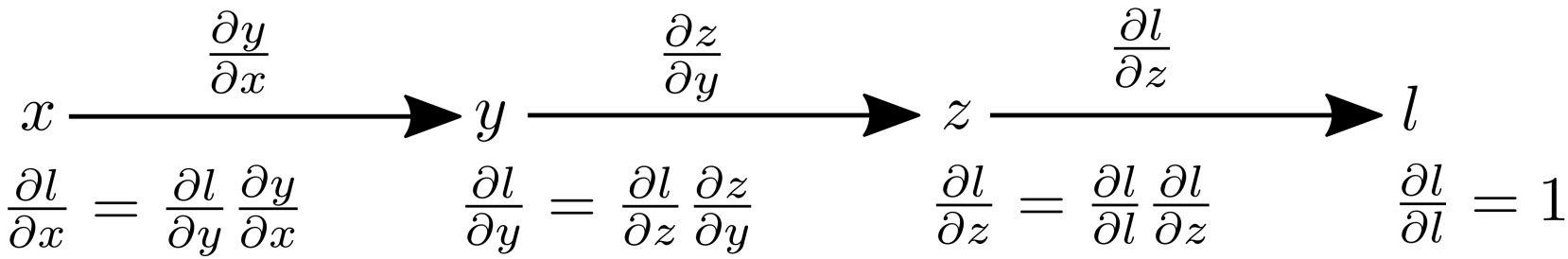
Simple backpropagation example



Simple backpropagation example



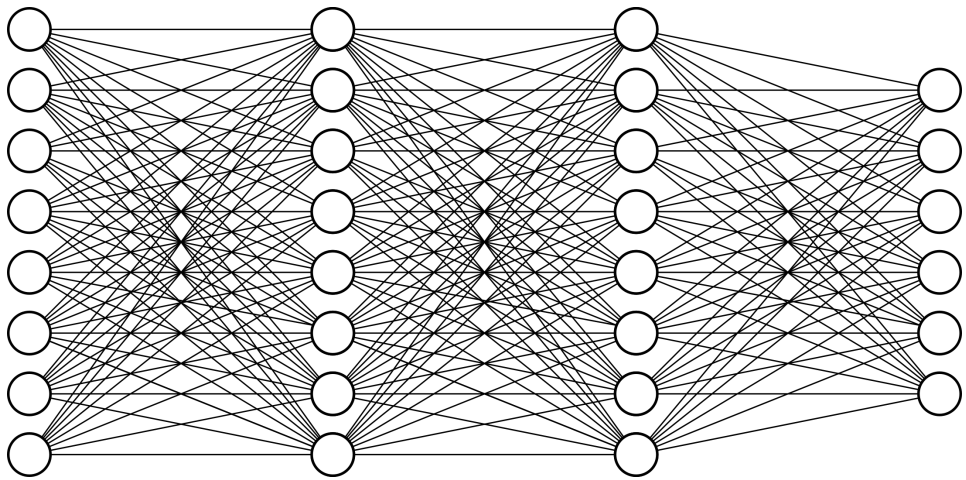
Simple backpropagation example



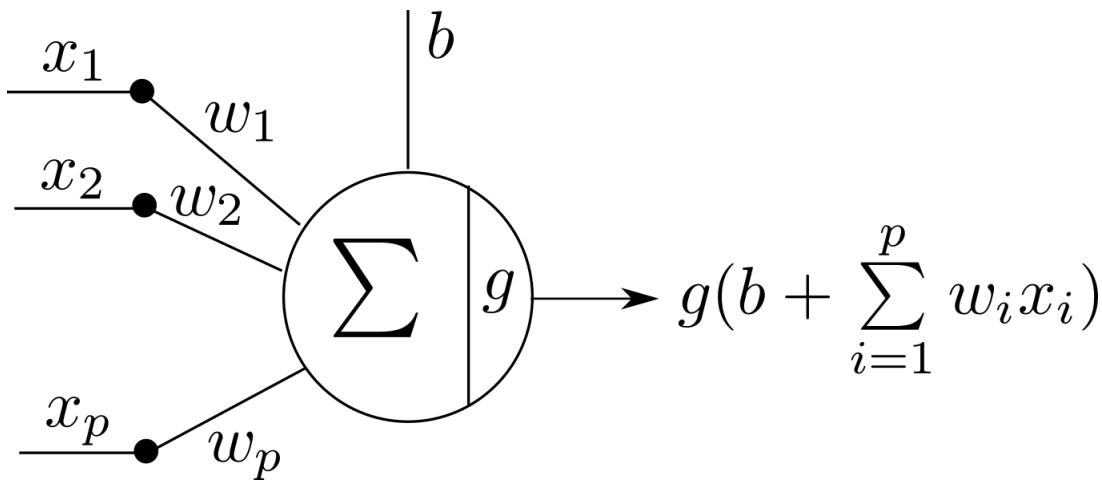
Feed-forward neural networks

Definition

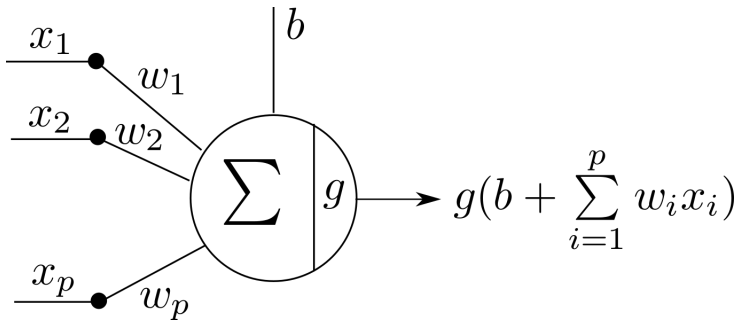
- A feed-forward neural networks is a computational graph without cycles
- Its computing units, the neurons, are organized in layers



Artificial neuron



Notations



With

$$\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix} = (w_1, \ldots, w_p)^T$$

and

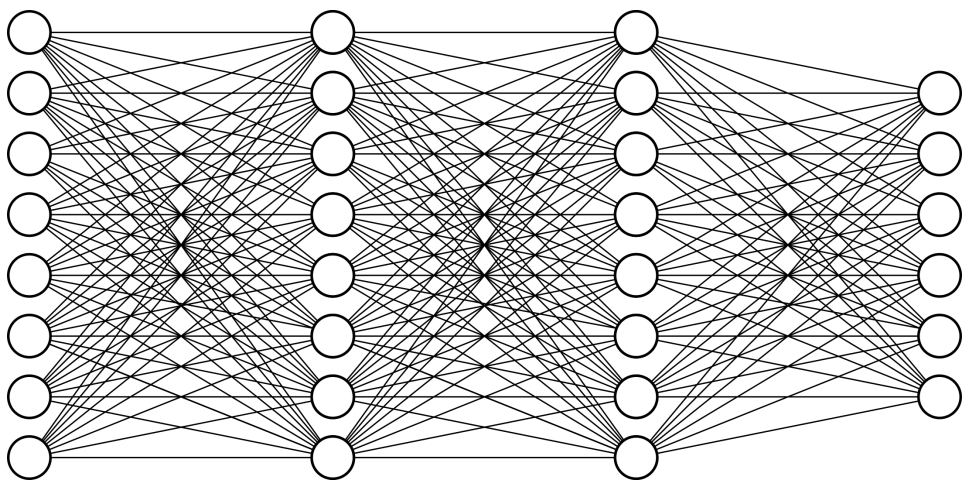
$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} = (x_1, \ldots, x_p)^T$$

We can simply write:

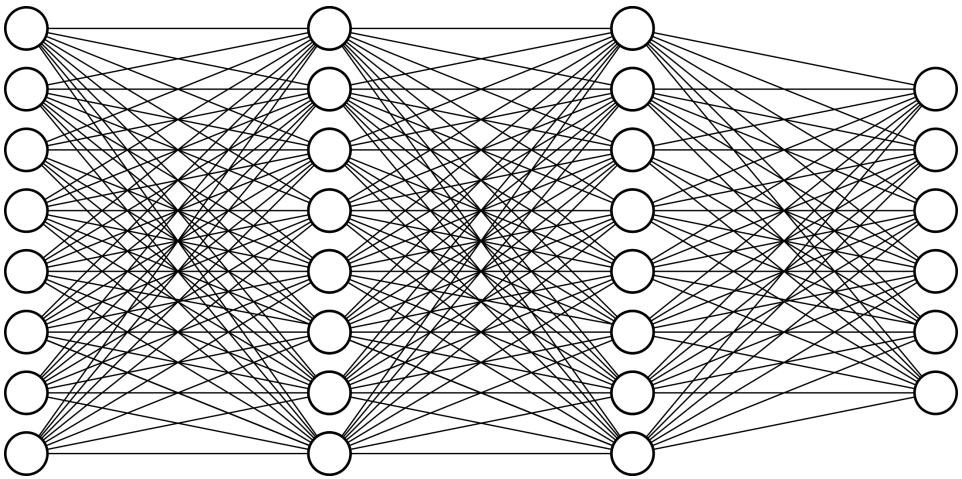
$$\mathrm{g}(b + \sum_{i=1}^{p} w_i x_i) = \mathrm{g}(b + \mathbf{w}^T \mathbf{x})$$

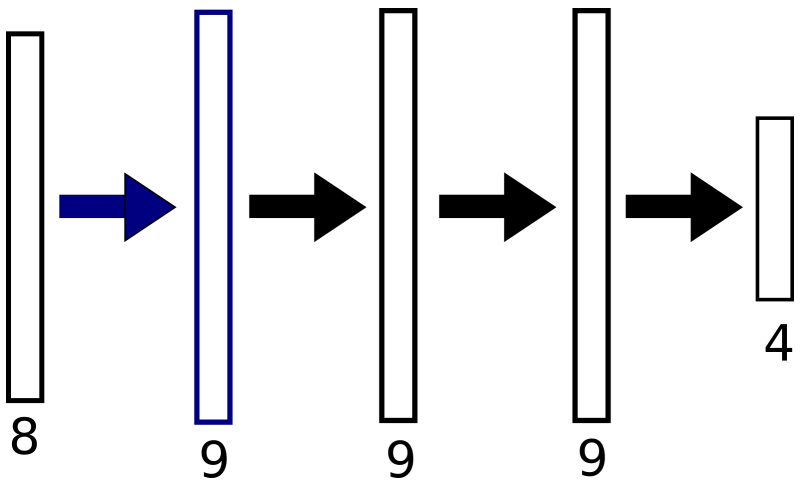
Fully-connected layer

- A layer is said to be fully-connected (FC) if each of its neurons is connected to all the neurons of the previous layer
- If a FC layer contains r neurons, and the previous layer q, then its weights are a 2D dimensional array (a matrix) of size $q \times r$



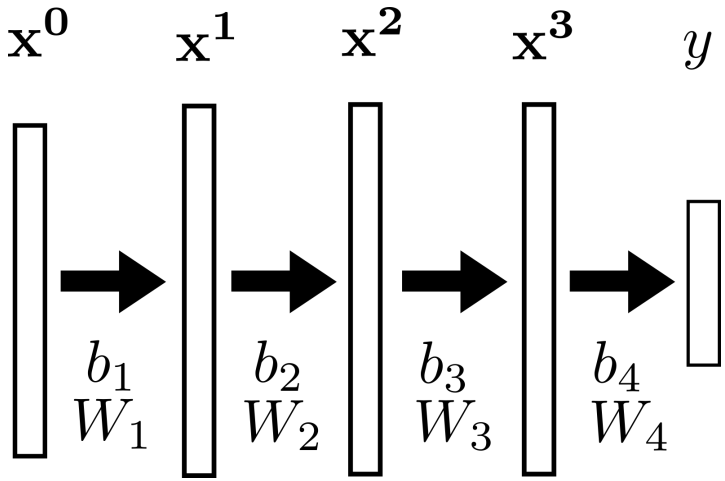
Graphical representation of NNs





- Data is organized into arrays, linked with operators
- A layer corresponds to an operator between arrays.

The equations of fully-connected layers

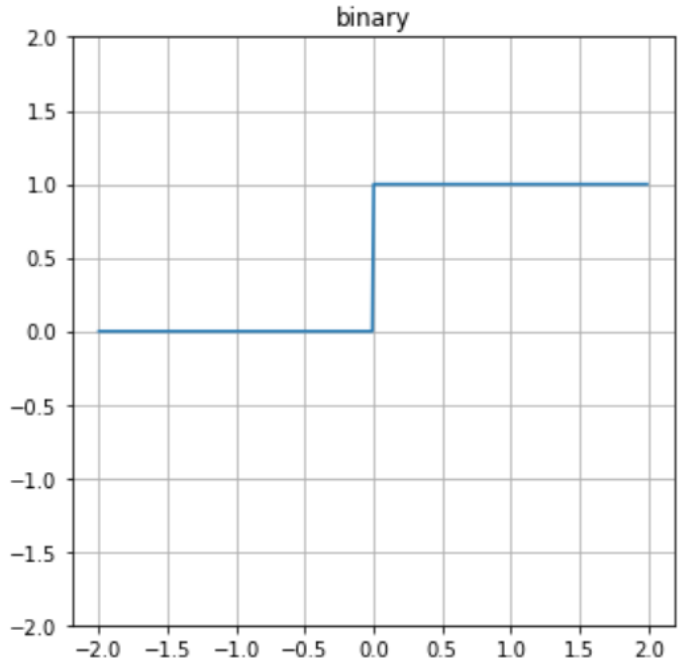


$$\mathbf{x}^i = \mathbf{g}_i(\mathbf{W}_i\mathbf{x}^{i-1} + \mathbf{b}_i),\ i = 1, 2, 3$$

$$y = \mathbf{g}_4(\mathbf{W}_4\mathbf{x}^4 + \mathbf{b}_4)$$

Activation: binary

$$g(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

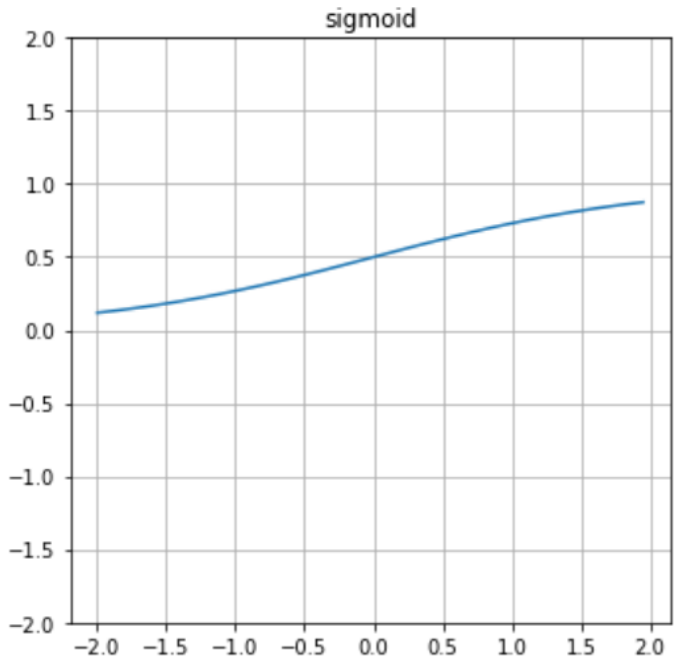


Remarks

- Biologically inspired
- \+ Simple to compute
- \+ High abstraction
- \- Gradient nil except on one point
- In practice, almost never used

Activation: sigmoid

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

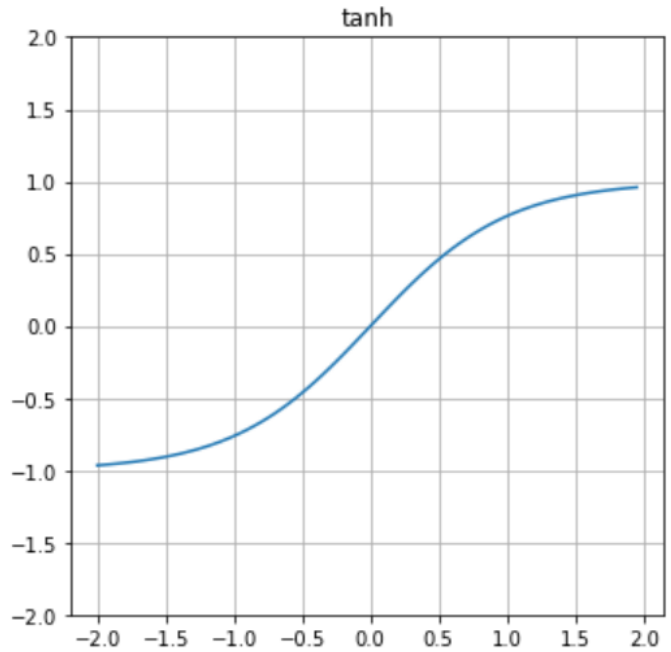


Remarks

+ Similar to binary activation, but with usable gradient
  - However, gradient tends to zero when input is far from zero
  - More computationally intensive

Activation: hyperbolic tangent

$$\mathrm{g}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

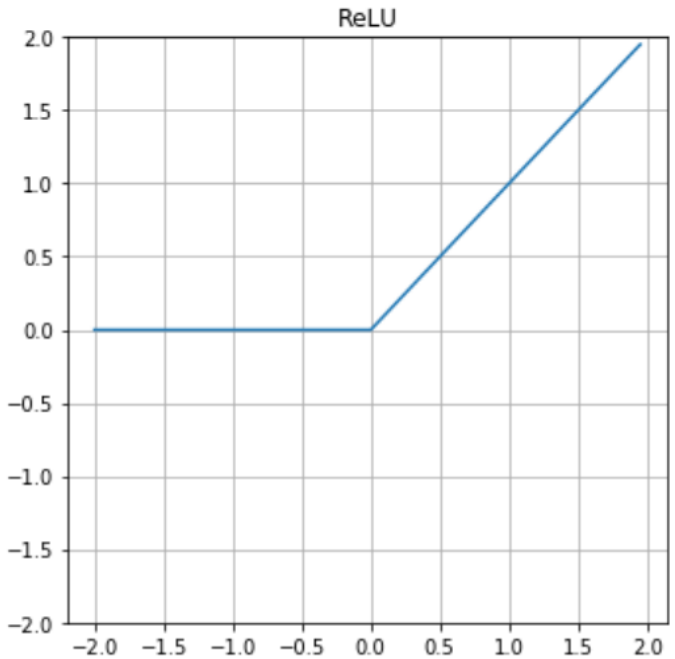


Remarks

- Similar to sigmoid

Activation: rectified linear unit (ReLU)

$$\mathrm{g}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

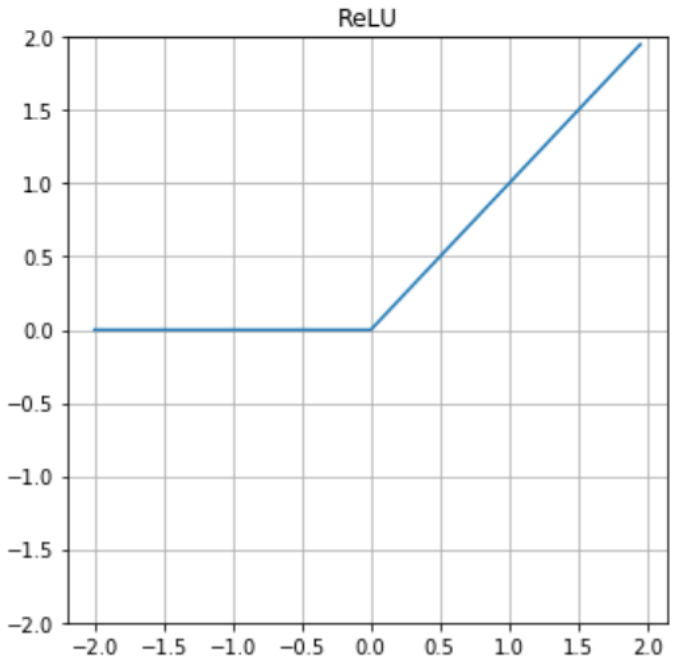


Remarks

+ Usable gradient when activated
+ Fast to compute
+ High abstraction

Activation: rectified linear unit (ReLU)

$$\mathrm{g}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$



Remarks

+ Usable gradient when activated
+ Fast to compute
+ High abstraction

ReLU is the most commonly used activation function.

Contents

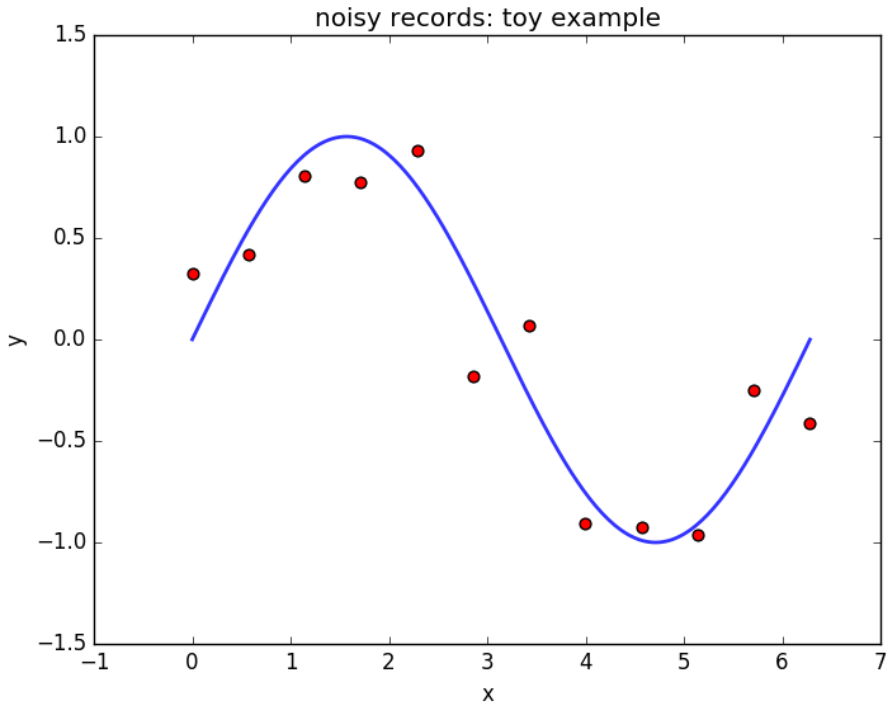
Machine Learning: basic definitions

- Machine Learning aims at predicting some output y from an input (or measurement) x:

$$y = f(x) \tag{1}$$

- In this formulation, Machine Learning aims at finding (learning) f from available data.
- The data that is used to learn f is called **training set**, denoted by $\mathbf{X}$.
- In this general formulation, there is no particular limitation as to the mathematical nature of x and y. Today, x will be an image, and y a class, a vector, or another image.

A simple example: polynomial curve fitting[1]



- From a set of measured points (x_i, y_i) (red), we would like to build a model to predict the value y for any given x.
- The true function is $g(x) = \sin(x)$ (displayed in blue).
- The measurements y_i are noisy outputs of that function, i.e.

$$y_i = \sin(x_i) + \epsilon \ , \qquad \epsilon \sim \mathcal{N}(0, 0.2) \tag{2}$$

[1] Example adapted from [Bishop, 2006]

A simple example: polynomial curve fitting

- We use the following polynomial model:

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m \\ &= \boldsymbol{\theta}^T \boldsymbol{\phi}(x) \end{aligned} \tag{3}$$

- Parameter vector: $\boldsymbol{\theta} = (a_0, a_1, \ldots, a_m)^T$
- Here, the initial measurement x is a scalar. In our model, we map x to a higher dimensional space:

$$\begin{aligned} \boldsymbol{\phi} : \mathbb{R}^P &\rightarrow \mathbb{R}^Q \\ x &\rightarrow \boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^m)^T \end{aligned} \tag{4}$$

- The model is linear in the parameters θ and linear in $\boldsymbol{\phi}$, but for $m > 1$, the model is not linear in x.

A simple example: polynomial curve fitting

- One classical approach is to minimize the least squared error between measured and predicted values:

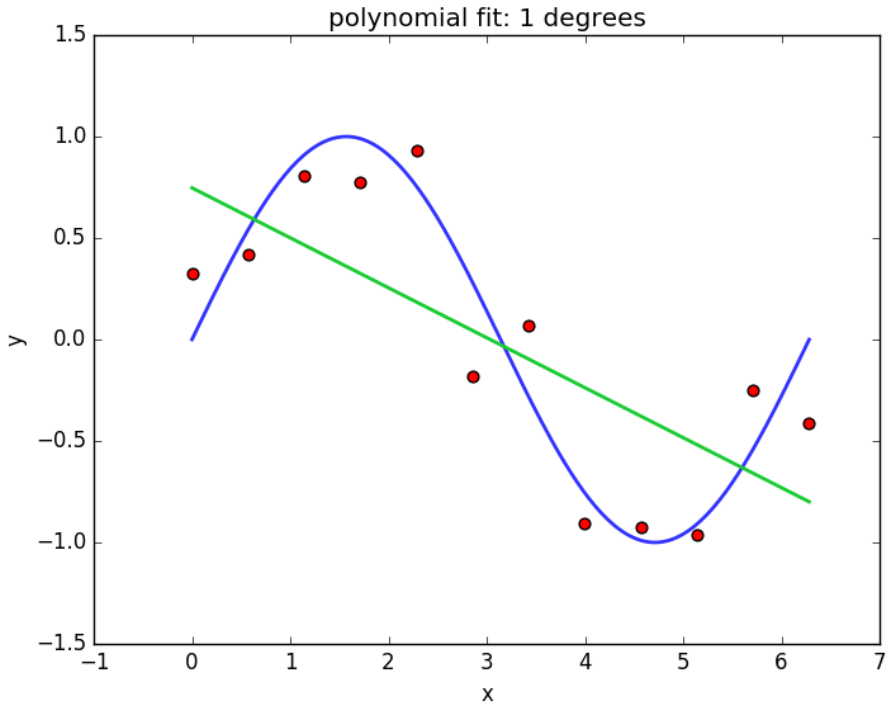
$$\begin{aligned} \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(x_i))^2 \end{aligned} \tag{5}$$

- This can be achieved by setting the gradient with respect to $\boldsymbol{\theta}$ to zero:

$$\nabla_{\boldsymbol{\theta}} L = (\frac{\partial L}{\partial a_0}, \frac{\partial L}{\partial a_1}, \ldots, \frac{\partial L}{\partial a_m})^T = 0 \tag{6}$$

- Unlike most optimization problems in this course, this leads to an analytical solution for $\boldsymbol{\theta}$. This is known as **linear regression**. For more details, we refer to [Hastie et al., 2009].

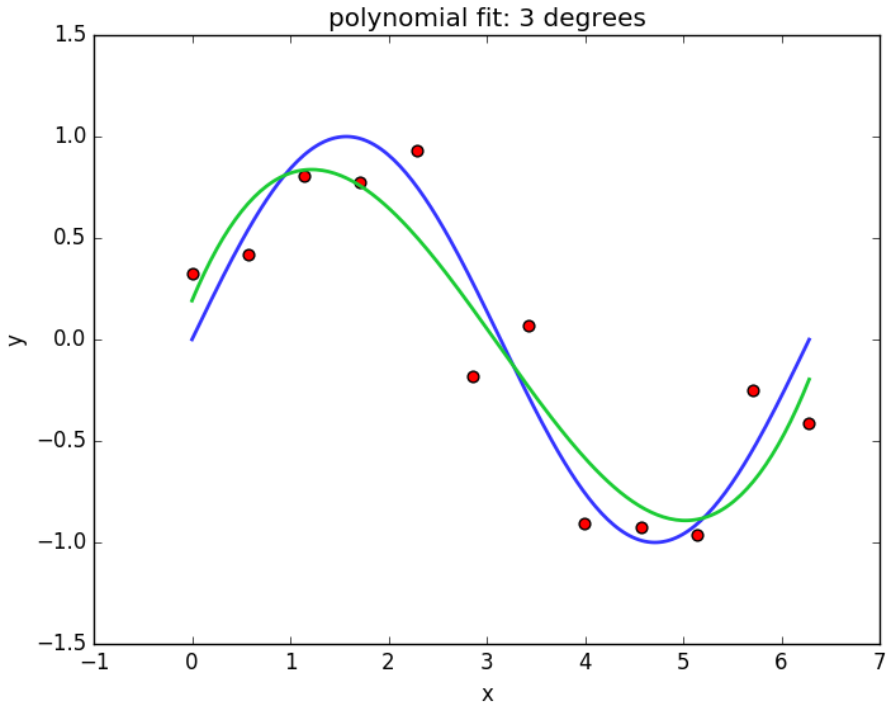
Overfitting and underfitting



$$\|\boldsymbol{\theta}\|^2 = 0.67$$

For $m = 1$, the model is linear in its inputs. The solution is not capable of modeling the measured data points; we get a poor approximation of the original function. The family of functions we have used was not complex enough to model the true data distribution. We also speak of **underfitting**.

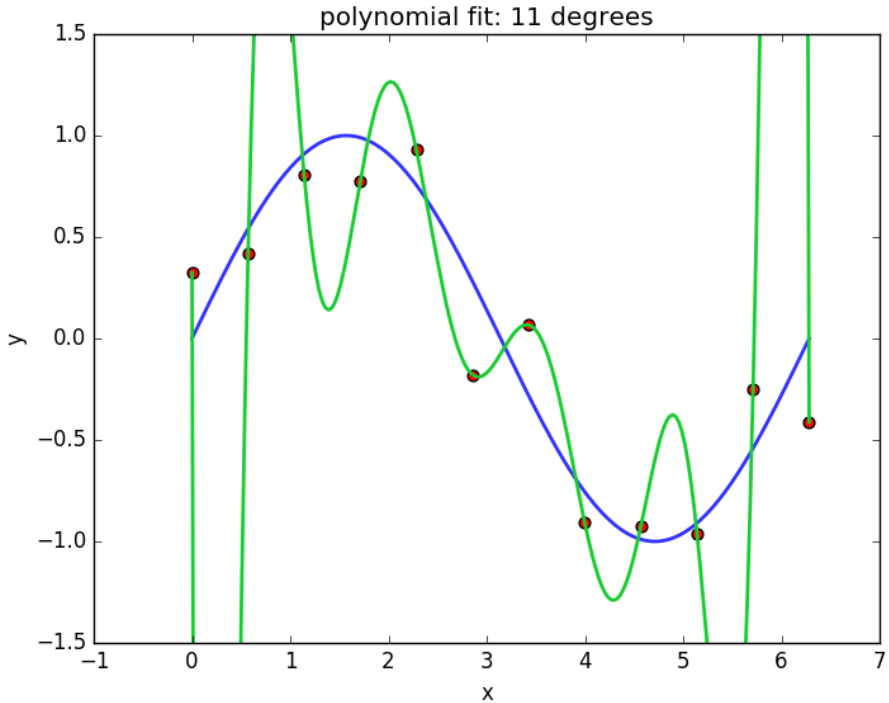
Overfitting and underfitting



$$\|\boldsymbol{\theta}\|^2 = 1.72$$

For $m = 3$, we obtain a solution that seems to be quite right: it is sufficiently complex to model the true data distribution, but not too complex to model the small variations which are due to noise.

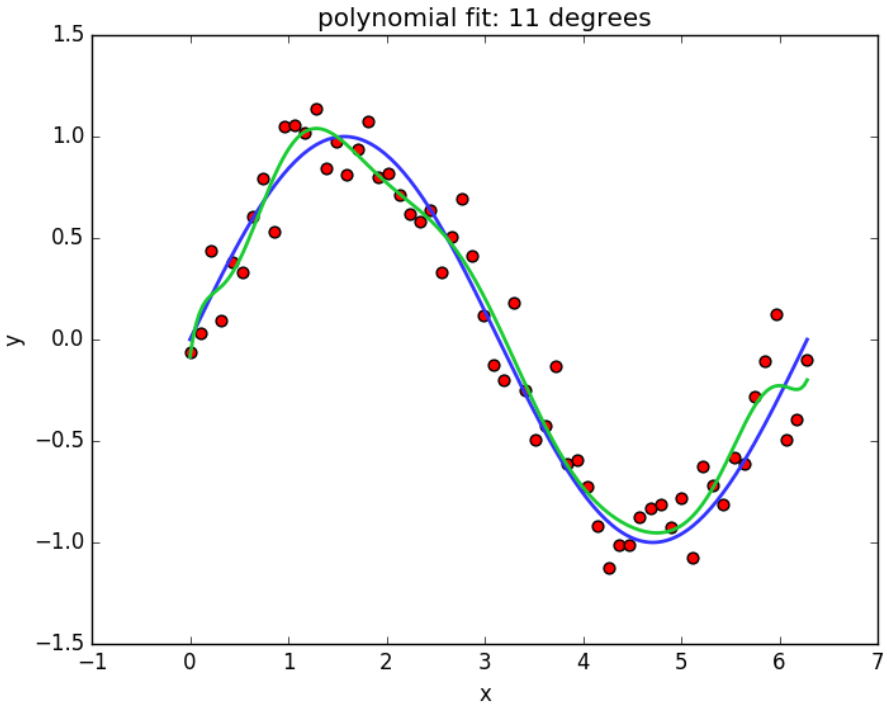
Overfitting and underfitting



$$\|\boldsymbol{\theta}\|^2 \approx 10^7$$

For $m = 11$, we obtain a solution that has zero error (the function passes through every point of the training set). But the coefficients with large absolute values that cancel each other precisely on the training points lead to a highly unstable function. We speak of **overfitting** and **poor generalization**.

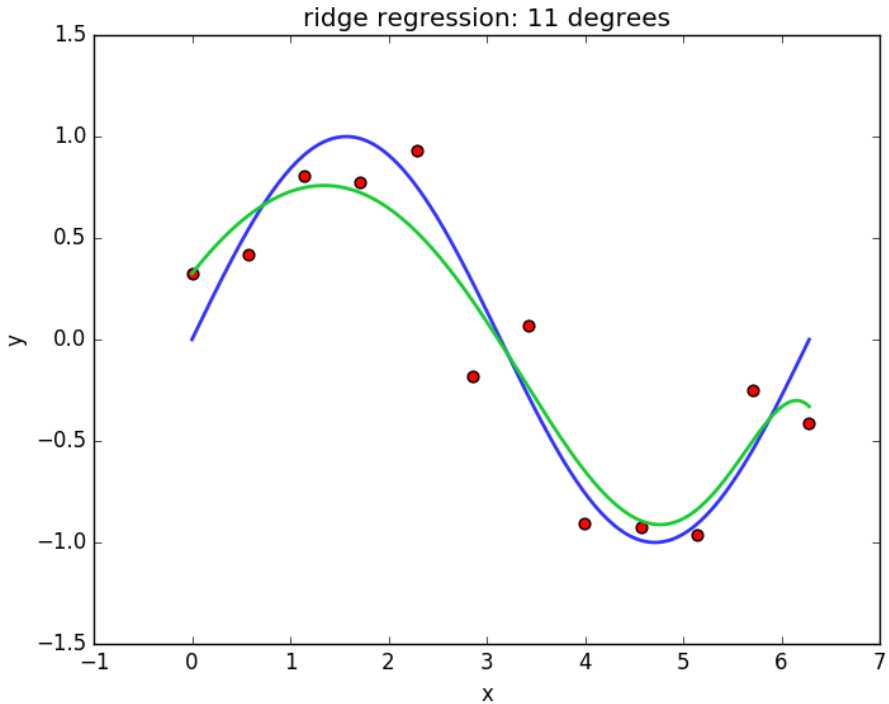
Overfitting and underfitting



$$\|\boldsymbol{\theta}\|^2 = 5647$$

One way of reducing overfitting is to increase the number of samples. Even if the function is complex, it cannot be "too wild", as it has to find a compromise between many training samples. This however implies the annotation (or measurement) of more samples.

Overfitting and underfitting



$$\|\boldsymbol{\theta}\|^2 = 0.41$$

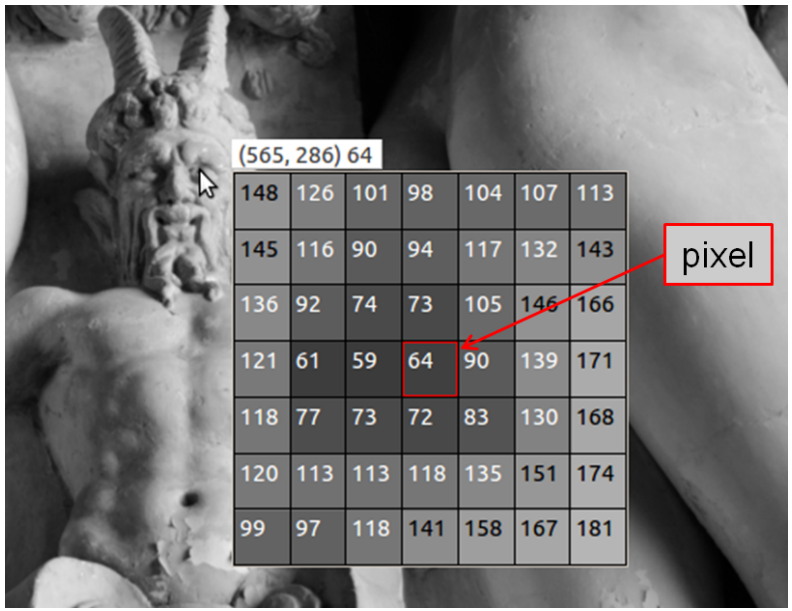
Another way of preventing overfitting without increasing the number of samples, is to add a penalization term in the optimization procedure. This is also known as **regularization**:

$$L = \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(x_i))^2 + \lambda \|\boldsymbol{\theta}\|^2 \tag{7}$$

A picture is worth a thousand words

Definition

- Classically, an image is a matrix of values belonging to $[0, \ldots, 255]$ (grey level images) or to $[0, \ldots, 255]^3$ (color images).
- More generally, an image is a q-dimensional array of values belonging to R^d.



Grey level values around the left eye of the faun

The role of annotated image databases

Image databases including *annotations* (typically some kind of high level information) are essential to the development of *supervised* machine learning methods for image analysis.

Annotations

- Image class
- Measure(s) obtained from the image
- Position of objects within the image
- Segmentation

MNIST database [Lecun et al., 1998]

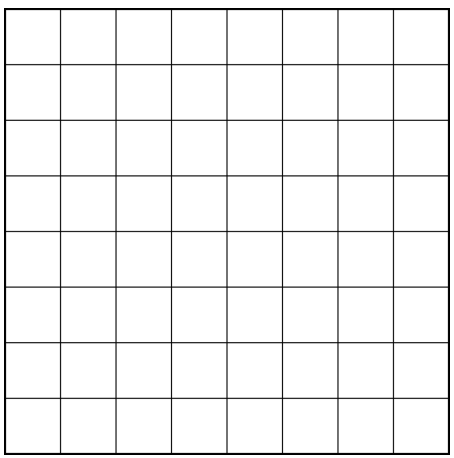
- The Modified National Institute of Standards and Technology (MNIST) database contains $60\,000$ training images of hand-written digits, and $10,000$ test images.
- Image size: 28×28
- It has been used since 1998
- Human performance on a similar database (NIST) is reported to be around 1.5% error [Simard et al., 1993]
- Best methods, based on convolutional neural networks, give around 0.21% test error.

MNIST database

Credits: Images from MNIST assembled by Josef Stepan (licensed under CC BY-SA 4.0)

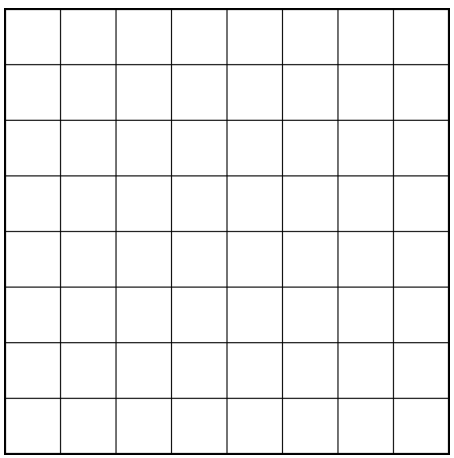
Layers representation

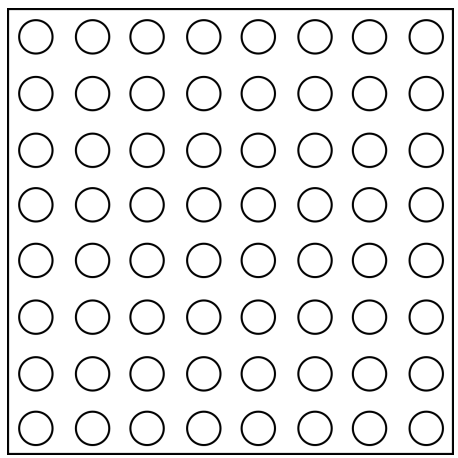
For illustration purposes, in the following slides images and filters will be displayed as rows of neurons – these can be seen as 1D arrays or as sections of 2D arrays.
We represent some connections between neurons. Each such connection is associated to a weight. The bias are not represented, to avoid clutter, but must not be forgotten.



Layers representation

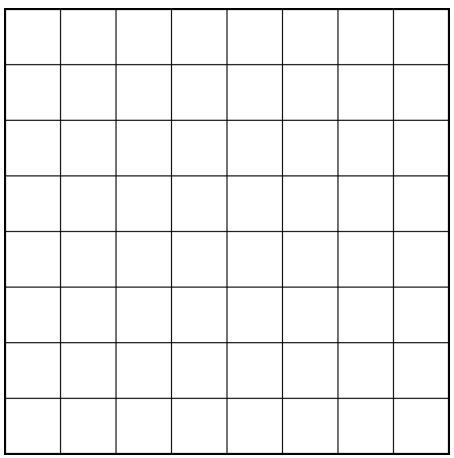
For illustration purposes, in the following slides images and filters will be displayed as rows of neurons – these can be seen as 1D arrays or as sections of 2D arrays.
We represent some connections between neurons. Each such connection is associated to a weight. The bias are not represented, to avoid clutter, but must not be forgotten.

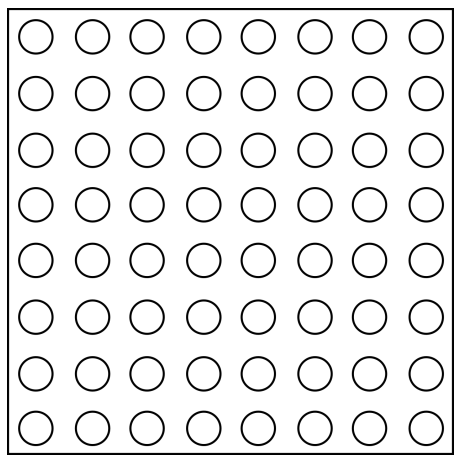


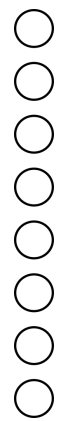


Layers representation

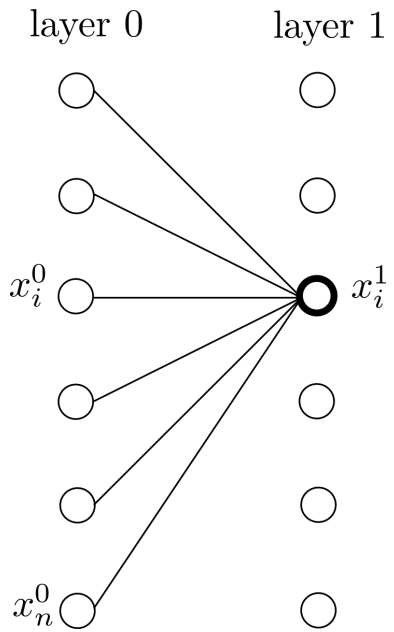
For illustration purposes, in the following slides images and filters will be displayed as rows of neurons – these can be seen as 1D arrays or as sections of 2D arrays.
We represent some connections between neurons. Each such connection is associated to a weight. The bias are not represented, to avoid clutter, but must not be forgotten.





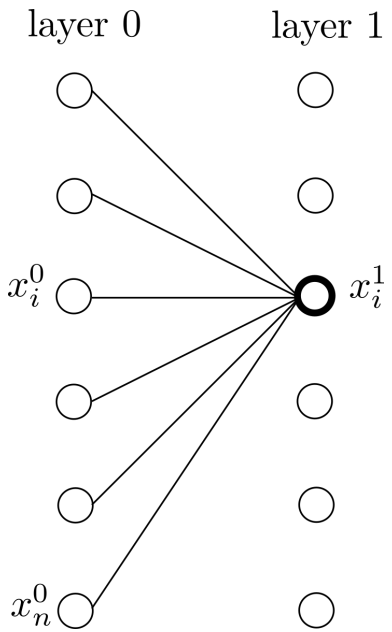


Towards convolutional layers

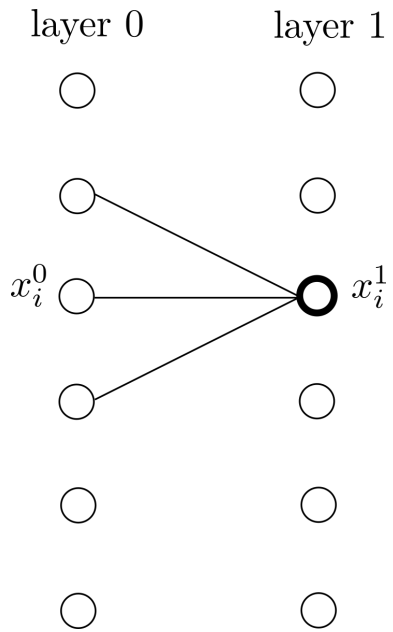


Fully connected layer:
$n(n+1)$ weights

Towards convolutional layers

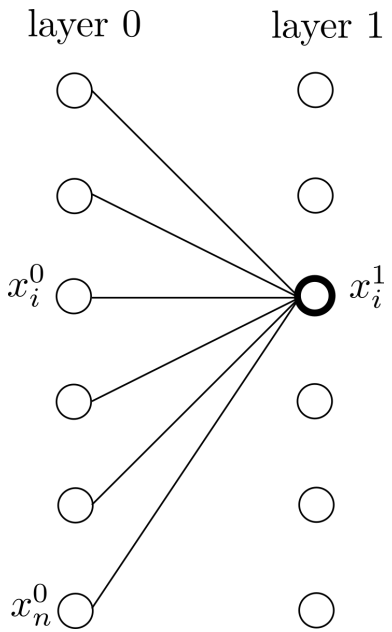


Fully connected layer:
$n(n+1)$ weights

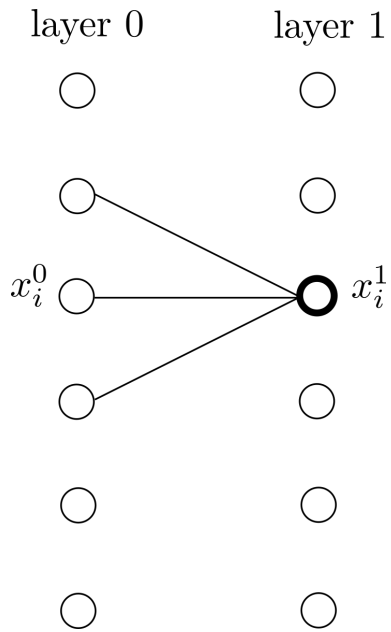


Locally conn. layer:
$n(s+1)$ weights

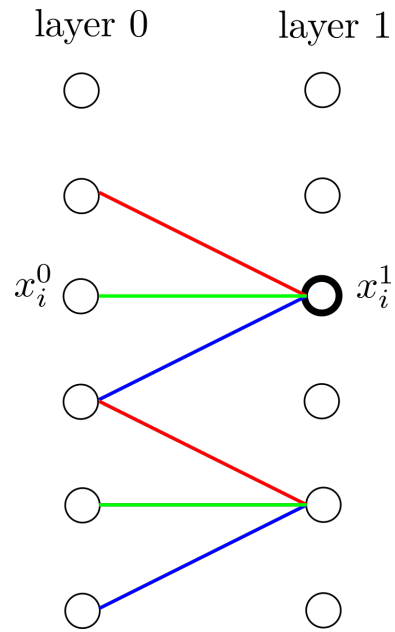
Towards convolutional layers



Fully connected layer:
$n(n+1)$ weights



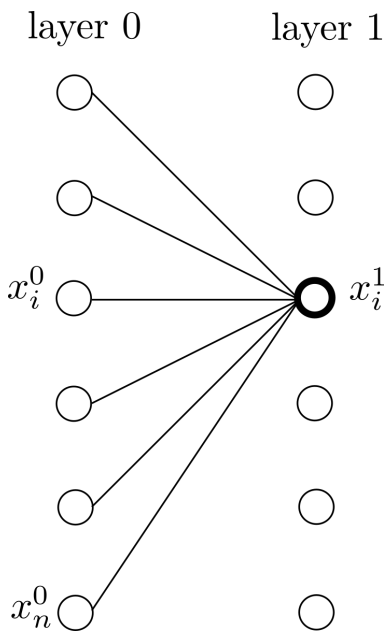
Locally conn. layer:
$n(s+1)$ weights



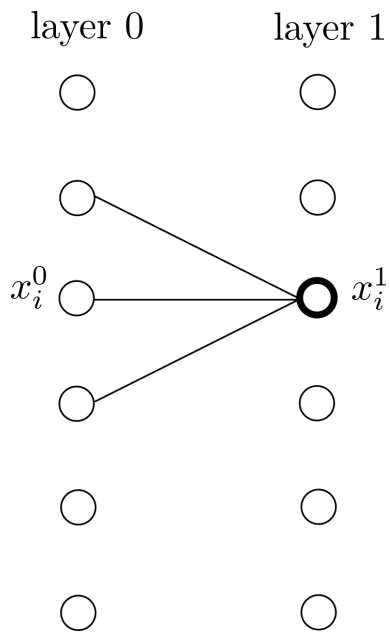
Weight replication: $s+1$ weights.
Convolutional layer.

Towards convolutional layers: some figures

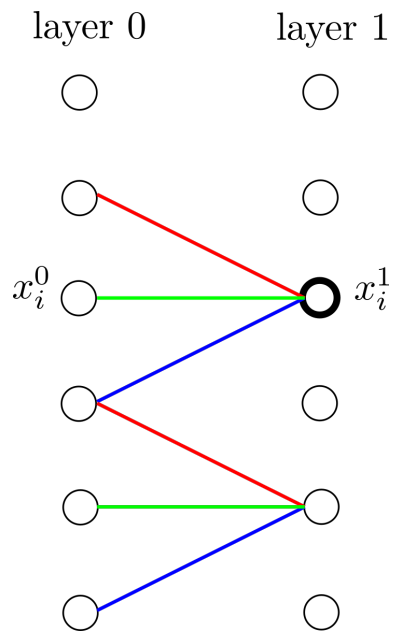
- 3×3 convolutions: $s = 9$
- Toy image: $n = 28 \times 28 = 784$
- Typical image: $n = 1000 \times 1000 = 10^6$



Fully connected layer:
$n(n+1)$ weights
$\approx 6.10^5$
$\approx 10^{12}$



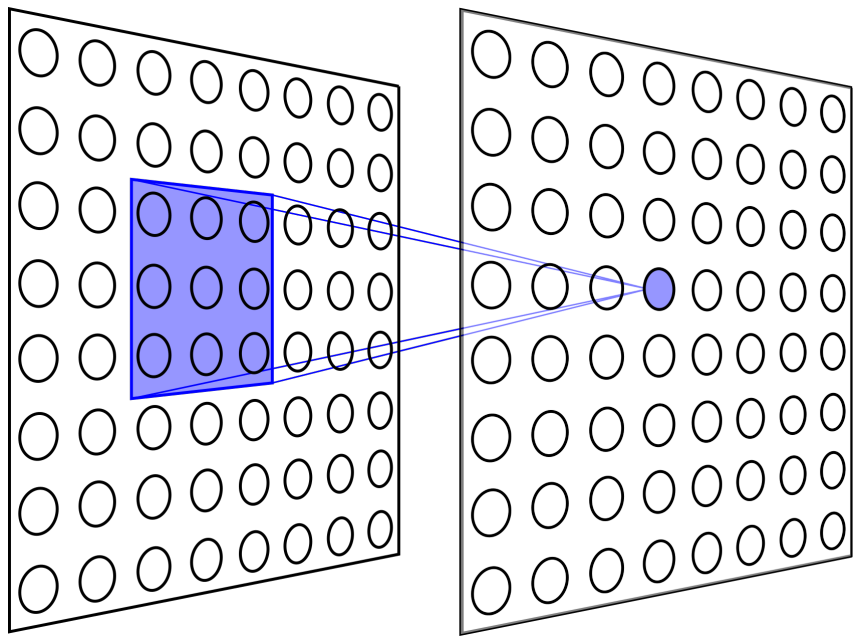
Locally conn. layer:
$n(s+1)$ weights
7840
10^7



Weight replication: $s+1$ weights.
10
10

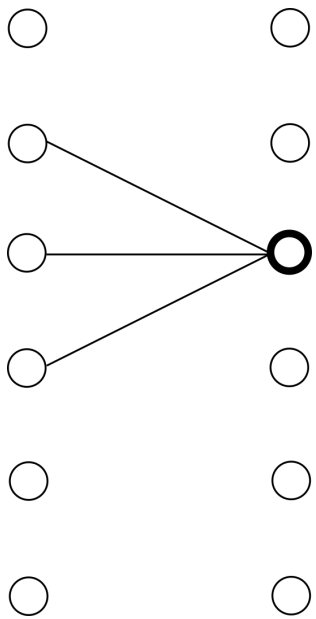
Convolutional layer illustration in 2D

- Illustration of a convolution of size 3×3



Stride

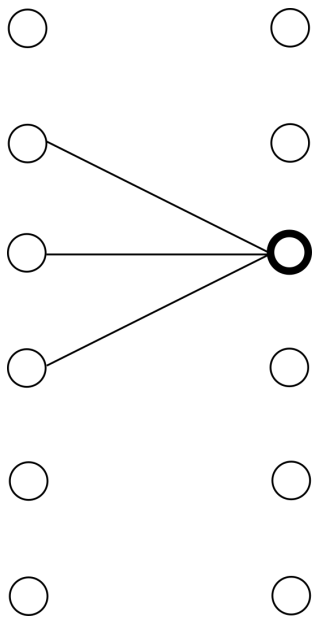
A convolutional layer can at the same time downsample the image by applying a sampling step, or *stride*.



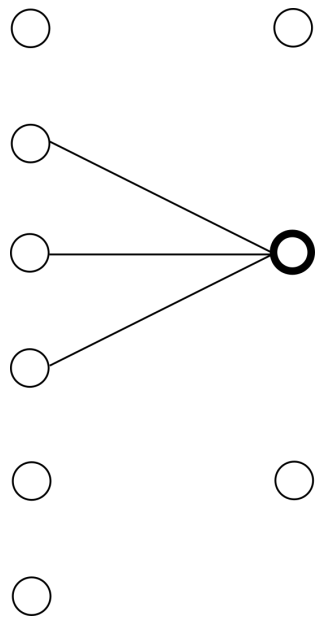
Stride 1

Stride

A convolutional layer can at the same time downsample the image by applying a sampling step, or *stride*.



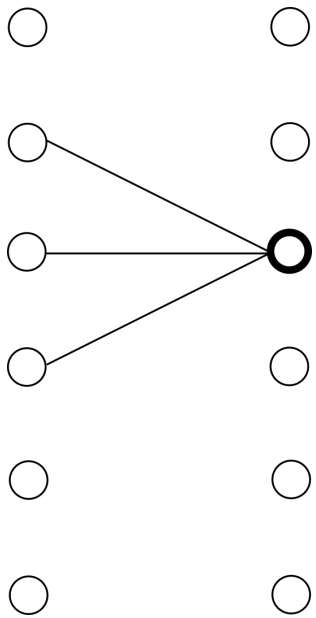
Stride 1



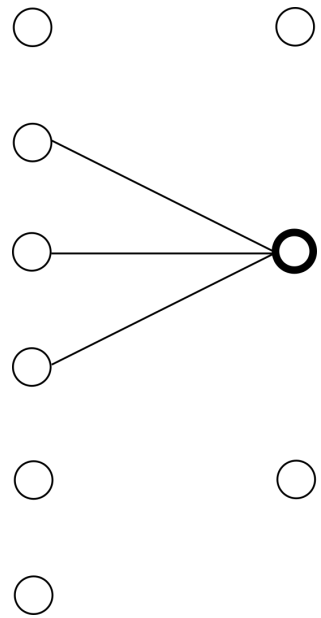
Stride 2

Stride

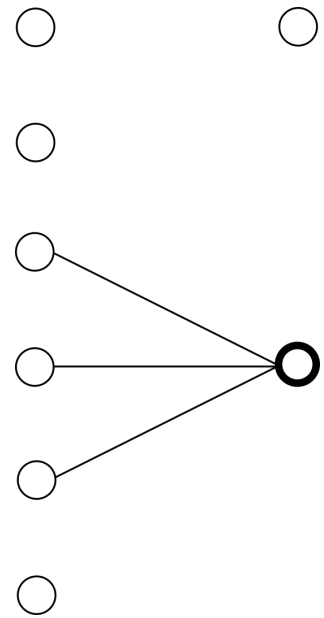
A convolutional layer can at the same time downsample the image by applying a sampling step, or *stride*.



Stride 1

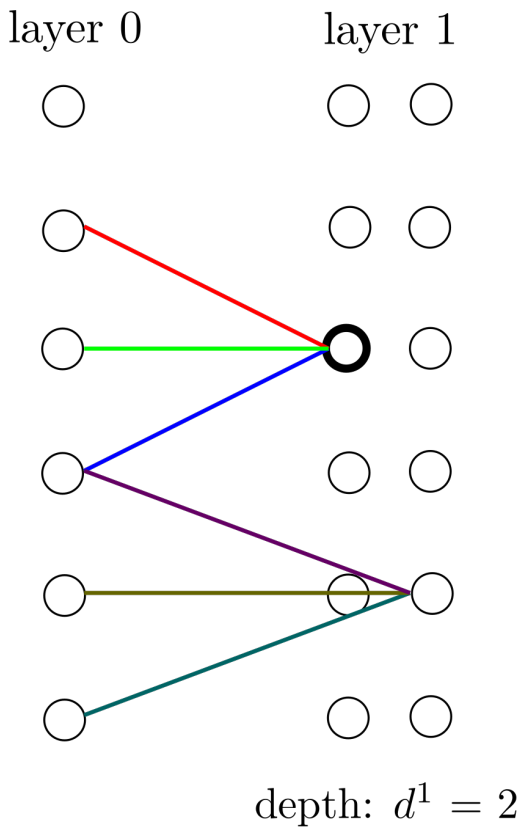


Stride 2



Stride 3

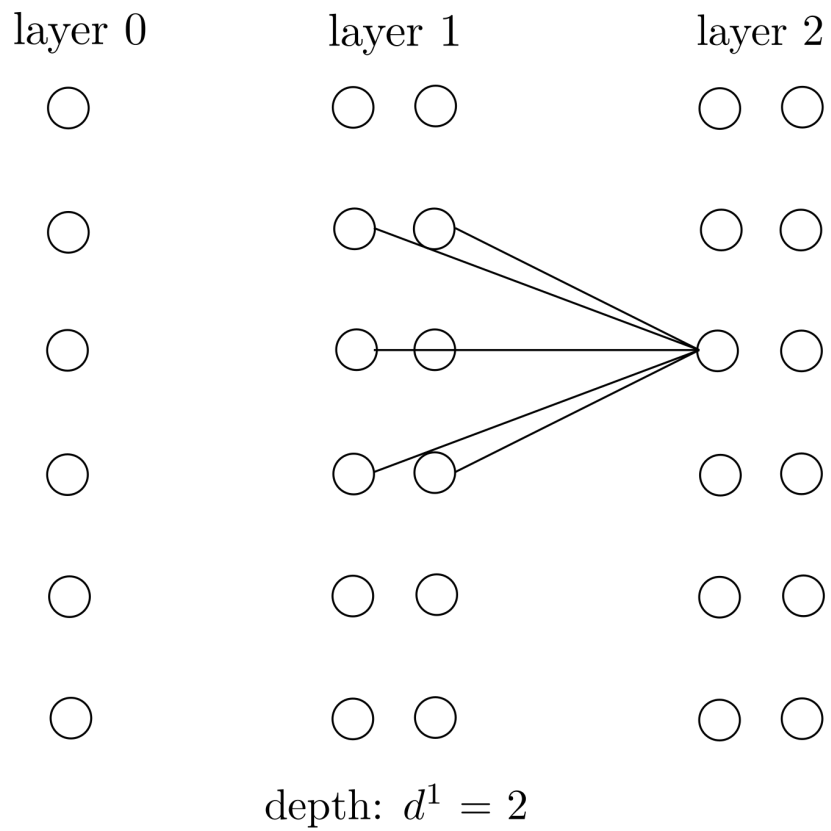
Several filters in the same convolutional layer



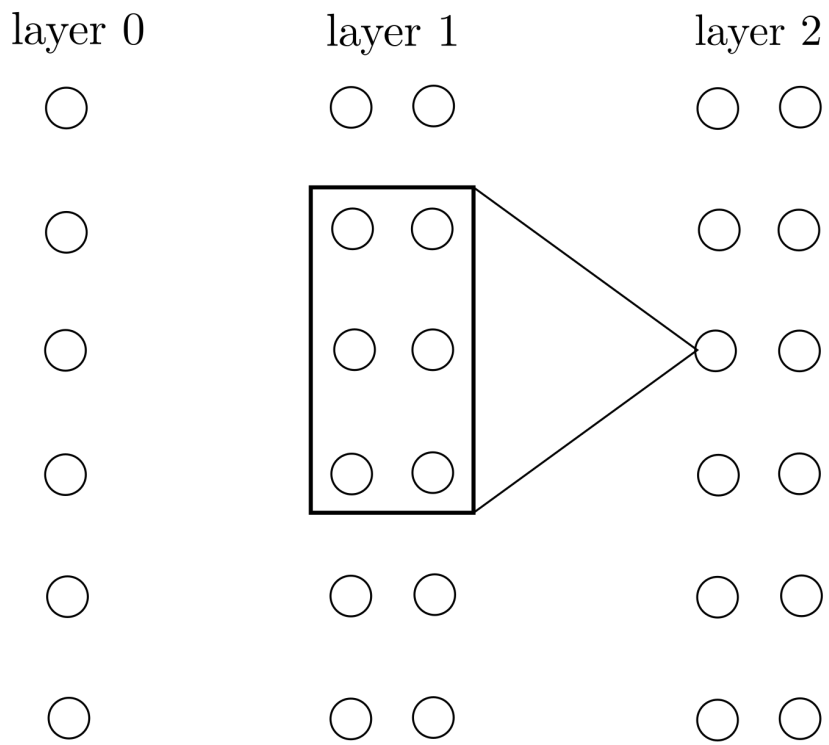
Note on vocabulary

The depth of a layer is often called the number of filters.

Several filters in the same convolutional layer

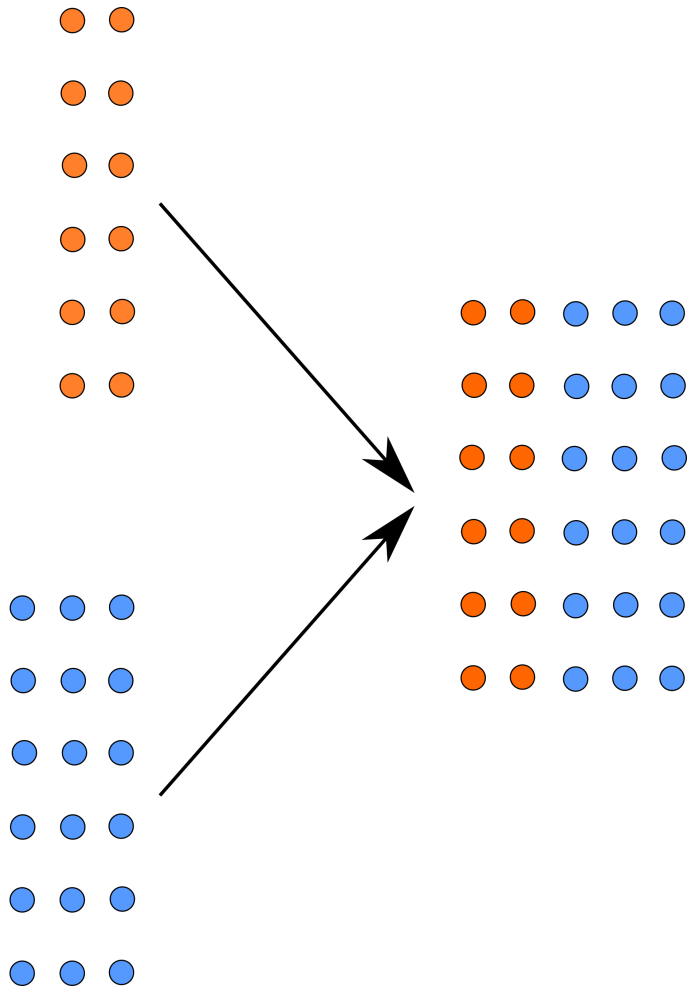


Several filters in the same convolutional layer

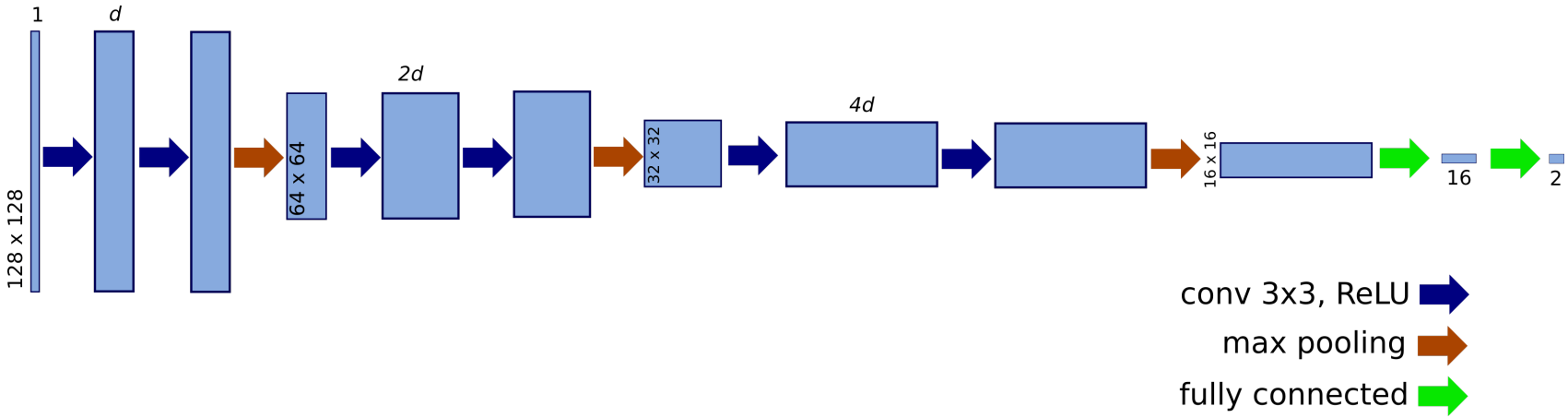


depth: $d^1 = 2$

Branch merging: concatenation

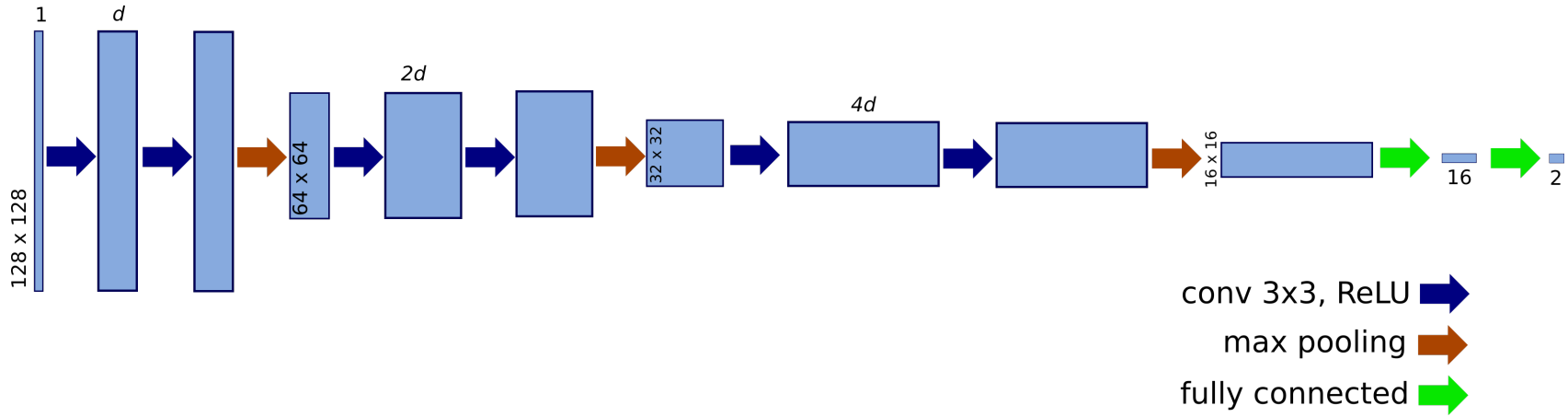


1D representations



Credits: NN is work of Robin Alais et al.
Fundus image by Mikael Häggström, used with permission (CC0).

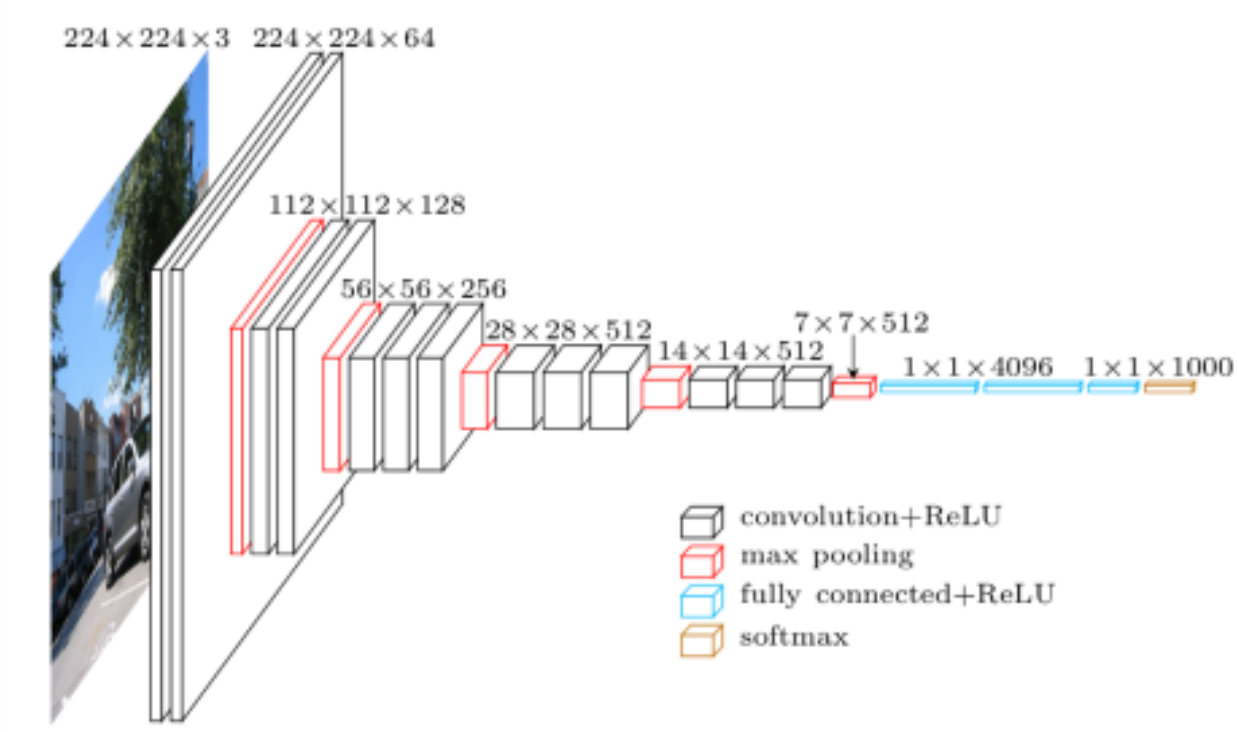
1D representations



This NN was used to estimate the position of the center of the macula on fundus images.

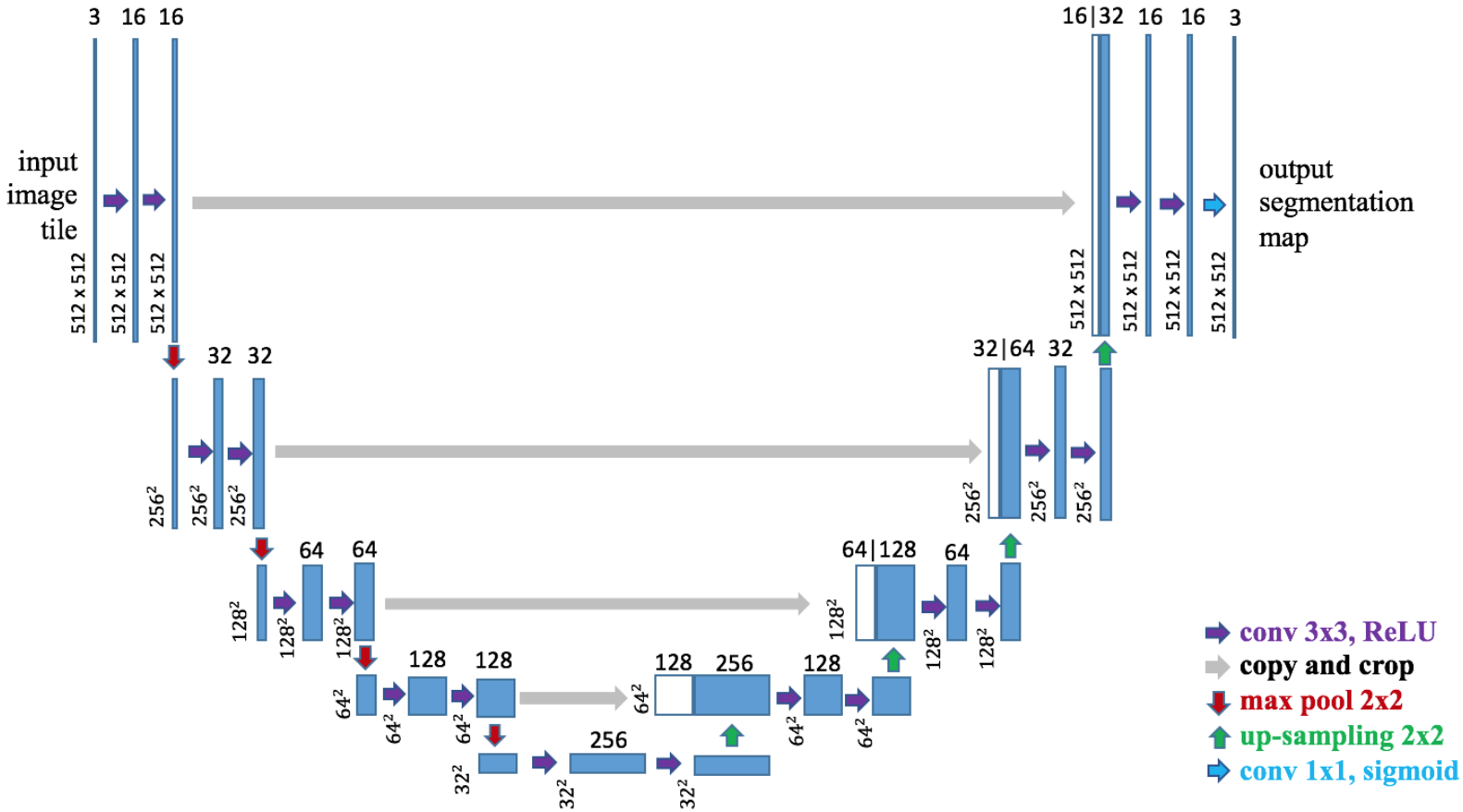
Credits: NN is work of Robin Alais et al. Fundus image by Mikael Häggström, used with permission (CC0).

2D representations



Credits: VGG16 (From https://www.cs.toronto.edu/ frossard/post/

U-Net architecture [Ronneberger et al., 2015]



Example: counting cells

image

image_60.png

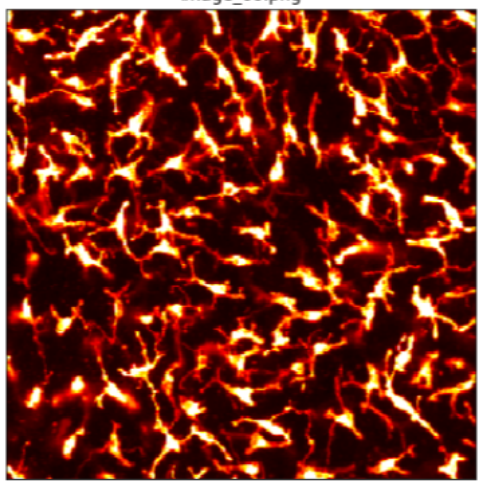
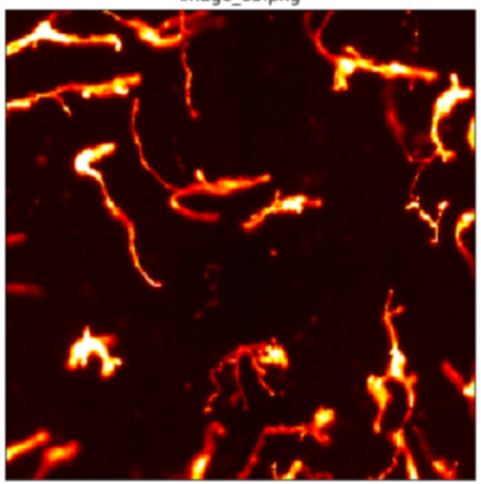
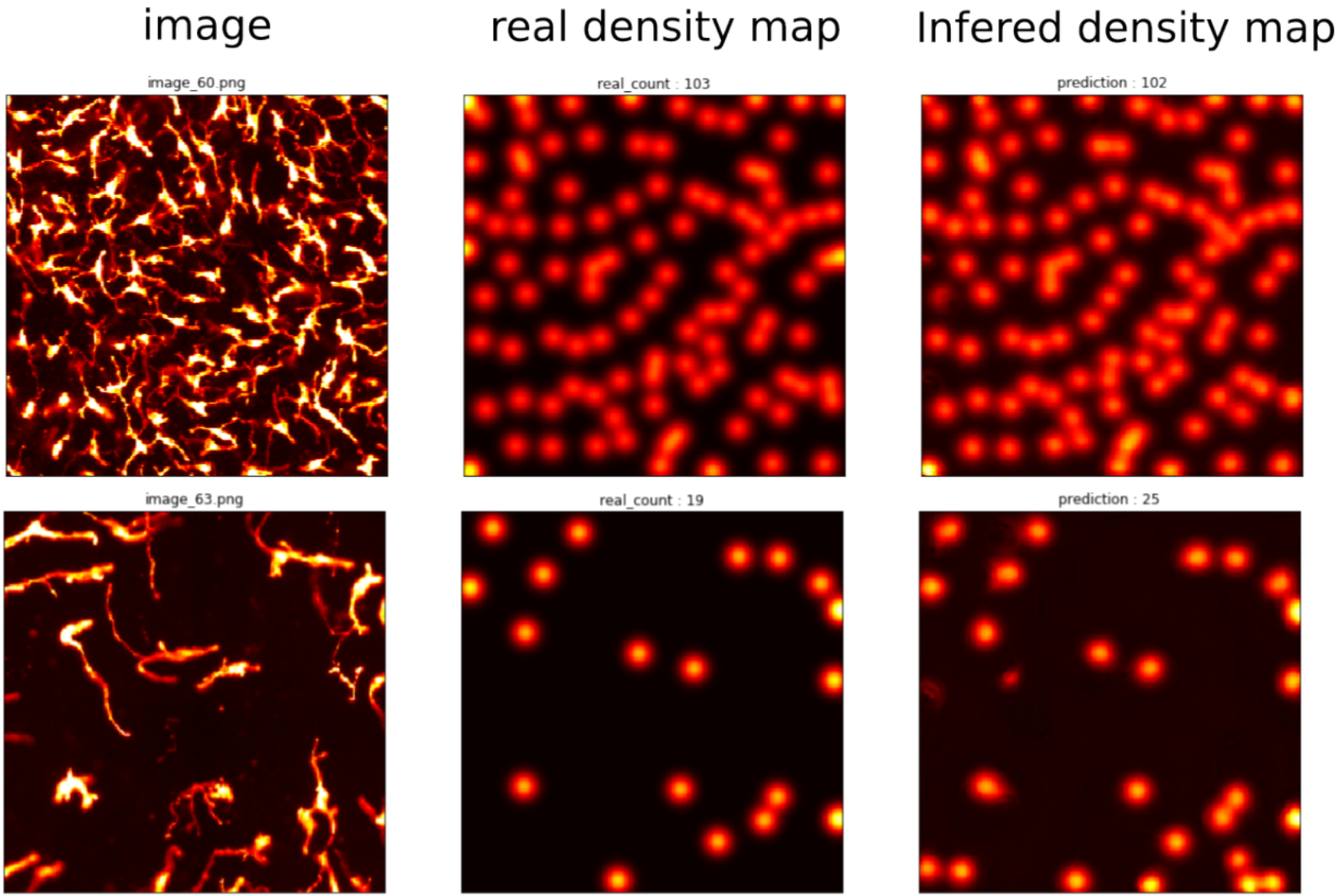


image_63.png



Credits: Tristan Lazard, master thesis. In collaboration with L'Oréal.

Counting cells

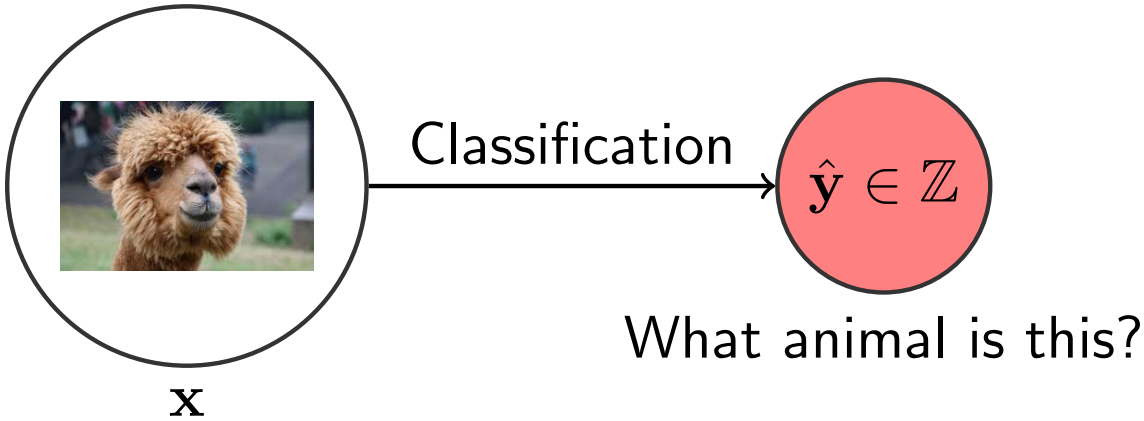


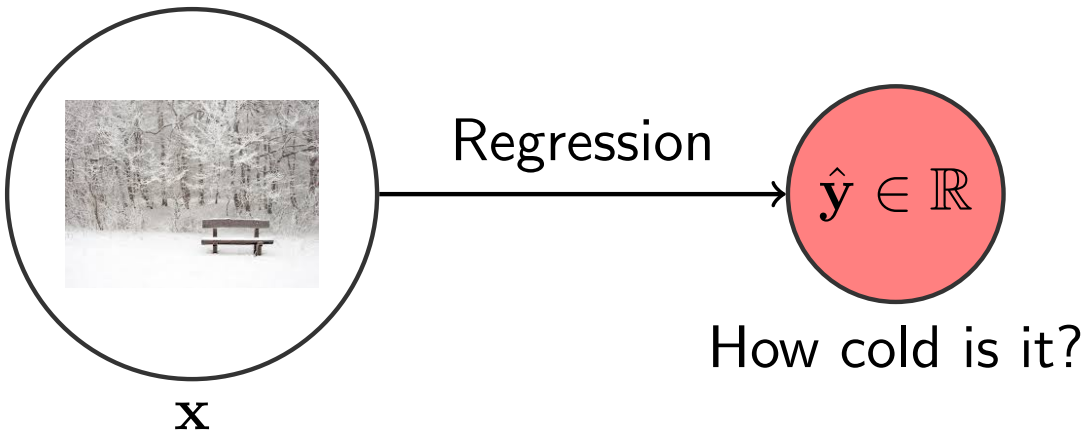
Credits: Tristan Lazard, master thesis. In collaboration with L'Oréal.

Contents

Supervised Learning

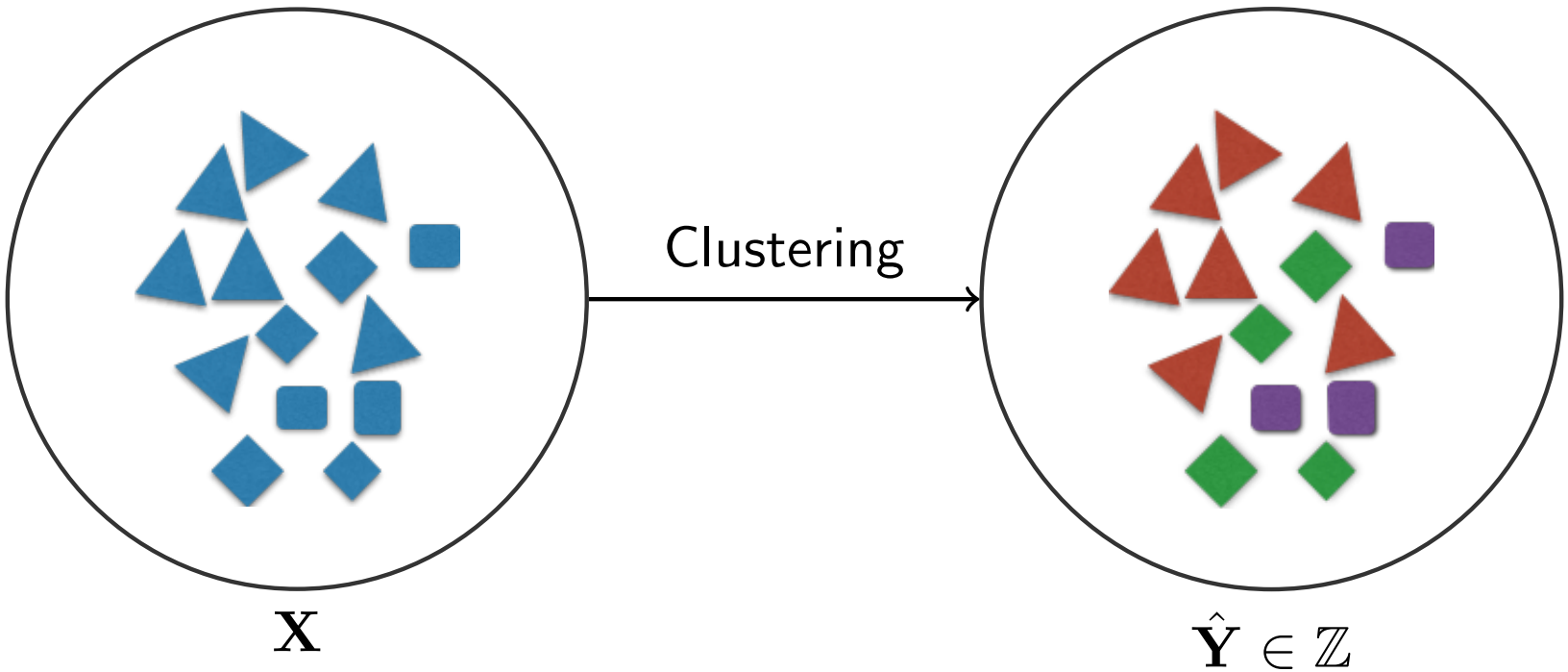
Given a labeled dataset $(\mathbf{X}, \mathbf{Y})$, we would like to learn a mapping from data space to label space.





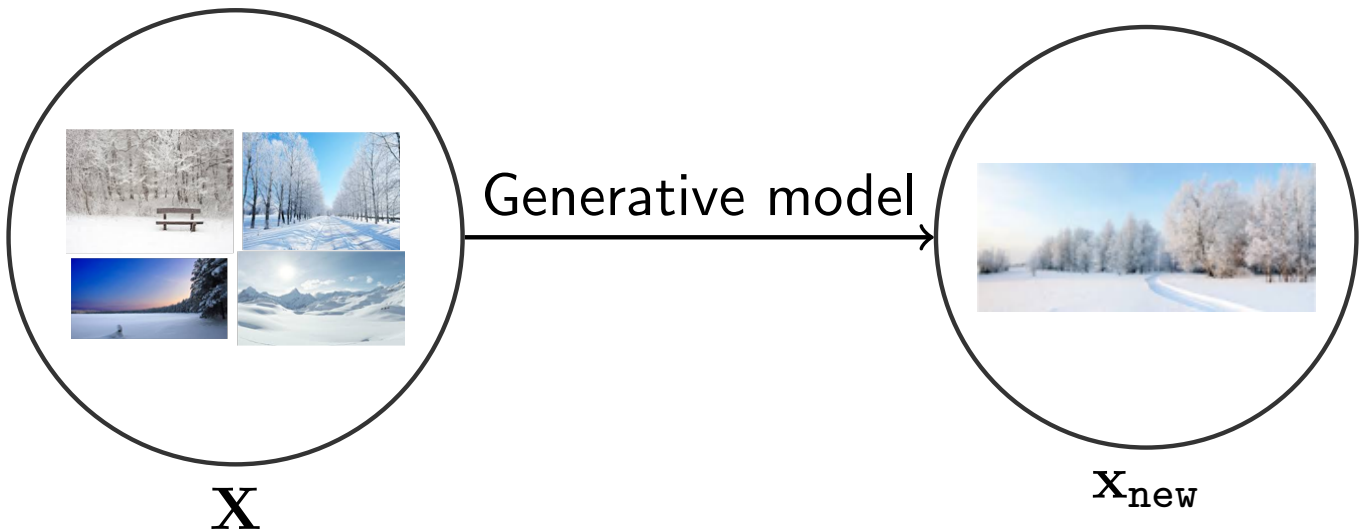
Unsupervised Learning: Clustering

Given an unlabeled dataset $(\mathbf{X})$, we would like to learn: How to group objects into categories?



Unsupervised learning: Generative Models

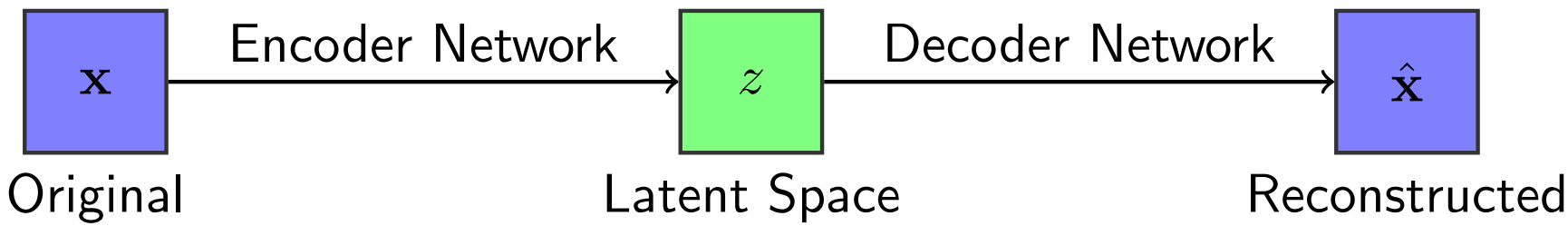
Given an unlabeled dataset $(\mathbf{X})$, we would like to learn: How to generate a new observation from the same distribution (unknown) of dataset?



Autoencoders

Autoencoders are neural networks whose purpose is twofold:

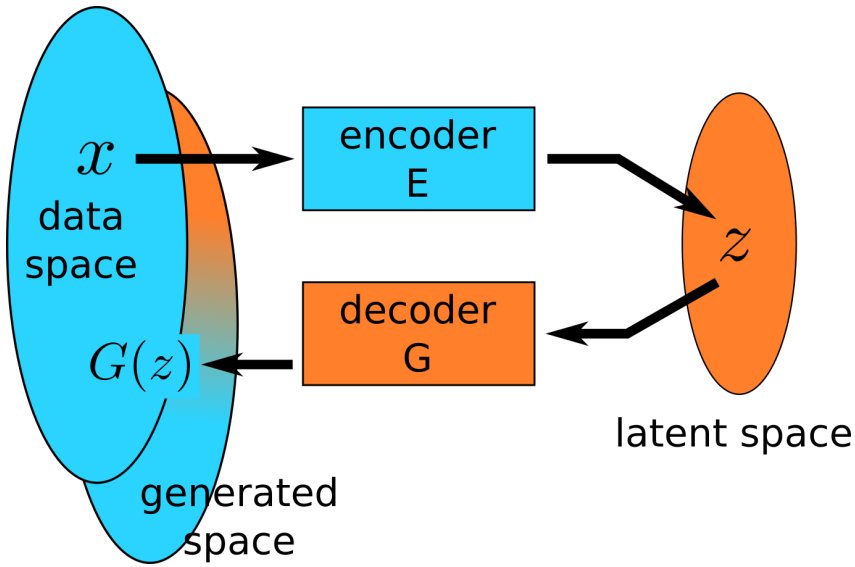
1. To compress some input data by transforming it from the input domain to another space, known as the *latent space* (code).
2. To take this latent representation and transform it back to the original space, such that the output is *similar* to the input.



The loss function for a given input vector is usually the reconstruction error:

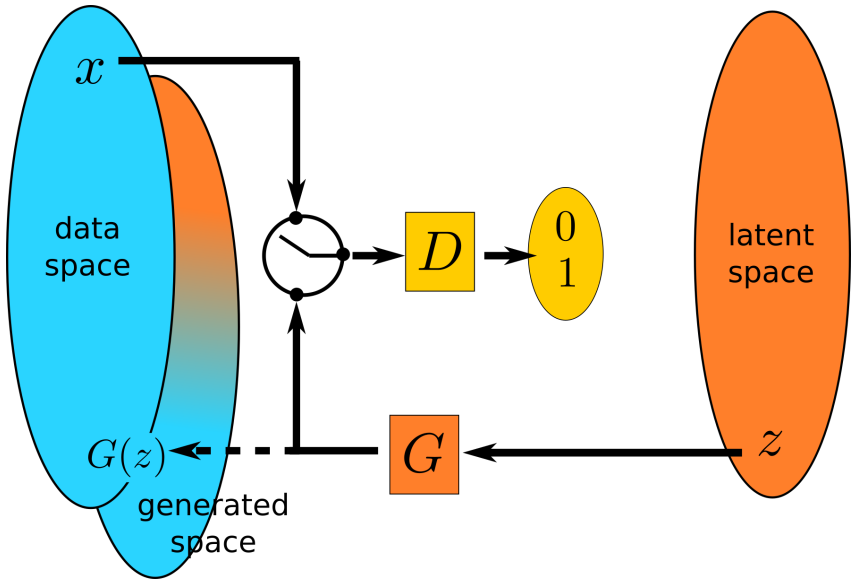
$$L(\mathbf{x}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

Autoencoders



- Encoder: E; decoder: G; autoencoder: $G \circ E$
- In most applications, the latent space is "smaller" than the data space.
- Objective: $\hat{x}$, i.e. $G \circ E(x)$, "close" to x
- When dealing with images, modern autoencoders use convolutional neural networks

Generative adversarial networks [Goodfellow et al., 2014]



- The discriminator D is optimized so that it correctly classifies images as real (1) or fake (0)
- The decoder or generator G is optimized so that the produced images are classified as real by the discriminator

Value function

$$V(G,D) = \mathbb{E}_{p_{\mathbf{x}}}(log(D(\mathbf{x}))) + \mathbb{E}_{p_z(z)}(log(1 - D(G(z))))$$

Which face is real?

Contents

Conclusion

- Deep learning allows to learn complex transformations between tensors, thanks to:
  - Smart methods and algorithms
  - Lots of data
  - Specialized hardware
- General artificial intelligence is still far away

References I

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

[LeCun, 1985] LeCun, Y. (1985). Une procedure d'apprentissage pour reseau a seuil asymmetrique (A learning scheme for asymmetric threshold networks). In *proceedings of Cognitiva 85*.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, number 9351 in Lecture Notes in Computer Science, pages 234–241. Springer International Publishing.

References II

[Simard et al., 1993] Simard, P., LeCun, Y., and Denker, J. S. (1993). Efficient pattern recognition using a new transformation distance. In *Advances in neural information processing systems*, pages 50–58.

[Werbos, 1982] Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In Drenick, R. F. and Kozin, F., editors, *System Modeling and Optimization*, Lecture Notes in Control and Information Sciences, pages 762–770. Springer Berlin Heidelberg.