Artificial neural networks and backpropagation

E. Decencière

MINES ParisTech
PSL Research University
Center for Mathematical Morphology

Contents

Contents

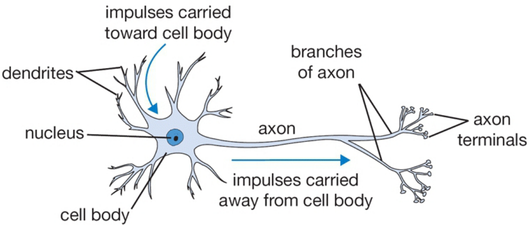
Artificial neural networks and deep learning history

For a very complete state of the art on deep learning, see the overview by Schmidhuber [Schmidhuber, 2015].

- 1958: Rosenblatt's perceptron [Rosenblatt, 1958]
- 1980's: the backpropagation algorithm (see, for example, the work of Le Cun [LeCun, 1985])
- 2006-: CNN implementations using Graphical Processing Units (GPU): up to a 50 speed-up factor.
- 2011-: super-human performances [Cireșan et al., 2011]
- 2012: Imagenet image classification won by a CNN [Krizhevsky et al., 2012].

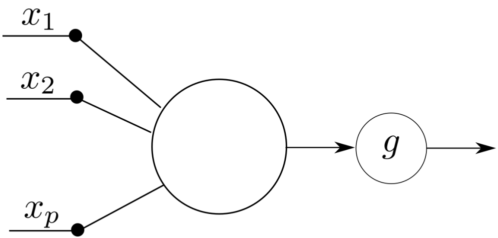
Contents

Neuron



- The human brain contains 100 billion (10^{11}) neurons
- A human neuron can have several thousand dendrites
- The neuron sends a signal through its axon if during a given interval of time the net input signal (sum on excitatory and inhibitory signals received through its dentrites) is larger than a threshold.

Artificial neuron

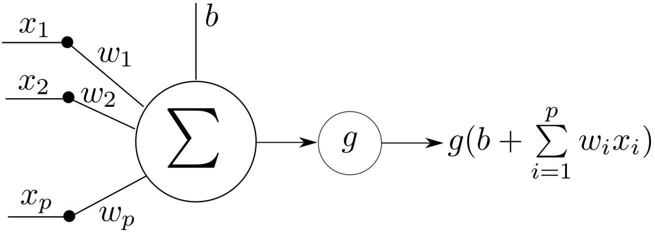


General principle

An artificial neuron takes p inputs $\{x_i\}_{1 \leq i \leq p}$, combines them to obtain a single value, and applies an activation function g to the result.

- The first artificial neuron model was proposed by [McCulloch and Pitts, 1943]
- Input and output signals were binary
- Input dendrites could be inhibitory or excitatory

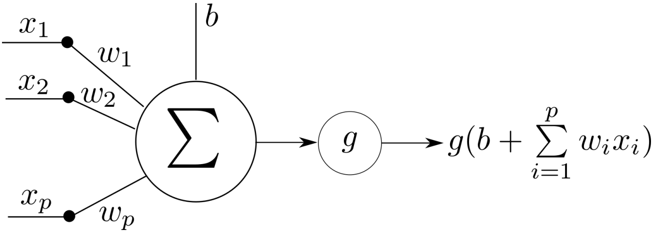
Modern artificial neuron



- The neuron computes a linear combination of the inputs x_i
  - The weights w_i are multiplied with the inputs
  - The bias b can be interpreted as a threshold on the sum
- The activation function g somehow decides, depending on its input, if a signal (the neuron's activation) is produced

Contents

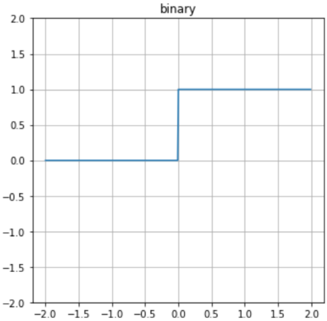
The role of the activation function



- The initial idea behind the activation function is that it works somehow as a gate
- If its input in "high enough", then the neuron is activated, i.e. a signal (other than zero) is produced
- It can be interpreted as a source of abstraction: information considered as unimportant is ignored

Activation: binary

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

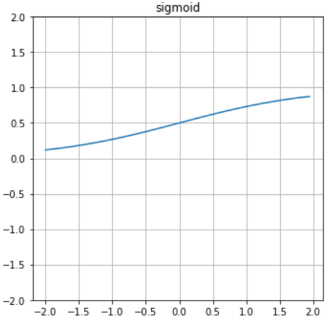


Remarks

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

Activation: sigmoid

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

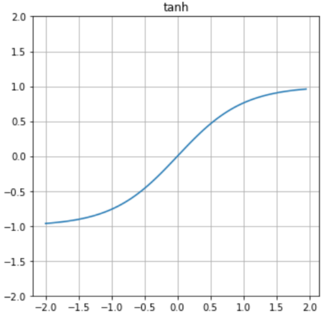


Remarks

+ 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

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

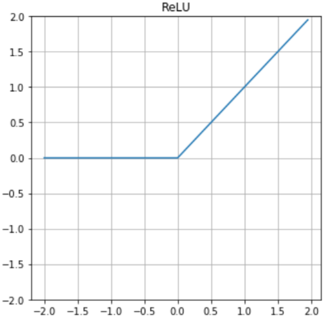


Remarks

- Similar to sigmoid

Activation: rectified linear unit

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

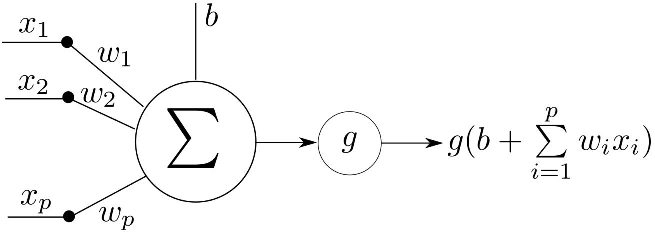


Remarks

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

Contents

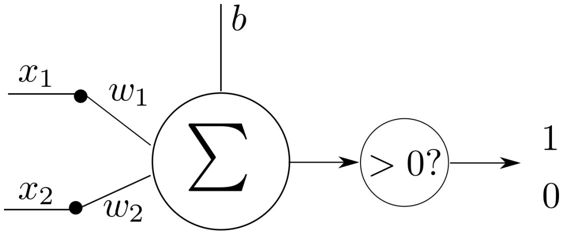
What can an artifical neuron compute?



In $\mathbb{R}^p$, $b + \sum_{i=0}^{p} w_i x_i = 0$ corresponds to a hyperplane. For a given point $\mathbf{x} = \{x_0, \ldots, x_p\}$, decisions are made according to the side of the hyperplane it belongs to.

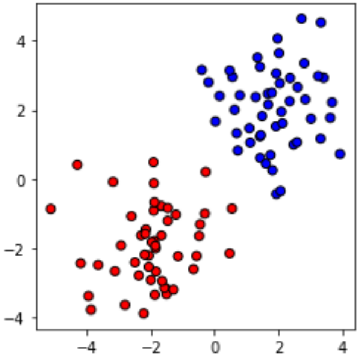
When the activation function is binary, we obtain a perceptron

Example of what we can do with a neuron

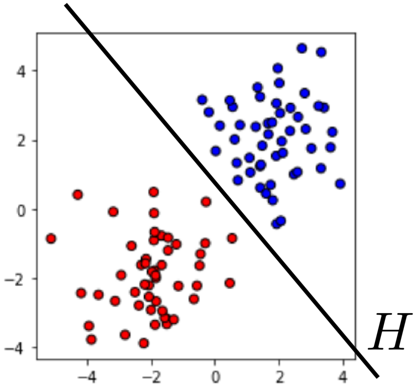


- $p = 2$: 2 dimensional inputs (can be represented on a screen!)
- Activation: binary
- Classification problem

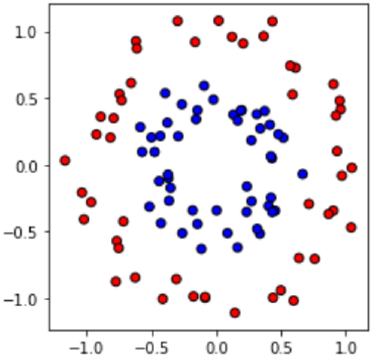
Gaussian clouds



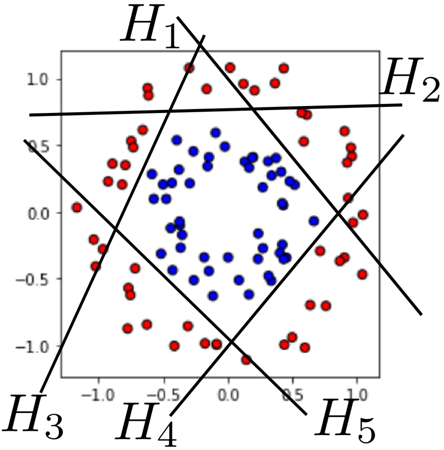
Gaussian clouds



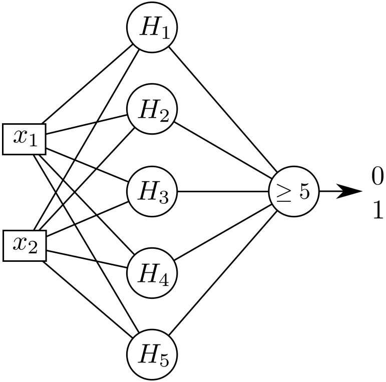
Circles



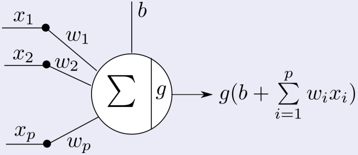
Circles



Solution



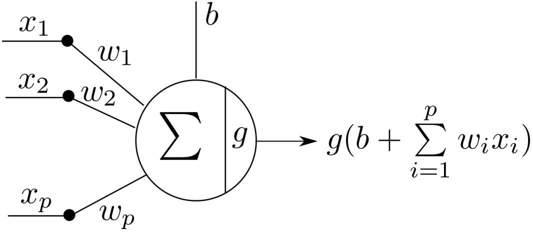
Artificial neuron compact representation



Contents

Contents

Notations



We will often use:

$$\mathbf{W} = (w_1, \ldots, w_p)^T$$

$$\mathbf{x} = (x_1, \ldots, x_p)^T$$

Therefore, we can simply write:

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

Neural network (NN)

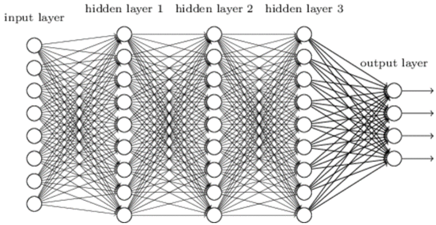
Definitions

- An (artificial) neural network is a directed graph, where:
  - the nodes are articial neurons and
  - the edges are connections between the neurons.
- The input layer is the set of neurons without incoming edges.
- The ouput layer is the set of neurons without outgoing edges.

Feed-forward neural networks

Definition

- A feed-forward neural networks is a NN without cycles
- A neuron belongs to layer q if the longest path in the graph between the input layer and the neuron is of length q.
- Any layers other than input and output layers are called hidden layers



(from http://www.jtoy.net)

Feed-forward neural networks

In the following of this course, except when otherwise specified, all NNs will be feed-forward. Indeed, this is the preferred type of NN for image processing.

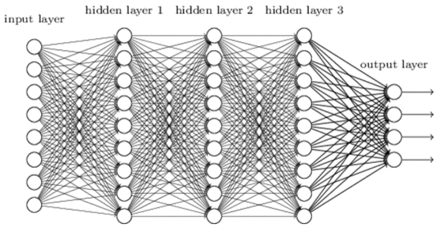
What about other architectures?

- Recurrent neural networks (RNN)
- Long short-term memory networks (LSTM)

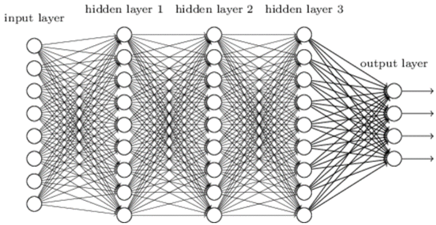
- More powerful than feed-forward NNs
- More biologically realistic
- Complex dynamics; more difficult to train
- Mainly used for processing temporal data

Fully-connected network

- A layer is said to be fully-connected if each of its neurons is connected to all the neurons of the previous and following layers
- A NN is said to be fully connected if all its hidden layers are fully connected

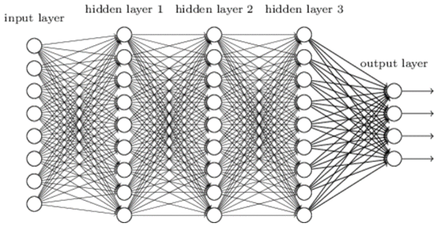


Number of parameters



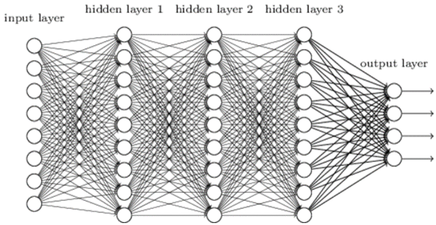
- How many parameters does the above network contain?

Number of parameters



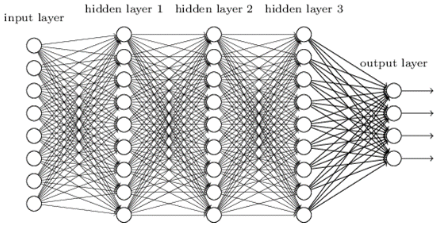
- How many parameters does the above network contain?
- First hidden layer:

Number of parameters



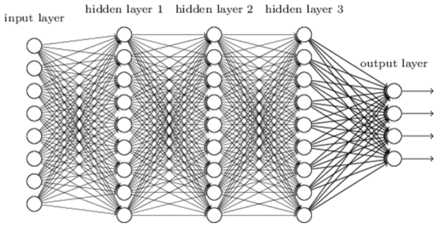
- How many parameters does the above network contain?
- First hidden layer:
- 9 neurons $\times 8$ neurons in the previous layer $+9$ biases $= 81$

Number of parameters



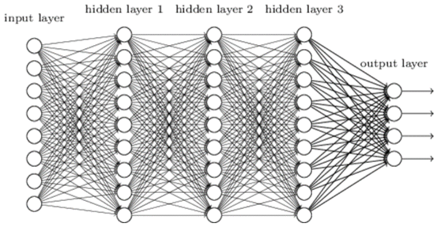
- How many parameters does the above network contain?
- First hidden layer:
- 9 neurons $\times 8$ neurons in the previous layer $+9$ biases $= 81$
- Second and third layers: $9 \times 9 + 9 = 90$

Number of parameters



- How many parameters does the above network contain?
- First hidden layer:
- 9 neurons $\times 8$ neurons in the previous layer $+9$ biases $= 81$
- Second and third layers: $9 \times 9 + 9 = 90$
- Output layer: $4 \times 9 + 4$

Number of parameters

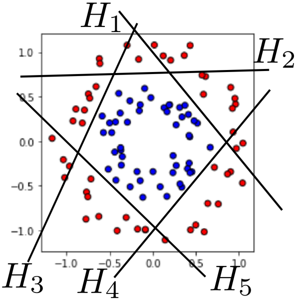


- How many parameters does the above network contain?
- First hidden layer:
- 9 neurons $\times 8$ neurons in the previous layer $+9$ biases $= 81$
- Second and third layers: $9 \times 9 + 9 = 90$
- Output layer: $4 \times 9 + 4$
- Total: 305 parameters

Contents

Universal approximation theorem

- We have previously seen that a neuron can be used as a linear classifier and that combining several of them one can build complex classifiers
- We will see that this observation can be generalized



Universal approximation theorem

Let f be a continuous real-valued function of $[0, 1]^p$ ($p \in \mathbb{N}^*$) and ϵ a strictly positive real. Let g be a non-constant, increasing, bounded real function (*the activation function*).
Then there exist an integer n, real vectors $\{\mathbf{W}_i\}_{1 \leq n}$ of $\mathbb{R}^p$, and reals $\{b_i\}_{1 \leq n}$ and $\{v_i\}_{1 \leq n}$ such that for all $\mathbf{x}$ in $[0, 1]^p$:

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} v_i \mathrm{g}(\mathbf{W}_i^T \mathbf{x} + b_i) \right| < \epsilon$$

A first version of this theorem, using sigmoidal activation functions, was proposed by [CYBENKO, 1989]. The version above was demonstrated by [Hornik, 1991].

Universal approximation theorem: what does it mean?

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} v_i \mathbf{g}(\mathbf{W}_i^T \mathbf{x} + b_i) \right| < \epsilon$$

This means that function f can be approximated with a neural network containing:

- an input layer of size p;
- a hidden layer containing n neurons with activation function g, weights $\mathbf{W}_i$ and biases b_i;
- an output layer containing a single neuron, with weigths v_i (and an identity activation function).

Universal approximation theorem in practice

- The number of neurons increases very rapidly with the complexity of the function
- Empirical evidence has shown that multi-layer architectures give better results

Universal approximation theorem in practice

- The number of neurons increases very rapidly with the complexity of the function
- Empirical evidence has shown that multi-layer architectures give better results

A NN can potentially have a lot of parameters. How can we set them?

Contents

Introduction

- We have seen that NNs have a lot of potential. However, how can the parameters $\boldsymbol{\theta} = (\mathbf{W}_i, \mathbf{b}_i)$ be set?
- What is our objective ?
- A very general solution, that is also the mostly used, is gradient descent

Learning problem

We recall that our training set contains n samples:

$$(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$$

We choose a family $f_{\boldsymbol{\theta}}$ of functions from $\mathbb{R}^p$ into $\mathbb{R}$, depending on our set of parameters $\boldsymbol{\theta}$, and find the value of $\boldsymbol{\theta}$ that minimizes a chosen loss function L:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}(L(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta}))$$

where $\mathcal{R}(\boldsymbol{\theta})$ is a regularization term.

For the time being, for the sake of simplicity, we will drop the regularization term until further notice

Loss function

A general form of the loss function is:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} d(y_i, f(\mathbf{x}_i, \theta))$$

where d is some disparity function (the more similar its parameters, the smaller its value).

Loss function: examples

Squared error

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \theta))^2$$

This loss function is mainly used in regression problems. However, it has also been used for binary classification problems.

Cross-entropy

In this case, $\mathbf{x}_i \in \{0, 1\}^p$ and $y_i \in [0, 1]$:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i ln(f(\mathbf{x}_i, \theta))$$

This loss function is used in binary classification problems, where the network's output can be interpreted as a probability of belonging to a class.

Gradient descent

Definition

Gradient descent is an optimization algorithm. For a derivable function L, a positive real γ (the learning rate) and a starting point $\boldsymbol{\theta}_0$, it computes a sequence of values:

$$\forall i \in \mathbb{N} : \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma \nabla L(\boldsymbol{\theta}_i)$$

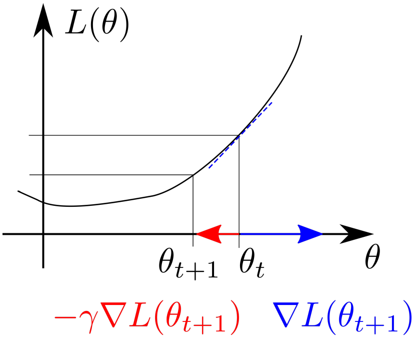
Property

If γ is small enough, then:

$$L(\boldsymbol{\theta}_{i+1}) \leq L(\boldsymbol{\theta}_i)$$

Gradient descent is an essential tool in optimization.

Gradient descent in the scalar case



$$\theta_{t+1} = \theta_t - \gamma\nabla L(\theta_t)$$

Gradient descent applied to neural networks

In the case of neural networks, the loss L depends on each parameter θ_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

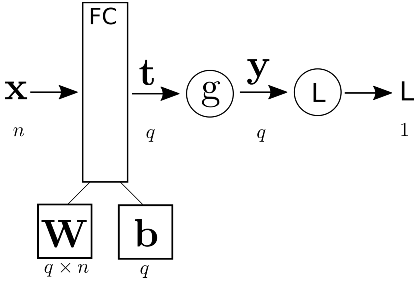
Let us introduce variables, x, y and z:

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

Then:

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

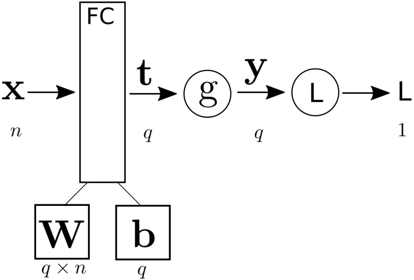
Backpropagation through a fully connected layer



Setup:

$$
\begin{aligned}
n, q &\in \mathbb{N}^* \\
\mathbf{x} &\in \mathbb{R}^n \\
\mathbf{W} &\in \mathbb{R}^q \times \mathbb{R}^n \\
\mathbf{b}, \mathbf{t}, \mathbf{y} &\in \mathbb{R}^q \\
L &\in \mathbb{R}
\end{aligned}
$$

Backpropagation through a fully connected layer



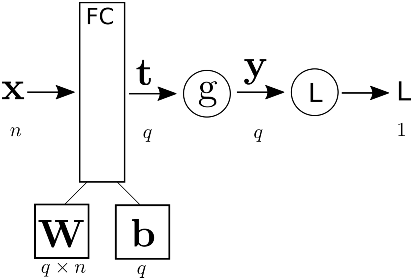
Local gradients:

$$
\begin{aligned}
\frac{\partial \mathbf{t}}{\partial \mathbf{W}} &= \mathbf{x}^t \\
\frac{\partial \mathbf{t}}{\partial \mathbf{b}} &= 1 \\
\frac{\partial \mathbf{y}}{\partial \mathbf{t}} &= \mathrm{g}'
\end{aligned}
$$

Forward pass:

$$
\begin{aligned}
\mathbf{t} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\
\mathbf{y} &= \mathrm{g}(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
L &= L(\mathbf{y})
\end{aligned}
$$

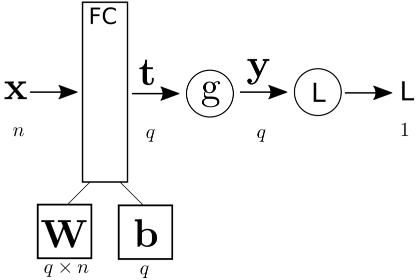
Backpropagation through a fully connected layer



Backpropagation:

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{t}} &= \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{t}} \\
&= \frac{\partial L}{\partial \mathbf{y}} \odot \mathrm{g}'(\mathbf{t})
\end{aligned}
$$

Backpropagation through a fully connected layer



Backpropagation:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{t}} . \frac{\partial \mathbf{t}}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{g}'(\mathbf{t}) . \mathbf{x}^t$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{g}'(\mathbf{t})$$

Contents

The triggering factor to the success of neural networks

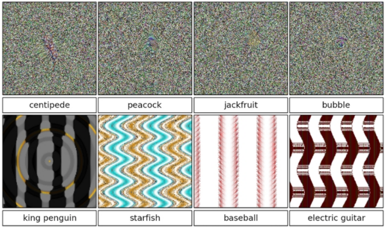
- Appropriate architectures: graphical processing units (GPUs)
- Optimized software
- Large annotated databases

Practical considerations

For a deep-learning solution to work, you need:

- A lot of annotated data
- A lot of fiddling (different architectures; hyper-parameters)
- GPUs, at least from training

Deep learning can produce astonishing results [Nguyen et al., 2015]...



The web giants

- Google, Facebook, Microsoft, Amazon etc. are actively investing in deep-learning
- Competition is intense
- Most of them are sharing their deep learning libraries

References I

[Cireşan et al., 2011] Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE.

[CYBENKO, 1989] CYBENKO, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

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

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

References II

[Nguyen et al., 2015] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.