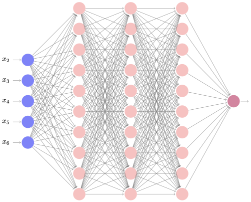


Perceptrons and Multilayer Perceptrons

Vincent Barra
LIMOS, UMR 6158 CNRS, Université Clermont Auvergne

Perceptron

Multilayer perceptrons

LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Threshold Logic Unit

Mc Culloch and Pitts, 1943

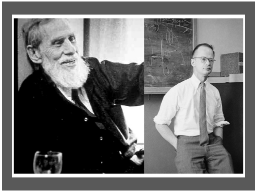
First mathematical model for a neuron
For $\boldsymbol{x}$ boolean vector, $w, b \in \mathbb{R}$:

$$f(x) = \mathbb{1}_{\{w \sum_i x_i + b \geq 0\}}$$

and in particular

- $OR(x, y) = \mathbb{1}_{\{x+y-0.5 \geq 0\}}$
- $AND(x, y) = \mathbb{1}_{\{x+y-1.5 \geq 0\}}$
- $NOT(x) = \mathbb{1}_{\{-x+0.5 \geq 0\}}$

Any Boolean function can be build with such units.



LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

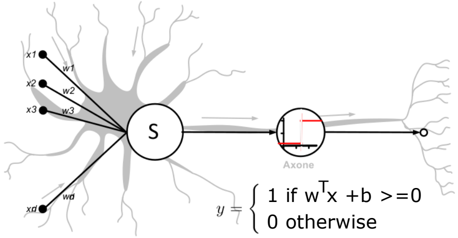
PERCEPTRON

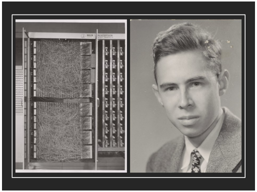
Rosenblatt 1957

Generalization: $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^d, \; b \in \mathbb{R}$

$$f(x) = \mathbb{1}_{\{\boldsymbol{w}^T \boldsymbol{x} + b \geq 0\}}$$

Relation to biology





LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES
cnrs
UCA UNIVERSITÉ Clermont Auvergne

Perceptron

A more general view

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$$

where

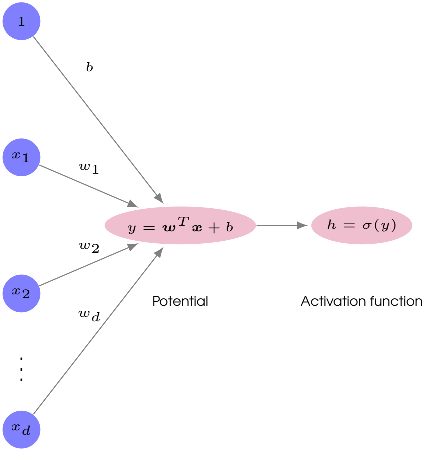
- $\boldsymbol{w}$: synaptic weights
- b: bias
- $\boldsymbol{w}^T\boldsymbol{x}$: post synaptic potential
- σ: activation function

LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Representing the Perceptron

Graphical representations

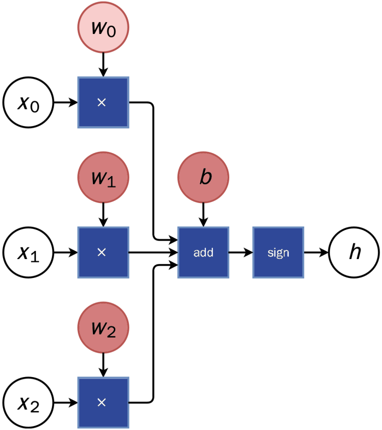
1. "Neural" representation
2.



Representing the Perceptron

Graphical representations

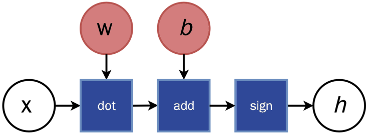
1. "Neural" representation
2. Computational graph
   - white nodes: inputs and outputs
   - red nodes: model parameters
   - blue nodes: operations



Representing the Perceptron

Graphical representations

1. "Neural" representation
2. Computational graph
   - white nodes: inputs and outputs
   - red nodes: model parameters
   - blue nodes: operations

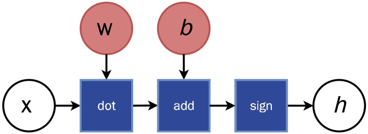


LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Representing the Perceptron

Basic brick

This unit is the basic brick of all neural networks



Learning the perceptron

Problem statement

How to build the model ?

- Input: Learning set $Z = \{(\boldsymbol{x}_i, y_i), i \in [\![1 \cdots n]\!], \boldsymbol{x}_i \in \mathbb{R}^{d+1}, y_i \in \mathbb{R}\}$
- Unknown: $\boldsymbol{w} \in \mathbb{R}^{d+1}$

LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

LEARNING THE PERCEPTRON

Problem statement

How to build the model ?

- Input: Learning set $Z = \left\{(\boldsymbol{x}_i, y_i), i \in [\![1 \cdots n]\!], \boldsymbol{x}_i \in \mathbb{R}^{d+1}, y_i \in \mathbb{R}\right\}$
- Unknown: $\boldsymbol{w} \in \mathbb{R}^{d+1}$

Key Idea

For each $\boldsymbol{x}_i \in Z$:

- expected output: y_i
- computed output: $h_i = \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) = f_{\boldsymbol{w}}(\boldsymbol{x})$

If $\mathcal{L} : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \to \mathbb{R}$ is a loss function

$$\hat{\boldsymbol{w}} = Arg \min_{\boldsymbol{w}} \sum_{(\boldsymbol{x}, y) \in Z} \mathcal{L}\left(f_{\boldsymbol{w}}(\boldsymbol{x}), y\right)$$

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

Examples of loss functions

Binary classification (-1/1)

1. Characteristic function: $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = \mathbb{1}_{y f_{\boldsymbol{w}}(x) \leq 0}$
2. Logistic loss : $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = ln\left(1 + e^{-y f_{\boldsymbol{w}}(x)}\right)$
3. binary cross-entropy: $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = -\left(y log(f_{\boldsymbol{w}}(x)) + (1-y) log(1 - f_{\boldsymbol{w}}(x))\right)$

Regression

1. Hinge loss : $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = (1 - y f_{\boldsymbol{w}}(x))_+ = max\,(0, 1 - y f_{\boldsymbol{w}}(x))$
2. MSE (L_2 loss) : $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = \| f_{\boldsymbol{w}}(x) - y \|^2$
3. Huber loss : $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = \begin{cases} \frac{1}{2\epsilon}(f_{\boldsymbol{w}}(x) - y)^2 & \text{if } |f_{\boldsymbol{w}}(x) - y| \geq \epsilon \\ 0 & \text{otherwise} \end{cases}$
4. Vapnik loss: $\mathcal{L}(f_{\boldsymbol{w}}(x), y) = \begin{cases} 0 & \text{if } |f_{\boldsymbol{w}}(x) - y| \leq \epsilon \\ |f_{\boldsymbol{w}}(x) - y| - \epsilon & \text{otherwise} \end{cases}$

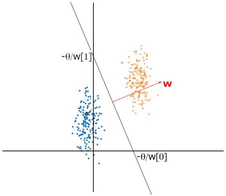
FIRST TRAINING ALGORITHM

Here, $\sigma(x) \in \{-1, 1\}$
Given a training set

$$Z = \{(\boldsymbol{x}_i, y_i), i \in [\![1 \cdots n]\!], \boldsymbol{x}_i \in \mathbb{R}^{d+1}, y_i \in \{-1, 1\}\}$$

this linear operator can be trained for a binary classification problem.

$\boldsymbol{w}^0 = \boldsymbol{0}$
$k = 0$
while $\exists i$ *such that* $y_i((\boldsymbol{w^k})^T \boldsymbol{x}_i) \leq 0$ **do**
  $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + y_i \boldsymbol{x}_i$
  $k = k + 1$
end



First training algorithm

Convergence iff:

- Points lie in a sphere of radius R:

$$(\forall i \in [\![1 \cdots n]\!])\ \|\boldsymbol{x}_i\| \leq R$$

- The two classes can be separated by a margin:

$$\exists \tilde{\boldsymbol{w}}, \|\tilde{\boldsymbol{w}}\| = 1\ \exists \gamma > 0,\ (\forall i \in [\![1 \cdots n]\!])\ \ y_i(\tilde{\boldsymbol{w}}^T \boldsymbol{x}_i) \geq \gamma/2$$

If so, the perceptron stops as soon as it finds a separating hyperplane.

FIRST TRAINING ALGORITHM

Convergence iff:

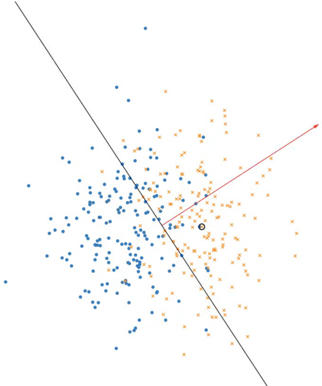
- Points lie in a sphere of radius R:

$$(\forall i \in [\![1 \cdots n]\!])\ \|\boldsymbol{x}_i\| \leq R$$

- The two classes can be separated by a margin:

$$\exists \tilde{\boldsymbol{w}}, \|\tilde{\boldsymbol{w}}\| = 1\ \exists \gamma > 0,\ (\forall i \in [\![1 \cdots n]\!])\ \ y_i(\tilde{\boldsymbol{w}}^T \boldsymbol{x}_i) \geq \gamma/2$$

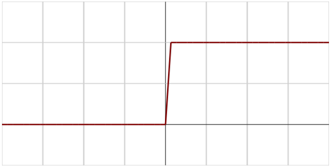
If so, the perceptron stops as soon as it finds a separating hyperplane. But what if the data is non linearly separable ?



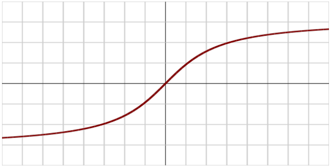
Second training algorithm

One possible solution: minimize the amount of errors.

1. Change σ function to make it differentiable



$\rightarrow$



2. Error

$$\ell(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in Z} \mathcal{L}\left(f_{\boldsymbol{w}}(\boldsymbol{x}), y\right)$$

3. Minimize the error w.r.t $\boldsymbol{w}$.

Second training algorithm

Gradient

At a local minimum the gradient is null: $\sum_{(\boldsymbol{x},y)\in Z} \nabla_{\boldsymbol{w}} \mathcal{L}\left(f_{\boldsymbol{w}}(\boldsymbol{x}), y\right) = \mathbf{0}$

SECOND TRAINING ALGORITHM

Gradient

At a local minimum the gradient is null: $\sum_{(\boldsymbol{x},y)\in Z} \nabla_{\boldsymbol{w}} \mathcal{L}\left(f_{\boldsymbol{w}}(\boldsymbol{x}), y\right) = \boldsymbol{0}$

Gradient Descent Algorithm

1. Initialization: $\boldsymbol{w} = \boldsymbol{w}_0$, $k = 0$
2. While (non stop)
   - 2.1 $\boldsymbol{g}_k = \frac{1}{|Z|} \sum_{(\boldsymbol{x},y)\in Z} \nabla_{\boldsymbol{w}} \mathcal{L}\left(f_{\boldsymbol{w_k}}(\boldsymbol{x}), y\right)$
   - 2.2 $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \boldsymbol{g}_k$
   - 2.3 $k = k + 1$

Additional ressource

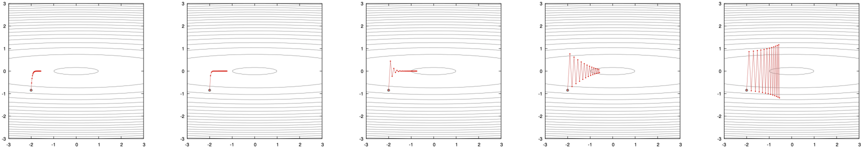
See Slides “toy example” and “Optimization for deep Learning”.

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES
UNIVERSITÉ Clermont Auvergne

SECOND TRAINING ALGORITHM

Algorithm parameters:

- stopping criterion
- η: learning rate
- Weight initialization



$\eta = 10^{-2}$ $\eta = 2.10^{-2}$ $\eta = 4.10^{-2}$ $\eta = 5.10^{-2}$ $\eta = 5.310^{-2}$

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

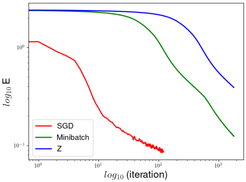
cnrs

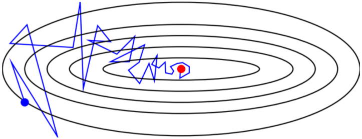
UCA UNIVERSITÉ Clermont Auvergne

SECOND TRAINING ALGORITHM

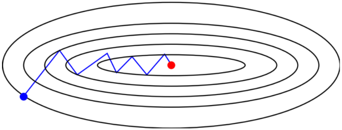
Different learning strategies

- Compute the error over all Z: real gradient descent
- Compute the error on one example only: stochastic gradient descent (SGD)
- Compute the error on a batch of example: batch learning (minibatch)





SGD



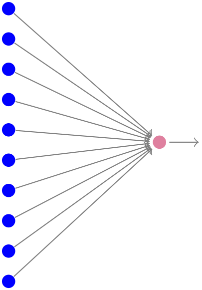
minibatch

But...
If we want to accurately classify the data (and allow a good generalization property), we need to find something else...

Stacking linear classifiers

A linear classifier of the form

$$\begin{aligned} f : \mathbb{R}^{d+1} &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \end{aligned}$$

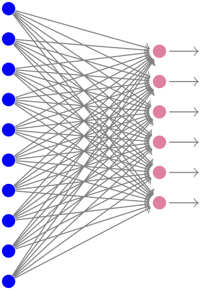


Stacking linear classifiers

A linear classifier of the form

$$\begin{aligned} f : \mathbb{R}^{d+1} &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \end{aligned}$$

can naturally be component-wise extended to any function $f : \mathbb{R}^{d+1} \to \mathbb{R}^c$

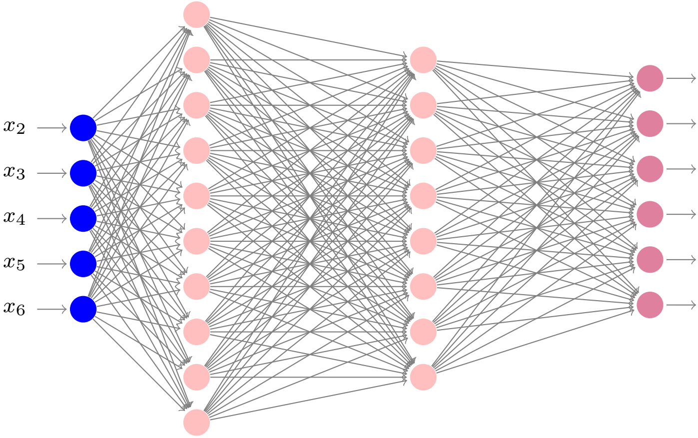


LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

And even...



LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne

The general structure can be defined using $\boldsymbol{x}^{(0)} = \boldsymbol{x}$ and

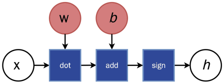
$$(\forall l \in [\![1 \cdots L]\!]) \quad \boldsymbol{x}^{(l)} = \sigma(\boldsymbol{w}^{(l)^T} \boldsymbol{x}^{(l-1)} + b^{(l)})$$

This is a *Multilayer Perceptron (MLP)*.

Building complex neural networks

$$h = \sigma(\boldsymbol{w}^T x + b)$$

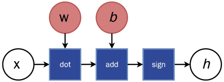
$h \in \mathbb{R}$,
$\boldsymbol{w}, x \in \mathbb{R}^{d+1}$
$b \in \mathbb{R}$



Building complex neural networks

$$h = \sigma(\boldsymbol{w}^T x + b)$$

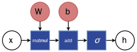
$h \in \mathbb{R}$,
$\boldsymbol{w}, x \in \mathbb{R}^{d+1}$
$b \in \mathbb{R}$



$\xrightarrow{\text{Parallel composition}}$

$$\boldsymbol{h} = \sigma(\boldsymbol{W}^T x + \boldsymbol{b})$$

$\boldsymbol{h} \in \mathbb{R}^q$
$\boldsymbol{W} \in \mathcal{M}_{d+1,q}(\mathbb{R})$
$\boldsymbol{b} \in \mathbb{R}^q$,
σ element-wise function

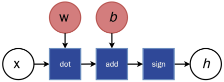


LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES
cnrs
UCA UNIVERSITÉ Clermont Auvergne

Building complex neural networks

$$h = \sigma(\boldsymbol{w}^T x + b)$$

$h \in \mathbb{R}$,
$\boldsymbol{w}, x \in \mathbb{R}^{d+1}$
$b \in \mathbb{R}$

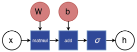


$\xrightarrow{\text{Parallel composition}}$

100× speed up

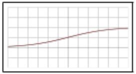
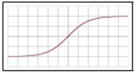
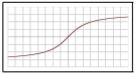
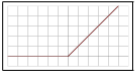
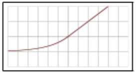
$$\boldsymbol{h} = \sigma(\boldsymbol{W}^T x + \boldsymbol{b})$$

$\boldsymbol{h} \in \mathbb{R}^q$
$\boldsymbol{W} \in \mathcal{M}_{d+1,q}(\mathbb{R})$
$\boldsymbol{b} \in \mathbb{R}^q$,
σ element-wise function



$\boldsymbol{h}$ is the output of a layer.

σ has to be non linear (otherwise equivalent to a perceptron).

Name	**Graph**	f	f'
Logistic / sigmoïd		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\big(1 - f(x)\big)$
tanh		$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f^2(x)$
atan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
ReLU		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$
Linear exponential		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

Learning the MLP

Expanding the gradient descent

- At step k of the gradient descent, need to evaluate

$$\nabla_\theta \mathcal{L}\left(f_\theta(\boldsymbol{x}), y\right)$$

- Evaluation of the total derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_j}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_j}$, $j \in [\![1 \ldots L]\!]$

$\Rightarrow$ Automatic differentiation on the computational graph

LIMOS Laboratoire d'informatique, de modélisation et d'optimisation des systèmes

cnrs

UCA Université Clermont Auvergne

LEARNING THE MLP

Expanding the gradient descent

- At step k of the gradient descent, need to evaluate

$$\nabla_\theta \mathcal{L}\left(f_\theta(\boldsymbol{x}), y\right)$$

- Evaluation of the total derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_j}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_j}$, $j \in [\![1 \dots L]\!]$

$\Rightarrow$ Automatic differentiation on the computational graph

Chain Rule

Let $g : \mathbb{R} \to \mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}$

$$f \circ g(x) = f(\boldsymbol{u}) = y \textbf{ where } \boldsymbol{u} = g(x) = (g_1(x) \dots g_m(x))^T = (u_1 \dots u_m)$$

Chain rule:

$$\frac{dy}{dx} = \sum_{j=1}^{m} \frac{\partial y}{\partial u_j} \underbrace{\frac{du_j}{dx}}_{\text{recursive}}$$

Learning the MLP

Automatic differentiation

- MLP = composition of differentiable functions
- The total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.

LIMOS Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
Université Clermont Auvergne

Learning the MLP

Automatic differentiation

- MLP = composition of differentiable functions
- The total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.

Automatic differentiation

1. Forward pass: values are all computed from inputs to outputs
2. Backward pass: the total derivatives are computed by walking through all paths from outputs to parameters in the computational graph and accumulating the terms.

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

Learning the MLP

Automatic differentiation

- MLP = composition of differentiable functions
- The total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.

Automatic differentiation

1. Forward pass: values are all computed from inputs to outputs
2. Backward pass: the total derivatives are computed by walking through all paths from outputs to parameters in the computational graph and accumulating the terms.

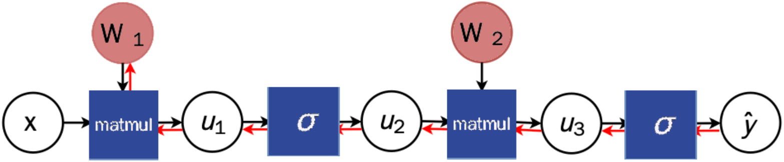
Additional ressource

See Slides "backpropagation" and "Vanishing gradient".

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Learning the MLP

Example: derivatives with respect to $\boldsymbol{W}_1$



1. Forward pass: $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ and $\hat{y}$ computed by traversing the graph, given $\boldsymbol{x}$,$\boldsymbol{W}_1$ and $\boldsymbol{W}_2$
2. Backward pass :

$$
\begin{aligned}
\frac{d\hat{y}}{d\boldsymbol{W}_1} &= \frac{\partial \hat{y}}{\partial \boldsymbol{u}_3}\frac{\partial \boldsymbol{u}_3}{\partial \boldsymbol{u}_2}\frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1}\frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{W}_1} \\
&= \frac{\partial \sigma(\boldsymbol{u}_3)}{\partial \boldsymbol{u}_3}\frac{\partial \boldsymbol{W}_2^T u_2}{\partial \boldsymbol{u}_2}\frac{\partial \sigma(\boldsymbol{u}_1)}{\partial \boldsymbol{u}_1}\frac{\partial \boldsymbol{W}_1^T u_1}{\partial \boldsymbol{W}_1}
\end{aligned}
$$

Evaluating the partial derivatives requires the intermediate values computed forward

UNIVERSAL APPROXIMATION

Theorem (Cybenko 1989; Hornik et al, 1991)

Let σ be a bounded, non-constant continuous function.
Let I_d denote the d-dimensional hypercube, and $C(I_d)$ denote the space of continuous functions on I_d.

$(\forall f \in C(I_d))(\forall \epsilon > 0)(\exists q > 0, v_i, \mathbf{w_i}, b_i, i \in [\![1 \ldots q]\!])$ such that

$$F(\mathbf{x}) = \sum_{i=1}^{q} v_i \sigma(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b)$$

satisfies

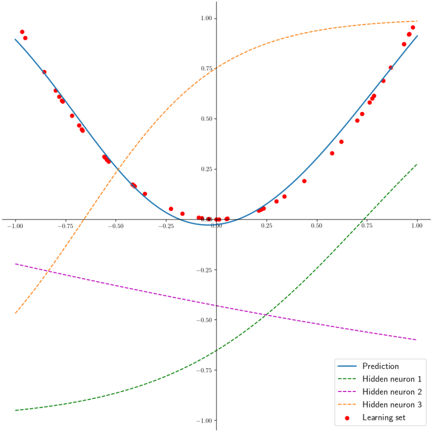
$$\sup_{\mathbf{x} \in I_d} \mid f(\mathbf{x}) - F(\mathbf{x}) \mid < \epsilon$$

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

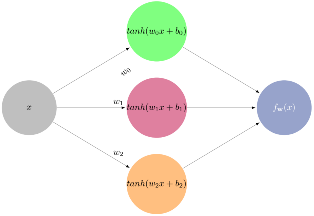
UNIVERSAL APPROXIMATION



$f(x) = x^2, |Z| = 50$

A simple example

- $|Z|$ points uniformly sampled (red) over the definition set
- 1 hidden layer MLP, 3 neurons.
- $tanh$ activation function, and linear output neurons
- network output : blue curve
- hidden neurons outputs: dashed curves



LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Universal approximation

Properties

- Guarantees that a single hidden layer network can represent any classification problem in which the boundary is locally linear (smooth)
- Does not inform about good/bad architectures, nor how they relate to the optimization procedure
- Generalizes to any non-polynomial (possibly unbounded) activation function, including the ReLU

Universal approximation

Theorem (Barron, 1992)

Let a one-hidden layer MLP with q hidden neurons , p inputs and $|Z| = n$. The mean integrated square error between the estimated network $\hat{F}$ and the target function f is bounded by

$$O\left(\frac{C_f^2}{q} + \frac{qp}{n} log(n)\right)$$

where C_f measures the global smoothness of f.

Properties

- Combines approximation and estimation errors.
- Provided enough data, guarantees that adding more neurons will result in a better approximation

Effect of depth

Theorem (Montúfar et al, 2014)

A MLP with ReLU as activation functions, p inputs, L hidden layers with $q \geq p$ neurons can compute functions having $\Omega\left(\left(\frac{q}{p}\right)^{(L-1)p} q^p\right)$ linear regions (asymptotic lower bound).

Properties

- The number of linear regions of deep models grows exponentially in L and polynomially in q.
- Even for small values of L and q, deep rectifier models are able to produce substantially more linear regions than shallow rectifier models.

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne