EE-559 – Deep learning

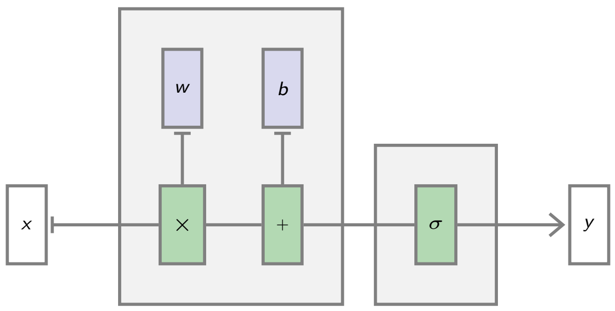
3b. Multi-Layer Perceptrons

François Fleuret
https://fleuret.org/dlc/

February 15, 2018

Combining multiple layers

For flexibility, we will separate the linear operators and the non-linearities in different blocks in our figures.

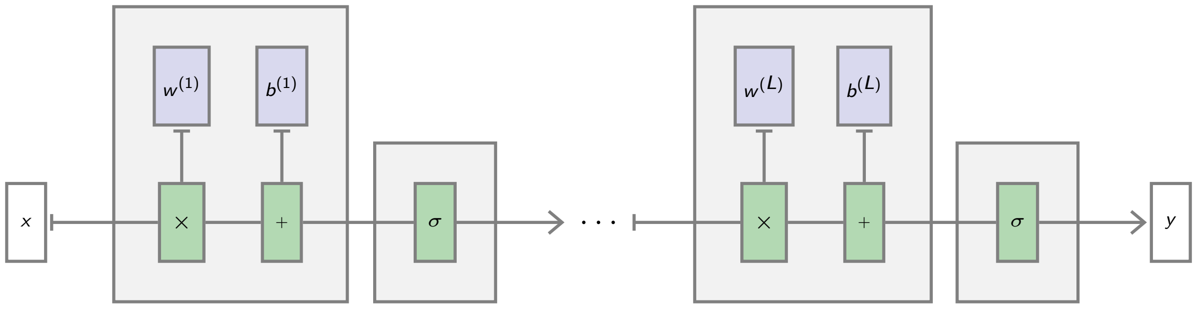


We can combine several "layers":

With $x^{(0)} = x$,

$$\forall l = 1, \dots, L,\ x^{(l)} = \sigma\left(w^{(l)} x^{(l-1)} + b^{(l)}\right)$$

and $f(x; w, b) = x^{(L)}$.



Such a model is a **Multi-Layer Perceptron (MLP).**

Note that if σ is a linear transformation,

$$\forall x \in \mathbb{R}^N,\ \sigma(x) = \alpha x + \beta \mathbb{I}$$

with $\alpha, \beta \in \mathbb{R}$, we have

$$\forall l = 1, \dots, L,\ x^{(l)} = \alpha w^{(l)} x^{(l-1)} + \alpha b^{(l)} + \beta \mathbb{I},$$

and the whole mapping is an affine transform

$$f(x; w, b) = A^{(L)} x + B^{(L)}$$

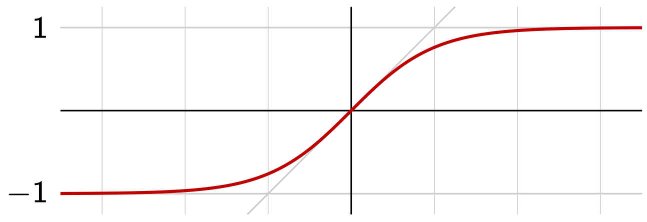
where $A^{(0)} = \mathbb{I}, B^{(0)} = 0$ and

$$\forall l < L,\ \begin{cases} A^{(l)} = \alpha\, w^{(l)} A^{(l-1)} \\ B^{(l)} = \alpha\, w^{(l)} B^{(l-1)} + \alpha\, b^{(l)} + \beta \mathbb{I} \end{cases}$$

⚠ Consequently, **the activation function should be non-linear**, or the resulting MLP is an affine mapping with a peculiar parametrization.

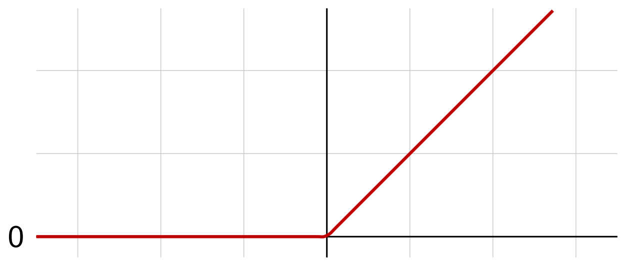
The two classical activation functions are the hyperbolic tangent

$$x \mapsto \frac{2}{1 + e^{-2x}} - 1$$



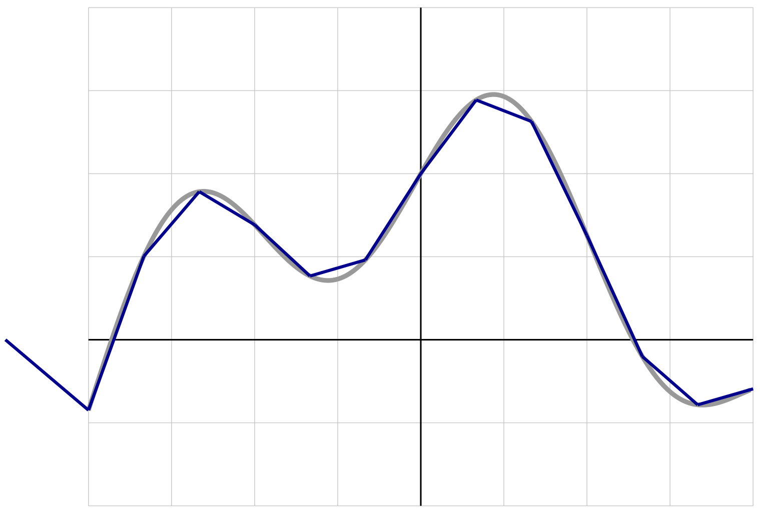
and the rectified linear unit (ReLU)

$$x \mapsto \max(0, x)$$



Universal approximation

We can approximate any $f \in \mathscr{C}([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions.



This is true for other activation functions under mild assumptions.

Extending this result to any $f \in \mathscr{C}([0,1]^D, \mathbb{R})$ requires a bit of work.

First, we can use the previous result for the sin function

$$\forall A > 0, \epsilon > 0,\ \exists N,\ (\alpha_n, a_n) \in \mathbb{R} \times \mathbb{R}, n = 1, \dots, N,$$
$$\text{s.t. } \max_{x \in [-A,A]} \left| \sin(x) - \sum_{n=1}^{N} \alpha_n \sigma(x - a_n) \right| \leq \epsilon.$$

And the density of Fourier series provides

$$\forall f \in \mathscr{C}([0,1]^D, \mathbb{R}), \delta > 0, \exists M, (v_m, \gamma_m, c_m) \in \mathbb{R}^D \times \mathbb{R} \times \mathbb{R}, m = 1, \dots, M,$$
$$\text{s.t. } \max_{x \in [0,1]^D} \left| f(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right| \leq \delta.$$

So, $\forall \xi > 0$, with

$$\delta = \frac{\xi}{2}, A = \max_{1 \leq m \leq M} \max_{x \in [0,1]^D} |v_m \cdot x + c_m|, \text{ and } \epsilon = \frac{\xi}{2 \sum_m |\gamma_m|}$$

we get, $\forall x \in [0,1]^D$,

$$\begin{aligned} & \left| f(x) - \sum_{m=1}^{M} \gamma_m \left(\sum_{n=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right) \right| \\ & \leq \underbrace{\left| f(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right|}_{\leq \frac{\xi}{2}} \\ & \quad \underbrace{+ \sum_{m=1}^{M} |\gamma_m| \underbrace{\left| \sin(v_m \cdot x + c_m) - \sum_{n=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right|}_{\leq \frac{\xi}{2 \sum_m |\gamma_m|}}}_{\leq \frac{\xi}{2}} \end{aligned}$$

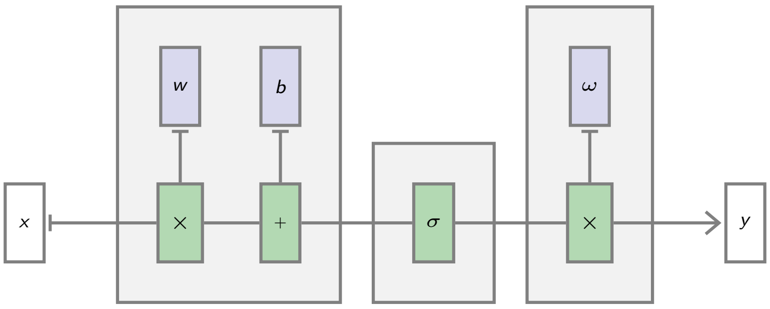
So we can approximate any continuous function

$$f : [0,1]^D \to \mathbb{R}$$

with a mapping of the form

$$x \mapsto \omega \cdot \sigma(w\, x + b),$$

where $b \in \mathbb{R}^K$, $w \in \mathbb{R}^{K \times D}$, and $\omega \in \mathbb{R}^K$, *i.e.* with a one hidden layer perceptron.



This is the **universal approximation theorem.**

⚠ A better approximation requires a larger hidden layer (larger K), and this theorem says nothing about the relation between the two. We will come back to that later.

Training and gradient descent

We saw that training consists of finding the model parameters minimizing an empirical risk or loss, for instance the mean-squared error (MSE)

$$\mathscr{L}(w, b) = \frac{1}{N} \sum_n \ell \left(f(x_n; w, b) - y_n \right)^2 .$$

Other losses are more fitting for classification, certain regression problems, or density estimation. We will come back to this.

In our previous examples we minimized the loss either with an analytic solution for the MSE, or with *ad hoc* recipes for the empirical error rate (k-NN and perceptron).

There is generally no *ad hoc* method. The logistic regression for instance

$$P_w(Y = 1 \mid X = x) = \sigma(w \cdot x + b), \text{ with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

leads to the loss

$$\mathscr{L}(w, b) = -\sum_n \log \sigma(y_n(w \cdot x_n + b))$$

which cannot be minimized analytically.

The general minimization method used in such a case is the **gradient descent**.

Given a functional

$$\begin{aligned} f : \mathbb{R}^D &\to \mathbb{R} \\ x &\mapsto f(x_1, \dots, x_D), \end{aligned}$$

its gradient is the mapping

$$\begin{aligned} \nabla f : \mathbb{R}^D &\to \mathbb{R}^D \\ x &\mapsto \left(\frac{\partial f}{\partial x_1}(x), \dots, \frac{\partial f}{\partial x_D}(x) \right). \end{aligned}$$

To minimize a functional

$$\mathscr{L} : \mathbb{R}^D \to \mathbb{R}$$

the gradient descent uses local linear information to iteratively move toward a (local) minimum.

For $w_0 \in \mathbb{R}^D$, consider an approximation of $\mathscr{L}$ around w_0

$$\tilde{\mathscr{L}}_{w_0}(w) = \mathscr{L}(w_0) + \nabla \mathscr{L}(w_0)^T (w - w_0) + \frac{1}{2\eta} \|w - w_0\|^2.$$

Note that the chosen quadratic term does not depend on $\mathscr{L}$.

We have

$$\nabla \tilde{\mathscr{L}}_{w_0}(w) = \nabla \mathscr{L}(w_0) + \frac{1}{\eta}(w - w_0),$$

which leads to

$$\underset{w}{\operatorname{argmin}}\, \tilde{\mathscr{L}}_{w_0}(w) = w_0 - \eta \nabla \mathscr{L}(w_0).$$

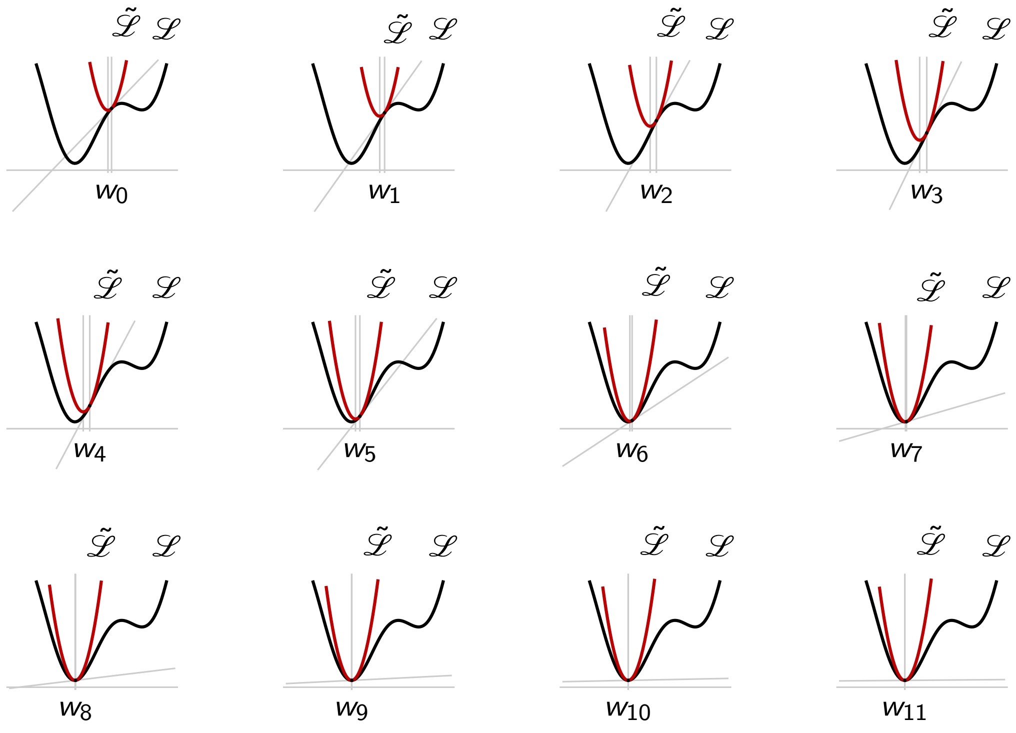
The resulting iterative rule takes the form of:

$$w_{t+1} = w_t - \eta \nabla \mathscr{L}(w_t).$$

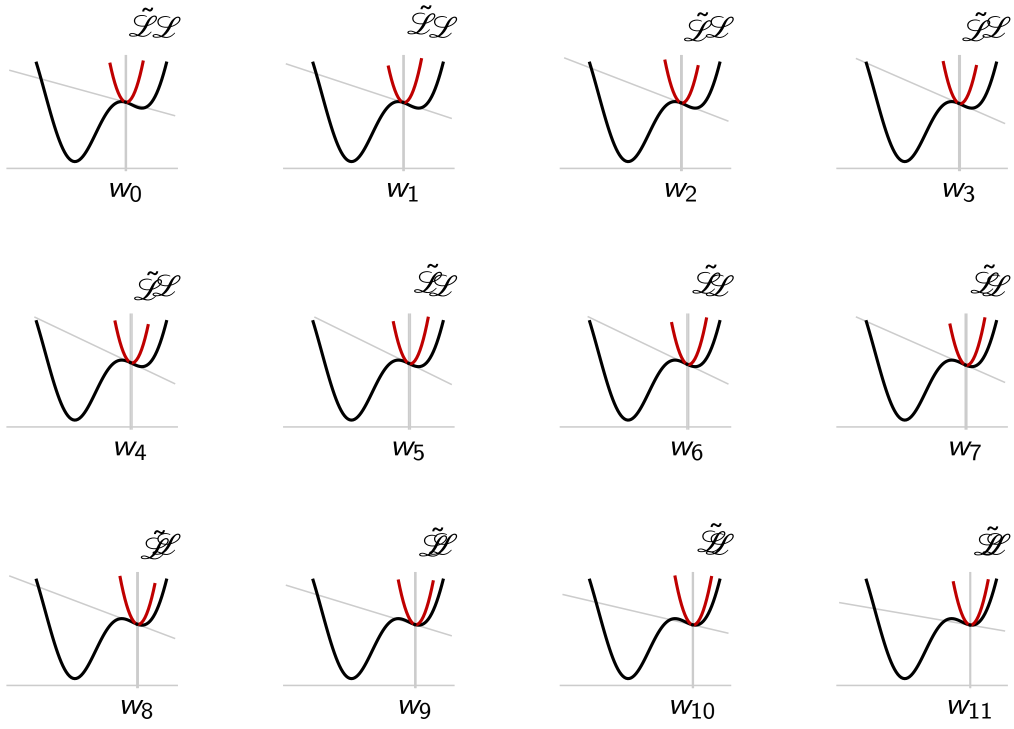
Which corresponds intuitively to "following the steepest descent".

This finds a **local** minimum, and the choices of w_0 and η are important.

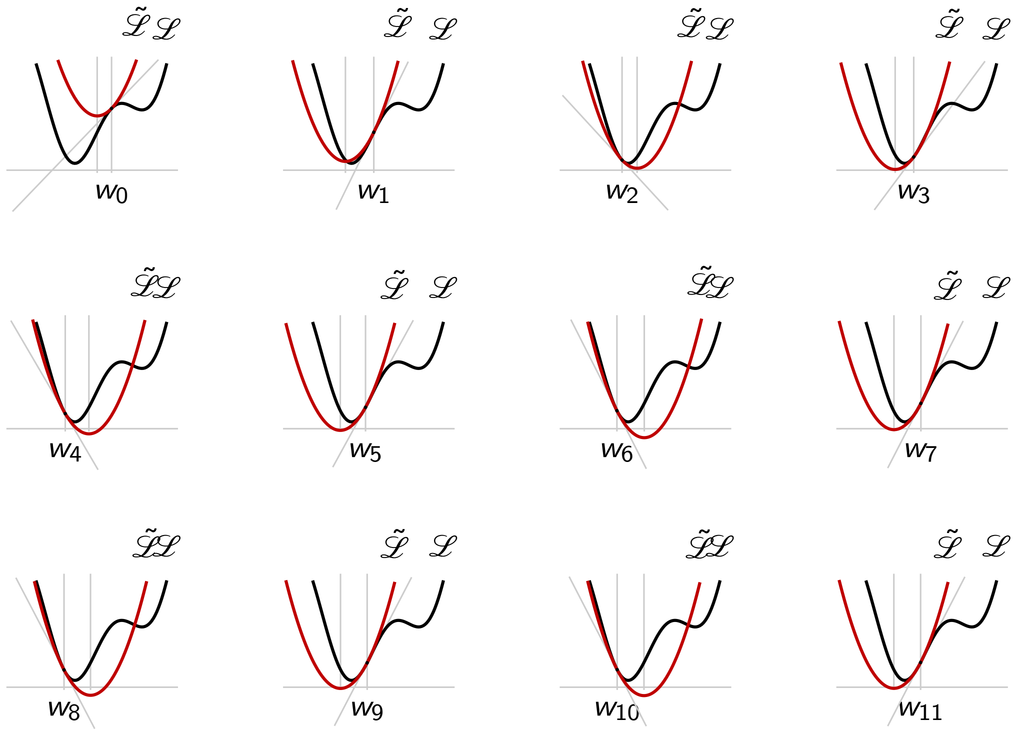
$\eta = 0.125$

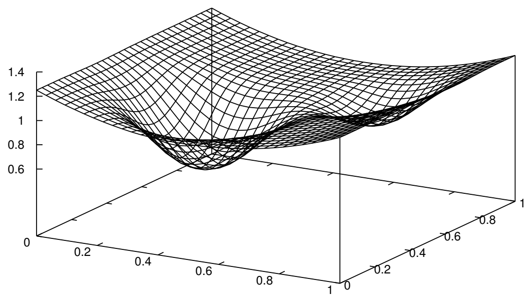
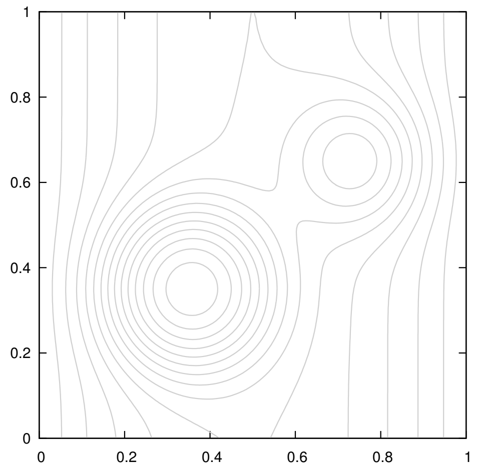
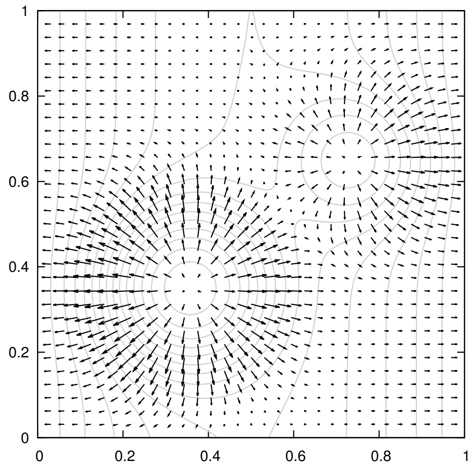
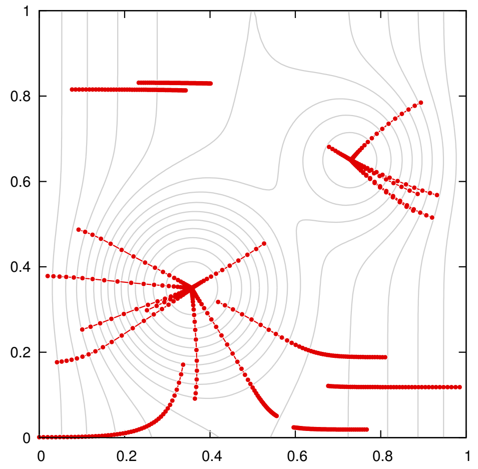


$\eta = 0.125$



$\eta = 0.5$



We saw that the minimum of the logistic regression loss

$$\mathscr{L}(w, b) = -\sum_n \log \sigma(y_n(w \cdot x_n + b))$$

does not have an analytic form.

We can derive

$$\forall d,\ \frac{\partial \mathscr{L}}{\partial w_d} = -\sum_n y_n\, x_{n,d}\, \sigma(-y_n(w \cdot x_n + b))$$

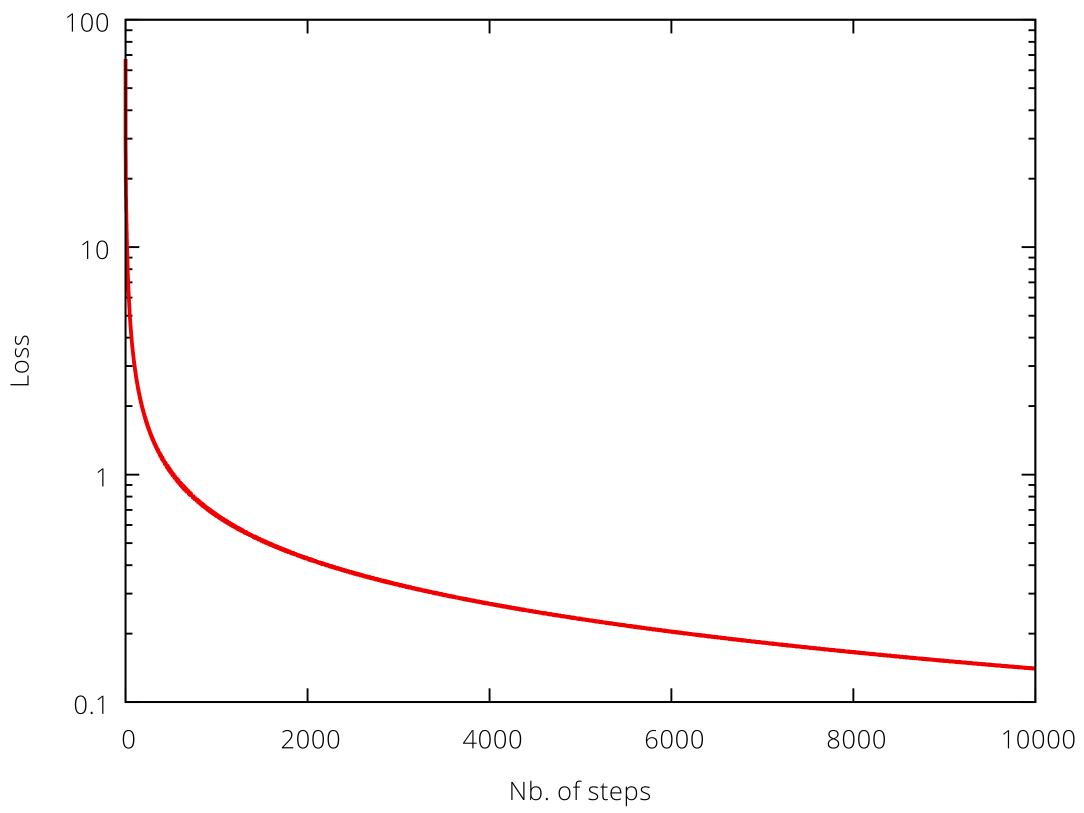
$$\frac{\partial \mathscr{L}}{\partial b} = -\sum_n y_n\, \sigma(-y_n(w \cdot x_n + b)).$$

Which can be implemented as

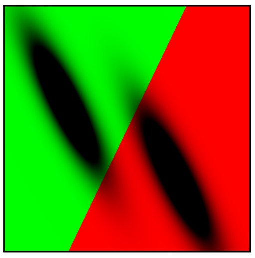
```
def gradient(x, y, w, b):
    dln_db = - y * ( - (x.mv(w) + b) * y).apply_(sigmoid)
    dln_dw = x * dln_db.view(-1, 1).expand_as(x)
    return dln_dw.sum(0), dln_db.sum()
```

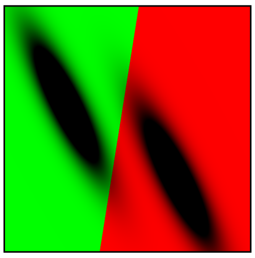
and the gradient descent as

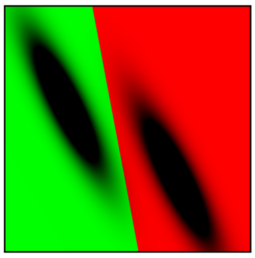
```
w = Tensor(dimension).normal_()
b = 0

eta = 1e-1

for k in range(nb_iterations):
    dw, db = gradient(x, y, w, b)
    w = w - eta * dw
    b = b - eta * db
```

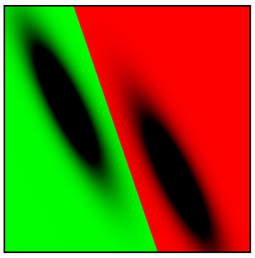


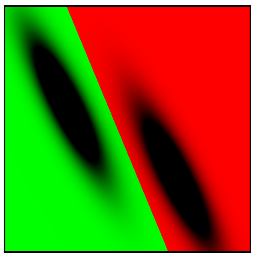
With 100 training points and $\eta = 10^{-1}$.

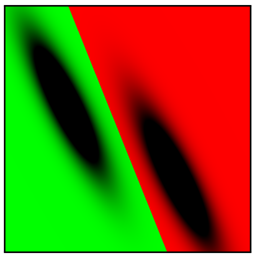

$n = 0$


$n = 10$


$n = 10^2$


$n = 10^3$


$n = 10^4$


LDA

Back-propagation

We want to train an MLP by minimizing a loss over the training set

$$\mathscr{L}(w, b) = \sum_n \ell(f(x_n; w, b), y_n).$$

To use gradient descent, we need the expression of the gradient of the loss with respect to the parameters:

$$\frac{\partial \mathscr{L}}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial b_i^{(l)}}.$$

So, with $\ell_n = \ell(f(x_n; w, b), y_n)$, what we need is

$$\frac{\partial \ell_n}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \ell_n}{\partial b_i^{(l)}}.$$

For clarity, we consider a single training sample x, and introduce $s^{(1)}, \dots, s^{(L)}$ as the summations before activation functions.

$$x^{(0)} = x \xrightarrow{w^{(1)}, b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)}, b^{(2)}} s^{(2)} \xrightarrow{\sigma} \dots \xrightarrow{w^{(L)}, b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; w, b).$$

Formally we set $x^{(0)} = x$, and $\forall l = 1, \dots, L$,

$$s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)}$$
$$x^{(l)} = \sigma\left(s^{(l)}\right)$$

and we set the output of the network is $f(x; w, b) = x^{(L)}$.

The core principle of the back-propagation algorithm is the "chain rule" from differential calculus:

$$(g \circ f)' = (g' \circ f) f'$$

which generalizes to longer compositions and higher dimensions

$$J_{f_N \circ f_{N-1} \circ \dots \circ f_1}(x) = \prod_{n=1}^{N} J_{f_n}(f_{n-1} \circ \dots \circ f_1(x)),$$

where $J_f(x)$ is the Jacobian of f at x, that is the matrix of the linear approximation of f in the neighborhood of x.

The linear approximation of a composition of mappings is the product of their individual linear approximations.

What follows is exactly this principle applied to a MLP.

We have

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

correct

and since $w_{i,j}^{(l)}$ influences ℓ only through $s_i^{(l)}$, the chain rule gives

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)},$$

correct

and similarly

$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}.$$

correct. derivative wrt bias is 1

We have

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

and since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$, the chain rule gives

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

Finally, we have trivially

$$\frac{\partial \ell}{\partial x_i^{(L)}} = (\nabla_1 \ell)_i$$

where $\nabla_1 \ell$ is the gradient of ℓ with respect to its first parameter, that is the predicted value. Also, $\forall l = 1, \dots, L-1$, since

$$s_h^{l+1} = \sum_i w_{h,i}^{l+1} x_i^{(l)} + b_h^{l+1},$$

correct

and $x_i^{(l)}$ influences ℓ only through the s_h^{l+1}, we have

$$\frac{\partial \ell}{\partial x_i^{(l)}} = \sum_h \frac{\partial \ell}{\partial s_h^{l+1}} \frac{\partial s_h^{l+1}}{\partial x_i^{(l)}} = \sum_h \frac{\partial \ell}{\partial s_h^{l+1}} w_{h,i}^{l+1}.$$

correct

why do we need to compute the gradient wrt to the sample ? coz read ML notes

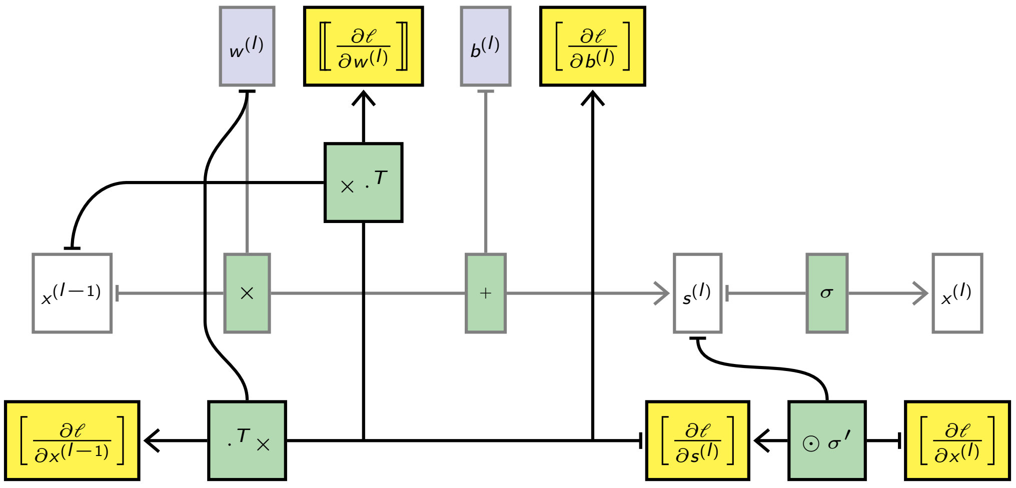
If $\psi : \mathbb{R}^N \to \mathbb{R}^M$, we will use the standard Jacobian notation

$$\left[\frac{\partial \psi}{\partial x}\right] = \begin{pmatrix} \frac{\partial \psi_1}{\partial x_1} & \dots & \frac{\partial \psi_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi_M}{\partial x_1} & \dots & \frac{\partial \psi_M}{\partial x_N} \end{pmatrix},$$

and if $\psi : \mathbb{R}^{N \times M} \to \mathbb{R}$, we will use the compact notation, also tensorial

$$\left[\!\left[\frac{\partial \psi}{\partial w}\right]\!\right] = \begin{pmatrix} \frac{\partial \psi}{\partial w_{1,1}} & \dots & \frac{\partial \psi}{\partial w_{1,M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi}{\partial w_{N,1}} & \dots & \frac{\partial \psi}{\partial w_{N,M}} \end{pmatrix}.$$

what operation converts a matrix to a scalar ?



$w^{(l)}$
$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right]$
$b^{(l)}$
$\left[\frac{\partial \ell}{\partial b^{(l)}}\right]$
$\times \cdot^T$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$
$\left[\frac{\partial \ell}{\partial x^{(l-1)}}\right]$
$\cdot^T \times$
$\left[\frac{\partial \ell}{\partial s^{(l)}}\right]$
$\odot \sigma'$
$\left[\frac{\partial \ell}{\partial x^{(l)}}\right]$

Forward pass

$$\forall n,\ x_n^{(0)} = x_n, \quad \forall l = 1, \dots, L, \ \begin{cases} s_n^{(l)} = w^{(l)} x_n^{(l-1)} + b^{(l)} \\ x_n^{(l)} = \sigma\left(s_n^{(l)}\right) \end{cases}$$

Backward pass

$$\begin{cases} \left[\frac{\partial \ell_n}{\partial x_n^{(L)}}\right] = \nabla_1 \ell_n \left(x_n^{(L)}\right) \\ \text{if } l < L, \left[\frac{\partial \ell_n}{\partial x_n^{(l)}}\right] = \left(w^{l+1}\right)^T \left[\frac{\partial \ell_n}{\partial s^{l+1}}\right] \end{cases} \qquad \left[\frac{\partial \ell_n}{\partial s^{(l)}}\right] = \left[\frac{\partial \ell_n}{\partial x_n^{(l)}}\right] \odot \sigma'\left(s^{(l)}\right)$$

$$\left[\!\left[\frac{\partial \ell_n}{\partial w^{(l)}}\right]\!\right] = \left[\frac{\partial \ell_n}{\partial s^{(l)}}\right] \left(x_n^{(l-1)}\right)^T \qquad \left[\frac{\partial \ell_n}{\partial b^{(l)}}\right] = \left[\frac{\partial \ell_n}{\partial s^{(l)}}\right].$$

Gradient step

$$w^{(l)} \leftarrow w^{(l)} - \eta \sum_n \left[\!\left[\frac{\partial \ell_n}{\partial w^{(l)}}\right]\!\right] \qquad b^{(l)} \leftarrow b^{(l)} - \eta \sum_n \left[\frac{\partial \ell_n}{\partial b^{(l)}}\right]$$

In spite of its hairy formalization, the backward pass is quite a simple algorithm: Just apply the chain rule again and again.

As for the forward pass, it can be expressed in tensorial form. Heavy computation is concentrated in linear operations, and all the non-linearities go into component-wise operations.

Regarding computation, the rule of thumb is that the backward pass is twice more expensive than the forward one.

Practical session:

`https://fleuret.org/dlc/dlc-practical-3.pdf`