Introduction to Deep Learning

Multi-Layered NNet and the back-propagation algorithm

Alexandre Allauzen

20/01/20

Outline

Previously: logistic regression

Linear classification and the sigmoid function

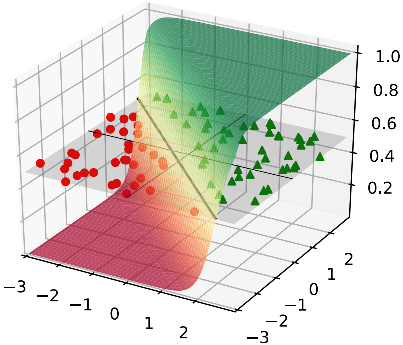
$$P(C=1|\boldsymbol{x}) = \sigma(w_0 + \mathbf{w}^t\boldsymbol{x}) = y$$

$$\sigma(a) = \frac{e^a}{1+e^a} = \frac{1}{1+e^{-a}}$$

$$a = w_0 + \mathbf{w}^t\boldsymbol{x}, a \in \mathbb{R}$$

$$a = w_0 + w_1x_1 + \cdots + w_Dx_D$$

$$\boldsymbol{\theta} = (\mathbf{w}), \mathcal{D} = (\boldsymbol{x}_{(i)}, c_{(i)})_{i=1}^n$$



Learning with (Stochastic) Gradient Descent

Minimize $\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = \sum_{i=1}^N l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$

$$l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)}) = -\underbrace{c_{(i)}log(y_{(i)})}_{c_{(i)}=1} - \underbrace{(1-c_{(i)})log(1-y_{(i)})}_{c_{(i)}=0}$$

Bias or not bias

A matter of notation

The bias can be explicite:

$$\color{red}{w_0} + \mathbf{w}^t\boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_D x_D$$

or implicite:

$$\mathbf{w} \cdot \boldsymbol{x} = \begin{pmatrix} \color{red}{w_0} \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} \cdot \begin{pmatrix} \color{red}{1} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$= \color{red}{w_0} + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$$

Gradient of the pointwise loss $w.r.t$ $\boldsymbol{w}$ - 1

$$\frac{\partial l(c_{(i)}, \boldsymbol{x}_{(i)}, \boldsymbol{\theta}_{(i)})}{\partial \boldsymbol{w}} = -c_{(i)} \frac{\partial \log y_{(i)}}{\partial \boldsymbol{w}} - (1 - c_{(i)}) \frac{\partial \log(1 - y_{(i)})}{\partial \boldsymbol{w}}$$

Setting $a = \boldsymbol{w}^t \boldsymbol{x}$, and consider the first term, the chain is

$$\boldsymbol{x} \rightarrow a \rightarrow y_{(i)} = \sigma(a) \rightarrow \log y_{(i)}$$

$$\begin{aligned}
\frac{\partial \log y_{(i)}}{\partial \boldsymbol{w}} &= \frac{\partial \log y_{(i)}}{\partial y_{(i)}} \frac{\partial y_{(i)}}{\partial a} \frac{\partial a}{\partial \boldsymbol{w}} = \frac{1}{y_{(i)}} \frac{\partial y_{(i)}}{\partial a} \frac{\partial a}{\partial \boldsymbol{w}} \\
&= \frac{1}{y_{(i)}} \frac{\partial \sigma(a)}{\partial a} \frac{\partial a}{\partial \boldsymbol{w}} \\
&= \frac{1}{y_{(i)}} \sigma(a)(1 - \sigma(a)) \frac{\partial a}{\partial \boldsymbol{w}} = \frac{1}{y_{(i)}} y_{(i)}(1 - y_{(i)}) \frac{\partial a}{\partial \boldsymbol{w}} \\
&= \frac{1}{y_{(i)}} y_{(i)}(1 - y_{(i)}) \boldsymbol{x} = (1 - y_{(i)}) \boldsymbol{x}
\end{aligned}$$

Gradient of the pointwise loss $w.r.t$ $\boldsymbol{w}$ - 2

Consider the second term

$$\frac{\partial \log(1-y_{(i)})}{\partial \boldsymbol{w}} = \frac{\partial \log(1-y_{(i)})}{\partial y_{(i)}} \frac{\partial y_{(i)}}{\partial a} \frac{\partial a}{\partial \boldsymbol{w}} = \frac{-1}{1-y_{(i)}} \frac{\partial y_{(i)}}{\partial a} \frac{\partial a}{\partial \boldsymbol{w}}$$
$$= \frac{-1}{1-y_{(i)}} y_{(i)}(1-y_{(i)})\boldsymbol{x} = -y_{(i)}\boldsymbol{x}$$

Putting everything together:

$$\begin{aligned}\frac{\partial l(c_{(i)}, \boldsymbol{x}_{(i)}, \boldsymbol{\theta}_{(i)})}{\partial \boldsymbol{w}} &= -c_{(i)}(1-y_{(i)})\boldsymbol{x} - (1-c_{(i)})(-y_{(i)})\boldsymbol{x} \\ &= -(c_{(i)} - c_{(i)}y_{(i)} - y_{(i)} + c_{(i)}y_{(i)})\boldsymbol{x} \\ &= -(c_{(i)} - y_{(i)})\boldsymbol{x}\end{aligned}$$

Update

Now consider the update, first for the class $c_{(i)} = 1$:

$$\begin{aligned}
\boldsymbol{w}_{(i+1)} &= \boldsymbol{w}_{(i)} + \eta(c_{(i)} - y_{(i)})\boldsymbol{x} \\
&= \boldsymbol{w}_{(i)} + \eta(1 - y_{(i)})\boldsymbol{x}, \text{ note that } e_{(i)} = 1 - y_{(i)} \geq 0 \\
\boldsymbol{w}^t_{(i+1)}\boldsymbol{x} &= \boldsymbol{w}^t_{(i)}\boldsymbol{x} + \eta e_{(i)}\boldsymbol{x}^t\boldsymbol{x} > \boldsymbol{w}^t_{(i)}\boldsymbol{x} \\
\sigma(\boldsymbol{w}^t_{(i+1)}\boldsymbol{x}) &\geq \sigma(\boldsymbol{w}^t_{(i)}\boldsymbol{x}) \text{ therefore } P(C = 1|\boldsymbol{x}) \nearrow
\end{aligned}$$

And for the class $c_{(i)} = 0$:

$$\begin{aligned}
\boldsymbol{w}_{(i+1)} &= \boldsymbol{w}_{(i)} + \eta(c_{(i)} - y_{(i)})\boldsymbol{x} \\
&= \boldsymbol{w}_{(i)} - \eta y_{(i)}\boldsymbol{x}, \text{ note that } e_{(i)} = y_{(i)} \geq 0 \\
\boldsymbol{w}^t_{(i+1)}\boldsymbol{x} &= \boldsymbol{w}^t_{(i)}\boldsymbol{x} - \eta e_{(i)}\boldsymbol{x}^t\boldsymbol{x} < \boldsymbol{w}^t_{(i)}\boldsymbol{x} \\
\sigma(\boldsymbol{w}^t_{(i+1)}\boldsymbol{x}) &\leq \sigma(\boldsymbol{w}^t_{(i)}\boldsymbol{x}) \text{ therefore } P(C = 1|\boldsymbol{x}) \searrow
\end{aligned}$$

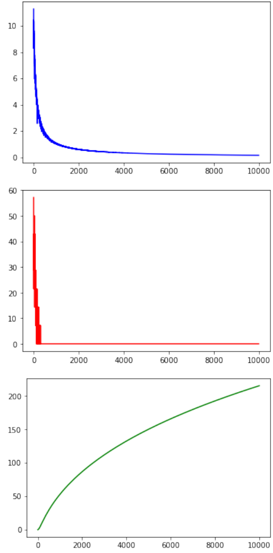
In both cases, the pointwise loss is reduced.

Unroll the updates

$$\begin{aligned}
\boldsymbol{w}_{(1)} &= \boldsymbol{w}_{(0)} + \eta(c_{(1)} - y_{(1)})\boldsymbol{x}_{(1)} \\
\boldsymbol{w}_{(2)} &= \boldsymbol{w}_{(1)} + \eta(c_{(2)} - y_{(2)})\boldsymbol{x}_{(2)} \\
&= \boldsymbol{w}_{(0)} + \eta\Big[(c_{(1)} - y_{(1)})\boldsymbol{x}_{(1)} + (c_{(2)} - y_{(2)})\boldsymbol{x}_{(2)}\Big] \\
\cdots &= \cdots \\
\boldsymbol{w}_{(i)} &= \boldsymbol{w}_{(0)} + \eta\sum_{k=1}^{i}(c_{(k)} - y_{(k)})\boldsymbol{x}_{(k)}
\end{aligned}$$

- $\boldsymbol{w}$ is the sum of the training examples, weighted by the error.
- A kind of lossy compression of the $\mathcal{D}$.
- During training, the error tends to 0, so the update.
- What is the difference, between batch and online training ?

Training curves

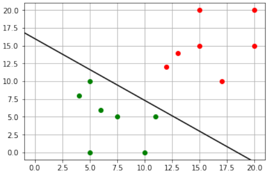


- Cumulated loss per epoch
- Classification error rate
- $\|\boldsymbol{w}\| + w_0^2$

The loss still decreases because the norm can increase ("forever").

Omitted exercise

Assume we have w_0 and $\boldsymbol{w}$ such that all the points are well classified.
Consider $w_0 = \alpha w_0$ and $\boldsymbol{w} = \alpha \boldsymbol{w}$, with $\alpha > 1$



1. what is the new classification error rate ?
2. does it change the decision boundary ?
3. what is the new loss function ?
4. propose a solution to overcome this issue, which is a kind of **overfitting**

$l2$ regularisation

Consider the new loss function as a tradeoff, with $\lambda > 0$ and usually $\lambda < 1$ (like η):

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \sum_{i=1}^{N} l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta}; \mathcal{D})}{\partial \boldsymbol{\theta}} = \boldsymbol{\nabla} + \lambda \boldsymbol{w}$$

Therefore the update becomes

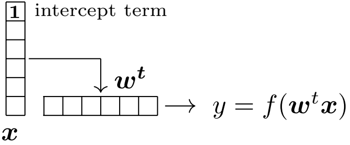
$$\begin{aligned} \boldsymbol{w}_{(i+1)} &= \boldsymbol{w}_{(i)} - \eta \Big(\boldsymbol{\nabla} + \boldsymbol{w} \Big) \\ &= (1 - \eta\lambda) \boldsymbol{w}_{(i)} - \eta \boldsymbol{\nabla} \end{aligned}$$

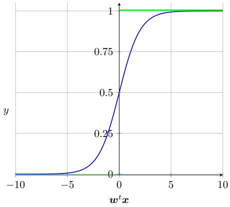
- A regularized objective function
- The new term is called regularization ($l2$)
- Helps to avoid a kind of **overfitting**

Outline

A choice of terminology

Logistic regression (binary classification)

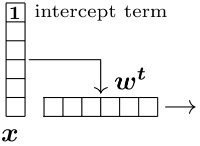




$$f(a = \boldsymbol{w}^t\boldsymbol{x}) = \frac{e^a}{1+e^a} = \sigma(a)$$

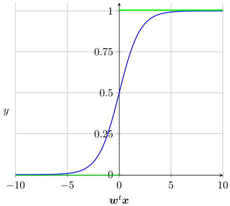
A choice of terminology

Logistic regression (binary classification)

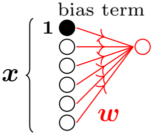


$\longrightarrow\ y = f(\boldsymbol{w}^t\boldsymbol{x})$

$$f(a = \boldsymbol{w}^t\boldsymbol{x}) = \frac{e^a}{1+e^a} = \sigma(a)$$



A single artificial neuron

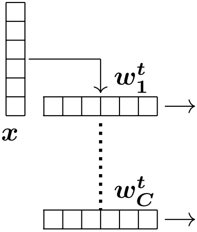


$y = f(\boldsymbol{w}^t\boldsymbol{x})$

- pre-activation : $a = \boldsymbol{w}^t\boldsymbol{x}$
- f: activation function
- Input values = input “neurones”
- $\boldsymbol{x}$: a vector of values, a layer

A choice of terminology - 2

From binary classification to C classes (Maxent)



$$y_1 = f(\boldsymbol{w_1}^t \boldsymbol{x})$$

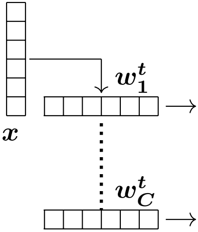
$$y_C = f(\boldsymbol{w_C}^t \boldsymbol{x})$$

$$\begin{aligned} y_k &= f(a_k = \boldsymbol{w_k}^t \boldsymbol{x}) = P(c = k|\boldsymbol{x}) \\ &= \frac{e^{a_k}}{\sum_{k'=1}^{C} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\boldsymbol{x})} \end{aligned}$$

$$\sum_{k=1}^{C} y_k = \sum_{k=1}^{C} P(c = k|\boldsymbol{x}) = 1$$

A choice of terminology - 2

From binary classification to C classes (Maxent)



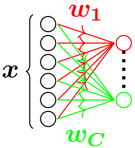
$$y_1 = f(\boldsymbol{w_1}^t\boldsymbol{x})$$

$$y_C = f(\boldsymbol{w_C}^t\boldsymbol{x})$$

$$y_k = f(a_k = \boldsymbol{w_k}^t\boldsymbol{x}) = P(c=k|\boldsymbol{x}) = \frac{e^{a_k}}{\sum_{k'=1}^{C} e^{a_{k'}}} = \frac{e^{a_k}}{Z(\boldsymbol{x})}$$

$$\sum_{k=1}^{C} y_k = \sum_{k=1}^{C} P(c=k|\boldsymbol{x}) = 1$$

A simple neural network

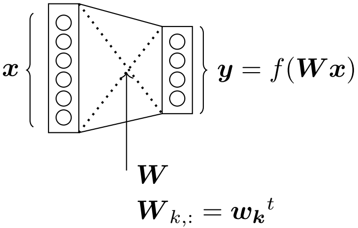


$$y_1 = f(\boldsymbol{w_1^t x})$$

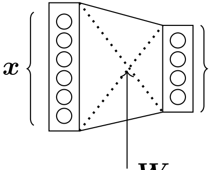
$$y_C = f(\boldsymbol{w_C^t x})$$

- $\boldsymbol{x}$: *input layer*
- $\boldsymbol{y}$: *output layer*
- each y_k has its parameters $\boldsymbol{w}_k$
- f is the **softmax** function

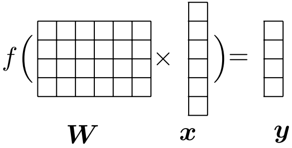
Two layers fully connected



Two layers fully connected



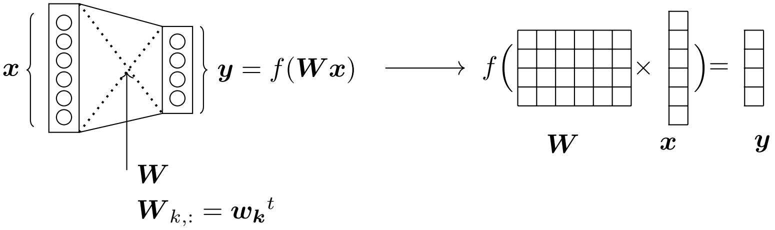
$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x}) \longrightarrow$



$\boldsymbol{W}$

$\boldsymbol{W}_{k,:} = \boldsymbol{w_k}^t$

Two layers fully connected



Activation f:

- f is usually a non-linear function
- f is a component wise function
- tanh, sigmoid, relu, ...

e.g the softmax function:

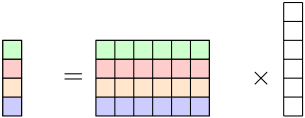
Dimensions:

- $\boldsymbol{x} : D \times 1$
- $\boldsymbol{W} : C \times D$
- $\boldsymbol{y} : (C \times \cancel{D) \times (D} \times 1) = C \times 1$

$$y_k = P(c = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w_k}^t \boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w_{k'}}^t \boldsymbol{x}}} = \frac{e^{\boldsymbol{W}_{k,:}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{W}_{k',:}\boldsymbol{x}}}$$

Matrix and Vector product

$$\mathbf{y} = \boldsymbol{W} \times \boldsymbol{x}$$



$$\begin{aligned} y_1 &= \boldsymbol{W}_{1,:} \times \boldsymbol{x} \\ y_2 &= \boldsymbol{W}_{2,:} \times \boldsymbol{x} \\ \dots &\ \dots \quad \dots \end{aligned}$$

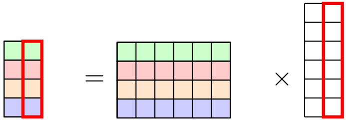
In terms of dimension:

$$\text{With} \left\{ \begin{array}{ll} \boldsymbol{x} & : (L_1 \times 1) \\ \boldsymbol{W} & : (L_2 \times C_2) \\ \boldsymbol{y} & : (L_2 \times 1) \end{array} \right. \Rightarrow (\boldsymbol{W}\boldsymbol{x}) : (L_2 \times 1) = (L_2 \times \underbrace{\cancel{C_2})(\cancel{L_1}}_{C_2 = L_1} \times 1)$$

Matrix-matrix product

$\boldsymbol{X}$ is a matrix of 2 columns, 2 vectors as $\boldsymbol{x}$:

$$\boldsymbol{Y} = \boldsymbol{W} \times \boldsymbol{X}$$

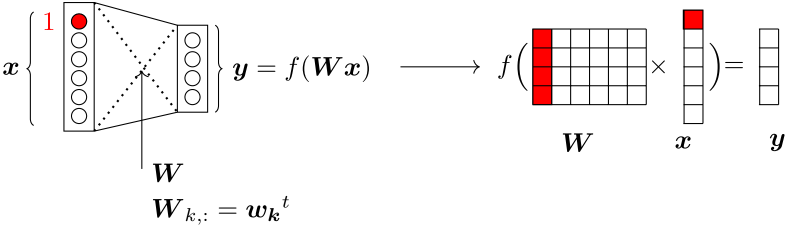


In terms of dimension:

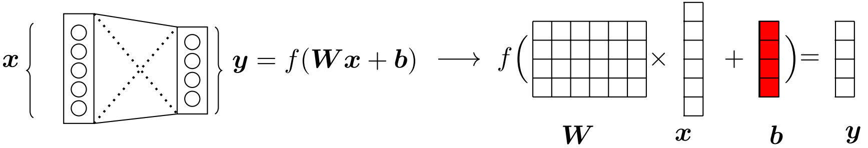
$$\text{With} \begin{cases} \boldsymbol{X} & : (L_1 \times C_1) \\ \boldsymbol{W} & : (L_2 \times C_2) \\ \boldsymbol{y} & : (L_2 \times C_1) \end{cases} \rightarrow (L_2 \times C_2) = (L_2 \times \underbrace{\cancel{C_2)(L_1}}_{C_2 = L_1} \times C_1)$$

Bias or not bias

Implicit Bias



Explicit bias



Classification with a simple neural network

Binary classification

- The input layer is a vector ($\boldsymbol{x}$), it encodes the data
- A single output neuron transform $\boldsymbol{x}$, $\boldsymbol{w}$ are its parameters
- With a sigmoïd activation, the loss function is the binary cross entropy, the log-loss, minus the log-likelihood, ...

Multiclass

- The input layer is a vector ($\boldsymbol{x}$)
- The output layer $\boldsymbol{y}$ contains one neuron per class
- It transforms $\boldsymbol{x}$, $\boldsymbol{W}$ are the parameters of the output layer
- $\boldsymbol{W}$ gathers the $\boldsymbol{w}_k = \boldsymbol{W}_{k,:}$
- With a softmax activation, the loss function is the cross entropy, the log-loss, minus the log-likelihood, ...

Other loss functions exist for classification

Regression with a simple neural network

Simple linear regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}),\ y \in \mathbb{R}$$

- The input layer is a vector ($\boldsymbol{x}$), it encodes the data
- A single output neuron for y ($\boldsymbol{w}$ are its parameters)
- The activation function depends on the output domain

Multi-variate linear regression

$$\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}),\ \boldsymbol{y} \in \mathbb{R}^C$$

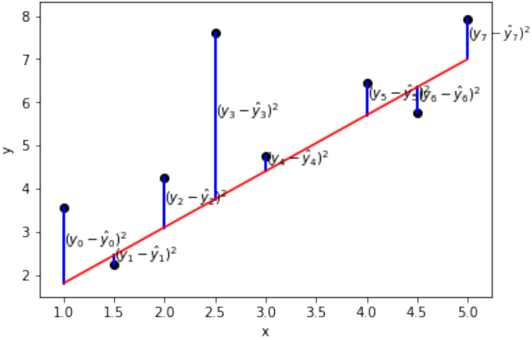
- The input layer is a vector ($\boldsymbol{x}$)
- The output layer $\boldsymbol{y}$ contains one neuron per output value
- It transforms $\boldsymbol{x}$ in $\boldsymbol{y}$, $\boldsymbol{W}$ are the parameters of the output layer

Loss for regression

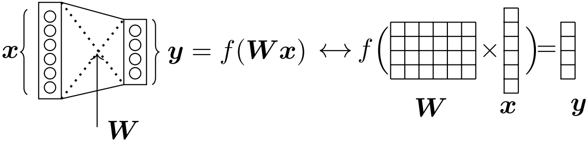
The Mean Squarred Error *a.k.a* MSE

$$\mathcal{D} = (\boldsymbol{x}_{(i)}, \boldsymbol{y}_{(i)})_{i=1}^{N},$$

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{y}_{(i)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}_{(i)})||^2$$



Two layers fully connected: another view

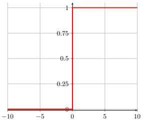


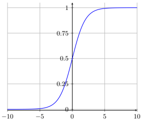
This basic brick implements a transformation of $\boldsymbol{x}$ in $\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x})$:

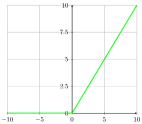
- A linear transformation $\boldsymbol{W}\boldsymbol{x}$
- Followed by a non-linear function

Example: a candidate made 6 interviews $\rightarrow \boldsymbol{x} \in \mathbb{R}^6$

- First compute 4 new scores : $\boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^4$, each is a linear combination of $\boldsymbol{x}$
- Apply a non-linearity to get $\boldsymbol{y}$







Outline

Remember: one artificial neuron

Linear classification and the sigmoid function

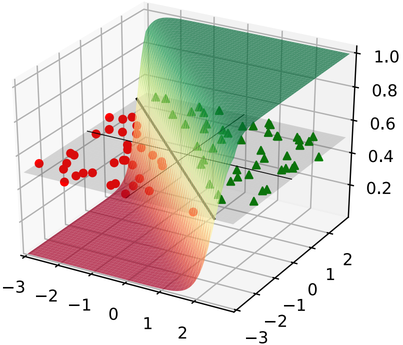
$$P(c=1|\boldsymbol{x}) = \sigma(\mathbf{w}^t\boldsymbol{x}) = y$$

$$\sigma(a) = \frac{e^a}{1+e^a} = \frac{1}{1+e^{-a}}$$

$$a = \mathbf{w}^t\boldsymbol{x}, a \in \mathbb{R}$$

$$a = w_0 + w_1x_1 + \cdots + w_Dx_D$$

$$\boldsymbol{\theta} = (\mathbf{w}), \mathcal{D} = (\boldsymbol{x}_{(i)}, c_{(i)})_{i=1}^n$$

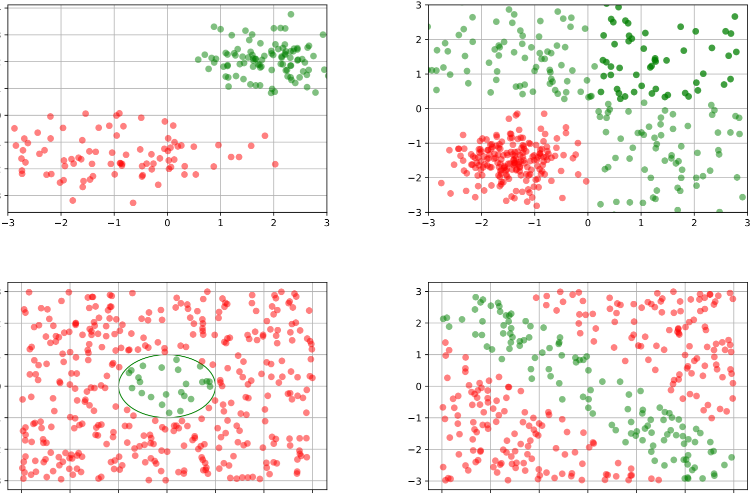


Learning with (Stochastic) Gradient Descent

Minimize $\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = \sum_{i=1}^N l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$

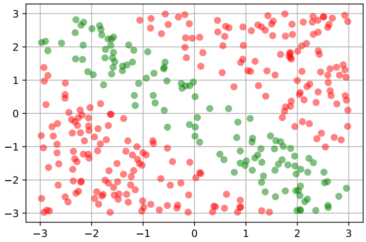
$$l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)}) = -\underbrace{c_{(i)}log(y_{(i)})}_{c_{(i)}=1} - \underbrace{(1-c_{(i)})log(1-y_{(i)})}_{c_{(i)}=0}$$

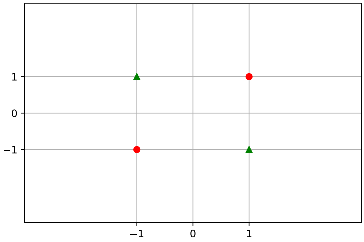
Limits of the linear separation



Case study : X-or

Starting point



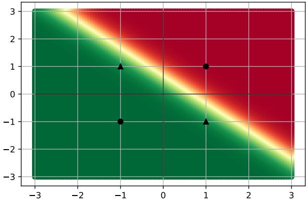


Hint

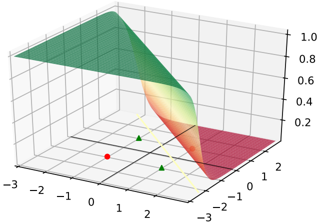
Try a first linear separation, maybe two.

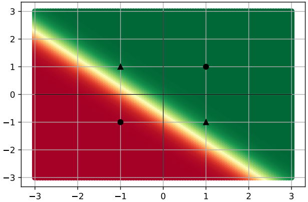
Case study : X-or

First try: linear classifiers

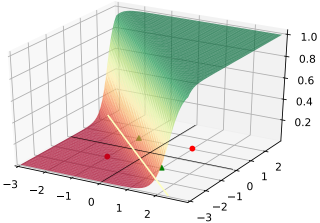


$$\mathbf{w} = [3, -4, -4]$$





$$\mathbf{w} = [2.5, 4, 4]$$

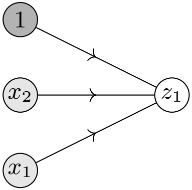


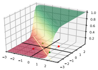
A first neural network

The multi-layer architecture

A first neural network

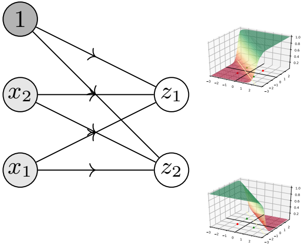
The multi-layer architecture





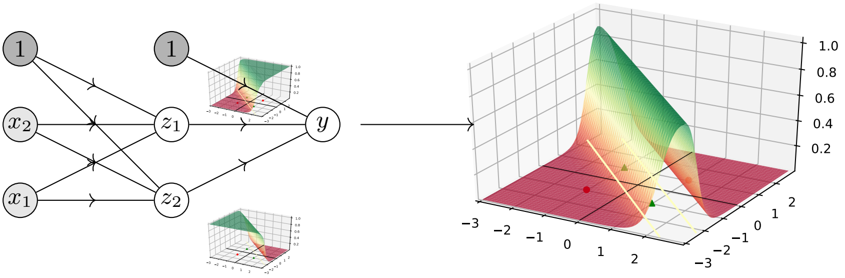
A first neural network

The multi-layer architecture



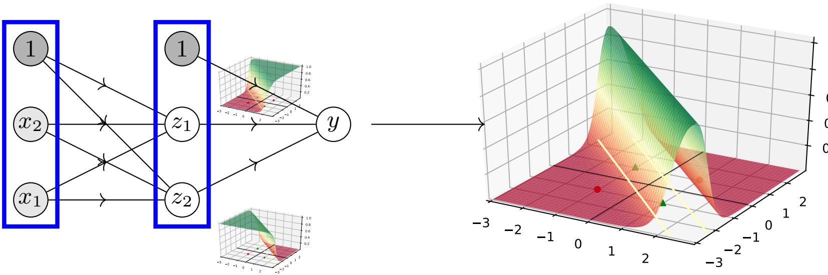
A first neural network

The multi-layer architecture



A first neural network

The multi-layer architecture



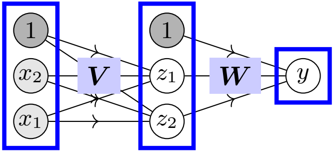
- The network is organized by layers: input ($\mathbf{x}$), hidden ($\mathbf{z}$), and output ($\mathbf{y}$)
- Two layers are fully connected
- The propagation of the input is sequential:

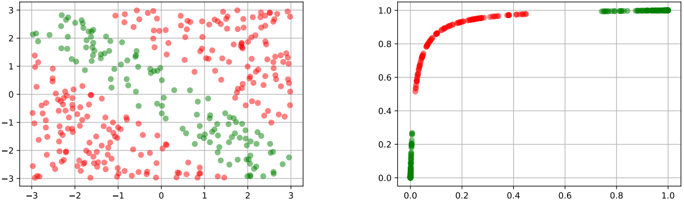
$$z_1 = f(\boldsymbol{v}_1^t \boldsymbol{x}) \text{ and } z_2 = f(\boldsymbol{v}_2^t \boldsymbol{x})) \qquad \Rightarrow \boldsymbol{z} = f(\boldsymbol{V}\boldsymbol{x})$$

$$y = f(\boldsymbol{w}^t \boldsymbol{z}) \qquad \Rightarrow y = f(\boldsymbol{W}\boldsymbol{z})$$

Another view of the neural network

Representation learning





- The network learns its own representation $\boldsymbol{z}$.
- The final decision is linear.

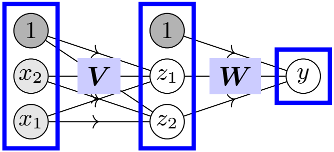
Summary

The multi-layer or feed-forward architecture

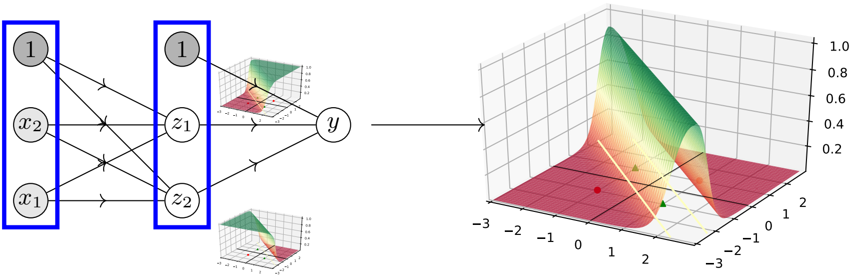
- 1 neuron = 1 value; 1 layer = 1 vector
- Two layers $(\boldsymbol{x}, \boldsymbol{z})$ fully connected:

$$\boldsymbol{z} = f(\boldsymbol{V}\boldsymbol{x})$$

- Inference: a sequential propagation
- Hidden layer $(\boldsymbol{z})$: the internal representation



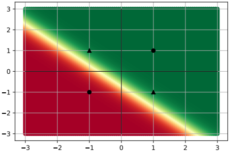
Exercise: the X-or network



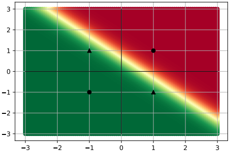
Compute the parameters of the output neuron ($\boldsymbol{W}$) ?
What is its activation function ?

Exercise : the X-or problem

From a boolean algebra point of view. Assume true value is 1 and false is 0



- Write the truth table.
- Provide an interpretation as a boolean operator



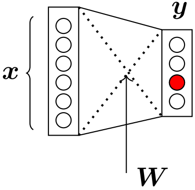
- Write the truth table.
- Provide an interpretation as a boolean operator

Write the boolean operation implemented by the previous neural network by merging the two previsous steps.

Outline

For a shallow network, with a single layer

$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x})$



$y_k = P(c = k|\boldsymbol{x})$

$\boldsymbol{\theta} = (\boldsymbol{W})$

Forward (inference)		Backward (gradient)
$l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})$	$\leftarrow \boldsymbol{y}$	$\dfrac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{W}} = \dfrac{\partial l(\boldsymbol{\theta}, \boldsymbol{x}_{(i)}, c_{(i)})}{\partial \boldsymbol{y}}$
$\boldsymbol{y}$	$= f(\boldsymbol{a})$	$\times \dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}$
$\boldsymbol{a}$	$= \boldsymbol{W}\boldsymbol{x}$	$\times \dfrac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}$

Review of the gradient computation

Without hidden layer (shallow net)

l is the shortcut for $l(c_{(i)}, \boldsymbol{x}_{(i)}, \boldsymbol{\theta}_{(i)})$

$$\frac{\partial l}{\partial \boldsymbol{W}} = \boxed{\frac{\partial l}{\partial y_{(i)}}} \times \boxed{\frac{\partial y_{(i)}}{\partial \boldsymbol{a}}} \times \boxed{\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}}$$

$\frac{\partial l}{\partial \boldsymbol{W}} \rightarrow$ Supervision

$\frac{\partial l}{\partial y_{(i)}} \rightarrow$ loss *vs* output

$\frac{\partial y_{(i)}}{\partial \boldsymbol{a}} \rightarrow$ activation

$\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \rightarrow$ gradient of $\boldsymbol{W}$

loss *vs* output + activation $\rightarrow$ Error signal from supervision and output

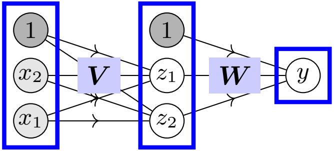
gradient of $\boldsymbol{W}$ $\rightarrow$ Contribution of $\boldsymbol{W}$ $\rightarrow \boldsymbol{x}$

In the matrix form

$$
\begin{aligned}
\nabla_{\boldsymbol{W}} = \frac{\partial l}{\partial \boldsymbol{W}} &= \boxed{\frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \boldsymbol{a}}} \times \boxed{\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}} \\
&= \boxed{\boldsymbol{\delta}_{\boldsymbol{W}}} \times \boxed{\boldsymbol{x}^t} \\
&= \boxed{(C, 1)} \times \boxed{(1, D)} \\
&= \boxed{\text{Gradient up to the pre-activation}} \times \boxed{\text{input of the layer}}
\end{aligned}
$$

- $\nabla_{\boldsymbol{W}}$ is a matrix of size (C, D)
- $\nabla_{\boldsymbol{W}}[i, j] = \boldsymbol{\delta}[i] \times \boldsymbol{x}[j]$

Gradient computation with one hidden layer



Two questions (or gradients):

- $\frac{\partial l}{\partial \boldsymbol{W}} = \ ?$
- $\frac{\partial l}{\partial \boldsymbol{V}} = \ ?$

Gradient computation with one hidden layer

Step 1: $\boldsymbol{W}$

$$\frac{\partial l}{\partial \boldsymbol{W}} = \frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \boldsymbol{a}} \times \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}$$

Supervision — loss *vs* output — activation — gradient of $\boldsymbol{W}$

$$\boldsymbol{\delta_W} \times \boldsymbol{z}^t$$

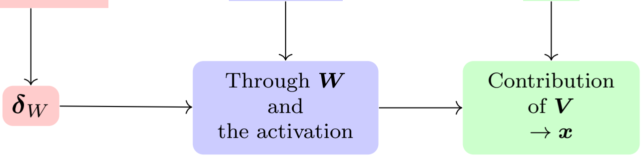
- Very similar to the shallow network, but the “input” is $\boldsymbol{z}$ instead of $\boldsymbol{x}$
- But $\boldsymbol{z}$ is computed by the network (parameters $\boldsymbol{V}$)

Gradient computation with one hidden layer

Step 2 : $\boldsymbol{V}$

Denote the pre-activation of the input layer $\boldsymbol{b} = \boldsymbol{V}\boldsymbol{x}$

$$\frac{\partial l}{\partial \boldsymbol{V}} = \frac{\partial l}{\partial y_{(i)}} \times \frac{\partial y_{(i)}}{\partial \boldsymbol{a}} \times \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{z}} \times \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \times \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{V}}$$



- The error signal $\boldsymbol{\delta}_W$ is back-propagated through $\boldsymbol{W}$,
- to get $\boldsymbol{\delta}_V$
- The update for $\boldsymbol{V}$ is $\nabla_{\boldsymbol{V}} = \boldsymbol{\delta}_V \boldsymbol{z}^t$

The back-propagation algorithm for a feed-forward network with one hidden layer

Introduced in (Rumelhart et al.1986)

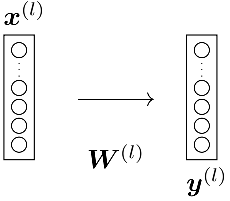
One iteration of the online version, for a given value of $\boldsymbol{\theta}$:

1. Foward propagation of $\boldsymbol{x}_{(i)}$ ($\rightarrow$ $\boldsymbol{z}$ and $\boldsymbol{y}_{(i)}$)
2. Compute the loss
3. Back-propagation and collect of the gradients:
    - output layer: $\boldsymbol{\delta}_W$ and $\nabla_{\boldsymbol{W}} = \boldsymbol{\delta}_W \boldsymbol{z}^t$
    - input layer: $\boldsymbol{\delta}_V$ and $\nabla_{\boldsymbol{V}} = \boldsymbol{\delta}_V \boldsymbol{x}^t$
4. Update parameters:

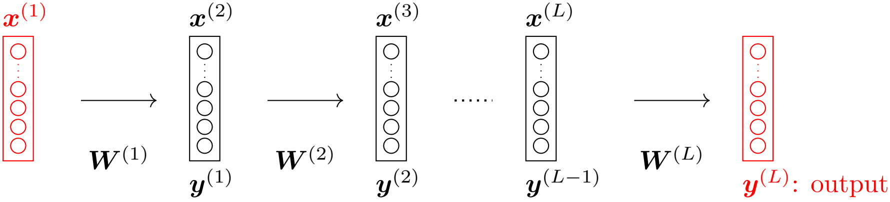
$$\boldsymbol{W} = \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}}$$
$$\boldsymbol{V} = \boldsymbol{V} - \eta \nabla_{\boldsymbol{V}}$$

Notations for a multi-layer neural network (feed-forward)

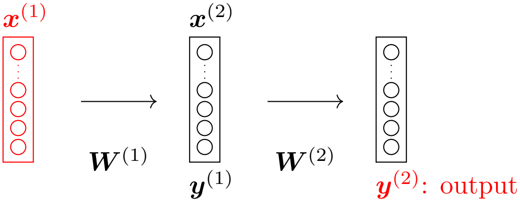
One layer, indexed by l



- $\boldsymbol{x}^{(l)}$: input of the layer l
- $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\ \boldsymbol{x}^{(l)})$
- stacking layers: $\boldsymbol{y}^{(l)} = \boldsymbol{x}^{(l+1)}$
- $\boldsymbol{x}^{(1)}$ = a data example



Example : with one hidden layer



$$\theta = (W^{(1)}, W^{(2)})$$

To learn, we need the gradients for:

- the output layer: $\nabla_{W^{(2)}}$
- the hidden layer: $\nabla_{W^{(1)}}$

Back-propagation: generalization

For a hidden layer l:

- The gradient at the pre-activation level:

$$\boldsymbol{\delta}^{(l)} = {f'}^{(l)}(\boldsymbol{a}^{(l)}) \circ \left(\boldsymbol{W}^{(l+1)^t}\boldsymbol{\delta}^{(l+1)}\right)$$

- The update is as follows:

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)} \boldsymbol{x}^{(l)^t}$$

The layer should keep:

- $\boldsymbol{W}^{(l)}$: the parameters
- $f^{(l)}$: its activation function
- $\boldsymbol{x}^{(l)}$: its input
- $\boldsymbol{a}^{(l)}$: its pre-activation associated to the input
- $\boldsymbol{\delta}^{(l)}$: for the update and the back-propagation to the layer $l-1$

Back-propagation: one training step

Pick a training example: $\boldsymbol{x}^{(1)} = \boldsymbol{x}_{(i)}$

Forward pass

For $l = 1$ to $(L-1)$

- Compute $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)})$
- $\boldsymbol{x}^{(l+1)} = \boldsymbol{y}^{(l)}$

$\boldsymbol{y}^{(L)} = f^{(L)}(\boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)})$

Backward pass

Init: $\boldsymbol{\delta}^{(L)} = \nabla_{\boldsymbol{a}^{(L)}}$
For $l = L$ to 2 // all hidden units

- $\boldsymbol{\delta}^{(l-1)} = f'^{(l-1)}(\boldsymbol{a}^{(l-1)}) \circ \left(\boldsymbol{W}^{(l)^t}\boldsymbol{\delta}^{(l)}\right)$
- $\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \eta_t \boldsymbol{\delta}^{(l)}\boldsymbol{x}^{(l)^t}$

$\boldsymbol{W}^{(1)} = \boldsymbol{W}^{(1)} - \eta_t \boldsymbol{\delta}^{(1)}\boldsymbol{x}^{(1)^t}$

Conclusion on back-propagation for one layer l

Training a NNet relies on forward-backward propagation.

Forward:

- get $\boldsymbol{x}^{(l)}$ for the previous layer;
- compute and send $\boldsymbol{y}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)})$.

Backward:

- get $\boldsymbol{\delta}^{(l)}$ as input from the up-comming layer;
- compute and send $\boldsymbol{\delta}^{(l-1)}$ to the previous layer;
- update parameters $\boldsymbol{W}^{(l)}$

Initialization recipes

A difficult question with several empirical answers.

One standard trick

$$\boldsymbol{W} \sim \mathcal{N}(0, \frac{1}{\sqrt{n_{in}}})$$

with n_{in} is the number of inputs

A more recent one

$$\boldsymbol{W} \sim \mathcal{U}\Big[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\Big]$$

with n_{in} is the number of inputs

Outline

Summary

Multi-layered Perceptron (MLP) or feed-forward NNet

- Artificial neurons are organizes in **layers**: $\rightarrow$ a vector
- Two layers are in general **fully connected**
  - A linear transformation parametrized by a matrix
  - followed by a pointwise non-linear function, the activation function
- A **feed-forward** architecture is a stack of layers fully connected

$\rightarrow$ Can approximate any functions depending on the number of hidden units.

Training by back-propagation

- After a forward pass (inference from input to the output)
- Backward pass (compute the gradients of each layer from the output to the input)

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.
1986.
Learning representations by back-propagating errors.
Nature, 323(6088):533–536, 10.