Introduction aux réseaux récurrents

Adrien CHAN-HON-TONG
ONERA

Introduction

Deep learning - apprentissage profond

Introduction

La classification binaire

On voudrait un système f qui prend une observation $x \in \mathbb{R}^D$
En déduit la classe de cette observation $f(x) \in \{-1, 1\}$

Exemple : à partir d'une image $x \in \mathbb{R}^D$, je voudrais savoir si c'est une image de chat $f(x) \in \{-1, 1\}$.
On peut sur un certain nombre d'exemples, vérifier si $f(x)$ est bien égale à la valeur voulue $y(x)$.

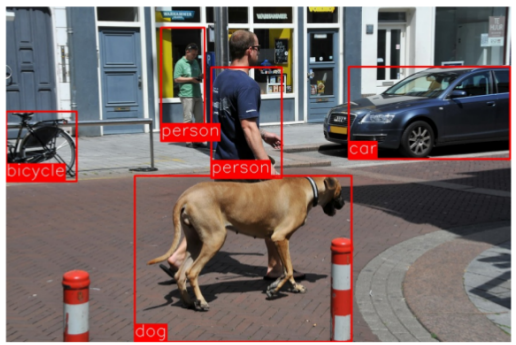
chat

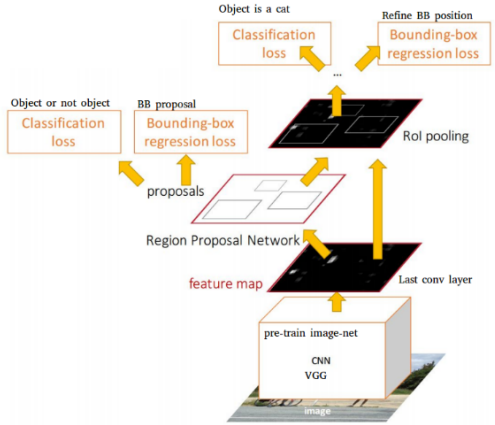
non chat

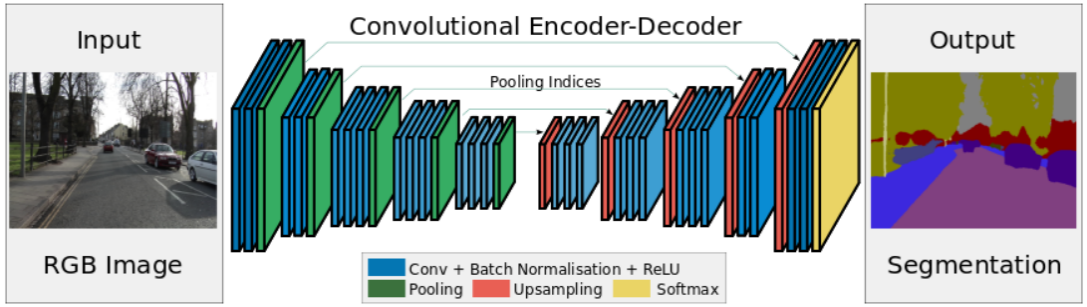
non chat

Introduction

Au delà de la classification

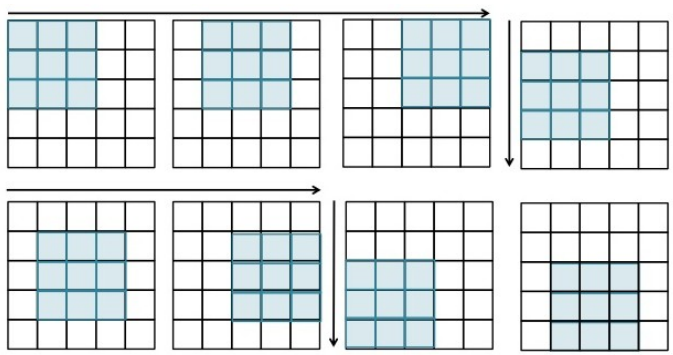


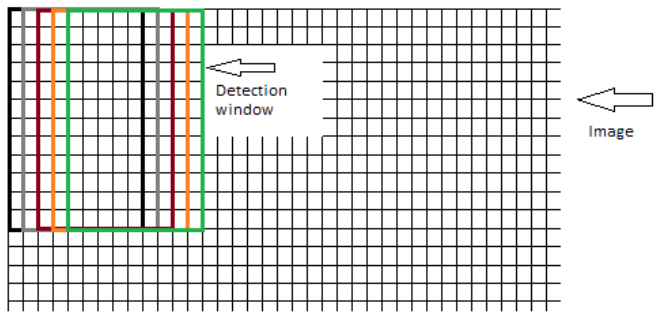




Introduction

Si on sait classer, on sait détecter et segmenter





Classification supervisée

- prend en entrée une **base d'apprentissage** : un ensemble d'observations étiquetées $x_1, y(x_1), ..., x_N, y(x_N)$ par la fonction désirée y (à valeur dans $\{-1, 1\}$).
- produit un modèle f : une fonction qui, à x, associe une classe $f(x) \in \{-1, 1\}$
- ce modèle est évalué sur une **base de test** disjointe de la base d'apprentissage $\chi_1, y(\chi_1), ..., \chi_N, y(\chi_N)$ en calculant :

$$\frac{1}{N}\sum_{n=1}^{N}|f(\chi_i) - y(\chi_i)| \approx \int_{\mathbb{R}^D} |f(x) - y(x)|P(x)\mathrm{d}x = E_P[|f - y|]$$

Du neurone au réseau

Le neurone

Dans les architectures de réseaux de neurones (profond ou pas), le neurone est un filtre linéaire couplé à une activation non linéaire par exemple $\max(0, x)$:

$$\begin{array}{rccc} neurone_{w,b} : & \mathbb{R}^D & \rightarrow & \mathbb{R} \\ & x & \rightarrow & \max(0, wx + b) \end{array}$$

$w \in \mathbb{R}^D$ et $b \in \mathbb{R}$ sont les **poids** du neurones.
$\max(0, x)$ est appelé **relu** - parfois noté $[x]_+$

Attention, pas d'activation sur le tout dernier neurone !
Plutôt la fonction signe.

Du neurone au réseau

La couche de neurone

Une couche de K de neurones est une séquence de K neurones prenant la même entrée, et, dont les K sorties sont regroupées en 1 vecteur :

$$\begin{array}{rccc} & \mathbb{R}^D & \rightarrow & \mathbb{R}^K \\ couche_{W,b} : & x & \rightarrow & \begin{pmatrix} neurone_{W_1,b_1}(x) \\ \dots \\ neurone_{W_K,b_K}(x) \end{pmatrix} \end{array}$$

$W \in \mathbb{R}^{K \times D}$ et $b \in \mathbb{R}^K$ sont les **poids** de chacun des K neurones.

Du neurone au réseau

Le réseau de neurone

Un réseau de neurones entièrement connectées (multi layer perceptron en anglais) de profondeur Q est un empilement de Q couche de neurones - la dernière est classiquement un seul neurone (sans activation) :

$$\begin{array}{rccc} reseau_\theta : & \mathbb{R}^D & \rightarrow & \mathbb{R} \\ & x & \rightarrow & C_{W_Q,b_Q}(C_{W_{Q-1},b_{Q-1}}(...(C_{W_1,b_1}(x))...)) \end{array}$$

$\theta = (W_1, b_1, ..., W_Q, b_Q)$ est un Q uplets de **poids** de couches de neurones - abrégé C

Du neurone au réseau

le dernier neurone

L'objectif du réseau est de classer l'entrée.
Pour cela, la classe prédite est le signe de la valeur du dernier neurones (sans le relu). Si le dernier neurones a une valeur négative, on dira que le réseau choisit la classe -1, et inversement, si la valeur est positive on dira que le réseau prédit la classe 1.

$$\text{signe}(f(x)) = y(x) \Leftrightarrow y(x)\text{signe}(f(x)) = 1 \Leftrightarrow y(x)f(x) > 0$$

La fonction signe est généralement implicite - on préfère juste avoir $y(x)f(x) > 0$

Apprentissage profond

Quand le réseau a beaucoup de couche, on parle de réseau profond.

Ce terme est apparu en 2012 avec un réseau à 10 couches dit Alexnet (*imagenet classification with deep convolutional neural networks*).

Les réseaux de neurones existent depuis 1950 mais la puissance de calcul n'était pas suffisante pour envisager des réseaux de neurones profonds.

Le neurone convolutif

Est ce qu'une image est $x \in \mathbb{R}^D$?

1 image RGB de 480x340 = 1 vecteur dans $[0, 255]^{163200}$

Différence image vecteur

- Le traitement d'images repose sur les opérations de **convolution** et **pooling** pour passer d'une image à un vecteur.
- le traitement se termine par une classification de vecteurs.

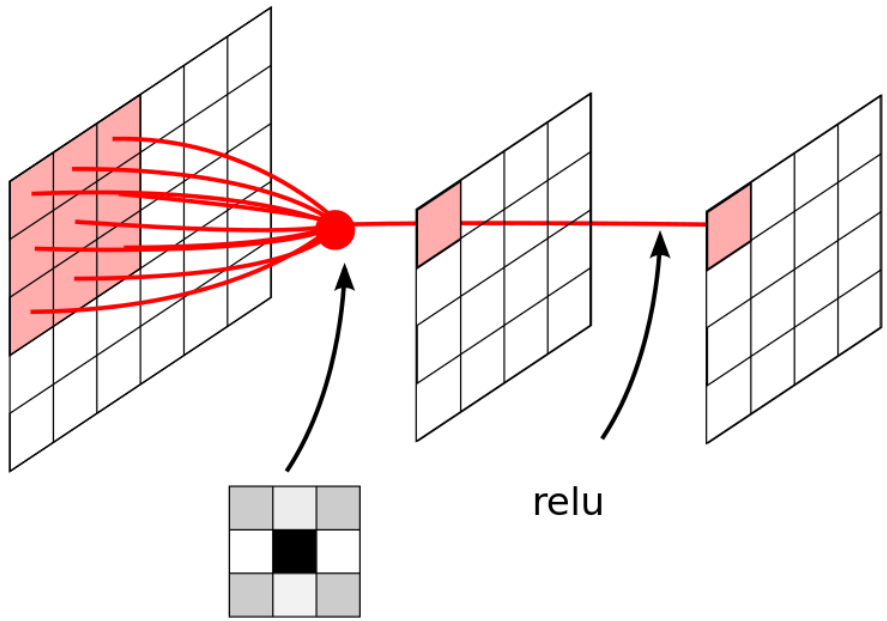
Convolution

- l'entrée est $x \in \mathbb{R}^{H \times W \times Ch}$
- les poids du neurones sont simplement $w \in \mathbb{R}^{(2\delta_H+1)\times(2\delta_W+1)\times Ch}$ (et b)
- la sortie est une image $\chi \in \mathbb{R}^{H \times W}$
- avec $\chi_{h,w}$ égal (avant activation) à :

$$\sum_{dh,dw,ch \in [-\delta_H,\delta_H]\times[-\delta_H,\delta_W]\times[1,Ch]} x_{h+dh,w+dw,ch} \times w_{dh,dw,ch}$$

La convolution est bien un filtre linéaire mais un filtre linéaire local prenant en compte l'aspect spatial des images, et s'applique identiquement en chaque pixel. Elle permet d'extraire de l'image des informations locales (contrastes, texture, bords...) avec un nombre de poids réduit.

Convolution



Convolution

$$conv_{w,b}(x) =$$

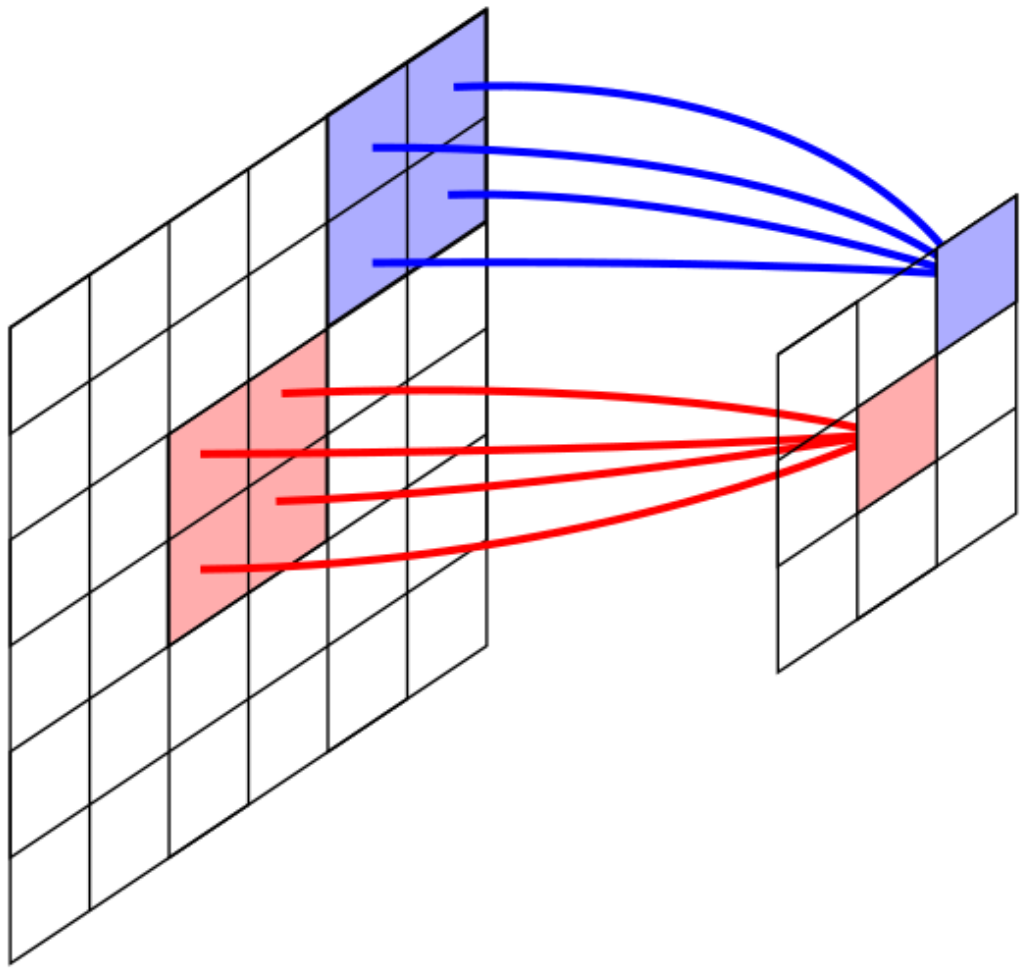
$$\left(\left[\sum_{\substack{dh \in [-\delta_H, \delta_H] \\ dw \in [-\delta_H, \delta_W] \\ ch \in [1, Ch]}} \begin{matrix} x_{h+dh,w+dw,ch} \\ \times w_{dh,dw,ch} \end{matrix} + b \right]_+ \right)_{\substack{h \in [1,H] \\ w \in [1,W]}}$$

Pooling

- l'entrée est $x \in \mathbb{R}^{H \times W \times Ch}$
- la sortie est une image $y \in \mathbb{R}^{\frac{H}{2} \times \frac{W}{2} \times Ch}$
- avec $\chi_{h,w,ch} = \max_{dh,dw \in [0,1]} x_{2h+dh,2w+dw,ch}$

Le pooling n'est pas vraiment une couche de neurones c'est plutôt un autre type d'activation, mais, avec une dimension spatiale. Elle permet d'introduire de l'invariance aux petites déformation.

Pooling



Convolution, Pooling, Stride

Classiquement : la convolution produit une sortie qui a la même taille que l'entrée (au effet de bord près) alors que le pooling produit une sortie 2 fois plus petite.

Cependant, la bonne notion est la notion de **stride** : la convolution a classiquement une stride de 1 car on déplace la fenêtre de 1 case alors que le pooling a une stride de 2 on déplace la fenêtre de 2 cases.
Rien n'interdit d'avoir des convolutions avec des strides supérieurs à 1 ou des pooling avec des strides de 1.

Convolution, Pooling, Stride, Padding

Dans la convolution la valeur en h, w est

$$\sum_{dh,dw,ch \in [-\delta_H,\delta_H]\times[-\delta_H,\delta_W]\times[1,Ch]} x_{h+dh,w+dw,ch} \times w_{dh,dw,ch}$$

Il y a un donc un effet de bord pour $h < \delta_H$...

Une solution à ce problème consiste à considérer au $x = 0$ en dehors de l'emprise initiale. Dans ce cas, la sortie a exactement la même taille que l'entrée.
Informatiquement, cela revient à travailler sur une image où on a rajouter δ_H lignes et δ_W colonnes de part et d'autre.

Conclusion partielle

- Qu'est ce qu'on cherche à faire ?
  $\Rightarrow$ **de la classification binaire**
- Comment fonctionnent ces réseaux de neurones ?
  $\Rightarrow$ **neurone / couche / réseau + convolution / pooling**

Quelles différences théoriques avec les autres méthodes d'apprentissage ?

Aucune

Sauf que les performances de ces algorithmes sont hautes, et, laissent envisager des applications critiques.

Est ce qu'on peut être sur que $f = y$?

On voudrait un système qui :

- prend une observation $x \in \mathbb{R}^D$
- propose une classe pour cette observation $f(x) \in \{-1, 1\}$
- la vraie classe est $y(x)$ (mais on ne connait pas y on peut l'estimer que sur un ensemble de points)

1

-1

-1

Est ce qu'on peut être sur que $f = y$?

Mais, on ne sait **pas** définir formellement le lien entre x et y.

On utilise des **exemples** pour définir le concept sous jacent.

Sachant que cette *définition* restera nécessairement **ambigue**.

Est ce qu'on peut être sur que $f = y$?

1

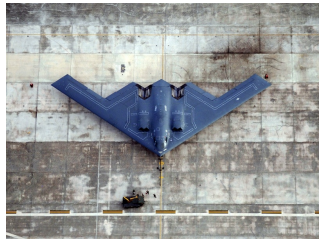
-1

-1

?

?

?



?

Est ce qu'on peut être sur que $f = y$?

1

-1

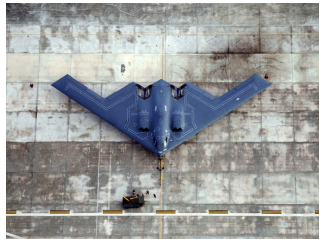
-1

Un détecteur de chat ?

1

-1

-1



-1

Est ce qu'on peut être sur que $f = y$?

1

-1

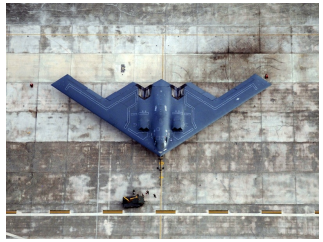
-1

Un détecteur d'animaux

1

1

-1



-1

Est ce qu'on peut être sur que $f = y$?

1

-1

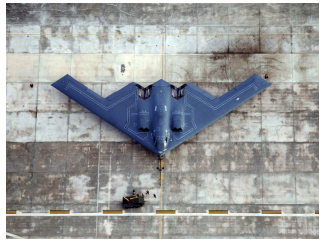
-1

Un détecteur de chat roux ?

-1

-1

-1



-1

Est ce qu'on peut être sur que $f = y$?

1

-1

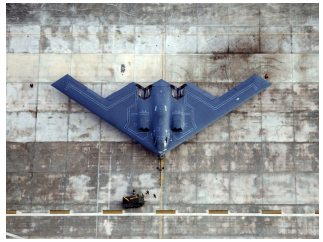
-1

Un détecteur de choses rassurantes ?

1

-1

1



-1

Est ce qu'on peut être sur que $f = y$?

NON

- Si on pouvait être sur que $f = y$, on pourrait directement utiliser y. Dans, ce cas il aurait été **inutile** d'utiliser de l'apprentissage !
- l'apprentissage est la **seule** méthode qui permet de traiter des problèmes non formalisées
- en contre partie, même si un oracle donne la solution y, il n'y a aucun moyen de tester que c'est bien la solution.
- a fortiori, on ne peut et on ne pourra jamais être sur que $f = y$ (même si potentiellement $f = y$)

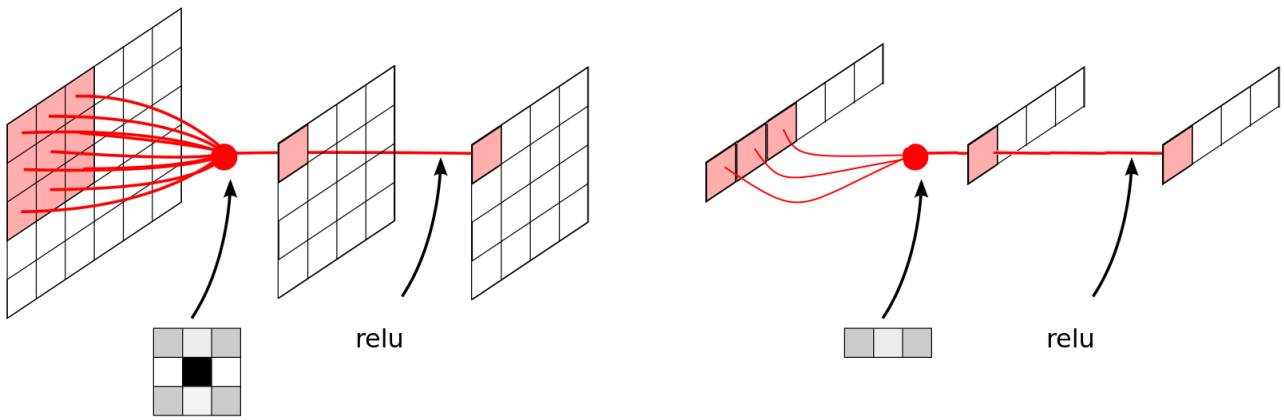
À retenir

Classification supervisée

- prend en entrée une **base d'apprentissage** : un ensemble d'observations étiquetées $x_1, y(x_1), ..., x_N, y(x_N)$ par la fonction désirée y (à valeur dans $\{-1, 1\}$).
- produit un modèle f : une fonction qui, à x, associe une classe $f(x) \in \{-1, 1\}$
- **on ne peut jamais être sur que** $f = y$.
- ce modèle est évalué sur une **base de test** disjointe de la base d'apprentissage $\chi_1, y(\chi_1), ..., \chi_N, y(\chi_N)$ en calculant : $\frac{1}{N} \sum_{n=1}^{N} |f(\chi_i) - y(\chi_i)|$ le taux d'erreur empirique.
- $\frac{1}{N} \sum_{n=1}^{N} |f(\chi_i) - y(\chi_i)| = 0$ n'implique pas $f = y$ (bien que plus N est grand plus c'est probable).

Le traitement de série temporel

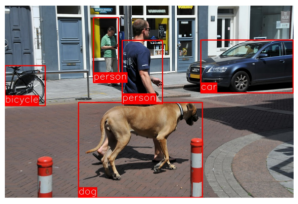
La convolution en 2D vs 1D

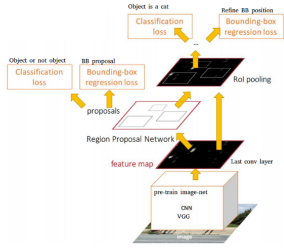


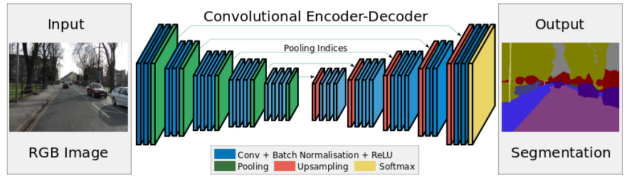
Pareil pour le pooling !

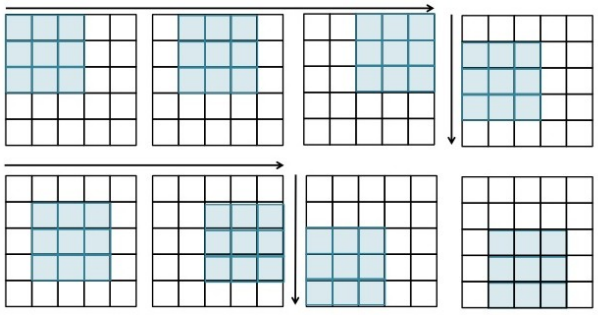
Le traitement de série temporel

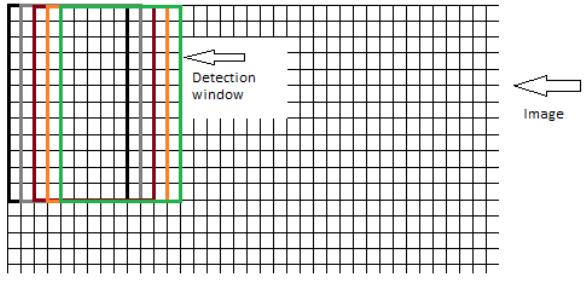
Classification, segmentation, détection 1D





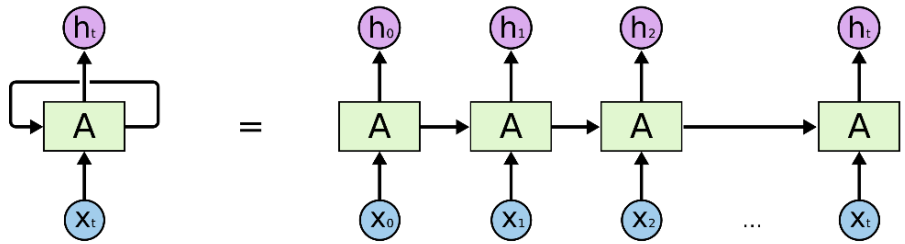






Le traitement de série temporel

Les réseaux récurrents



Le traitement de série temporel

neurone vs convolution vs recurrence

L'entrée est un signal de durée T et de dimension D (ex : le nombre d'électrode) :

- un neurone simple aura $T \times D$ paramètres (il prend le signal entier et le transforme en 1 valeur)
- une convolution de taille τ a $\tau \times D$ paramètres (elle prend le signal entier et produit un signal de même durée et de dimension 1)
- un réseau récurrent a D paramètres et une mémoire M

$\Rightarrow$ la récurrence permet de prendre en compte un large horizon temporel avec un nombre de paramètre borné - mais est plus complexe à optimiser !

Le traitement de série temporel

convolution vs recurrence

Sur un problème simple, les 2 marcheront.

Sur un problème trop dur, aucun des 2 ne marchera.

Sur un problème intérmédiaire, il faut utiliser les a priori :
Est ce que mon signal d'entrée est informatif ?
Quel est l'horizon temporel pertinent ? etc...

Pour finir, 2 mots d'optimisation

La descente de gradient

f est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$ alors
$\forall u, h \in \mathbb{R}^D,\ f(u+h) = f(u) + \nabla f_u | h + \varepsilon(h)h$
avec $\varepsilon(h) \underset{h \to 0}{\to} 0$
Donc si $\nabla f_u \neq 0$ alors il existe $\lambda > 0$ tel que $f\left(u - \lambda \nabla f_u\right) < f(u)$

Pour finir, 2 mots d'optimisation

La descente de gradient en machine learning

u correspond à l'ensemble des poids

f est la fonction de perte, elle doit être minimale quand on a atteint le comportement espéré

exemple :
$(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}$

$$f(u) = \sum_{i=1}^{n} relu(1 - y_i reseau_u(x_i))$$

En pratique, on utilise une descente de gradient stochastique.

À retenir

Le neurone

$$neurone_{w,b} : \begin{array}{ccc} \mathbb{R}^D & \rightarrow & \mathbb{R} \\ x & \rightarrow & wx + b \end{array}$$

- associé à une fonction d'activation (sauf à la fin)
- regroupé en couche
- le réseau est un empilement de couche
- il existe des neurones convolutifs quand l'information est porté par les voisins
- réseau récurrent : moins de poids pour un horizon donné mais moins stable

À retenir

Classification supervisée

- prend en entrée une **base d'apprentissage** : un ensemble d'observations étiquetées $x_1, y(x_1), ..., x_N, y(x_N)$ par la fonction désirée y (à valeur dans $\{-1, 1\}$).
- produit un modèle f : une fonction qui, à x, associe une classe $f(x) \in \{-1, 1\}$
- **on ne peut jamais être sur que** $f = y$.
- ce modèle est évalué sur une **base de test** disjointe de la base d'apprentissage $\chi_1, y(\chi_1), ..., \chi_N, y(\chi_N)$ en calculant : $\frac{1}{N} \sum_{n=1}^{N} |f(\chi_i) - y(\chi_i)|$ le taux d'erreur empirique.
- $\frac{1}{N} \sum_{n=1}^{N} |f(\chi_i) - y(\chi_i)| = 0$ n'implique pas $f = y$ (bien que plus N est grand plus c'est probable).