

Réseau de neurones

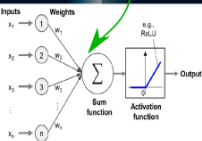
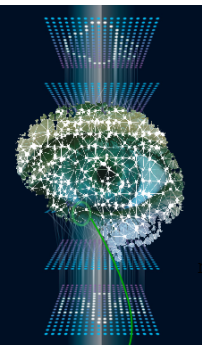
Cours 5 ML Master DAC

Nicolas Baskiotis

`nicolas.baskiotis@sorbonne-universite.fr`

équipe MLIA,
Institut des Systèmes Intelligents et de Robotique (ISIR)
Sorbonne Université

S2 (2022-2023)



Résumé des épisodes

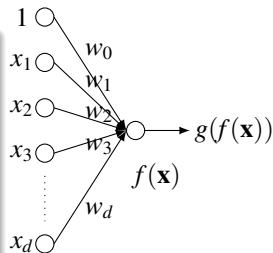
Problématique de l'apprentissage supervisé

- Ensemble d'apprentissage $\{(\mathbf{x}^i, y^i)\} \in X \times Y$, ensemble de fonctions \mathcal{F}
- un coût $L(\hat{y}, y) : Y \times Y \rightarrow \mathbb{R}^+$, trouver $f^* = \operatorname{argmin}_{f \in \mathcal{F}} \sum_i L(f(\mathbf{x}^i), y^i)$

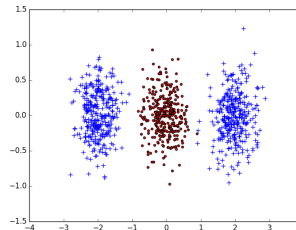
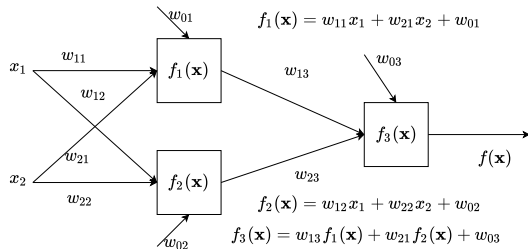
Perceptron

- Hypothèse linéaire : $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i$
- Coût perceptron : $L(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -f_{\mathbf{w}}(\mathbf{x})y)$
- Gradient :

$$\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} 0 & \text{si } (-y < \mathbf{w} \cdot \mathbf{x}) < 0 \\ -y\mathbf{x} & \text{sinon} \end{cases}$$

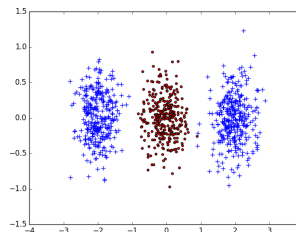
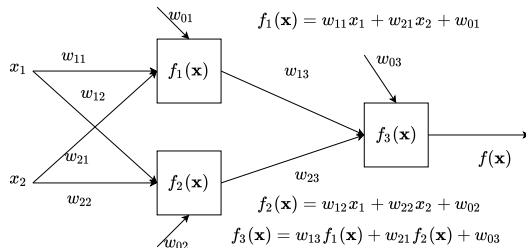


Deux neurones



- Combiner des neurones \Rightarrow suffisant ?

Deux neurones



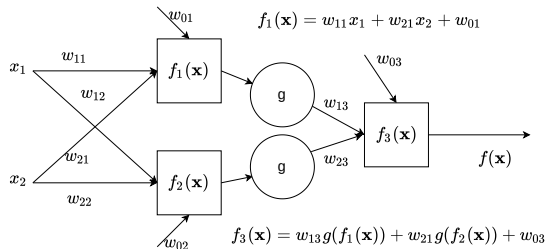
- Combiner des neurones \Rightarrow suffisant ?

$$f(\mathbf{x}) = w_{03} + w_{13}(w_{01} + w_{11}x_1 + w_{12}x_2) + w_{23}(w_{02} + w_{12}x_1 + w_{22}x_2)$$

$$= w_{03} + w_{13}w_{01} + w_{23}w_{02} + x_1(w_{13}w_{11} + w_{23}w_{12}) + x_2(w_{11}w_{21} + w_{23}w_{22})$$

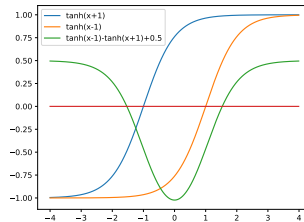
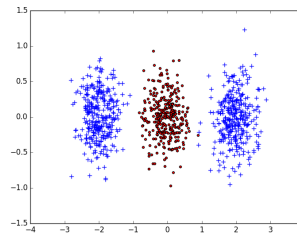
Non ! il faut introduire de la non linéarité, sinon équivalent à un perceptron ...

Deux neurones

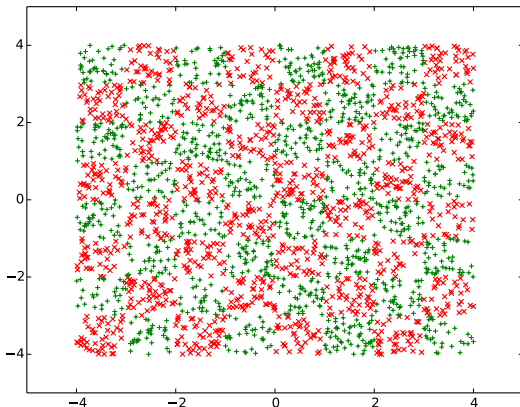


- Quelle non-linéarité ?

- ▶ Fonction *signe* ?
- ⇒ dérivée problématique ...
- ▶ Fonctions *tanh*, *sigmoïde*, ...



Et pour ce problème ?

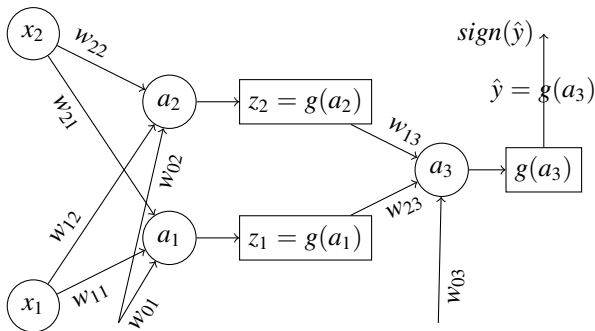


Si vous aviez droit à n'importe quelle fonction dans un neurone ?

Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)

Pour l'inférence



Inférence

Avec $g(x) = \tanh(x)$

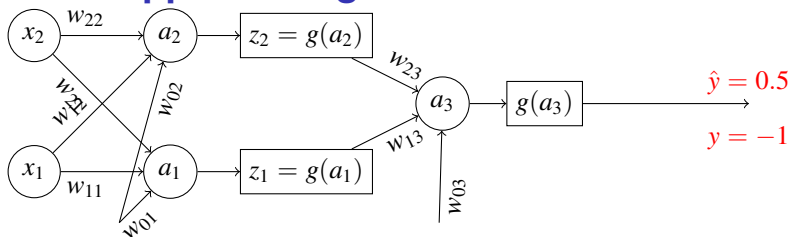
- $a_1 = w_{01} + w_{11}x_1 + w_{21}x_2$
- $a_2 = w_{02} + w_{12}x_1 + w_{22}x_2$
- $z_1 = g(a_1)$
- $z_2 = g(a_2)$
- $a_3 = w_{03} + w_{13}z_1 + w_{23}z_2$
- $\hat{y} = g(a_3)$

⇒ prédiction : $sign(\hat{y})$

Vocabulaire

- Inférence : *pas forward*
- g fonction d'activation (non linéarité du réseau)
- a_i activation du neurone i
- z_i sortie du neurone i (transformée non linéaire de l'activation).

Pour l'apprentissage



Objectif : apprendre les poids

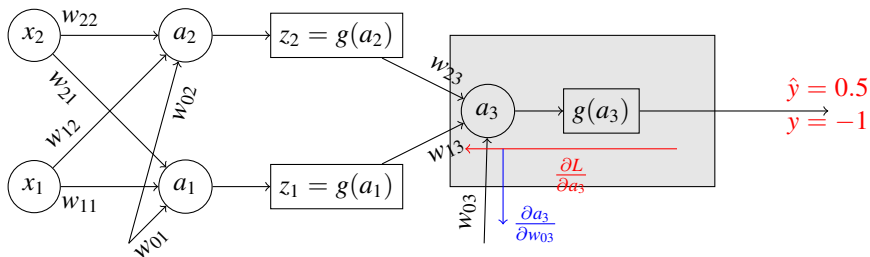
- Choix d'un coût : moindres carrés
 $L(\hat{y}, y) = (\hat{y} - y)^2$ (pourquoi est ce un bon choix ?)
- Mais à quel(s) neurone(s) et comment répartir l'erreur entre les poids ?

⇒ Rétro-propagation de l'erreur :

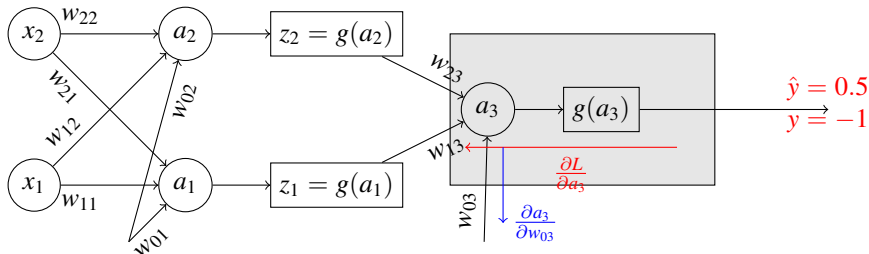
- ▶ corriger un peu tous les poids ...
- ▶ en estimant la part de chacun dans l'erreur
- ▶ en commençant par la fin et en figeant au fur et à mesure le réseau

⇒ descente de gradient : on cherche à calculer tous les $\frac{\partial L(\hat{y}, y)}{\partial w_{ij}}$

Pour l'apprentissage



Pour l'apprentissage



Pour les poids de la dernière couche : gradient $\nabla_{w_{i3}} L(\hat{y}, y)$

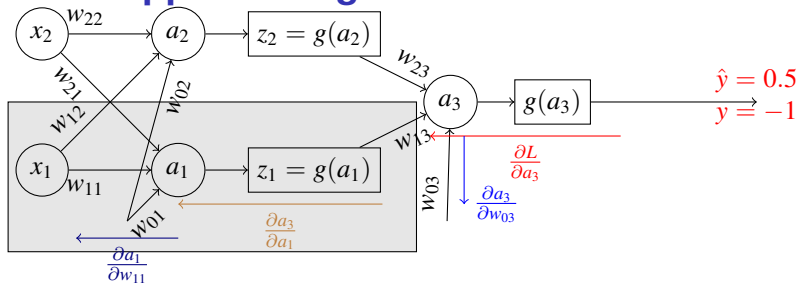
On a : $L(\hat{y}, y) = (g(a_3) - y)^2 = (g(w_{03} + w_{13}z_1 + w_{23}z_2) - y)^2$

$$\frac{\partial L}{\partial w_{i3}} = \frac{\partial L}{\partial a_3} \frac{\partial a_3}{\partial w_{i3}} \quad \text{avec} \quad \left| \begin{array}{l} \frac{\partial L}{\partial a_3} = \frac{\partial L}{\partial g(a_3)} \frac{\partial g(a_3)}{\partial a_3} = \frac{\partial (g(a_3) - y)^2}{\partial a_3} = 2g'(a_3)(g(a_3) - y) \\ \frac{\partial a_3}{\partial w_{i3}} = \frac{\partial w_{03} + w_{13}z_1 + w_{23}z_2}{\partial w_{i3}} = z_i \end{array} \right.$$

Soit

$$\frac{\partial L}{\partial w_{i3}} = 2g'(a_3)(\hat{y} - y)z_i$$

Pour l'apprentissage



Pour les poids de la première couche: w_{i1} par exemple

$$\frac{\partial L}{\partial w_{i1}} = \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial w_{i1}} \quad \text{avec} \quad \left| \begin{array}{l} \frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial a_3} \frac{\partial a_3}{\partial a_1} = \frac{\partial L}{\partial a_3} g'(a_1) w_{13} \\ \frac{\partial a_1}{\partial w_{i1}} = \frac{w_{01} + w_{11}x_1 + w_{21}x_2}{\partial w_{i1}} = x_i \end{array} \right.$$

Soit

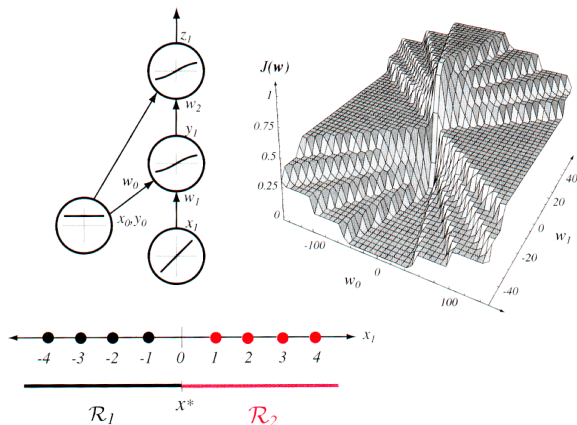
$$\underbrace{\frac{\partial L}{\partial w_{i1}}}_{\text{correction de } w_{i1}} = \frac{\partial L}{\partial a_1} x_i = \underbrace{\frac{\partial L}{\partial a_3}}_{\text{erreur à propager}} \underbrace{g'(a_1) w_{13}}_{\text{poids de la connexion}} x_i$$

$\Rightarrow \frac{\partial L}{\partial a_i}$: joue le rôle de l'erreur à retro-propager dans les couches inférieures.

Plan

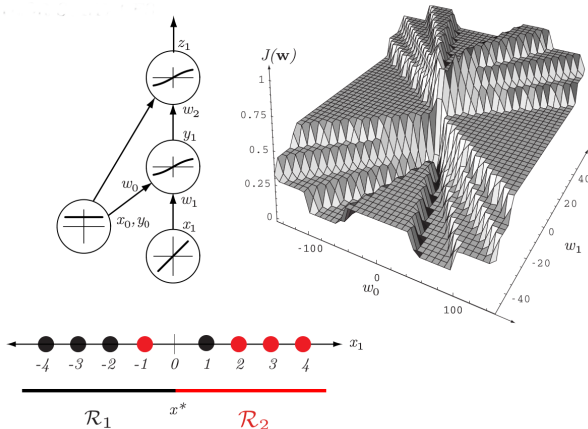
- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)**
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)

Analyse de la surface d'erreur

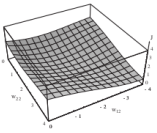
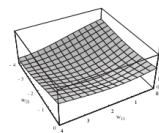
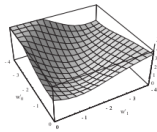
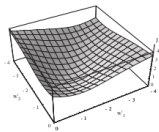
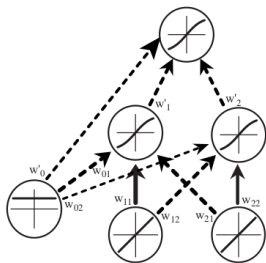


Analyse de la surface d'erreur

Figure 12.11: Surface d'erreur

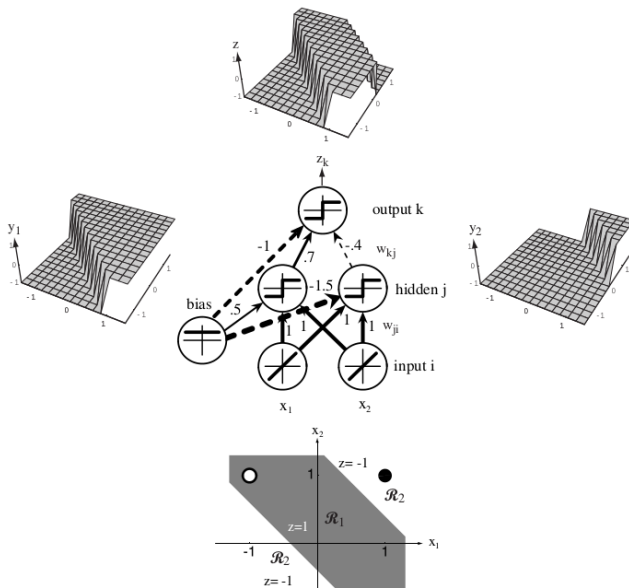


Analyse de la surface d'erreur



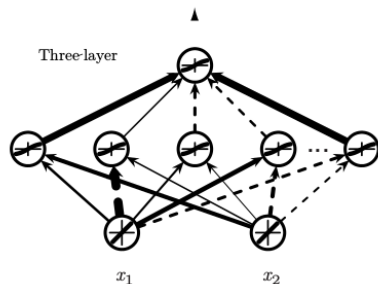
Exemple

Le XOR selon [Duda et al 00]

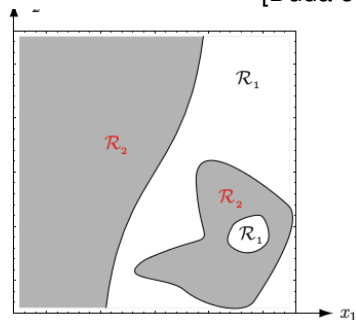


Exemple

Non convexité des régions apprises



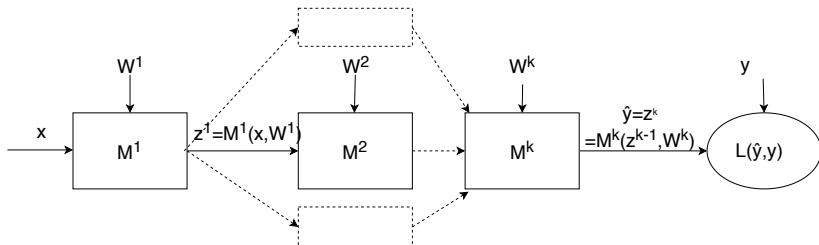
[Duda et al 00]



Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire**
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)

Assemblage de modules

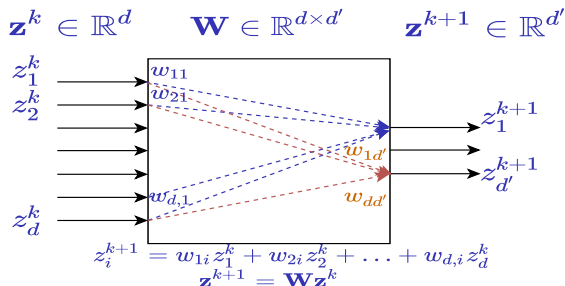


Un module M^k est caractérisé

- par ses entrées : le résultat de la couche précédente z^{k-1} (et potentiellement d'autres variables)
- par ses paramètres W^k
- produit une sortie $z^k = M^k(z^{k-1}, W^k)$

D'un point de vue formel, il n'y a pas de différences entre les paramètres du module et les entrées : ce sont tous des arguments de la fonction du module.

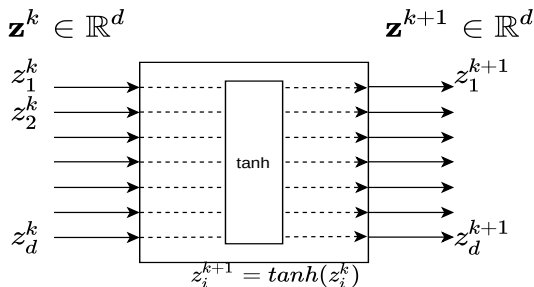
Type usuel de modules : Module linéaire



Transformation linéaire paramétrée de \mathbb{R}^d vers $\mathbb{R}^{d'}$

- $\mathbf{z}^k = M^k(\mathbf{z}^{k-1}, W^k) = W^k \mathbf{z}^{k-1}$ avec $W^k \in \mathbb{R}^d \times \mathbb{R}^{d'}$, $\mathbf{z}^{k+1} \in \mathbb{R}^{d'}$
- Chaque sortie $z_i^{k+1} = W_{i,\cdot}^k \mathbf{z}^k = \langle \mathbf{w}_i^k, \mathbf{z}^k \rangle$ correspond au calcul d'un perceptron
- La matrice W^k est l'empilement des \mathbf{w}_i , poids de chaque perceptron.

Type usuel de modules : Module d'activation



Fonction d'activation de \mathbb{R}^d vers \mathbb{R}^d

- tangente hyperbolique :

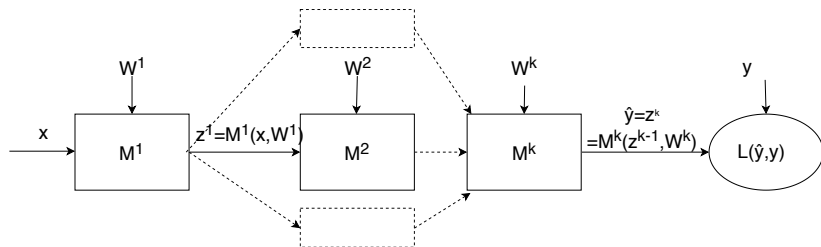
$$M^k(\mathbf{z}^{k-1}, 0) = \tanh(\mathbf{z}^{k-1}) = (\tanh(z_1^{k-1}), \tanh(z_2^{k-1}), \dots, \tanh(z_d^{k-1}))$$

- sigmoïde : $M^k(\mathbf{z}^{k-1}, 0) = \sigma(\mathbf{z}^{k-1}) = (\sigma(z_1^{k-1}), \sigma(z_2^{k-1}), \dots, \sigma(z_d^{k-1}))$

- ReLU :

$$M^k(\mathbf{z}^{k-1}, 0) = \text{ReLU}(\mathbf{z}^{k-1}) = (\max(0, z_1^{k-1}), \max(0, z_2^{k-1}), \dots, \max(0, z_d^{k-1}))$$

Type usuel de modules : Module de coût

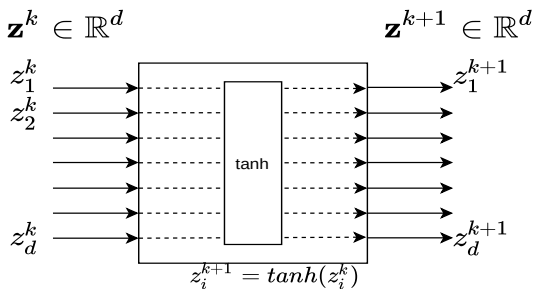


Fonction de coût

Bloc final : deux entrées, la supervision et la sortie du réseau $\hat{y} = z^k$.

- MSE : $L(\hat{y}, y) = \|\hat{y} - y\|^2$
- Negative Log-Likelihood : $L(\hat{y}, y) = -\sum_{i=1}^d y_i \log \hat{y}_i$
- KL-divergence : $L(\hat{y}, y) = -\sum_{i=1}^d y_i \log \frac{\hat{y}_i}{y_i}$

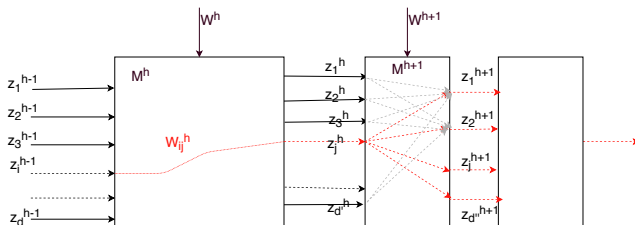
Rétro-propagation du gradient



Pour apprendre le réseau :

- Pour chaque module, il faut calculer $\nabla_{w^k} L(\hat{y}, y)$
- Cas simple : paramètres constants (module d'activation), le gradient est nul (il n'y a rien à apprendre pour ce module)
- Rétro-propagation pour les autres.

Zoom sur un module linéaire

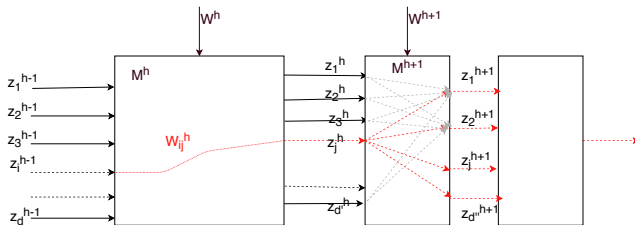


Rétro-propagation pour M^h , $z^h = M(z^{h-1}, W^h)$ avec $W^h \in \mathbb{R}^{d^{h-1}} \times \mathbb{R}^{d^h}$

- $\frac{\partial L}{\partial w_{ij}^h} = \sum_{k=1}^{d^h} \frac{\partial L}{\partial z_k^h} \frac{\partial z_k^h}{\partial w_{ij}^h} = \frac{\partial L}{\partial z_j^h} \frac{\partial z_j^h}{\partial w_{ij}^h} = \frac{\partial L}{\partial z_j^h} \frac{\partial M^h(z^{h-1}, W^h)}{\partial w_{ij}^h}$ (w_{ij}^h n'influe que sur z_j^h)
- $\frac{\partial L}{\partial z_j^h} = \sum_{k=1}^{d^{h+1}} \frac{\partial L}{\partial z_k^{h+1}} \frac{\partial z_k^{h+1}}{\partial z_j^h} = \sum_{k=1}^{d^{h+1}} \frac{\partial L}{\partial z_k^{h+1}} \frac{\partial M^{h+1}(z^h, W^{h+1})}{\partial z_j^h}$
- On introduit $\delta_j^h = \frac{\partial L}{\partial z_j^h} = \sum_{k=1}^{d^{h+1}} \delta_k^{h+1} \frac{\partial M^{h+1}(z^h, W^{h+1})}{\partial z_j^h}$: $\frac{\partial L}{\partial w_{ij}^h} = \delta_j^h \frac{\partial M^h(z^{h-1}, W^h)}{\partial w_{ij}^h}$

⇒ Les δ_i^h sont calculés en partant de la fin du réseau

Zoom sur un module linéaire



Rétro-propagation pour $M^h, z^h = M(z^{h-1}, W^h)$ avec $W^h \in \mathbb{R}^{d^{h-1}} \times \mathbb{R}^{d^h}$

Avec
$$\delta_j^h = \frac{\partial L}{\partial z_j^h} = \sum_{k=1}^{d^{h+1}} \delta_k^{h+1} \frac{\partial M^{h+1}(z^h, W^{h+1})}{\partial z_j^k} : \frac{\partial L}{\partial w_{ij}^h} = \delta_j^h \frac{\partial M^h(z^{h-1}, W^h)}{\partial w_{ij}^h}$$

Pour chaque module, on a besoin :

- du gradient $\nabla_{W^h} M^h(z^{h-1}, W^h)$ par rapport à ses paramètres : maj des paramètres (nul si pas de paramètres)
- du gradient $\nabla_{z^{h-1}} M^h(z^{h-1}, W^h)$ par rapport à ses entrées : rétro-propagation de l'erreur

Complexité et expressivité

- Efficacité en apprentissage
 - ▶ En $O(|w|)$ pour chaque passe d'apprentissage où $|w|$ est le nombre de poids
 - ▶ Il faut typiquement plusieurs centaines de passes (voir plus loin)
 - ▶ Il faut typiquement recommencer plusieurs dizaines de fois un apprentissage en partant avec différentes initialisations des poids
- Efficacité en reconnaissance
 - ▶ Possibilité de temps réel

Expressivité

- Quelle influence du nombre de couches ?
 - du nombre de neurones par couche ?
- ⇒ Une couche cachée suffit pour un apprentissage universel ! Mais ...

Vers les réseaux profonds

Problème : plus le réseau est profond plus il est dur à entraîner

- le gradient s'évapore (*vanishing*)
- le sur-apprentissage est très favorisé

Quelques solutions

- utiliser des architectures peu propices au sur-apprentissage (convolutives, RBM, ...)
- Early-stopping
- Apprentissage en bruitant les données d'entrées

⇒ pas suffisant

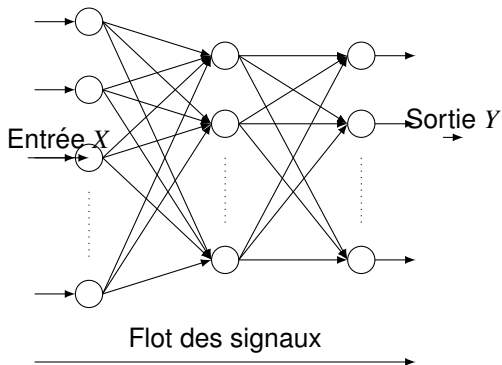
- Première passe d'apprentissage pour "bien initialiser" les couches ⇒ (apprentissage couche par couche, auto-encoders)
- Drop-out : permet de limiter le sur-apprentissage (éteindre/supprimer un nombre de neurones aléatoirement pendant l'apprentissage)
- utilisation de fonctions d'activation spécifiques (ReLU etc)
- utilisation d'architecture spécifiques (couches résiduelles etc)

Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)**
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)

Topologie typique

Couche d'entrée Couche cachée Couche de sortie



Pour chaque neurone k , la sortie z_k

$$z_k = g \left(\sum_{j=0}^d w_{j,k} z_j \right) = g(a_k)$$

où

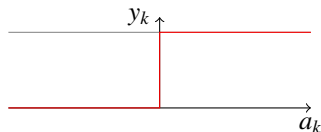
- $w_{j,k}$: poids de la connexion de cellule j à la cellule k
- a_k : activation de la cellule k
 $a_k = \sum_{j=0}^d w_{j,k} z_j$
- g : fonction d'activation

Apprentissage :

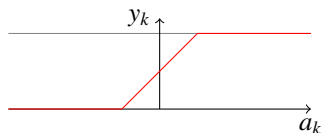
- Minimiser la fonction de coût $L(W, \{X, Y\})$ en fonction du paramètre $W = (w_{i,j})$
- Algorithme de rétro-propagation de gradient $\Delta w_{i,j} \propto \frac{\partial L}{\partial w_{i,j}}$

Fonction d'activation

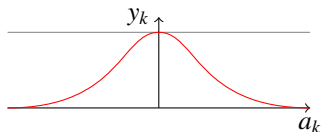
- Fonction à seuil



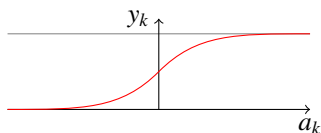
- Fonction à rampe



- Fonction radiale



- Fonction sigmoïde



- ▶ $g(a) = \frac{1}{1 + \exp(-a)}$
- ▶ $g'(a) = g(a)(1 - g(a))$

Algorithme

- 1 Présentation d'un/des exemple(s) parmi l'ensemble d'apprentissage
- 2 Calcul de l'état du réseau (phase forward)
- 3 Calcul de l'erreur avec un coût donné : e.g. $= (y - \hat{y})^2$
- 4 Calcul des gradients (par l'algorithme de rétro-propagation du gradient)
- 5 Modification des poids
- 6 Critère d'arrêt (sur l'erreur, nombre de présentation d'exemples...)
- 7 Retour en 1

La rétro-propagation de gradient

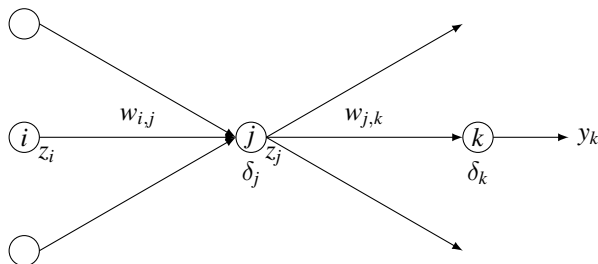
- Le problème :
 - ▶ Détermination des responsabilités (credit assignment problem)
 - ▶ Quelle connexion est responsable, et de combien, de l'erreur ?
- Principe :
 - ▶ Calculer l'erreur sur une connexion en fonction de l'erreur sur la couche suivante
- Deux étapes :
 - 1 Evaluation des dérivées de l'erreur par rapport aux poids
 - 2 Utilisation de ces dérivées pour calculer la modification de chaque poids

La rétro-propagation de gradient

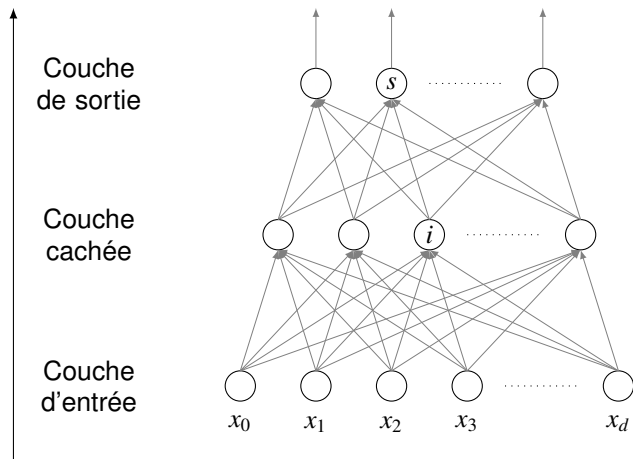
- a_i : activation de la cellule i
- z_i : sortie de la cellule i
- δ_i : erreur attachée à la cellule i

cellule cachée cellule de sortie

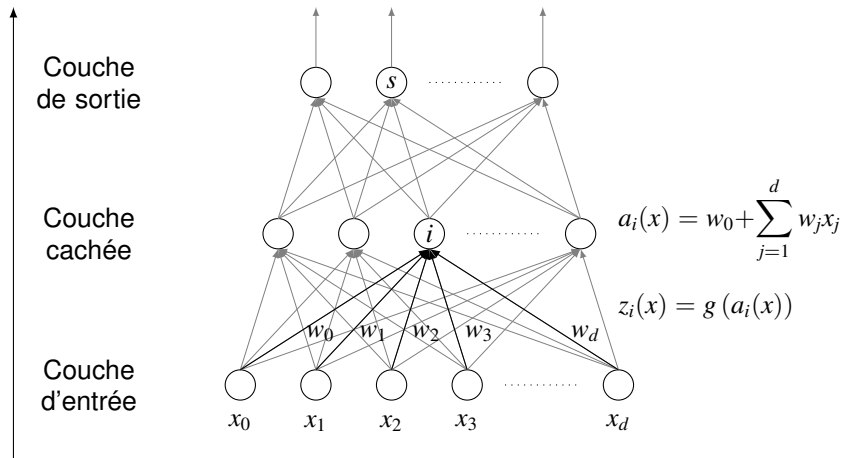
↓ ↓



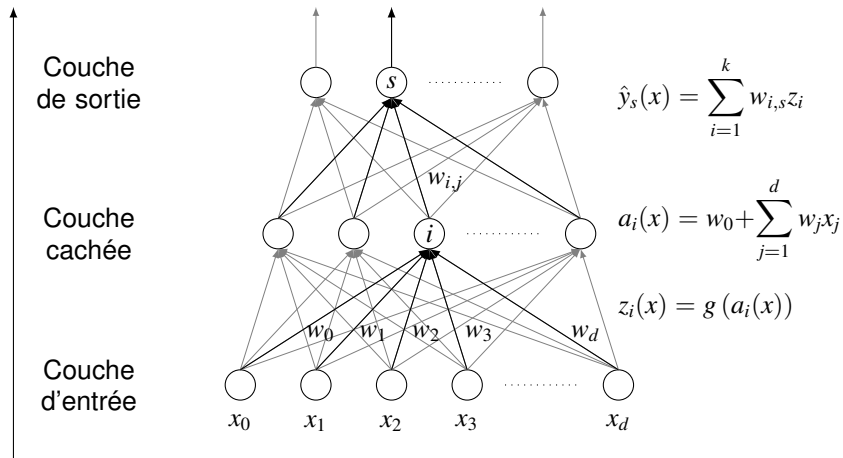
Passe avant (forward) Illustrations J.-N. Vittaut



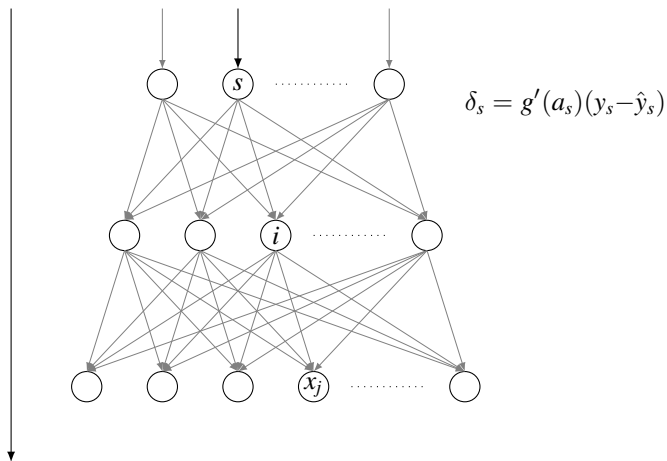
Passe avant (forward) Illustrations J.-N. Vittaut



Passe avant (forward) Illustrations J.-N. Vittaut

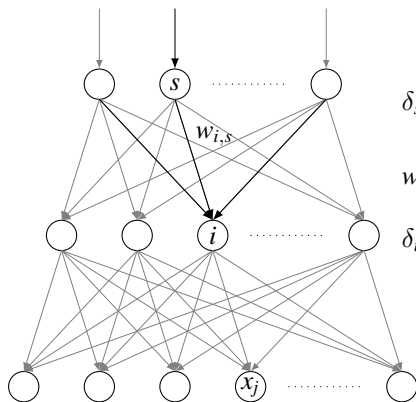


Passe arrière (backward)



- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

Passe arrière (backward)



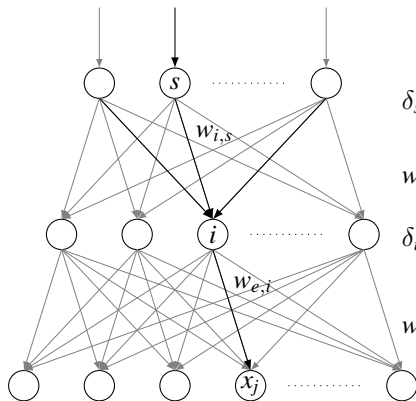
$$\delta_s = g'(a_s)(y_s - \hat{y}_s)$$

$$w_{i,s}^{t+1} = w_{i,s}^t - \eta(t) \delta_s a_i$$

$$\delta_i = g'(a_i) \sum_{s \in \text{coucheSuiv.}} w_{i,s} \delta_s$$

- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

Passe arrière (backward)



$$\delta_s = g'(a_s)(y_s - \hat{y}_s)$$

$$w_{i,s}^{t+1} = w_{i,s}^t - \eta(t) \delta_s a_i$$

$$\delta_i = g'(a_i) \sum_{s \in \text{couche Suiv.}} w_{i,s} \delta_s$$

$$w_{x_j,i}^{t+1} = w_{x_j,i}^t - \eta(t) \delta_i x_j$$

- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

La rétro-propagation de gradient

- 1. Evaluation de l'erreur L due à chaque connexion : $\frac{\partial L}{\partial w_{i,j}}$
 - calculer l'erreur sur la connexion $w_{i,j}$ en fonction de l'erreur après la cellule j

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}} = \delta_j z_i$$

- Pour les cellules de la couche de sortie :

$$\delta_k = \frac{\partial L}{\partial a_k} = g'(a_k)(y_k - \hat{y}_k)$$

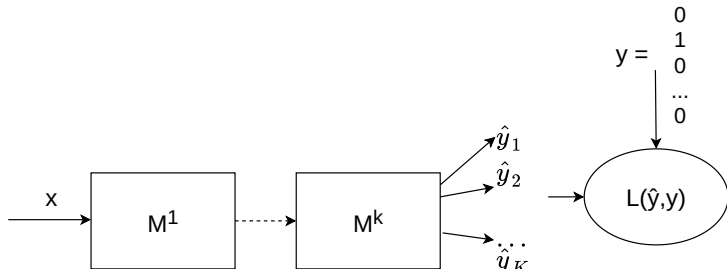
- Pour les cellules d'une couche cachée :

$$\delta_j = \frac{\partial L}{\partial a_j} = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = g'(a_j) \cdot \sum_k w_{j,k} \delta_k$$

Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe**
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)

Multi-classe



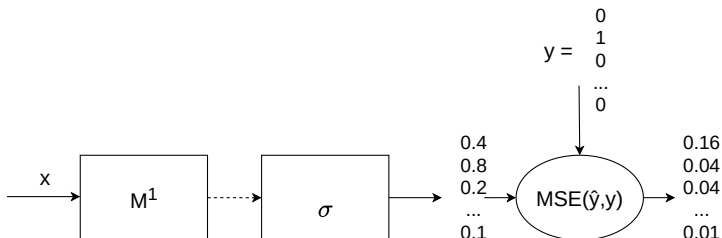
Quand il faut prédire K classes

- K sorties
- Utilisation de vecteurs 1-hot pour la supervision:

$$\mathbf{y} = (0, 0, \dots, 1, \dots, 0)$$

avec $y_i = 0$ pour i différent de la bonne classe,
 $y_k = 1$ pour k l'indice de la bonne classe.

Utilisation de la MSE



Fonction de coût problématique

- Sortie du réseau entre 0 et 1 \Rightarrow utilisation d'une sigmoïde
- Mais :
 - ▶ La similarité au vecteur de sortie n'est pas le plus important, c'est l'argmax qui nous intéresse le plus
 - ▶ Pas plus d'efforts mis sur la maximisation de la sortie de la bonne classe que la minimisation des autres sorties

Coût Cross-entropique

SoftMax : Transformer les sorties en distribution

$$\text{SoftMax}(\mathbf{z})_i = e^{z_i} / \left(\sum_{j=1}^K e^{z_j} \right), \quad \sum_{i=1}^K \text{SoftMax}(\mathbf{z})_i = 1$$

Coût Cross-entropique

- $CE(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^K y_i \log(\hat{y}_i)$
- Dans le cas où \mathbf{y} est un vecteur one-hot de la classe k :
 $CE(\mathbf{y}, \hat{\mathbf{y}}) = - \log(\hat{y}_k)$

- Combinaison SoftMax et Cross-entropie :

$$CE(\mathbf{y}, \text{SoftMax}(\mathbf{z})) = -z_k + \log \left(\sum_{j=1}^K e^{z_j} \right), \quad \frac{\partial CE(\mathbf{y}, \text{SoftMax}(\mathbf{z}))_i}{\partial z_i} = \text{Softmax}(\mathbf{z})_i - 1_{i=k}$$

Cross-entropie binaire

Pour le multi-label en particulier, cross-entropie sur chaque sortie :

$$BCE(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^K y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning**
- 7 Bestiaire (incomplet)

Quelques révolutions du Deep Learning - 1

ImageNet

- Challenge de classification d'images 2009
- Arrivée du Deep en 2012 - écrase la concurrence
- Emergences des modèles les plus connus : AlexNet, Inception, ResNet, ...

Résultats depuis 2010

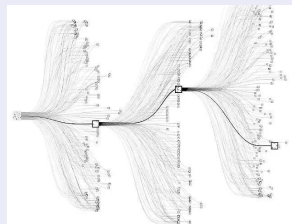


Quelques révolutions du Deep Learning - 2

Deep Q-Learning/Alpha[Go/Zero]

- Capable de jouer à des jeux génériques sans connaître les règles
- 2015 : bat pour la première fois un joueur professionnel de Go
- 2017 : bat le champion du monde de Go

Arbre d'exploration

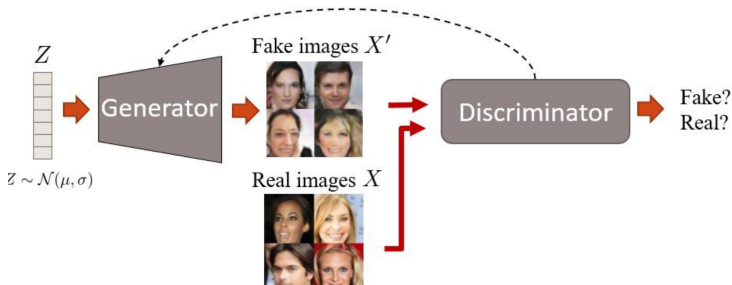
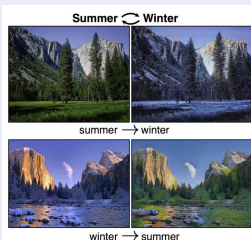


Quelques révolutions du Deep Learning - 3

Generative Adversarial Network (2014)

- Nouveau paradigme d'apprentissage : faussaire contre policier
- Enormément de domaines d'application, en particulier la génération d'image
- Capable d'apprendre des caractéristiques latentes

Transfert de style



Quelques révolutions du Deep Learning - 4

Modèles de Langues

- Origine : word2vec (2013) (apprentissage de représentation), explosion des performances avec le concept d'attention (Transformer, 2017)
- Ont révolutionné tout en TAL : traduction, ChatBot, Recherche d'Information, ...
- Multiples modèles : GPT, BERT, ELMO, Galactica

ChatGPT chatbot



What is brandolini's law?



Brandolini's law, also known as the "Bullshit Asymmetry Principle," is a humorous aphorism that states: "The amount of energy needed to refute bullshit is an order of magnitude bigger than to produce it." In other words, it takes more effort and time to debunk a false or misleading statement than it does to make the statement in the first place.

The law is named after Alberto Brandolini, an Italian software developer who coined the term in a tweet in 2013. Brandolini's law is often used to describe the challenges of debunking misinformation or false claims, particularly in the age of social media, where false and misleading information can spread quickly and easily.

While Brandolini's law is not a formal scientific principle, it is a useful way of illustrating the challenges of combating misinformation and the importance of carefully evaluating information before accepting it as true.

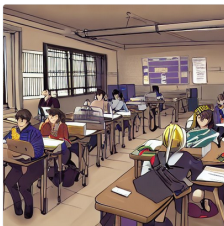
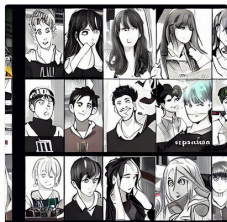


Et quand on mélange toutes ces révolutions ...

On obtient *DALL-E*, *Stable Diffusion* et autres modèles du genre

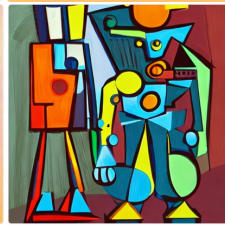
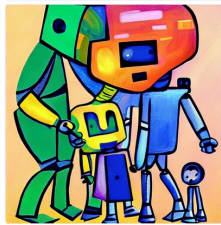
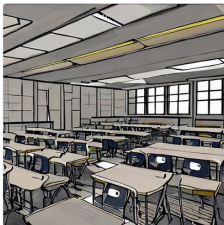
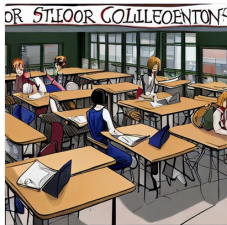
classroom of college students, sorbonne university,| trending in artst

Generate image



a friendly robot with kids, trending in artstation, picasso style

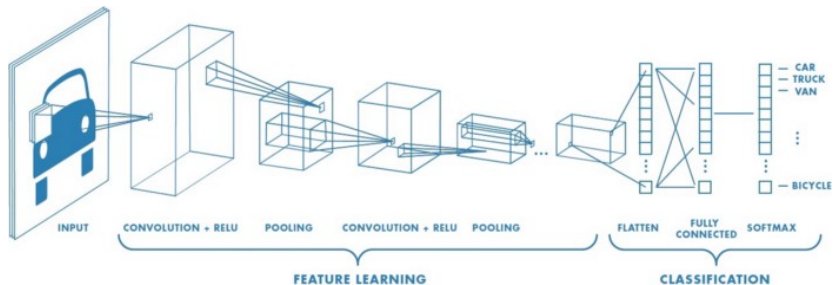
Generate image



Plan

- 1 Réseau à deux couches
- 2 Exemples de réseau MLP (Multi Layer Perceptron)
- 3 Apprentissage du réseau : Vision modulaire
- 4 Apprentissage d'un réseau linéaire multi-couche (Multi Layer Perceptron)
- 5 Apprentissage multi-classe
- 6 La(es) révolution(s) Deep Learning
- 7 Bestiaire (incomplet)**

Réseaux convolutifs



Une convolution

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |



| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

5 x 5 – Image Matrix

3 x 3 – Filter Matrix

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Convolved Feature

Application d'une transformation linéaire sur toutes les régions de l'image

Couche de *Pooling*

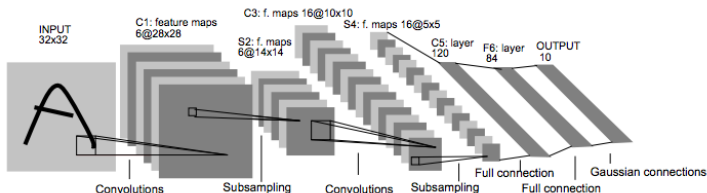


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Pooling (ou subsampling) : Réduire la dimensionnalité de sortie

- Max Pooling : on prend le max sur une fenêtre
- Average Pooling : on fait la moyenne
- Sum Pooling : la somme

...

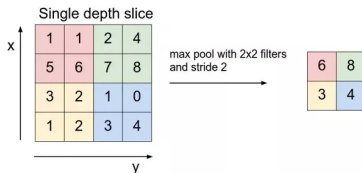
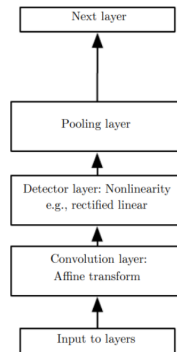
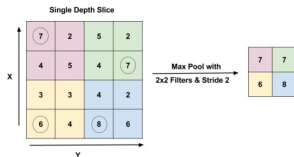
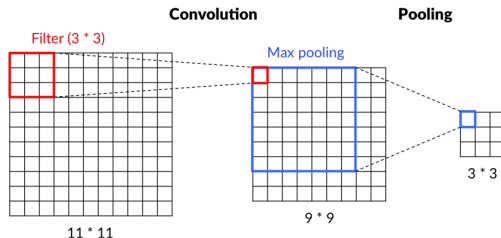


illustration :
<https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>

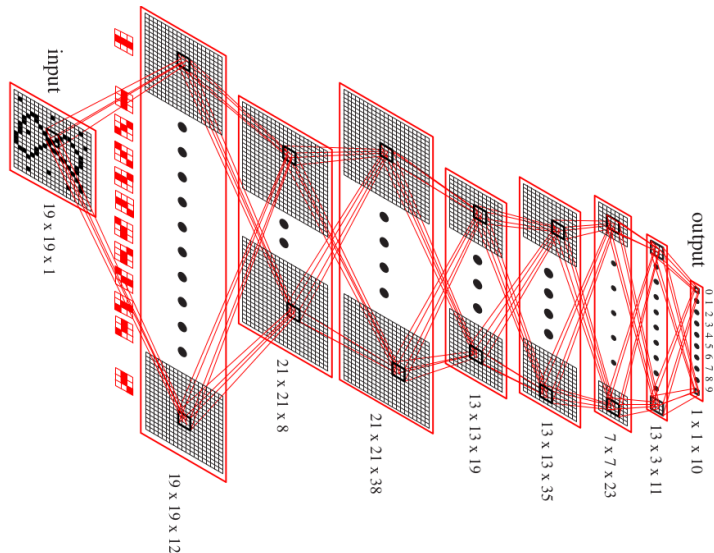
Couche convolutionnelle usuelle



Exemple

Reconnaissance de caractères

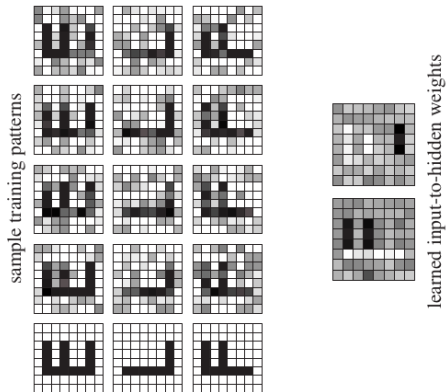
[Duda et al 00]



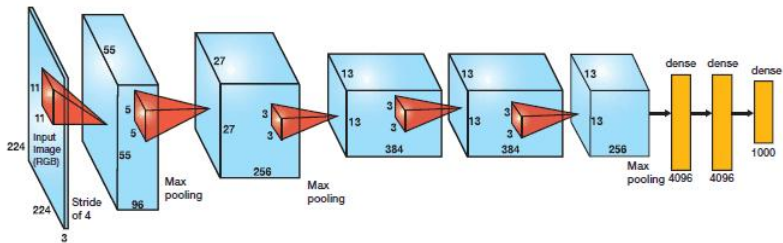
Example

Reconnaissance de caractères (couches internes)

[Duda et al 00]

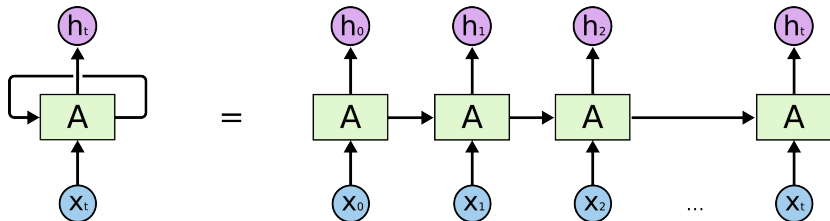


AlexNet (2012)



- 11×11 , 5×5 , 3×3 convolutions
- Max-pooling, ReLU activations
- Dropout et Data-augmentation

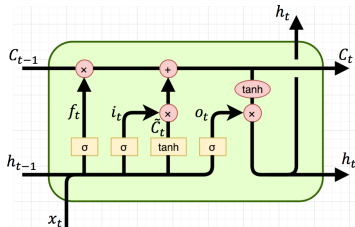
Réseaux récurrents



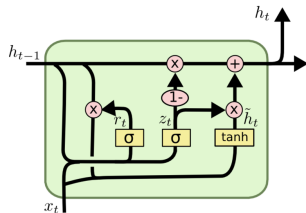
Pour les séquences

- L'objectif est de prédire l'élément suivant d'une séquence : (x^0, x^1, \dots, x^t)
- Hypothèse : $x^{t+1} = f(x^t, h^t)$ l'élément suivant d'une séquence dépend des éléments précédents et d'un état mémoire latent.
- Le réseau prédit pour chaque passage l'état h^t latent.
- Un autre réseau est utilisé pour "décoder" l'état mémoire en la valeur x^t associée.

Long Short Term Memory (LSTM) et Gated Recurrent Unit (GRU)



(a) Long Short-Term Memory

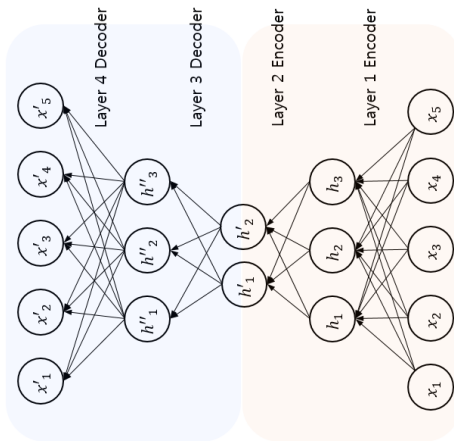


(b) Gated Recurrent Unit

Deux variantes des RNNs

- Un processus de mémoire mis-à-jour automatiquement
- Permet de se "souvenir" (d'avoir une information persistente d'état en état)

Auto-encoders



Apprentissage non-supervisé

Objectif :

- apprendre une “compression” des données utile pour l'apprentissage
- $f^{-1}(f(x)) \approx x$: la sortie doit être proche de l'entrée

Intérêts :

- clustering des données
- lissage/débruitage
- visualisation
- ...

Et beaucoup d'autres

- RBMs (Restricted Boltzmann Machine)
- V.A.E. (Variational Auto-encoder)
- G.A.N. (Generative Adversarial Network)
- Mécanisme d'attention
- Stochastic Unit (Reinforce, Gumbel-Softmax, Straight-Through estimator)

La suite en M2 ! (et en projet ...)

