Introduction aux outils de l'intelligence artificielle
Réseaux de neurones

V. Monbet

[1] Université de Rennes/UFR Mathématiques

Outline

Rappels sur la régression linéaire

Régression linéaire univariée : on explique la variable y
à partir d'une variable x

$$y = \beta_0 + \beta_1 x + \epsilon$$

Régression linéaire multivariée : on explique la variable y
à partir de d variables $x_1, x_2, \cdots, x_d$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d + \epsilon$$

ce qui s'écrit aussi

$$y = W\mathbf{x} + b + \epsilon$$

avec le prédicteur $\mathbf{x} = (x_1, \cdots, x_d)^T$, l'intercept $b = \beta_0$ et le vecteur de poids $W = (\beta_1, \cdots, \beta_d)$

Sachant les observations $(y_i, \mathbf{x}_i)_{i=1,\cdots,n}$, on estime b et W en minimisant la fonction de perte des moindres carrés.

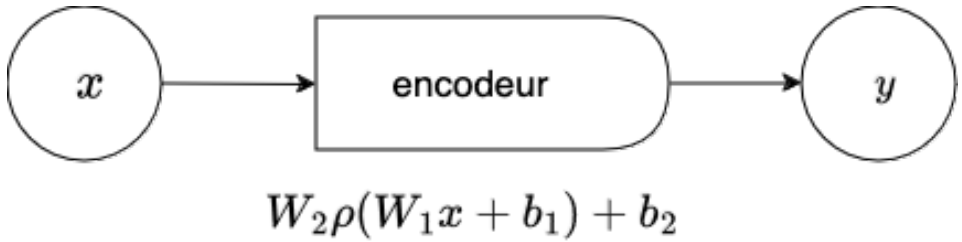
Exemple (multivarié) :

- y : temps de freinage de la voiture
- $\mathbf{x}$: (vitesse de la voiture, poids de la voiture, état des pneus, humidité de la route)

Outline

Réseau de neurones

Un **réseau de neurones "feedforward"** est un algorithme permettant de traiter une entrée $x \in \mathbb{R}^d$ et de renvoyer une sortie $y \in \mathbb{R}$ (ou $\mathbb{R}^k$).



L'algorithme implique l'application répétée de deux opérations simples :

1. une transformation linéaire affine,

$$A(x) = Wx + b$$

2. une fonction d'activation non linéaire ρ
appliquée coordonnée par coordonnée.

Dans le cas le plus simple, les deux opérations sont répétées plusieurs fois, par composition.

Un neurone = perceptron

Un neurone seul est défini comme suit :

$$\Phi(x) = \rho(Wx + b)$$

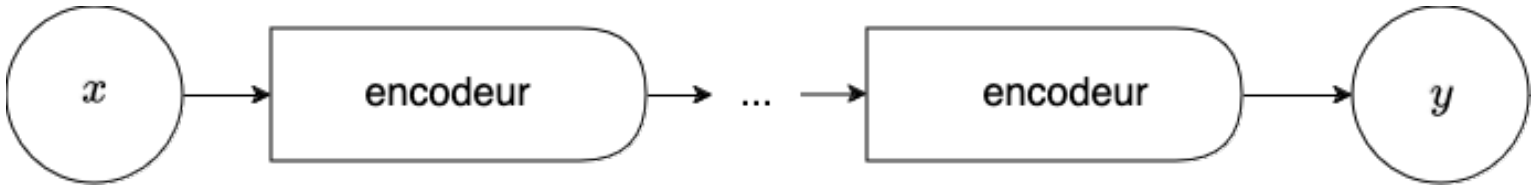
avec ρ la **fonction d'activation**, W les **poids** et b le **biais**.

On l'appelle **perceptron**.

Les paramètres inconnus sont les poids W et le biais b.

Réseau de neurones

Un **réseau de neurones "feedforward"** traite l'information par composition



1. Notons l'**entrée**, $\hat{x}^0 = x$
2. Pour $1 \leq \ell \leq L$,

$$x^\ell = A^\ell\left(\rho(x^{\ell-1})\right) = W^\ell \rho(\hat{x}^{\ell-1}) + b^\ell$$

3. La **sortie** est $y = \Phi(x) = x^L$ (dans le cas de la régression)

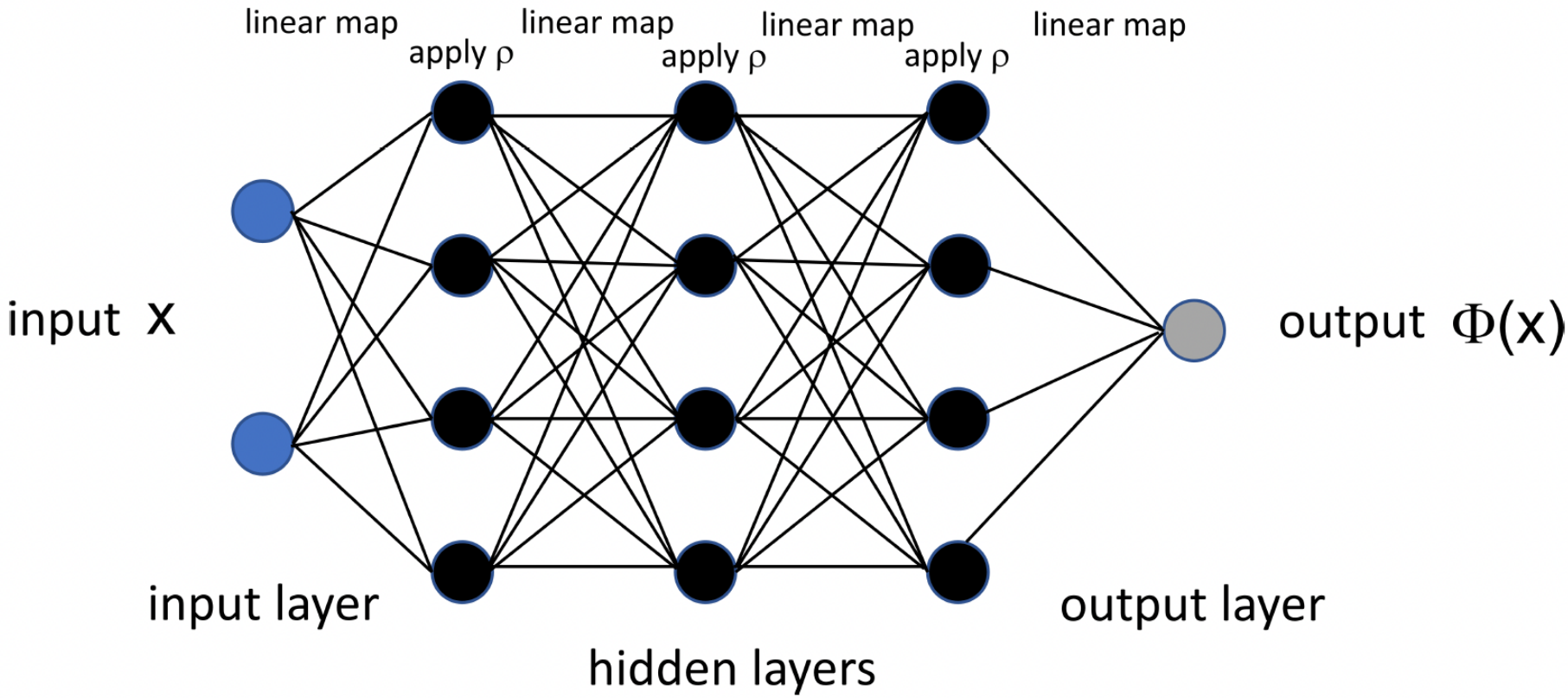
Les valeurs x^ℓ, $1 \leq \ell \leq L-1$, qui ne sont pas directement observées correspondent aux **couches latentes** [1] ou **variables latentes** du réseau de neurones.

Remarque : on verra plus loin que les valeurs des vecteurs de poids W^ℓ et de biais b^ℓ, $1 \leq \ell \leq L$, sont estimées par la minimisation d'une fonction de perte.

1. On les appelle aussi **couches cachées**

Réseau de neurones

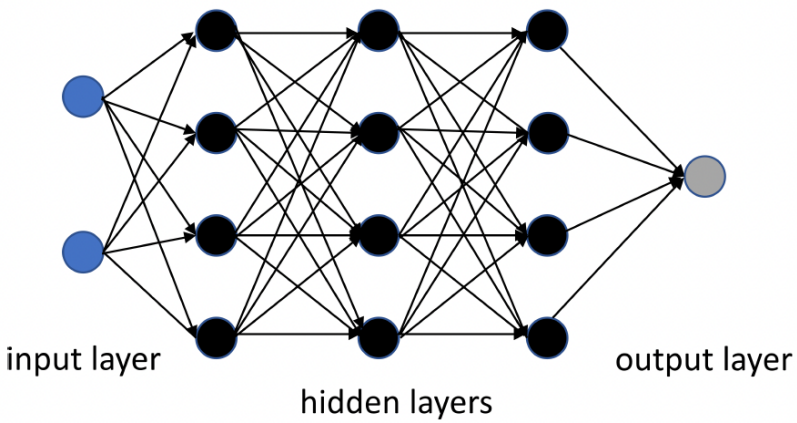
Représentation graphique d'un réseau de neurones avec une entrée $x \in \mathbb{R}^2$, une sortie $\Phi(x) \in \mathbb{R}$ et 4 couches (L=4) dont 3 couches cachées.



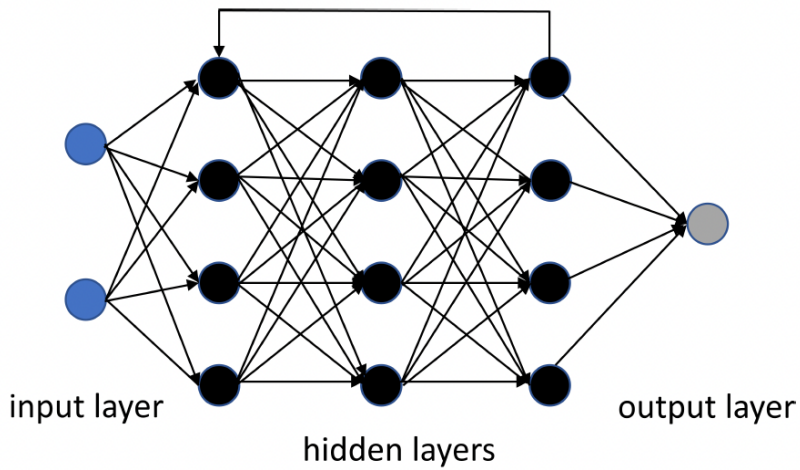
L'information est propagée de l'entrée vers la sortie
$\rightarrow$ **architecture feedforward**

Réseau de neurones

Quand l'information est propagée de l'entrée vers la sortie, le réseau de neurones est appelé **réseau de neurones feed-forward** ou **perceptron multicouches**.
Les réseaux neuronaux qui incluent des connexions vers l'arrière sont appelés **réseaux neuronaux récurrents** (non étudiées dans ce cours).



Feedforward neural network



Recurrent neural network

Outline

Réseau de neurones feedforward : un peu d'histoire

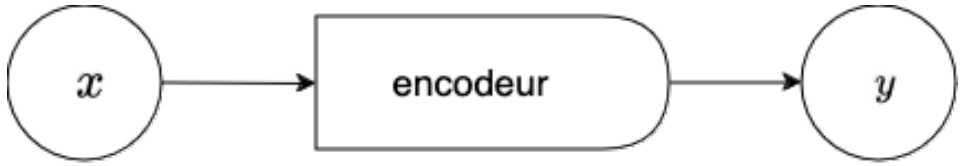
Les réseaux neuronaux feedforward sont également appelés **perceptrons multicouches (MLP)** et familièrement appelés réseaux neuronaux "vanilla".

- En 1958, un perceptron multicouches (multilayer perceptron MLP), composé d'une couche d'entrée, d'une couche cachée avec des poids aléatoires d'une couche de sortie, a été introduit par Frank Rosenblatt dans son livre Perceptron. La fonction d'activation ρ était la fonction heavy-side.

- En 1967, S. Amari a pour la première fois estimé les poids d'un réseau de neurones à l'aide d'un algorithme de descente de gradient stochastique pour un problème de classification.

Réseau de neurones feedforward

Un réseau de neurones feedforward à une couche cachée est un réseau peu profond.



$$W_2\rho(W_1x + b_1) + b_2$$

Dans ce cas, on applique une transformation linéaire à l'entrée $x \in \mathbb{R}^d$, puis la fonction d'activation ρ et de nouveau une transformation linéaire

$$\Phi(x) = W^2\rho\big(W^1x + b^1\big) + b^2$$

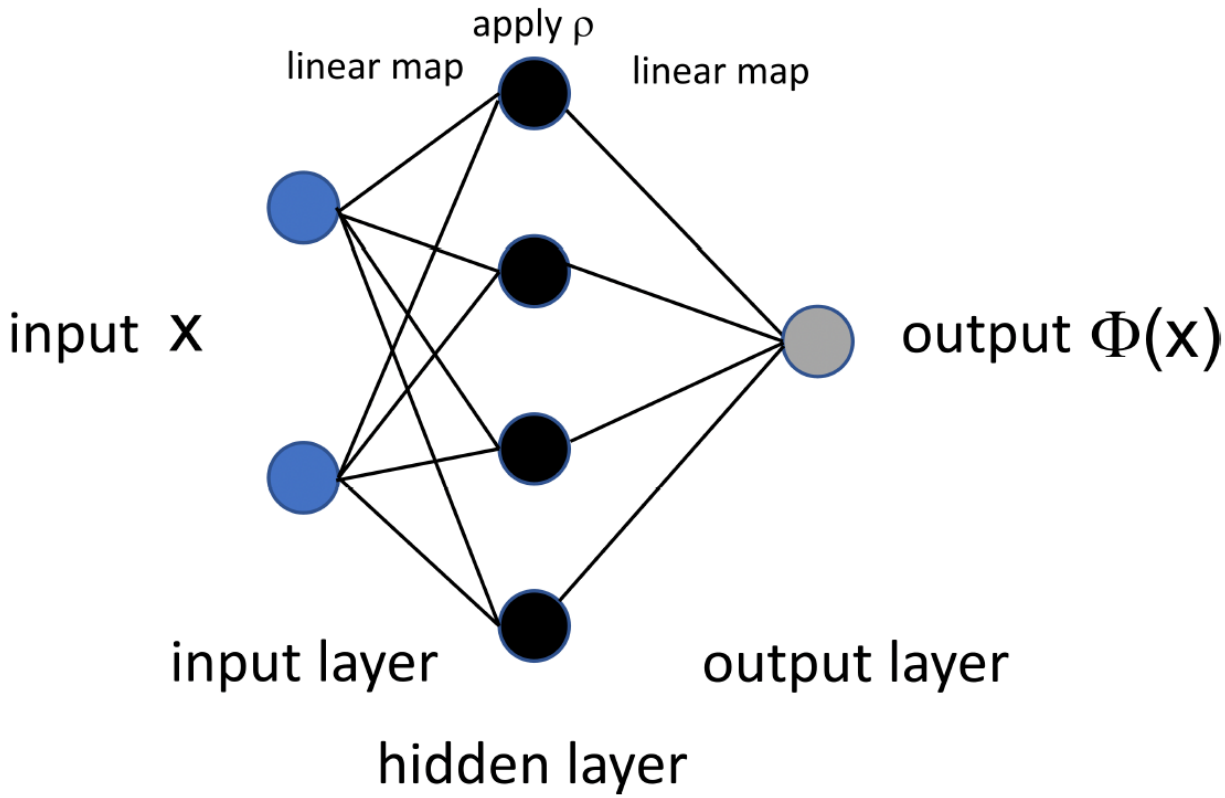
avec $W^1 \in \mathbb{R}^{d,N}$, $b^1 \in \mathbb{R}^N$, $W^2 \in \mathbb{R}^{m,1}$, $b^2 \in \mathbb{R}$ où N est le nombre de neurones cachés.

Si l'entrée est univariée ($d = 1$), on peut écrire

$$\Phi(x) = W^2\rho\Big(\sum_{j=1}^{N} W_j^1x + b_j^1\Big) + b^2$$

Exercice :
1. Représenter un réseau de neurones peu profond à une couche cachée contenant 4 neurones (L=2), avec une entrée $x \in \mathbb{R}^2$ (ie deux variables) et une sortie $\Phi(x) \in \mathbb{R}$. 2. Ecrire l'équation permettant de calculer $\Phi(x)$ et préciser la taille des matrices de poids et des vecteurs de biais.

Représentation graphique d'un réseau de neurones peu profond à une couche cachée contenant 4 neurones (L=2), avec une entrée $x \in \mathbb{R}^2$ (ie deux variables) et une sortie $\Phi(x) \in \mathbb{R}$.

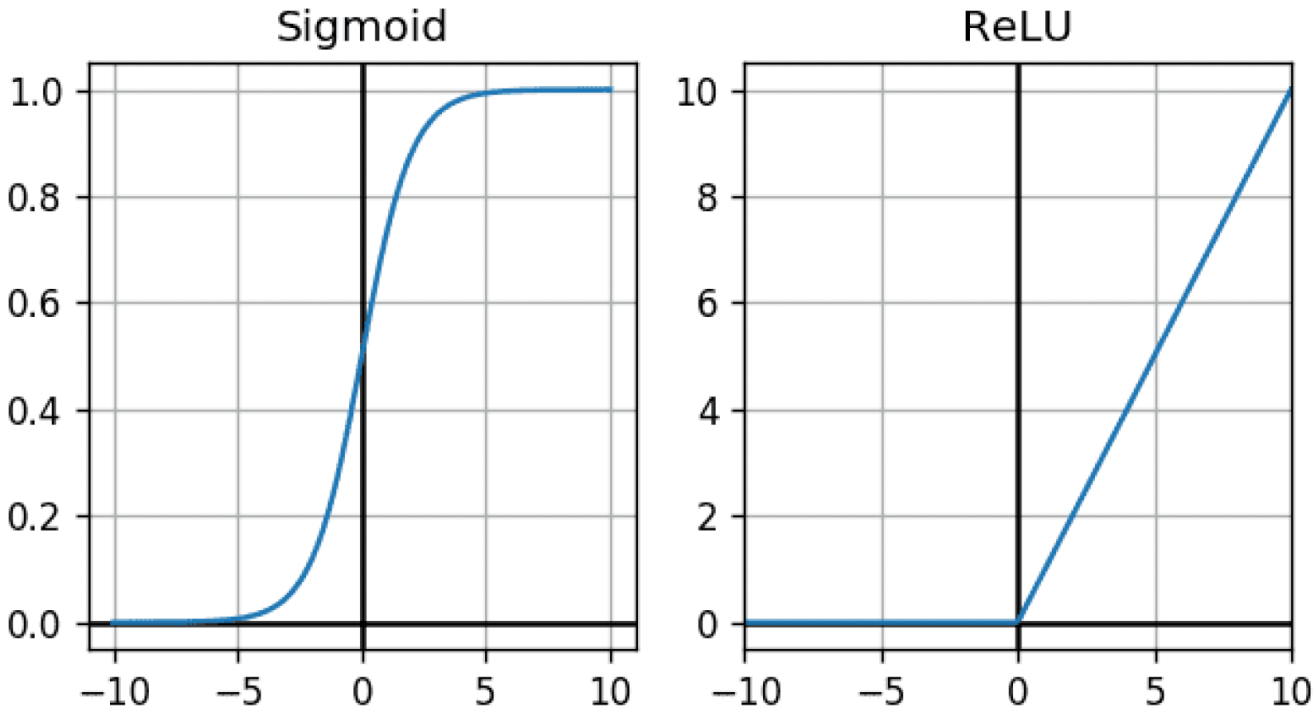


$$\Phi(x) = W^2 \rho(W^1 x + b^1) + b^2$$

Fonction d'activation

Les deux **fonctions d'activations** ρ les plus courantes sont

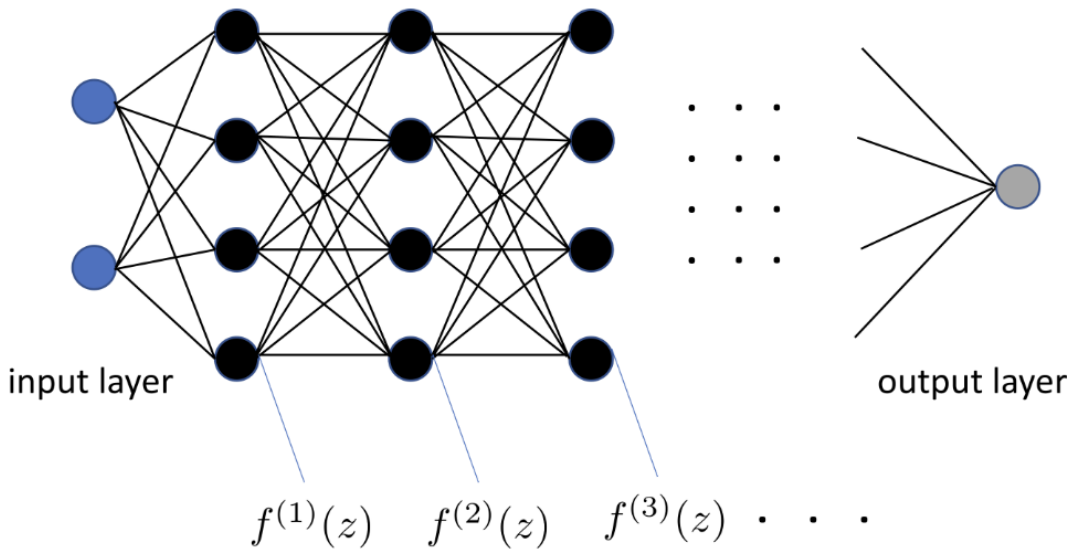
- la fonction **sigmoid** $\rho(x) = \frac{1}{1+e^{-x}}$
- la fonction **Rectified linear unit (ReLU)** : $\rho(x) = max(x, 0)$



La fonction RELU est la plus utilisée, mais la fonction sigmoid a l'avantage d'être différentiable. Une alternative à la fonction sigmoid est la fonction tanh.

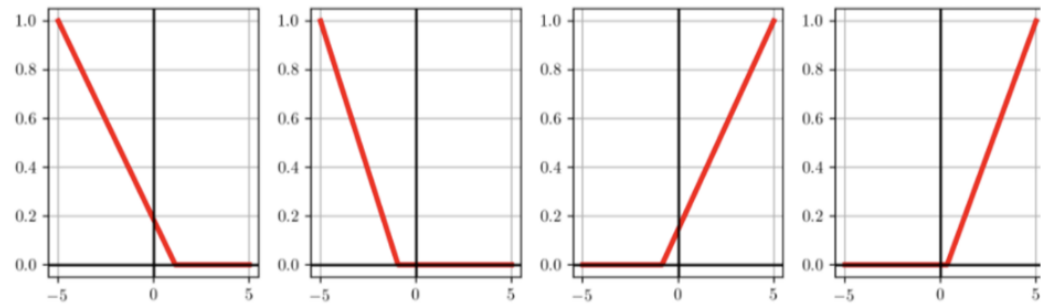
Le pouvoir des réseaux de neurones

En combinant des fonctions simples, un réseau de neurones peut représenter des formes complexes.
On considère un réseau de neurones à, au moin,s 3 couches cachées comme représenté ci-dessous.
On va montrer les sorties des neurones selon deux fonctions d'activation différentes.

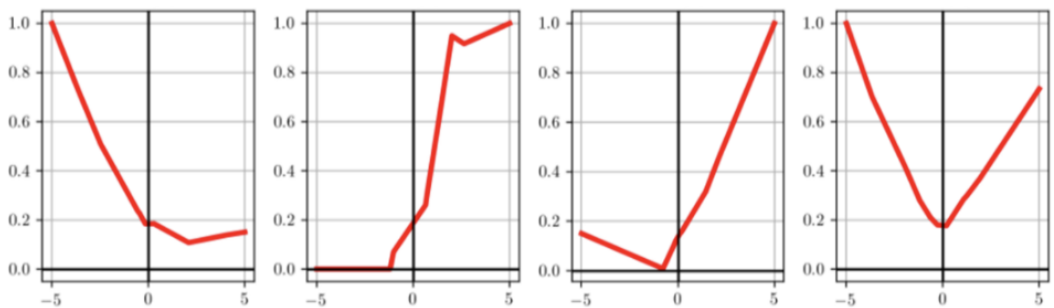


Fonction d'activation

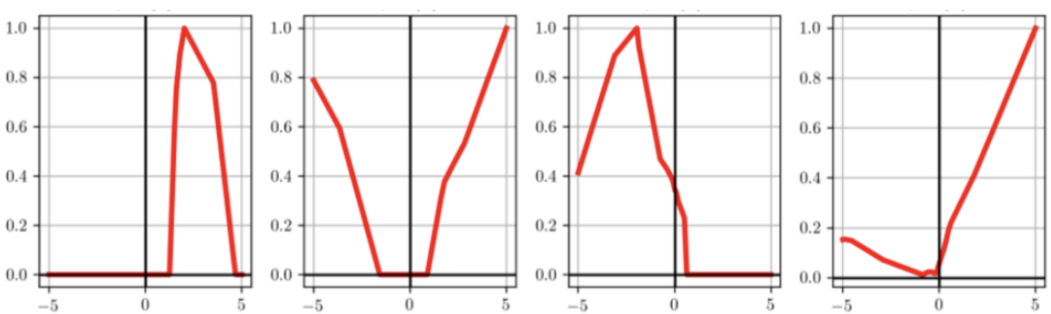
Ici, on montre la sortie des neurones des 3 premières couches, pour des entrées aléatoires, en utilisant la fonction d'activation RELU.



$f^{(1)}(z)$



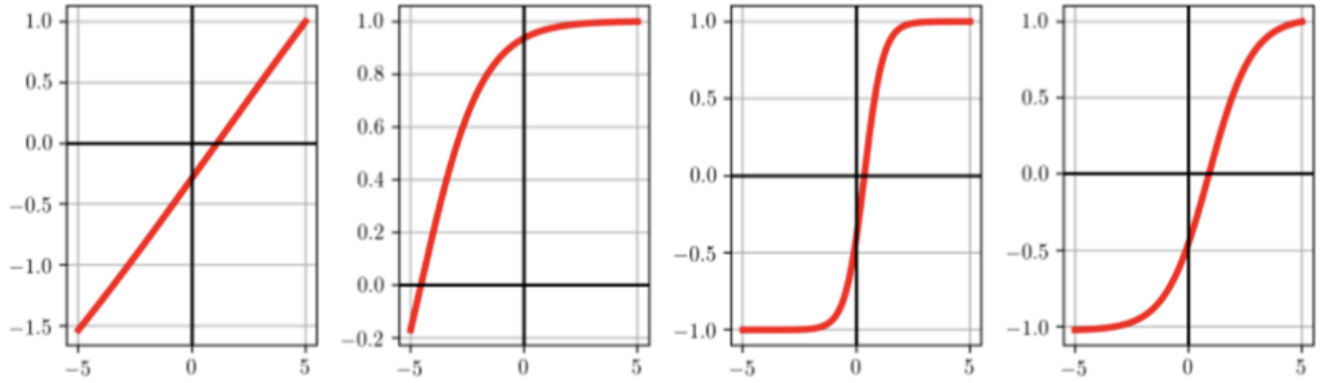
$f^{(2)}(z)$



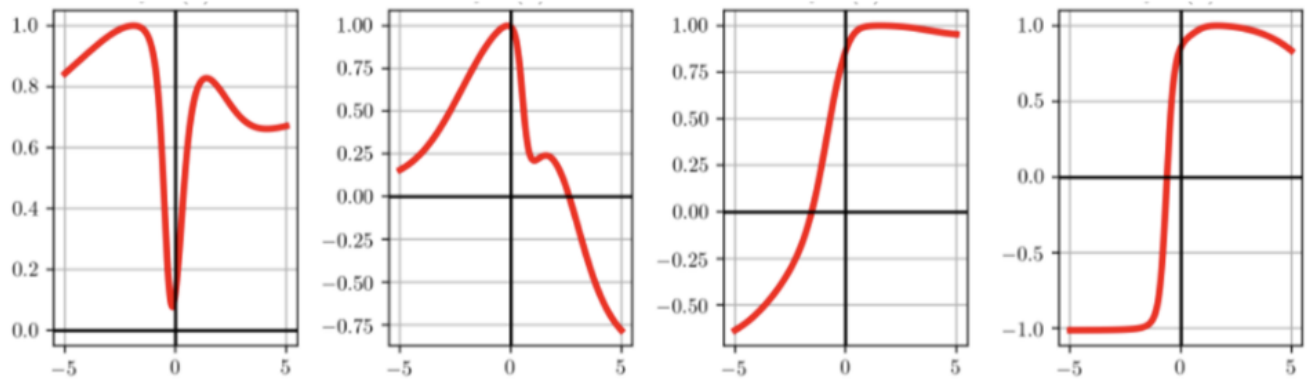
$f^{(3)}(z)$

Fonction d'activation

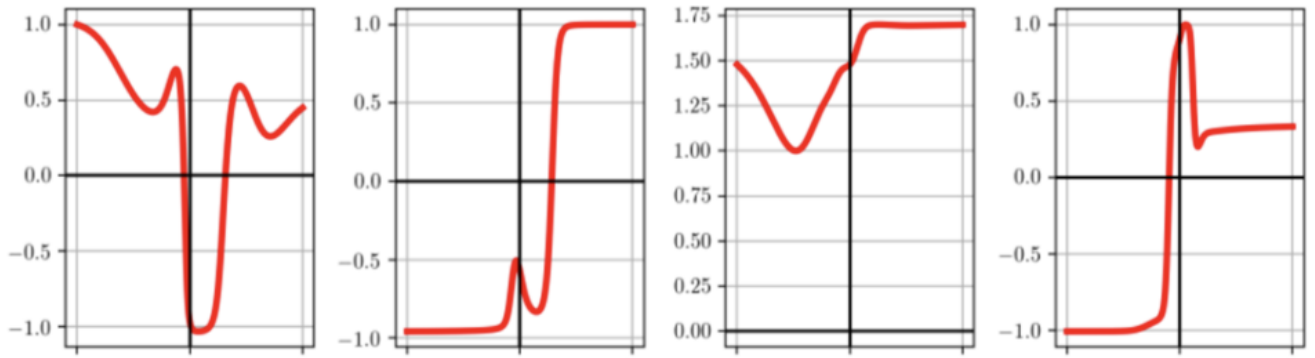
Ici, on montre la sortie des neurones des 2 premières couches, pour des entrées aléatoires, en utilisant la fonction d'activation tanh.



$f^{(1)}(z)$



$f^{(2)}(z)$



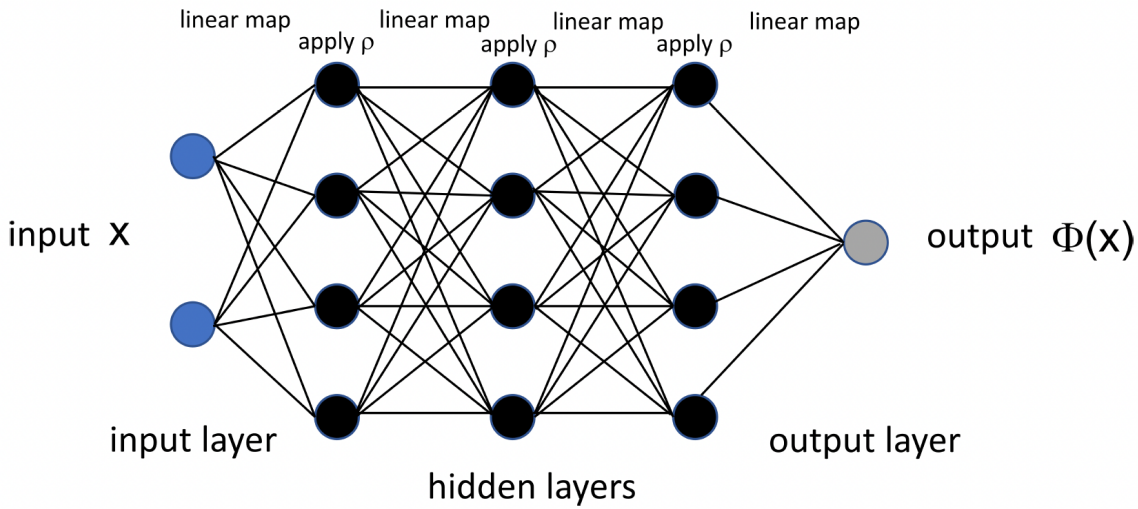
$f^{(3)}(z)$

Réseau de neurones profond (deep neural network)

Un réseau de neurones profond a plusieurs couches cachées (parfois un très grand nombre)

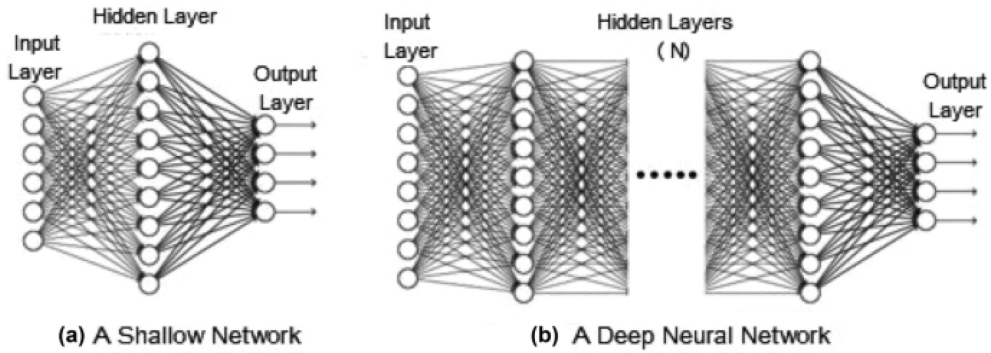
$$\Phi(x) = W^4\rho(W^3\rho(W^2\rho(W^1x + b^1) + b^2) + b^3) + b^4$$

Dans cet exemple, on a un réseau de neurones avec 4 couches dont 3 couches cachées.



Remarque : pour que le réseau Φ soit bien défini, il faut que les dimensions des matrices W^ℓ et des vecteurs b^ℓ soient cohérentes.

Réseau de neurones profond (deep neural network)



- Les architectures modernes sont le plus souvent très profondes.
- Une architecture profonde implique un grand nombre de paramètres (poids).
- Concevoir un réseau profond avec un grand nombre de paramètres peut être utile pour apprendre des modèles complexes pour des tâches non triviales, mais cela peut également conduire à un surajustement [2] important si l'ensemble d'apprentissage est petit ou si la tâche à accomplir est simple.

2. Donner un exemple !

Outline

Approximation universelle

Il existe des résultats théoriques qui montrent que les réseaux de neurones peuvent approcher toute fonction multivariée avec une précision fixée.

Definition

Une classe de fonctions $\mathcal{F}$ est appelée **approximateur universel** sur un compact $\mathcal{S}$ si pour toute fonction continue g et toute précision $\epsilon > 0$, il existe $f \in \mathcal{F}$ telle que

$$\sup_{x \in S} |f(x) - g(x)| \geq \epsilon$$

Approximation universelle

Les théorèmes d'approximation universelle disent que, quelle que soit la fonction que nous essayons d'apprendre, un réseau de neurones suffisamment profond sera capable de représenter cette fonction.

Theorem

(Hornik, 1991) Supposons que ρ est une fonction C^∞, non polynomiale. Alors la classe des réseaux de neurones peu profonds est un approximateur universel sur $[0,1]^d$. d est la dimension d'entrée du réseau de neurones.

Remarque : le réseau peut être arbitrairement large.

Theorem

Kidger and Lyons, 2020 Supposons que ρ est une fonction continue non affine qui est C^1 en au moins un point, avec une dérivée non nulle en ce point. La classe des réseaux de neurones profonds où le nombre de neurones N_ℓ pour chaque couche ℓ peut être borné par $D + d + 2$ est un approximateur universel sur $[0,1]^d$. d est la dimension d'entrée et D est la dimension de sortie du réseau de neurones.

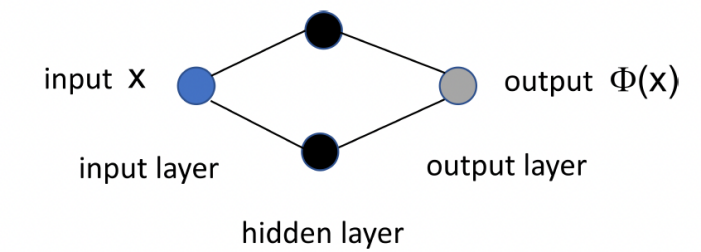
Remarque : le réseau peut être arbitrairement profond.

- Les théorèmes d'approximation universelle indiquent que les réseaux neuronaux ont la capacité d'approcher avec une certaine précision n'importe quelle fonction continue multivariée.
- Le pouvoir de représentation d'un réseau de neurones augmente avec le nombre de neurones et de couches.
- Attention : même si un réseau de neurones est capable de représenter une fonction, l'algorithme d'entraînement peut échouer à apprendre cette fonction spécifique.
- L'apprentissage peut échouer
  (i) parce que l'algorithme d'optimisation utilisé pour l'entraînement peut ne pas être capable de trouver la valeur des paramètres correspondant à la fonction désirée ou
  (ii) parce que l'algorithme d'entraînement pourrait choisir la mauvaise fonction résultant d'un surajustement.

Exemple : la fonction triangle

$$T(x) = \begin{cases} 2x & \text{si} \quad 0 \leq x \leq 1/2 \\ 2(1-x) & \text{si} \quad 1/2 \leq x \leq 1 \end{cases}$$

On peut approcher la fonction T par un réseau de neurones à 1 couche cachée contenant 2 unités cachées et la fonction d'activation RELU.



On a un réseau de neurones à 7 paramètres

$$\Phi(x) = [w_{21} \quad w_{22}]\rho\left(\begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} x + \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix}\right) + b_2$$

avec

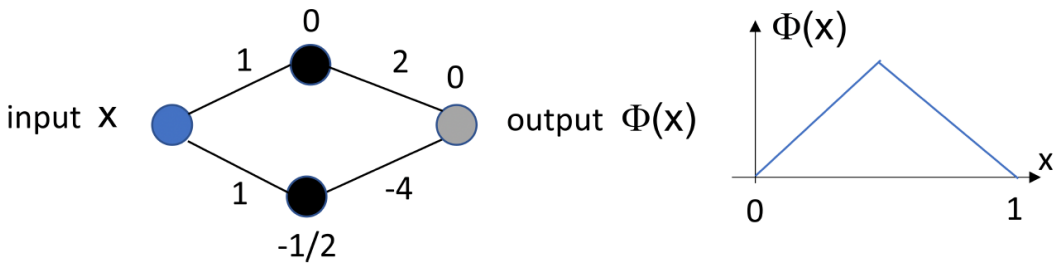
$$\rho(wx + b) = (wx + b)_+$$

On peut réécrire

$$\Phi(x) = w_{21}(w_{11}x + b_{11})_+ + w_{22}(w_{12}x + b_{12})_+ + b_2$$

Par ailleurs, on peut vérifier que

$$T(x) = 2(x - 0)_+ - 4(x - 1/2)_+$$



On peut remarquer que chaque terme de Φ est associé à une portion d'une fonction linéaire par morceaux.

- On peut représenter n'importe quelle fonction triangle avec un réseau de neurones à 2 neurones cachés et des fonctions d'activation RELU.
- Plus généralement, toute fonction linéaire par morceaux peut être représentée par un réseau de neurones et des fonctions d'activation RELU.

Outline

Intelligence artificielle, machine learning

Les réseaux de neurones sont utilisés pour automatiser des tâches très variées

- Régression non linéaire
  ex : prédire le prix d'un logement, prédire une variable clinique, prédire une variable atmosphérique
- Classification
  ex : reconnaitre le contenu d'une image
- Génération d'images
  ex : création de logos
- Génération de texte
  ex : chatGPT, traduction automatique

D'un problème à l'autre, on change l'architecture du réseau de neurones, la fonction d'activation sur la couche de sortie et la fonction de perte.

Dans ce cours, on s'intéresse à la régression.

Régression

Soit $\{(x_1, y_1), \cdots, x_n, y_n)\}$ un ensemble d'apprentissage où $x_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$.

La prédiction du réseau de neurones pour une entrée donnée x_i est le résultat de la passe avant

$$\Phi(x_i; \theta)$$

avec θ le vecteur de tous les poids w et biais b.

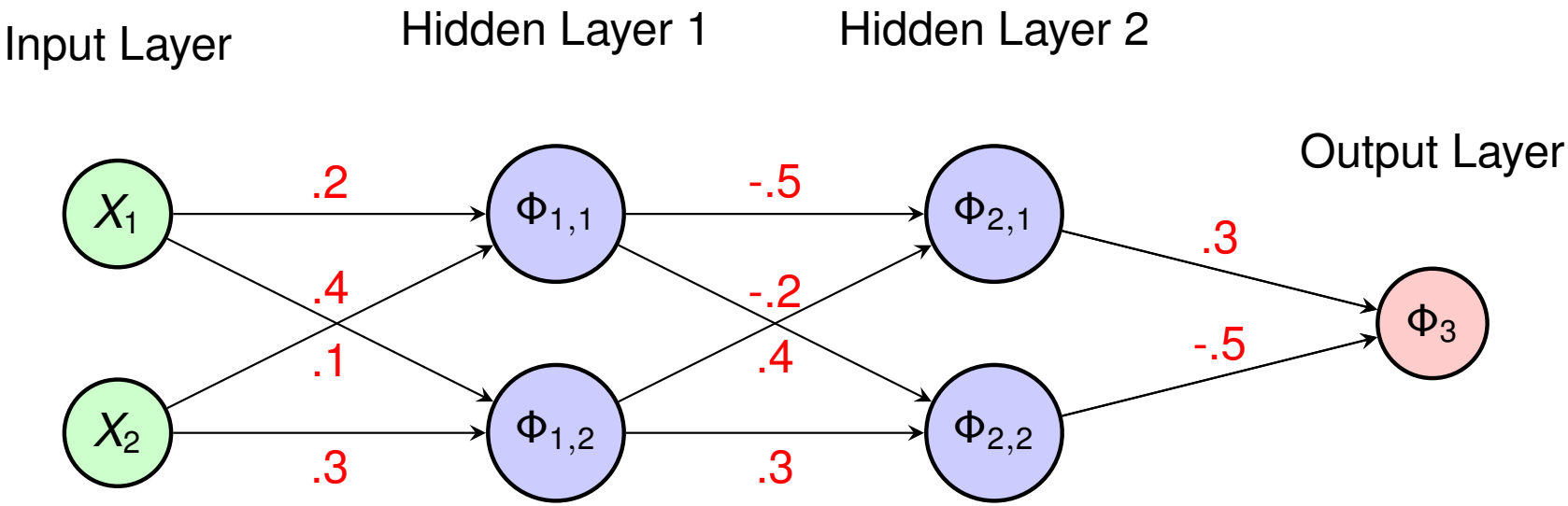
Comme dans le cas de la régression linéaire, on utilise l'erreur aux moindres carrés pour mesurer la qualité de la prédiction

$$L(x, y; \theta) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \Phi(x_i; \theta)\right)^2$$

Pour estimer les paramètres θ, on a besoin de minimiser la fonction de perte et donc de calculer $\Phi(x_i; \theta)$ pour tout i. Or la fonction Φ n'admet pas de forme analytique simple : on l'évalue numériquement par la passe avant.

Exemple de passe avant

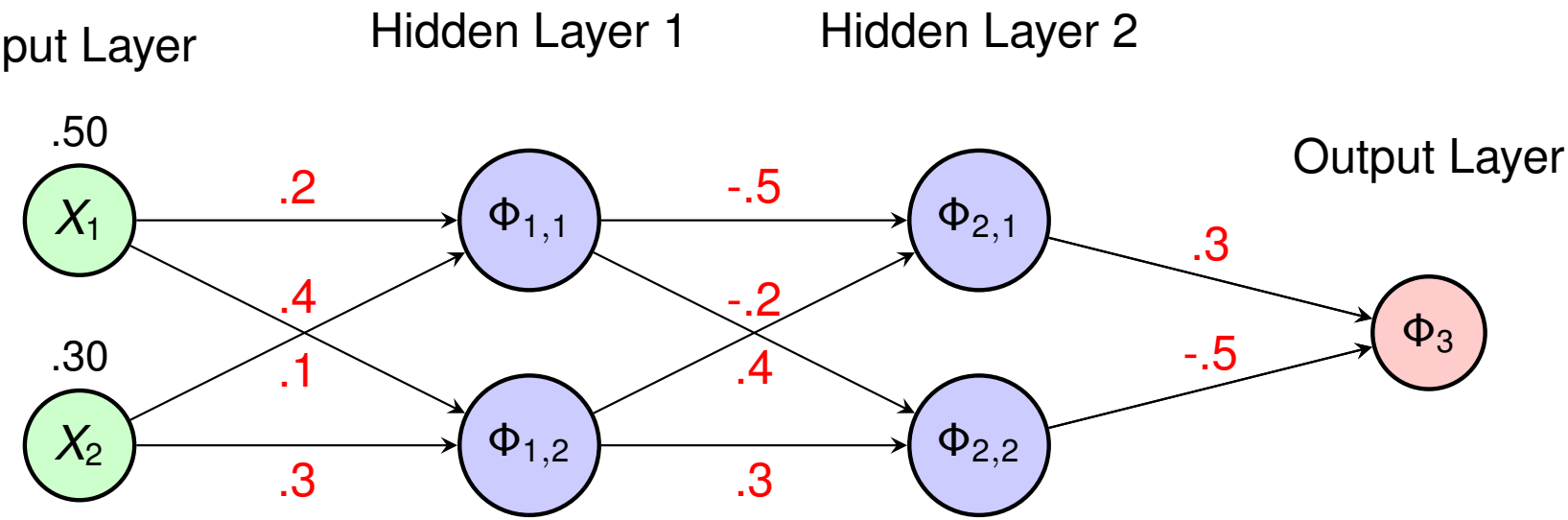
On considère un réseau de neurones à 2 entrées, 1 sortie et 2 couches cachées, contenant chacune 2 unités. On suppose que les biais sont nuls (réseau sans biais). La fonction d'activation de chacune des couches cachées est la fonction RELU.



Exercice : exécuter une passe avant pour calculer la sortie correspondant à l'entrée $X = (.5, .3)$.

Exemple de passe avant

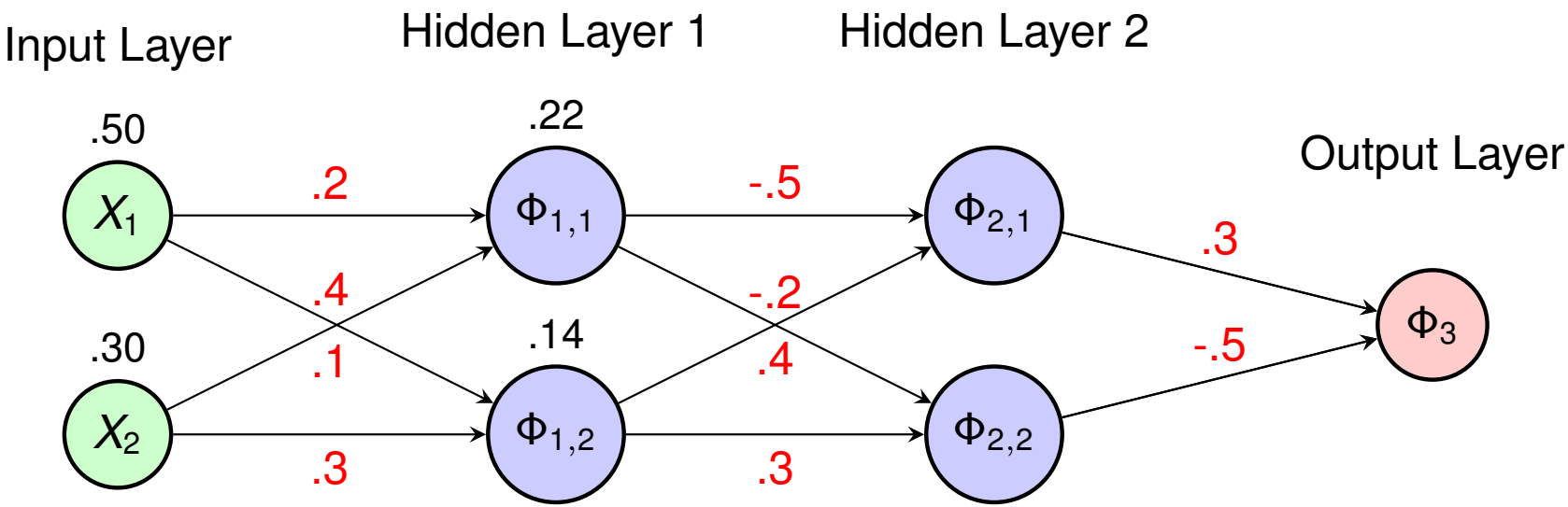
Entrée $X = (.5, .3)$



Exemple de passe avant

Première couche cachée (fonction d'activation RELU)

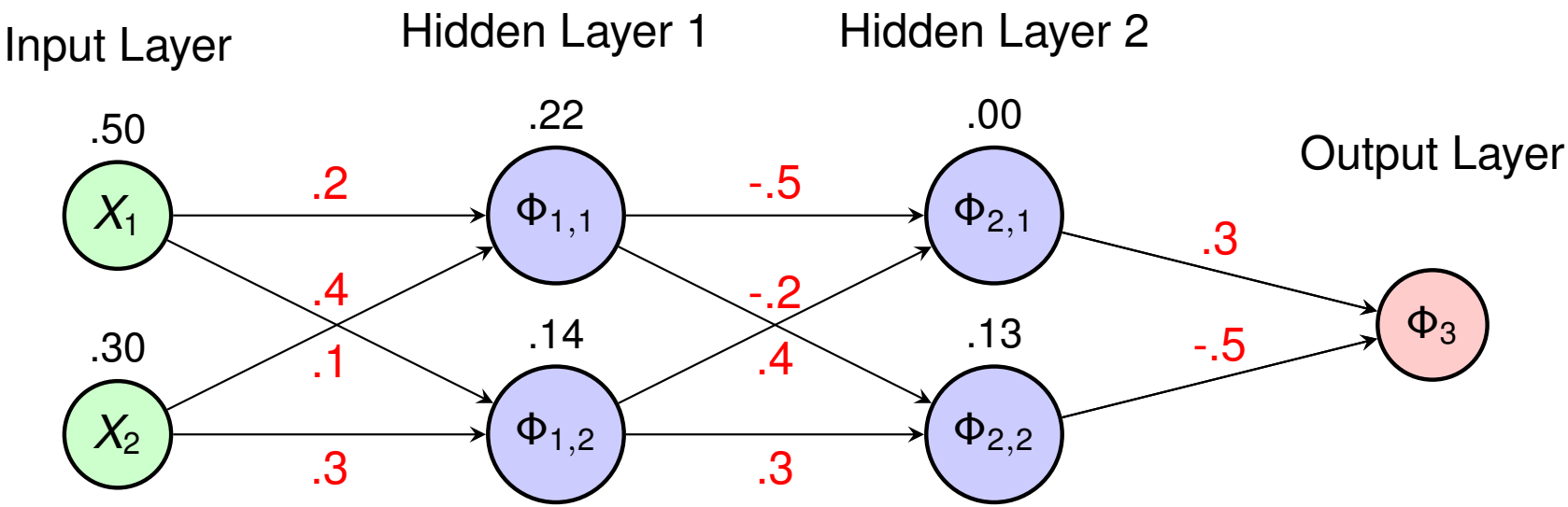
$$\Phi_{11}(X) = (.5 * .2 + .3 * .4)_+, \quad \Phi_{12}(X) = (.5 * .1 + .3 * .3)_+$$



Exemple de passe avant

Deuxième couche cachée (fonction d'activation RELU)

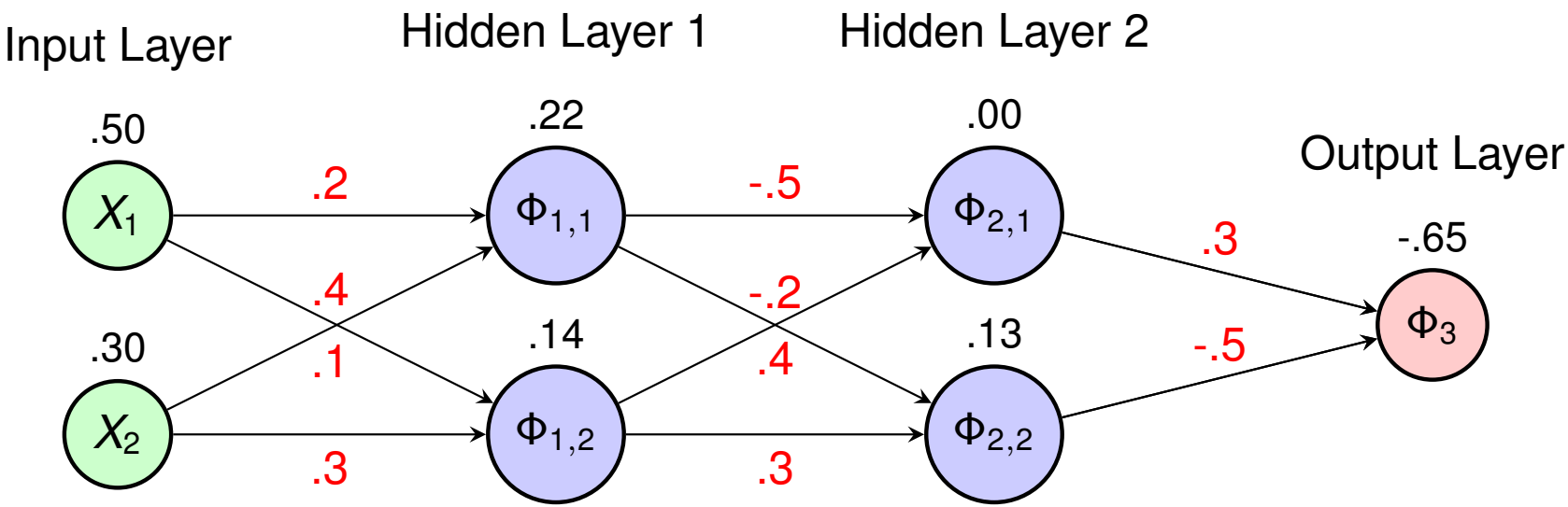
$$\Phi_{21}(X) = (-.22 * .5 - .14 * ..2)_+ = 0, \quad \Phi_{22}(X) = (.22 * .4 + .14 * .3)_+$$



Exemple de passe avant

Couche de sortie (fonction d'activation identité)

$$\Phi_3(X) = (.00 * .3 - .13 * ..5)_+ = -0.65$$



Outline

Minimum de la fonction de perte

Pour estimer les paramètres d'un réseau de neurones MLP sachant un ensemble d'apprentissage $\{(x_i, y_i), i = 1 \cdots, n\}$, il faut trouver le minimum de la fonction de perte.

La fonction de perte des moindres carrés est dérivable mais elle n'est pas convexe : pour un réseau de neurones donné, elle admet plusieurs minima locaux.

Un algorithme de descente du gradient initialisé avec un vecteur de paramètres va converger vers un des minima locaux.

Minimum de la fonction de perte

Le minimum de la fonction de perte d'un réseau de neurone "multi-layer perceptron" (MLP) n'est pas très facile à localiser car ce n'est pas une fonction facile à différencier.

Dans un MLP, la sortie dépend de tous les poids, donc l'erreur obtenue au dernier nœud de sortie dépend également de tous les poids.

Pour calculer la dérivée de la fonction de perte par rapport aux poids, nous devons rétropropager l'erreur jusqu'au nœud d'entrée à partir du nœud de sortie.

La **rétropropagation** est un algorithme qui permet de calculer le gradient d'un MLP. C'est un des éléments clé de la force des réseaux de neurones.

Rétropropagation

Dans un réseau de neurones feedforward, considérons l'erreur associée à une donnée x_i. Il s'agit de l'erreur ente la sortie obtenue $\Phi_L(x_i)$ et le résultat attendu y_i, avec L l'indice de la dernière couche.

Notons $E_L(i)$, l'erreur associée à l'observation x_i au nœud L et

$$e_L(i) = y_i - \Phi_L(x_i)$$

et $E_L(i)$ l'erreur aux moindres carrés à la sortie du noeud L

$$E_L(i) = \frac{1}{2} e_L(i)^2$$

Pour mettre à jour les poids, on utilise une descente de gradient : pour tout poids $w_{\ell m}$

$$w_{\ell m} \leftarrow w_{\ell m} - \eta \frac{\partial E_L(i)}{\partial w_{\ell m}}$$

Calcul du gradient : formule de composition

Un réseau de neurones est une composition de fonctions. On va donc utiliser la formule de composition soit le gradient de fonctions composées.
Soient F, g et h des fonctions différentiables [3]

$$\frac{\partial F(g(t), h(s))}{\partial t} = \frac{\partial F(g(t), h(s))}{\partial g(t)} \frac{\partial g(t)}{\partial t}$$

$$\frac{\partial F(g(t), h(s))}{\partial s} = \frac{\partial F(g(t), h(s))}{\partial h(s)} \frac{\partial h(s)}{\partial s}$$

Exercice
Soit F la fonction composée suivante, avec F et F_1 différentiables
$F(w_1, w_2) = F(F_1(w_1), w_2)$. Exprimer la dérivée de F par rapport à w_1 en fonction de la dérivée partielle de F_1 par rapport à w_1.

3. différentiable est l'équivalent de dérivable pour les fonctions à plusieurs variables

Calcul du gradient : formule de composition

Un réseau de neurones est une composition de fonctions. On va donc utiliser la formule de composition soit le gradient de fonctions composées.
Soient F, g et h des fonctions différentiables [4]

$$\frac{\partial F(g(t), h(s))}{\partial t} = \frac{\partial F(g(t), h(s))}{\partial g(t)} \frac{\partial g(t)}{\partial t}$$

$$\frac{\partial F(g(t), h(s))}{\partial s} = \frac{\partial F(g(t), h(s))}{\partial h(s)} \frac{\partial h(s)}{\partial s}$$

Exercice (corrigé)
Soit F la fonction composée suivante, avec F et F_1 différentiables $F(w_1, w_2) = F(F_1(w_1), w_2)$. Exprimer la dérivée de F par rapport à w_1 en fonction de la dérivée partielle de F_1 par rapport à w_1.

Alors on a

$$\frac{\partial F(F_1(w_1), w_2)}{\partial w_1} = \frac{\partial F(F_1(w_1), w_2)}{\partial F_1(w_1)} \frac{\partial F_1(w_1)}{\partial w_1}$$

4. différentiable est l'équivalent de dérivable pour les fonctions à plusieurs variables

Calcul du gradient

L'erreur à la sortie de la dernière couche s'écrit

$$E_L(i) = \frac{1}{2}\left(y_i - \Phi_L(x_i)\right)^2$$

avec

$$\Phi_L(x_i) = w_L \Phi_{L-1}(x_i)$$

et

$$\Phi_{L-1}(x_i) = \rho(w_{L-1}\Phi_{L-2}(x_i))$$

etc.

Remarque - on a allégé les notations ; en réalité $\Phi_K(x_i) = \Phi_K(w_K, x_i)$ Pour un poids de la couche de sortie (couche L), on a

$$\frac{\partial E_L(i)}{\partial w_L} = \frac{\partial E_L(i)}{\partial \Phi_L(x_i)} \frac{\partial \Phi_L(x_i)}{\partial w_L} = \left(y_i - \Phi_L(x_i)\right)\Phi_{L-1}(x_i)$$

On remarque que toutes les quantités nécessaires pour obtenir ces éléments du gradient ont été calculées lors de la passe avant.

L'erreur à la sortie de la dernière couche s'écrit

$$E_L(i) = \frac{1}{2}\left(y_i - \Phi_L(x_i)\right)^2$$

avec

$$\Phi_L(x_i) = w_L \Phi_{L-1}(x_i)$$

et

$$\Phi_{L-1}(x_i) = \rho(w_{L-1}\Phi_{L-2}(x_i))$$

etc.

Pour un poids de la dernière couche cachée (couche L-1), on peut écrire

$$\begin{aligned}\frac{\partial E_L(i)}{\partial w_{L-1}} &= \frac{\partial E_L(i)}{\partial \phi_L(x_i)} \frac{\partial \phi_L(x_i)}{\partial \phi_{L-1}(x_i)} \frac{\partial \phi_{L-1}(x_i)}{\partial w_{L-1}} \\ &= \left(y_i - \Phi_L(x_i)\right) w_L \Phi_{L-2}(x_i) \rho'(w_{L-1}\Phi_{L-2}(x_i))\end{aligned}$$

Remarques :
- si la fonction d'activation ρ est la fonction RELU alors

$$\rho'(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ 0 & \text{sinon} \end{cases}$$

- toutes les quantités nécessaires pour obtenir ces éléments du gradient ont été calculées lors de la passe avant.

Gradient de la fonction de perte

On a montré, par le calcul, que le gradient de $E_L(x_i)$ en fonction des poids w se calcule par une passe arrière le long du réseau de neurones.

Le gradient de la fonction de perte :

$$\nabla L(w) = \nabla\left(\frac{1}{2}\sum_{i=1}^{n}(y_i - e_L(x_i))^2\right) = \sum_{i=1}^{n}\left(\nabla E_L(x_i)\right)\left(y_i - e_L(x_i)\right)$$

Toutes les quantités nécessaires ont déjà été calculées : le calcul du gradient ne coûte par cher.
C'est une des forces des réseaux de neurones : on utilise naturellement les algorithmes de descente de gradient, à un coût très raisonnable.

Exemple de rétropropagation

Nous allons illustrer la rétropropagation avec un réseau simple comportant une seule couche cachée à 2 unités cacchées. reprendre l'exemple précédent, comme ça on a déjà fait la passe avant

Outline

Outline