

# Réseaux de neurones

Bassem Ben Hamed

Ecole Nationale d'Electronique et des Télécommunications de Sfax

Hammamet, 18-22 février 2025

- 1 Introduction
- 2 Réseaux Feed-Forward (FFNN)
- 3 Entraînement du réseau

# Table des matières

1 Introduction

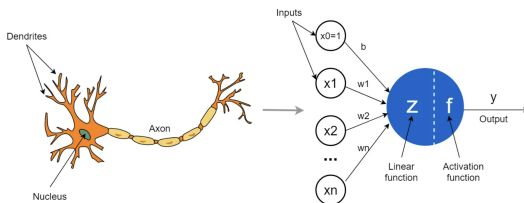
2 Réseaux Feed-Forward (FFNN)

3 Entraînement du réseau

# Réseaux de Neurones Artificiels (ANN)

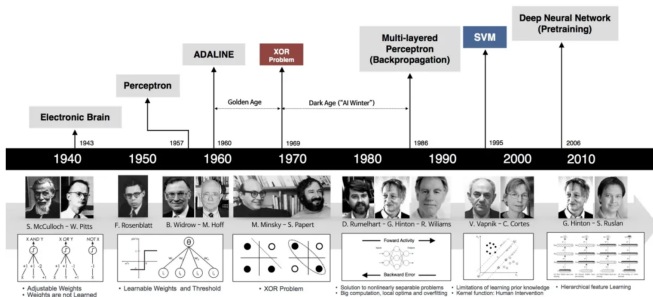
## Definition

- Un réseau de neurones artificiels est un modèle d'apprentissage inspiré du fonctionnement des neurones biologiques.
- Il est composé de neurones organisés en couches : entrée, cachées et sortie.



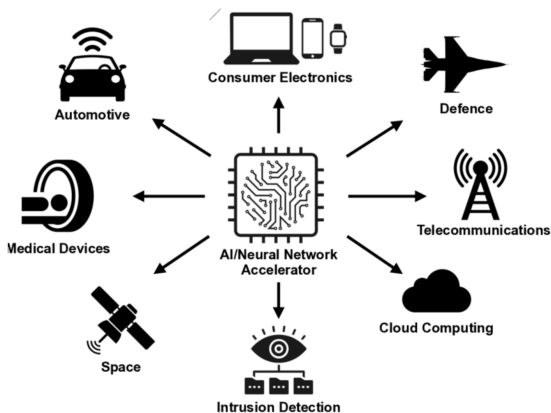
# Historique

- 1943 : McCulloch et Pitts proposent un premier modèle formel de neurone artificiel.
- 1958 : Rosenblatt développe le perceptron, l'un des premiers modèles d'apprentissage automatique.
- Années 1980 : Redécouverte des réseaux multicouches avec l'algorithme de rétropropagation.
- Années 2010 : Avancées majeures grâce au deep learning et aux réseaux convolutifs.



# Applications des Réseaux de Neurones

- Reconnaissance d'images et de la parole.
- Traitement du langage naturel.
- Jeux et intelligence artificielle (AlphaGo, DALL-E).
- Véhicules autonomes et robotique.



# Table des matières

1 Introduction

2 Réseaux Feed-Forward (FFNN)

3 Entraînement du réseau

Les modèles linéaires pour la régression et la classification sont basés sur des combinaisons linéaires de fonctions de base non linéaires  $\phi_j(x)$ , et prennent la forme suivante :

$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right) \quad (1)$$

où  $f()$  est une fonction d'activation non linéaire dans le cas de la classification et est l'identité dans le cas de la régression.

# Extension du Modèle aux Fonctions de Base Paramétriques

- L'objectif est d'étendre ce modèle en faisant en sorte que les fonctions de base  $\phi_j(x)$  dépendent de paramètres, puis de permettre l'ajustement de ces paramètres, ainsi que des coefficients  $w_j$ , lors de l'entraînement.
- Les réseaux de neurones utilisent des fonctions de base qui suivent la même forme que (1), avec des fonctions de base non linéaires dépendant d'une combinaison linéaire des entrées, dont les coefficients sont des paramètres adaptatifs.

Le modèle de réseau de neurones est décrit par une série de transformations fonctionnelles. Nous commençons par construire  $M$  combinaisons linéaires des variables d'entrée  $x_1, \dots, x_D$  sous la forme :

$$a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad (2)$$

où  $j = 1, \dots, M$ . Les paramètres  $w_{ji}^{(1)}$  sont les poids et  $w_{j0}^{(1)}$  sont les biais.

Chaque  $a_j$  est ensuite transformé à l'aide d'une fonction d'activation non linéaire  $h()$  pour obtenir :

$$z_j = h(a_j) \quad (3)$$

Les  $z_j$  correspondent aux sorties des fonctions de base dans (1), appelées unités cachées dans le contexte des réseaux de neurones.

Les unités cachées sont ensuite combinées linéairement pour obtenir les activations des unités de sortie :

$$a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad (4)$$

où  $k = 1, \dots, K$  et  $K$  est le nombre total de sorties. Cette transformation correspond à la deuxième couche du réseau.

# Fonction d'Activation pour les Sorties

Les activations des unités de sortie sont transformées à l'aide d'une fonction d'activation appropriée pour donner les sorties du réseau  $y_k$ . Selon la nature du problème, la fonction d'activation peut être l'identité (pour la régression), une sigmoïde (pour la classification binaire) ou une fonction softmax (pour les problèmes multiclassés).

$$y_k = \sigma(a_k) \quad (5)$$

avec

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \quad (6)$$

La fonction globale du réseau prend la forme suivante pour les fonctions d'activation sigmoïdes des unités de sortie :

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=1}^M w_{kj}^{(2)} h \left( \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \quad (7)$$

Le réseau est donc une fonction non linéaire des variables d'entrée  $\mathbf{x}$  aux variables de sortie  $y_k$ , contrôlée par un vecteur de paramètres  $\mathbf{w}$  ajustables.

# Interprétation Graphique du Réseau de Neurones

Le réseau peut être représenté sous la forme d'un diagramme où chaque nœud représente une transformation fonctionnelle. Le processus d'évaluation de la fonction du réseau correspond à une propagation avant de l'information à travers le réseau.

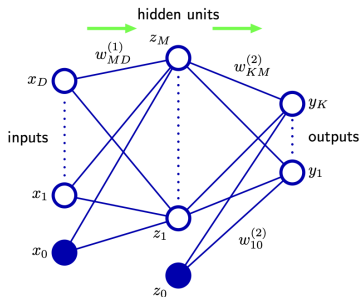


Diagramme du réseau neuronal à deux couches correspondant à (7)

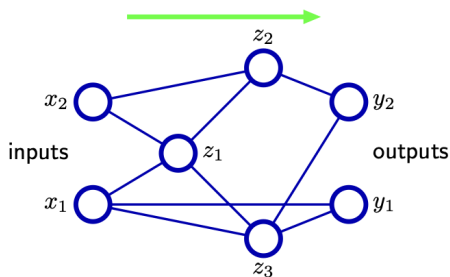
Bien que les diagrammes de réseaux ne représentent pas des modèles graphiques probabilistes, on peut leur donner une interprétation probabiliste dans le cadre des approches bayésiennes ou de la maximum-likelihood pour l'entraînement des paramètres.

# Réseaux de Neurones avec Fonctions d'Activation Linéaires

- Si les fonctions d'activation des unités cachées sont linéaires, un réseau à deux couches peut toujours être réduit à un réseau sans unités cachées, car la composition des transformations linéaires est elle-même une transformation linéaire.
- Cependant, si le nombre d'unités cachées est inférieur au nombre d'unités d'entrée ou de sortie, les transformations générées ne sont pas les plus générales possibles.

# Propriétés d'Approximation des Réseaux Feed-Forward

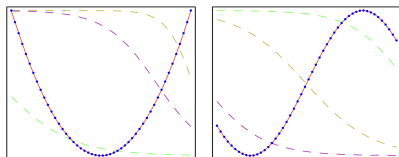
Les propriétés d'approximation des réseaux feed-forward ont été largement étudiées. Les réseaux de neurones sont dits des approximateurs universels. Par exemple, un réseau à deux couches avec des sorties linéaires peut approximer uniformément n'importe quelle fonction continue sur un domaine d'entrée compact avec une précision arbitraire, à condition d'avoir un nombre suffisant d'unités cachées.



Propriétés d'approximation d'un réseau à deux couches

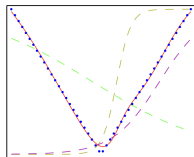
# Réseau de Neurones Multicouches avec Skip-Layer

Le réseau peut être généralisé pour inclure des connexions de type skip-layer, permettant de relier directement les entrées aux sorties, et ainsi d'améliorer l'efficacité du modèle.

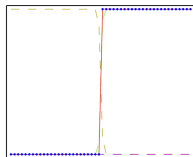


(a)

(b)



(c)



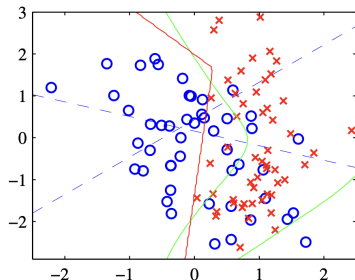
(d)

Illustration de la capacité d'un perceptron multicouche à approximer

4 fonctions différentes : (a)  $f(x) = x^2$ , (b)  $f(x) = \sin(x)$ , (c)  $f(x) = |x|$  et (d)  $f(x) = H(x)$  où  $H$  est la fonction Heaviside.

# Exemple de Réseau Sparse

Un autre exemple de réseau est le réseau sparse, où toutes les connexions possibles entre les unités ne sont pas présentes. Ce type de réseau est souvent utilisé dans les réseaux neuronaux convolutifs.



Les lignes bleues en pointillé montrent les contours  $z = 0.5$  pour chacune des unités cachées, et la ligne rouge montre la surface de décision  $y = 0.5$  pour le réseau. À titre de comparaison, la ligne verte représente la limite de décision optimale calculée à partir des distributions utilisées pour générer les données.

# Conclusion

Les réseaux de neurones sont des approximateurs universels capables de modéliser une large gamme de fonctions. L'entraînement du réseau nécessite des méthodes efficaces, notamment basées sur le maximum de vraisemblance et les approches bayésiennes.

# Table des matières

- 1 Introduction
- 2 Réseaux Feed-Forward (FFNN)
- 3 Entraînement du réseau**

- Un réseau de neurones est une fonction non-linéaire paramétrique reliant  $\mathbf{x}$  (entrée) à  $\mathbf{y}$  (sortie).
- L'entraînement consiste à déterminer les paramètres en minimisant une fonction d'erreur.
- Une approche classique : minimisation de l'erreur quadratique moyenne.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2 \quad (8)$$

- Objectif : minimiser  $E(\mathbf{w})$  pour obtenir  $\mathbf{w}_{ML}$ .
- Entraînement basé sur la maximisation de la vraisemblance.

## Pourquoi une approche probabiliste ?

- Les réseaux de neurones modélisent l'incertitude et permettent d'inférer des distributions de sortie.
- Chaque sortie  $t$  est considérée comme une variable aléatoire conditionnée par  $\mathbf{x}$  et les poids  $\mathbf{w}$ .

## Hypothèse de distribution gaussienne :

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \quad (9)$$

## Pourquoi une distribution gaussienne ?

- Hypothèse naturelle pour des variables continues.
- Facilite les calculs analytiques.

- $\beta$  est la précision (inverse de la variance) du bruit dans la sortie du modèle.
- Contrôle le degré de confiance du modèle dans ses prédictions.
- Relation avec l'erreur quadratique moyenne :

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n\}^2 \quad (10)$$

- Une faible valeur de  $\beta$  signifie une grande incertitude dans les prédictions.

$$E(\mathbf{w}) = \frac{\beta}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \quad (11)$$

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n\}^2 \quad (12)$$

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)} \quad (13)$$

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t (1 - y(\mathbf{x}, \mathbf{w}))^{1-t} \quad (14)$$

- Utilisation de la fonction sigmoïde.
- Modélisation par une distribution de Bernoulli.

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad (15)$$

- Adaptée à la classification binaire.
- Meilleure convergence et généralisation que l'erreur quadratique.

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad (16)$$

$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \quad (17)$$

- Utilisation de la fonction softmax.
- Chaque observation appartient à une classe unique parmi  $K$ .

- Régression : sorties linéaires, erreur quadratique.
- Classification binaire : sigmoïde, entropie croisée.
- Classification multiclasse : softmax, entropie croisée.
- Correspondance naturelle entre activation et fonction d'erreur.

- Objectif : Trouver un vecteur de poids  $\mathbf{w}$  qui minimise la fonction d'erreur  $E(\mathbf{w})$
- Pour un petit pas  $\delta\mathbf{w}$  dans l'espace des poids :

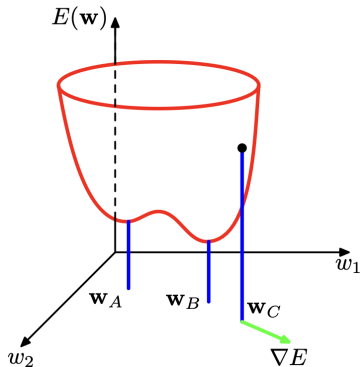
$$\delta E \simeq \delta\mathbf{w}^\top \nabla E(\mathbf{w})$$

- Au minimum, le gradient s'annule :

$$\nabla E(\mathbf{w}) = \mathbf{0}$$

- Types de points stationnaires :
  - Minimum global : valeur minimale absolue de  $E(\mathbf{w})$
  - Minima locaux : valeurs minimales relatives de  $E(\mathbf{w})$
  - Points selles et maxima

# Vue géométrique de la fonction d'erreur $E(\mathbf{w})$



Vue géométrique de la fonction d'erreur  $E(\mathbf{w})$  représentée comme une surface au-dessus de l'espace des poids. Le point  $\mathbf{w}_A$  est un minimum local et  $\mathbf{w}_B$  est le minimum global. En tout point  $\mathbf{w}_C$ , le gradient local de la surface d'erreur est donné par le vecteur  $\nabla E$

- Solution analytique impossible  $\rightarrow$  approche numérique itérative
- Procédure générale d'actualisation des poids :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta \mathbf{w}^{(\tau)}$$

où  $\tau$  représente l'étape d'itération

- Caractéristiques importantes :
  - Multiples minima locaux équivalents : dans un réseau à  $M$  unités cachées, existence de  $M!2^M$  points équivalents
  - Le minimum global n'est pas toujours nécessaire
  - Comparaison de plusieurs minima locaux souvent suffisante

# Approximation Quadratique Locale - Développement de Taylor

- Développement de Taylor de la fonction d'erreur  $E(\mathbf{w})$  autour d'un point  $\hat{\mathbf{w}}$  :

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

où :

- $\mathbf{b}$  est le gradient de  $E$  évalué en  $\hat{\mathbf{w}}$  :

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}}$$

- $\mathbf{H}$  est la matrice Hessienne avec éléments :

$$(\mathbf{H})_{ij} \equiv \left. \frac{\partial^2 E}{\partial w_i \partial w_j} \right|_{\mathbf{w}=\hat{\mathbf{w}}}$$

# Approximation Quadratique au Minimum Local

Au point minimum  $\mathbf{w}^*$  :

- Le gradient s'annule :  $\nabla E = 0$
- L'approximation devient :

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- Équation aux valeurs propres de la Hessienne :

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

où les vecteurs propres forment une base orthonormée :

$$\mathbf{u}_i^\top \mathbf{u}_j = \delta_{ij}$$

# Géométrie de l'Approximation Quadratique

- Décomposition sur la base des vecteurs propres :

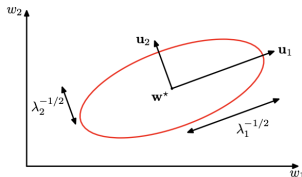
$$\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i$$

- Expression de la fonction d'erreur :

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2$$

- Condition de minimum local : matrice Hessienne définie positive

$$\mathbf{v}^\top \mathbf{H} \mathbf{v} > 0 \quad \text{pour tout } \mathbf{v}$$



# Utilisation de l'Information du Gradient

- Complexité de l'approximation quadratique :
  - Matrice Hessienne  $\mathbf{H}$  (symétrique) et vecteur  $\mathbf{b}$  contiennent  $\frac{W(W+3)}{2}$  éléments indépendants
  - $W$  = dimensionnalité du vecteur de poids  $\mathbf{w}$
- Sans utilisation du gradient :
  - Nécessite  $O(W^2)$  évaluations de fonction
  - Chaque évaluation requiert  $O(W)$  étapes
  - Complexité totale :  $O(W^3)$  opérations
- Avec utilisation du gradient :
  - Chaque évaluation de  $\nabla E$  apporte  $W$  informations
  - Rétropropagation requiert seulement  $O(W)$  étapes
  - Complexité totale réduite à  $O(W^2)$  opérations

**Conclusion :** L'utilisation du gradient via la rétropropagation constitue la base des algorithmes pratiques d'entraînement des réseaux de neurones.

- Mise à jour des poids par descente de gradient :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

où  $\eta > 0$  est le taux d'apprentissage

- Méthodes par lots (batch) :
  - Utilise l'ensemble du jeu d'entraînement à chaque itération
  - Évalue  $\nabla E$  sur toutes les données
  - Plus lent mais plus stable
- Alternatives plus efficaces :
  - Gradients conjugués
  - Méthodes quasi-Newton
  - Garantie de décroissance de l'erreur à chaque itération

# Descente de Gradient Stochastique

- Fonction d'erreur décomposable :

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

- Mise à jour stochastique :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

- Avantages de la méthode en ligne :
  - Gestion efficace de la redondance dans les données
  - Possibilité d'échapper aux minima locaux
  - Adaptation rapide aux nouvelles données
- Variantes :
  - Séquentielle : parcours ordonné des données
  - Stochastique : sélection aléatoire avec remise
  - Mini-batch : compromis entre batch et stochastique

- L'objectif est de trouver une technique efficace pour évaluer le gradient d'une fonction d'erreur  $E(\mathbf{w})$  pour un réseau de neurones à propagation avant.
- Cela peut être réalisé en utilisant un schéma de passage de messages locaux connu sous le nom de *rétropropagation de l'erreur*, ou simplement *rétropropagation*.

# Deux Étapes Distinctes de l'Entraînement

La plupart des algorithmes d'entraînement impliquent une procédure itérative pour minimiser une fonction d'erreur, avec des ajustements de poids dans une séquence d'étapes.

À chaque étape, nous distinguons deux phases :

- 1 Évaluation des dérivées de la fonction d'erreur par rapport aux poids
- 2 Utilisation de ces dérivées pour calculer les ajustements des poids

# Évaluation des Dérivées de la Fonction d'Erreur

De nombreuses fonctions d'erreur d'intérêt pratique comprennent une somme de termes

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w}) \quad (18)$$

On considère l'évaluation de  $\nabla E_n(\mathbf{w})$  pour un terme de la fonction d'erreur, qui peut être

- Utilisé directement pour l'optimisation séquentielle
- Accumulé sur l'ensemble d'entraînement pour les méthodes par lots

# Modèle Linéaire Simple

Considérons d'abord un modèle linéaire simple

$$y_k = \sum_i w_{ki} x_i \quad (19)$$

Avec une fonction d'erreur pour un motif d'entrée particulier  $n$

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \quad (20)$$

Le gradient par rapport à un poids  $w_{ji}$  est

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \quad (21)$$

Dans un réseau à propagation avant général, chaque unité calcule :

$$a_j = \sum_i w_{ji} z_i \quad (22)$$

$$z_j = h(a_j) \quad (23)$$

où  $z_i$  est l'activation d'une unité qui envoie une connexion à l'unité  $j$ , et  $w_{ji}$  est le poids associé.

Pour chaque motif, on a

- On fournit le vecteur d'entrée au réseau
- On calcule les activations de toutes les unités cachées et de sortie par application successive des équations

Ce processus est appelé *propagation avant* car il représente un flux d'information vers l'avant.

# Évaluation des Dérivées

Pour évaluer  $\frac{\partial E_n}{\partial w_{ji}}$ , on applique la règle de la chaîne :

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \quad (24)$$

On introduit une notation utile

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \quad (25)$$

De la définition de  $a_j$ , on a

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \quad (26)$$

Ainsi

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \quad (27)$$

Pour les unités de sortie, en utilisant la fonction de lien canonique

$$\delta_k = y_k - t_k \quad (28)$$

Pour les unités cachées, en utilisant la règle de la chaîne

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad (29)$$

# Formule de Rétropropagation

Cela conduit à la formule de rétropropagation

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (30)$$

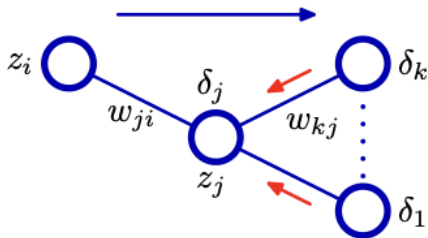


Illustration du calcul de  $\delta_j$  pour l'unité cachée  $j$  par rétropropagation des  $\delta$  depuis les unités  $k$  auxquelles l'unité  $j$  envoie des connexions.

---

**Algorithm 1** Rétropropagation de l'Erreur

---

- 1: Appliquer un vecteur d'entrée  $\mathbf{x}_n$  au réseau et propager vers l'avant en utilisant  $a_j = \sum_i w_{ji}z_i$  et  $z_j = h(a_j)$  pour trouver les activations de toutes les unités cachées et de sortie.
  - 2: Évaluer les  $\delta_k$  pour toutes les unités de sortie en utilisant  $\delta_k = y_k - t_k$ .
  - 3: Rétropropager les  $\delta$  en utilisant  $\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$  pour obtenir  $\delta_j$  pour chaque unité cachée dans le réseau.
  - 4: Utiliser  $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$  pour évaluer les dérivées requises.
-

Pour les méthodes par lots, la dérivée de l'erreur totale  $E$  est obtenue par :

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \quad (31)$$

La dérivation suppose que chaque unité a la même fonction d'activation  $h(\cdot)$ , mais peut être généralisée pour permettre à différentes unités d'avoir des fonctions d'activation individuelles.

# Calcul de la Rétropropagation (1)

Architecture du réseau :

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=1}^M w_{kj}^{(2)} h \left( \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Notations :

- $h$  : fonction d'activation de la couche cachée
- $\sigma$  : fonction d'activation sigmoïde de sortie
- $w_{ji}^{(1)}$  : poids couche d'entrée vers couche cachée
- $w_{kj}^{(2)}$  : poids couche cachée vers sortie

# Calcul de la Rétropropagation (2)

Définissons les activations :

$$a_j^{(1)} = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j^{(1)})$$

$$a_k^{(2)} = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

$$y_k = \sigma(a_k^{(2)})$$

Erreur pour un exemple :

$$E_n = - \sum_k t_k \ln y_k + (1 - t_k) \ln(1 - y_k)$$

# Calcul de la Rétropropagation (3)

Dérivées par rapport aux poids de sortie :

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} z_j$$
$$\delta_k^{(2)} = (y_k - t_k) \sigma'(a_k^{(2)})$$

Dérivées par rapport aux poids de la couche cachée :

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i$$
$$\delta_j^{(1)} = h'(a_j^{(1)}) \sum_k w_{kj}^{(2)} \delta_k^{(2)}$$

# Algorithme de Mise à Jour

Mise à jour des poids :

$$w_{kj}^{(2)}(t+1) = w_{kj}^{(2)}(t) - \eta \delta_k^{(2)} z_j$$

$$w_{ji}^{(1)}(t+1) = w_{ji}^{(1)}(t) - \eta \delta_j^{(1)} x_i$$

Algorithme complet :

- 1 Propagation avant : calculer  $z_j$  puis  $y_k$
- 2 Calculer  $\delta_k^{(2)}$  pour la couche de sortie
- 3 Rétropropager vers la couche cachée : calculer  $\delta_j^{(1)}$
- 4 Mettre à jour tous les poids