

Neural Networks

Réseaux de Neurones

April 2023

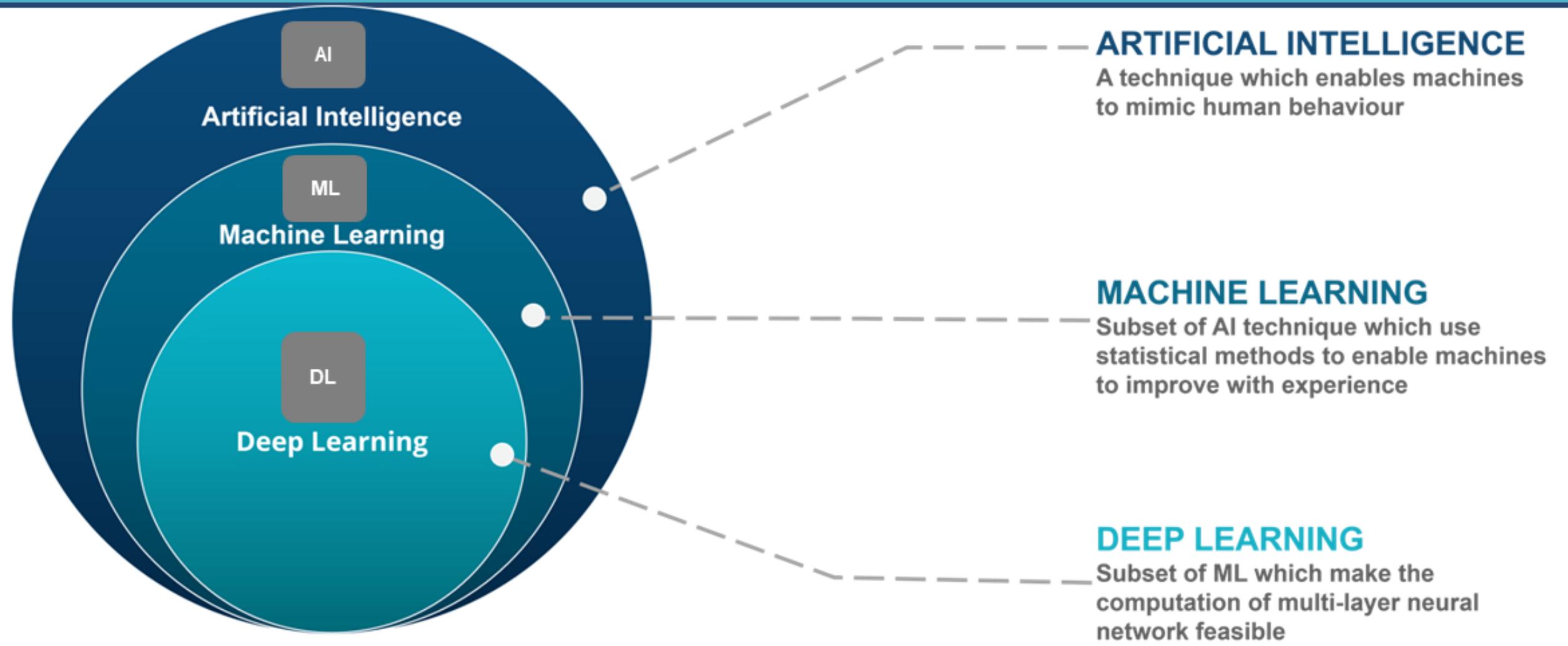


Algorithmes d'apprentissage automatique

1. Arbres de décision : utilisés pour les tâches de classification et de régression.
2. Support Vector Machines (SVM) : utilisées pour les tâches de classification et de régression.
3. Naïve Bayes Classifier : utilisé pour les tâches de classification.
4. K-Nearest Neighbors (KNN) : utilisé pour les tâches de classification et de régression.
5. Forêt aléatoire : utilisée pour les tâches de classification et de régression.
6. Régression logistique : utilisée pour les tâches de classification.
7. Gradient Boosting: utilisé pour les tâches de classification et de régression.
8. Réseaux de neurones : utilisés pour les tâches de classification et de régression.
9. K-Means Clustering: utilisé pour les tâches d'apprentissage non supervisées.
10. Il existe de nombreux autres algorithmes d'apprentissage automatique tels que les modèles de Markov cachés (HMM), la maximisation des attentes (EM) et les réseaux de croyances profondes (DBN). Il existe également de nombreux autres algorithmes d'apprentissage profond tels que les machines de Boltzmann, les réseaux antagonistes génératifs (GAN) et l'apprentissage par renforcement profond.

Algorithmes d'apprentissage profond

1. Réseaux de neurones convolutifs (CNN): utilisés pour analyser des images et des vidéos.
2. Réseaux neuronaux récurrents (RNN): utilisés pour le traitement du langage naturel et la reconnaissance vocale.
3. Réseaux antagonistes génératifs (GAN): utilisés pour générer des images et des vidéos.
4. Réseaux de mémoire à long terme (LSTM) : utilisés pour le traitement du langage naturel et la reconnaissance vocale.
5. Autoencodeurs : utilisés pour apprendre des représentations compressées de données.
6. Reinforcement Learning : utilisé pour l'entraînement des robots et des véhicules autonomes.



Artificial Intelligence

TRANSFORMA
INSIGHTS

Mimicking of
human
intelligence by
computers

Business rules
including
IFTTT

Machine Learning

Application of
statistical
techniques

Learn to
improve with
experience

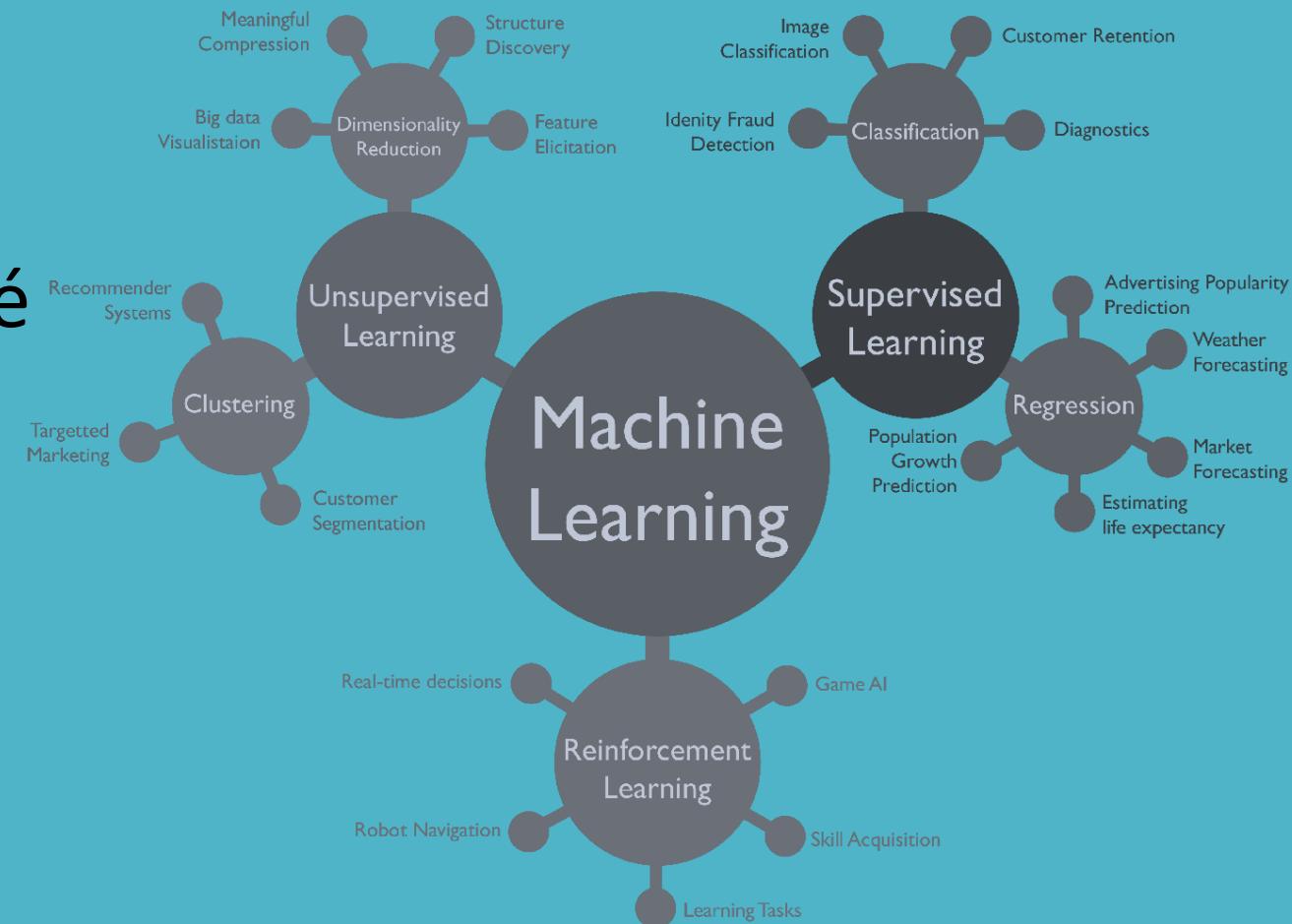
Deep Learning

Self-training software

Application of neural
networks

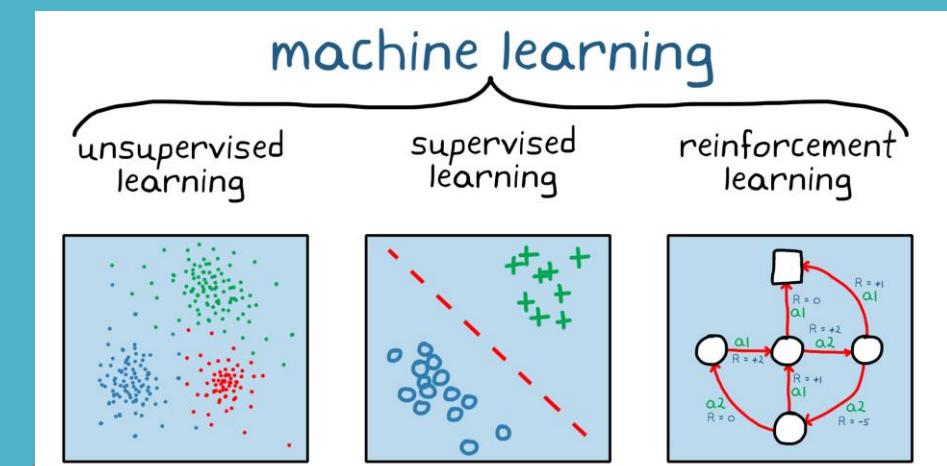
Types d'apprentissage automatique

1. Apprentissage supervisé
2. Apprentissage non supervisé
3. Apprentissage par renforcement
4. Apprentissage semi-supervisé



Apprentissage par renforcement

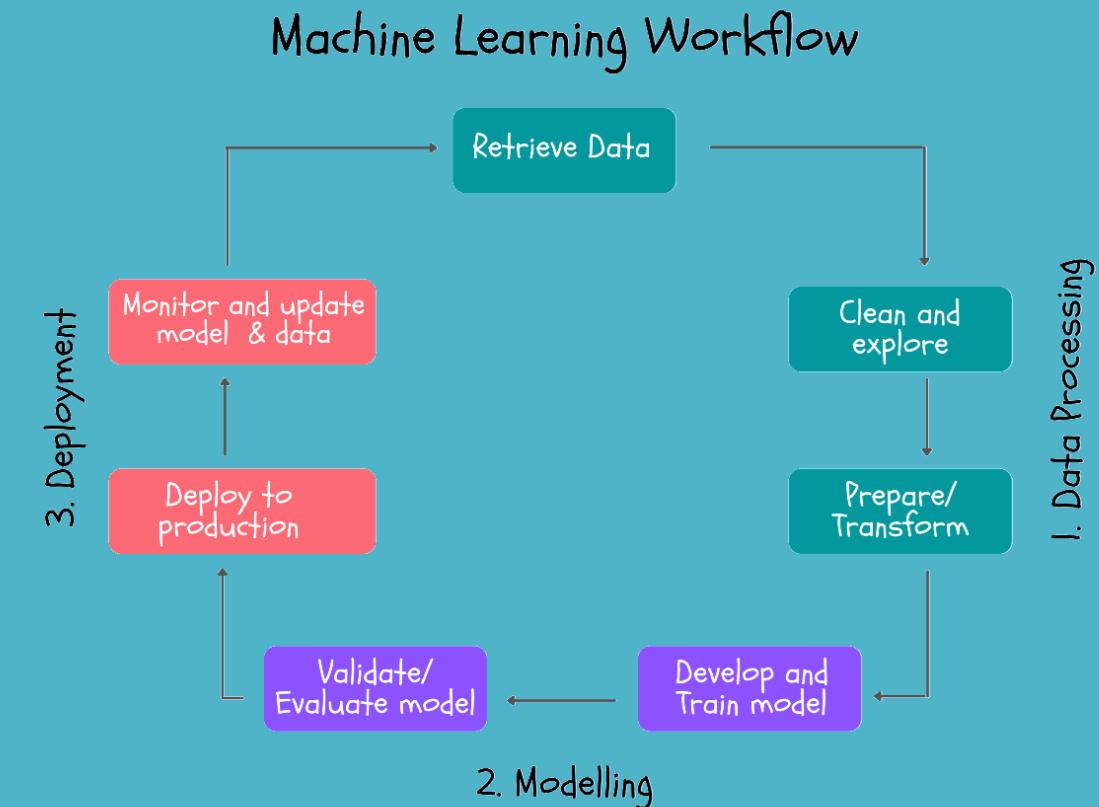
- L'apprentissage par renforcement est un type d'apprentissage automatique où un agent est formé pour prendre des mesures dans un environnement afin de maximiser une récompense.
- Ce type d'apprentissage automatique est utilisé pour entraîner des robots et des véhicules autonomes.
- Les algorithmes courants d'apprentissage par renforcement incluent le Q-learning et la recherche d'arbres de Monte Carlo.



Model	Learning task
Supervised Learning Algorithms	
Nearest Neighbor	Classification
Naive Bayes	Classification
Decision Trees	Classification
Classification Rule Learners	Classification
Linear Regression	Numeric prediction
Regression Trees	Numeric prediction
Model Trees	Numeric prediction
Neural Networks	Dual use
Support Vector Machines	Dual use
Unsupervised Learning Algorithms	
Association Rules	Pattern detection
k-means clustering	Clustering
Meta-Learning Algorithms	
Bagging	Dual use
Boosting	Dual use
Random Forests	Dual use

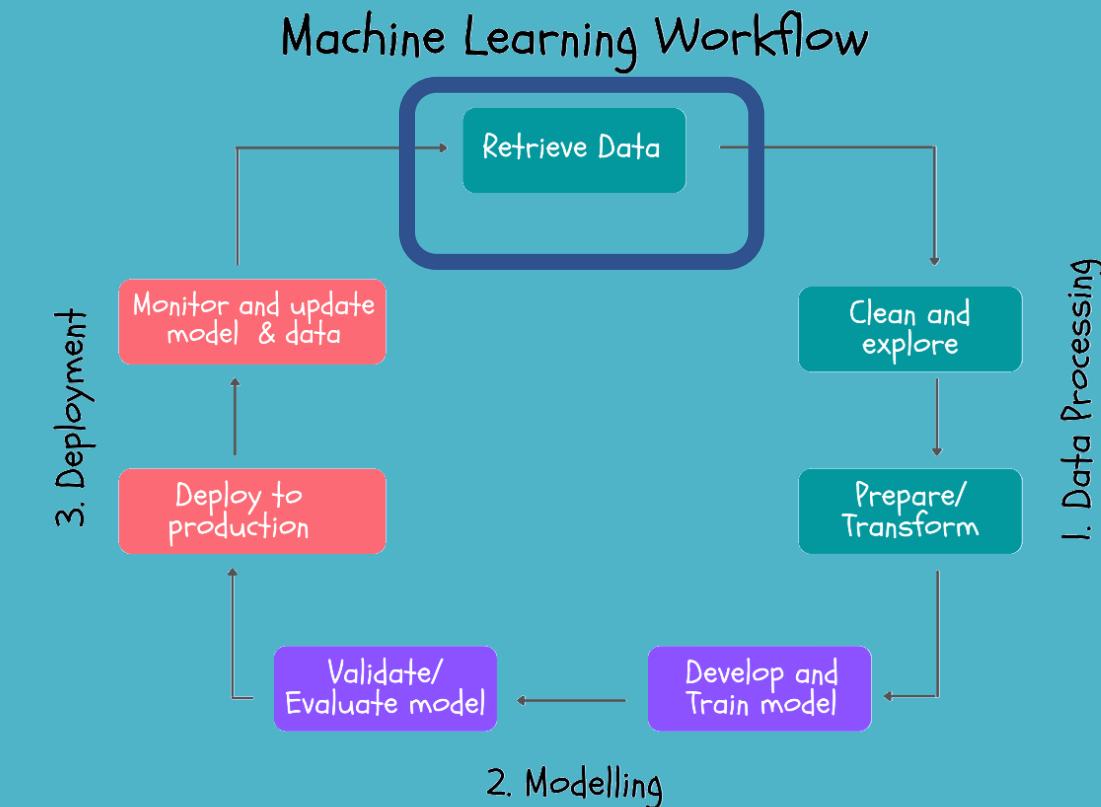
Flux de travail d'apprentissage automatique

- **Collecte de données** : collecte de données provenant de diverses sources.
- **Prétraitement des données** : nettoyage, transformation et normalisation des données.
- **Model Training** : formation du modèle sur les données traitées.
- **Évaluation du modèle** : évaluation du modèle sur les données de test.
- **Déploiement de modèle** : déploiement du modèle pour une utilisation dans un environnement de production.
- **Surveillance du modèle** : surveillance de la précision et des performances du modèle.



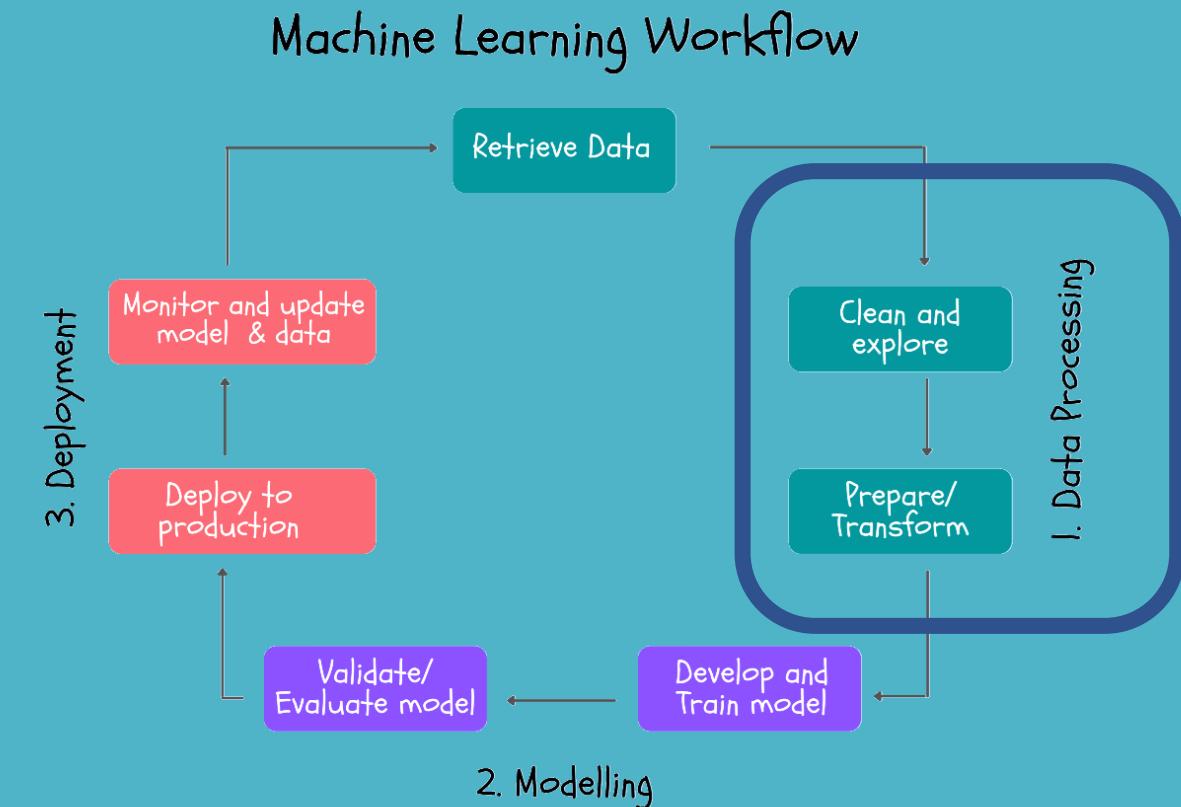
Machine learning : collecte de données

- La collecte de données à partir de diverses sources est la première étape du flux de travail d'apprentissage automatique.
- Cela implique la collecte de données à partir de bases de données, d'API Web et d'autres sources.
- Il est important de s'assurer que les données sont complètes, exactes et pertinentes pour la tâche à accomplir.



Machine learning : Prétraitement des données

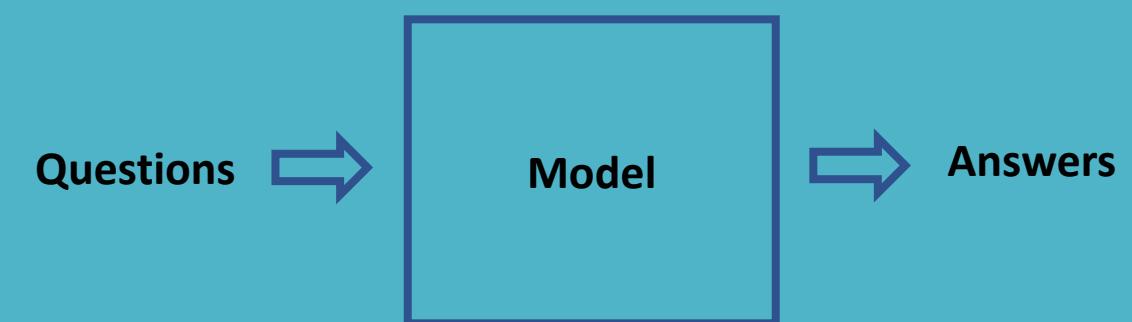
- Nettoyez, transformez et normalisez les données.
- Cela implique de supprimer les valeurs aberrantes, de transformer les variables catégorielles en variables numériques et de mettre à l'échelle les données.
- Il est également important de diviser les données en ensembles d'apprentissage, de validation et de test.



Phase 1: Training – deriving a model

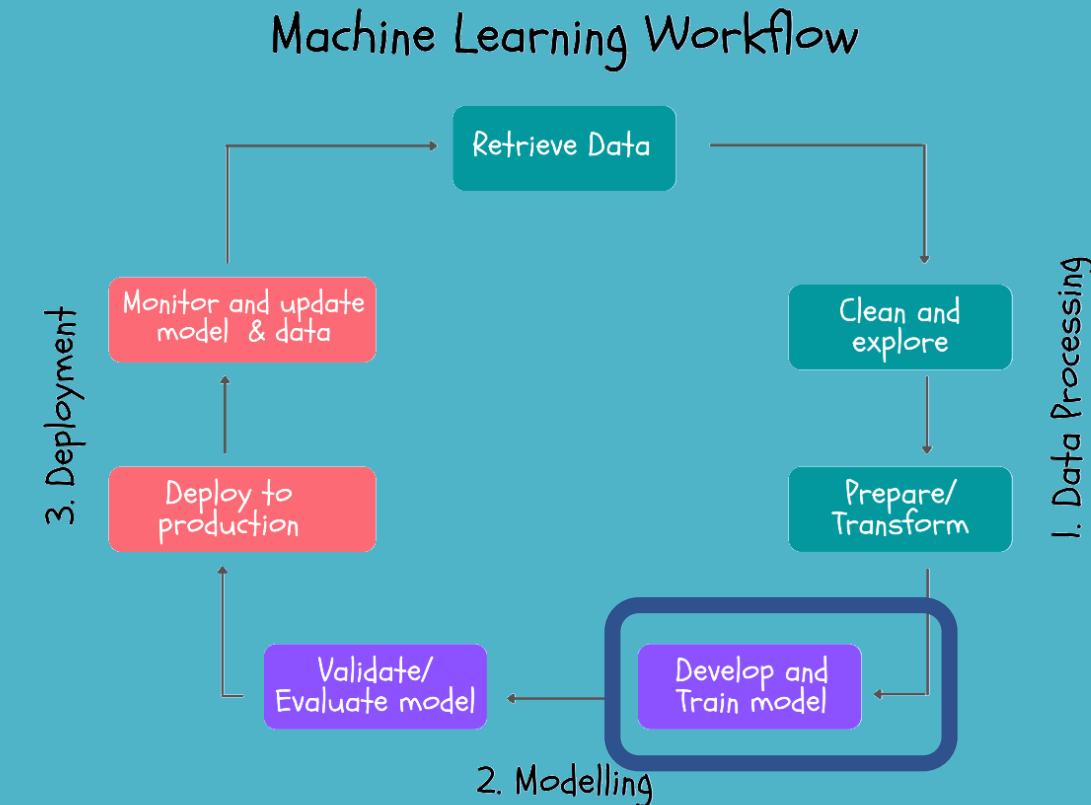


Phase 2: Inference – using the model



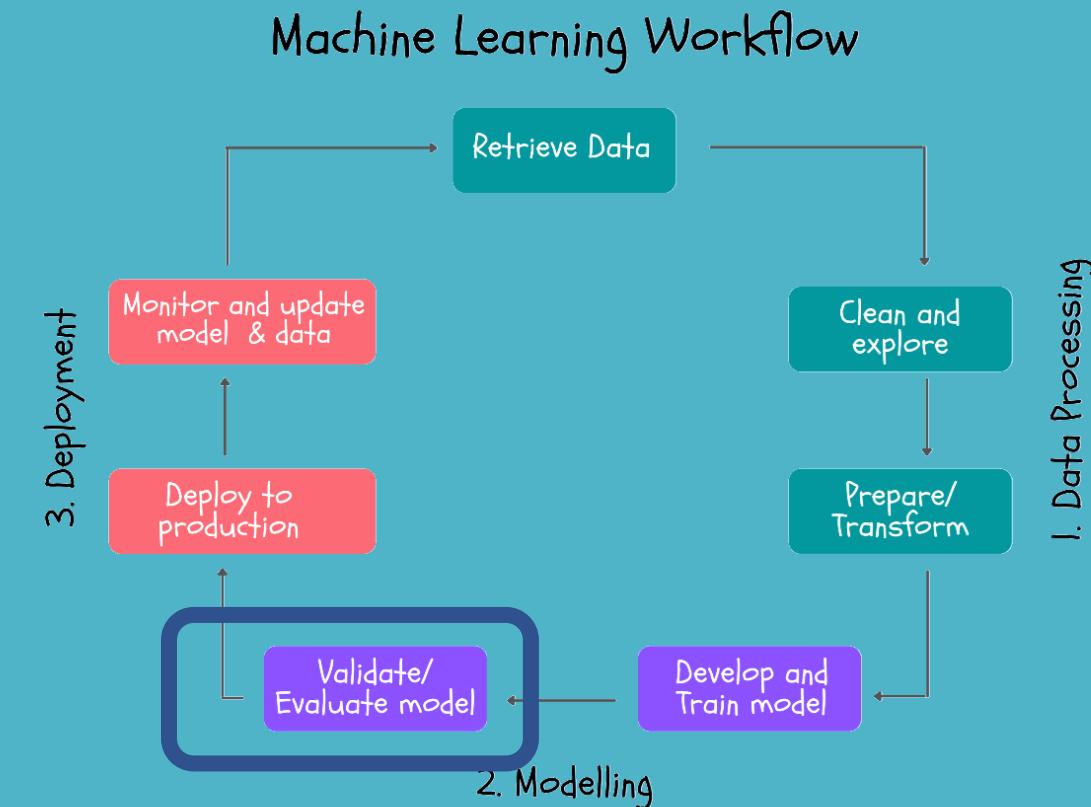
Machine learning : formation de modèles

- Après le prétraitement des données, le modèle est formé sur les données traitées.
- Cela implique d'ajuster le modèle aux données et d'ajuster les paramètres.
- Il est important de choisir le bon modèle pour la tâche à accomplir et de s'assurer qu'il ne surajuste pas ou ne sous-ajuste pas les données.



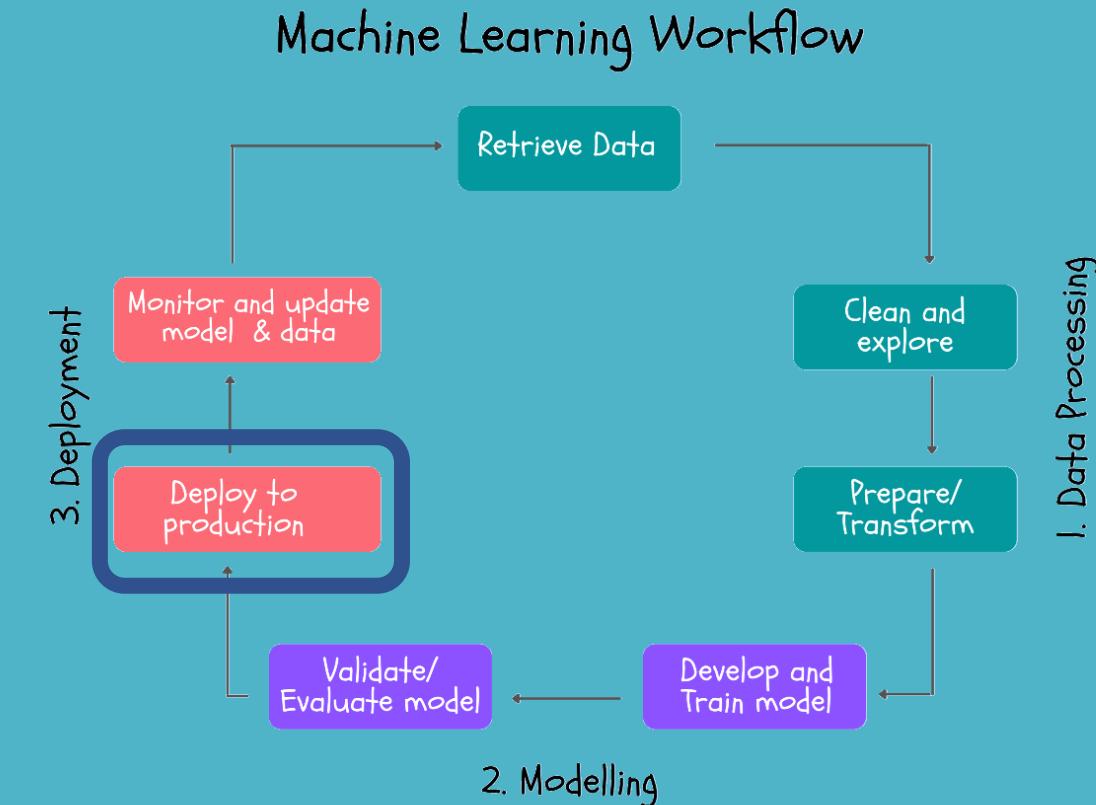
Machine learning : évaluation de modèle

- Après l'entraînement du modèle, il est évalué sur les données de test.
- Cela implique de mesurer la précision et la performance du modèle et d'apporter les ajustements nécessaires.
- Les mesures courantes pour évaluer les modèles d'apprentissage automatique incluent l'exactitude, la précision, le rappel et le score F1.
- Collecte de données : collecte de données auprès de



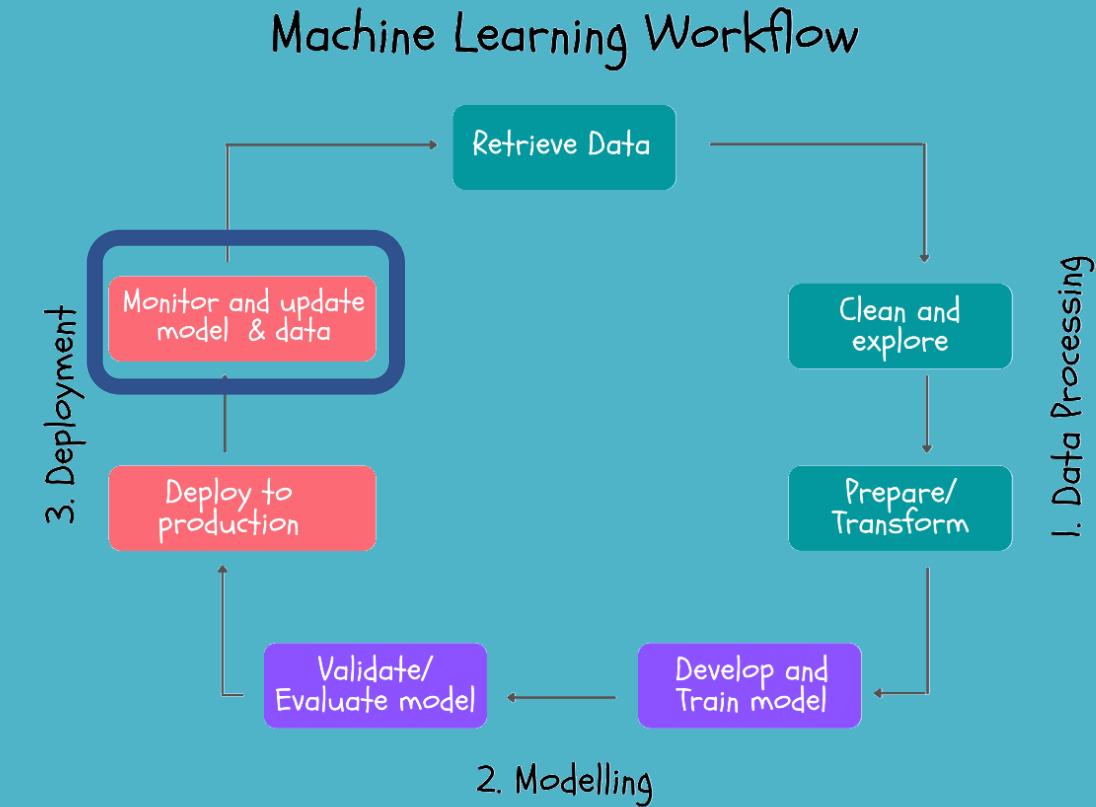
Machine learning : déploiement de modèles

- Le modèle est ensuite déployé dans un environnement de production, ce qui lui permet d'être utilisé en continu dans des applications réelles.
- Il est important de tenir compte du coût de la formation et du déploiement du modèle, ainsi que des implications en matière de confidentialité et de sécurité des données.



Machine learning : Surveillance des modèles

- Le modèle est ensuite surveillé pour la précision et la performance.
- Cela implique de mettre à jour le modèle au besoin et de suivre ses performances au fil du temps.
- Les mesures courantes pour la surveillance des modèles d'apprentissage automatique incluent l'exactitude, la précision, le rappel et le score F1



Qu'est-ce qu'un réseau neurones ?

- Les réseaux neuronaux sont une classe d'algorithmes d'apprentissage automatique conçus pour modéliser la structure et la fonction du cerveau humain.
- Ils sont constitués de couches interconnectées de neurones artificiels qui traitent et transforment les données d'entrée, afin de produire des prédictions ou des classifications de sortie.
- Chaque neurone d'un réseau neuronal reçoit une ou plusieurs entrées, applique une fonction de transformation à ces entrées et produit une sortie qui est transmise à la couche suivante de neurones.
- Les poids des connexions entre les neurones déterminent la force et la direction de la connexion, et sont ajustés pendant l'entraînement afin d'optimiser les performances du réseau.
- Les réseaux neuronaux peuvent être entraînés à l'aide d'une variété d'algorithmes d'optimisation, tels que la descente de gradient, afin de minimiser une perte donnée ou une fonction d'erreur.
- Ils peuvent gérer une variété de types de données, y compris des données numériques, catégorielles et textuelles, et peuvent apprendre des modèles et des relations complexes dans les données sans avoir besoin d'une ingénierie explicite des fonctionnalités.
- Il existe plusieurs types de réseaux neuronaux, y compris les réseaux neuronaux feedforward, les réseaux de neurones convolutifs et les réseaux neuronaux récurrents, chacun avec ses propres forces et applications.
- Les réseaux neuronaux ont atteint des performances de pointe sur de nombreuses tâches difficiles, telles que la reconnaissance d'images, la reconnaissance vocale et le traitement du langage naturel, et sont largement utilisés dans l'industrie et le milieu universitaire.
- Bien que les réseaux neuronaux puissent être complexes et difficiles à interpréter, les progrès récents dans les techniques de visualisation et d'explicabilité ont facilité la compréhension et l'interprétation du fonctionnement interne de ces modèles.
- Dans l'ensemble, les réseaux neuronaux sont un outil puissant pour résoudre un large éventail de problèmes et ont le potentiel de révolutionner de nombreux domaines, des soins de santé et de la finance au transport et à la fabrication.

Pourquoi utiliser des réseaux neurones ?

- Les réseaux neuronaux sont une classe d'algorithmes d'apprentissage automatique qui s'inspirent de la structure et de la fonction du cerveau humain et sont capables d'apprendre des modèles et des relations complexes dans les données.
- Ils sont très polyvalents et peuvent être utilisés pour un large éventail d'applications, y compris la reconnaissance d'images et de paroles, le traitement du langage naturel, les systèmes de recommandation et la robotique, entre autres.
- Les réseaux neuronaux peuvent gérer une variété de types de données, y compris des données numériques, catégorielles et textuelles, ainsi que des données avec des dépendances temporelles ou spatiales.
- Ils sont capables d'apprendre automatiquement des fonctionnalités et des représentations à partir de données d'entrée brutes, sans avoir besoin d'une ingénierie explicite des fonctionnalités.
- Les réseaux neuronaux peuvent être formés sur de grands ensembles de données et sont capables de bien généraliser à des données invisibles, ce qui les rend très efficaces pour résoudre des problèmes complexes.
- Ils sont hautement évolutifs et peuvent être parallélisés sur plusieurs processeurs ou GPU, ce qui permet une formation et une inférence efficaces sur de grands ensembles de données.
- Avec les progrès récents des techniques d'apprentissage profond, telles que les réseaux de neurones convolutifs et les réseaux de neurones récurrents, les réseaux neuronaux ont atteint des performances de pointe sur de nombreuses tâches difficiles, notamment la reconnaissance d'images, la reconnaissance vocale et le traitement du langage naturel.
- Les réseaux neuronaux ont un haut degré de flexibilité et peuvent être personnalisés et réglés pour répondre aux exigences spécifiques d'un problème ou d'une application donnée.
- Bien que les réseaux neuronaux puissent être complexes et difficiles à interpréter, les progrès récents dans les techniques de visualisation et d'explicabilité ont facilité la compréhension et l'interprétation du fonctionnement interne de ces modèles.
- Dans l'ensemble, les réseaux neuronaux sont un outil puissant pour résoudre un large éventail de problèmes et ont le potentiel de révolutionner de nombreux domaines, des soins de santé et de la finance au transport et à la fabrication.

NEURAL NETWORK

10001
10100
10001
10101
10010
01011

29%

76%

G - 4/2

RTI - 675.8965 - 874.8374
SH - 456/9583 - 472.8921
G - 8943 - 6754

B - 784.905
RT - 7832 - 9043 - 7841
[004] 895 - 785 S
UT - 847 - 26

DS - 2893 - 0004

FG - 8374 - 9031

R - 7.4

15.9438

46.8725

14.847

KU - 0073

F - 2.4

RO - 2893

VA - 2904

A - 7/3

54.059 - 9504

84.903 - 8942

89.049 - 1948

Q - 9973

F - 8.2

63.7843

G - 3948

D - 7/2

27.0479

B - 3/8

31.375

68.3742

31.8472

17.352

C - 5/9

11.376

IO - 0782

PV - 1783

57.0062

RO - 9472

29.625

S - 8932

9.3056

R - 3.8

38.0104

IT - 0837

33.2748

FN - 9062

54.7831

J [002] - 8953 - 34

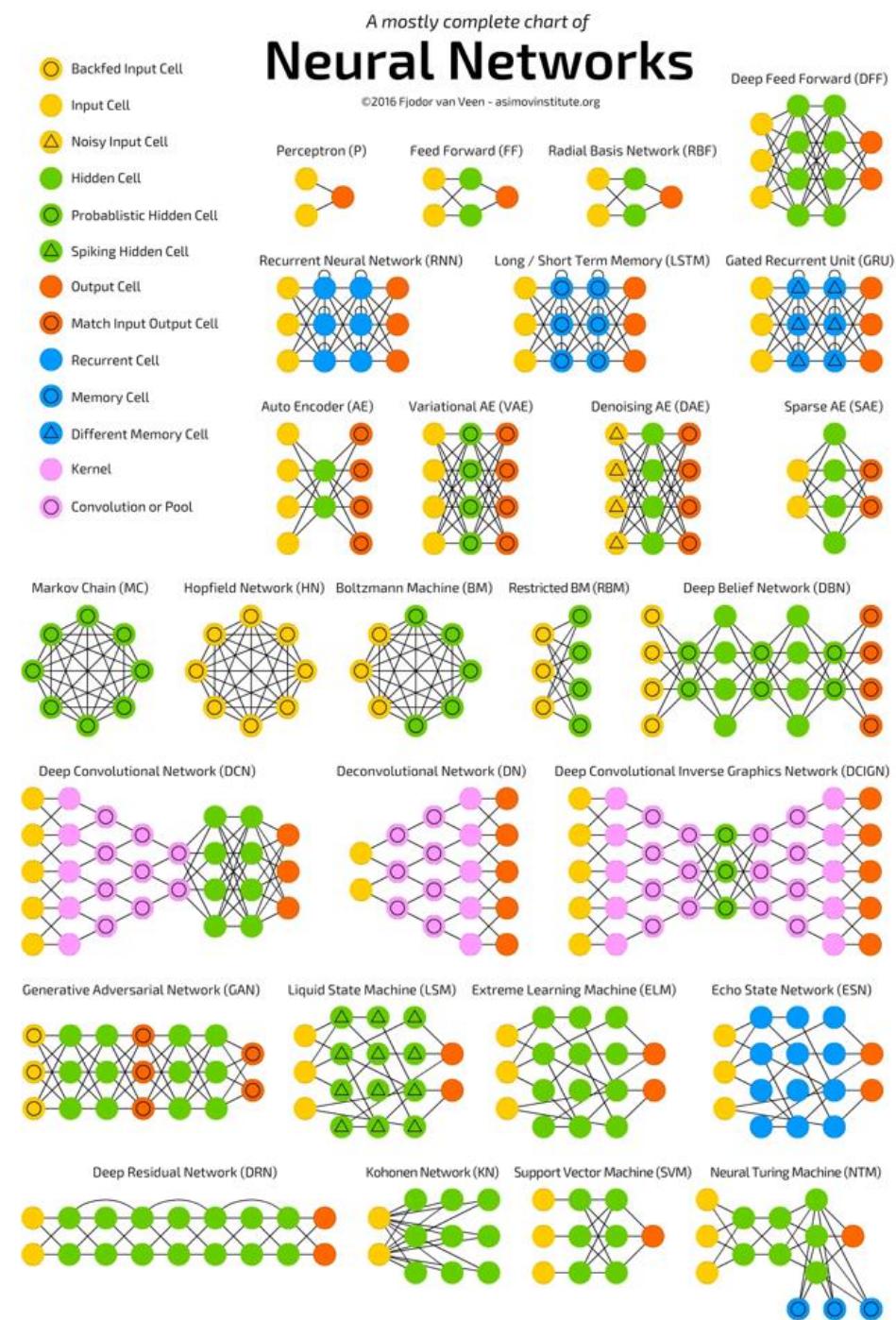
D5 - 900 - 895 M

3894 - 9032

22.106

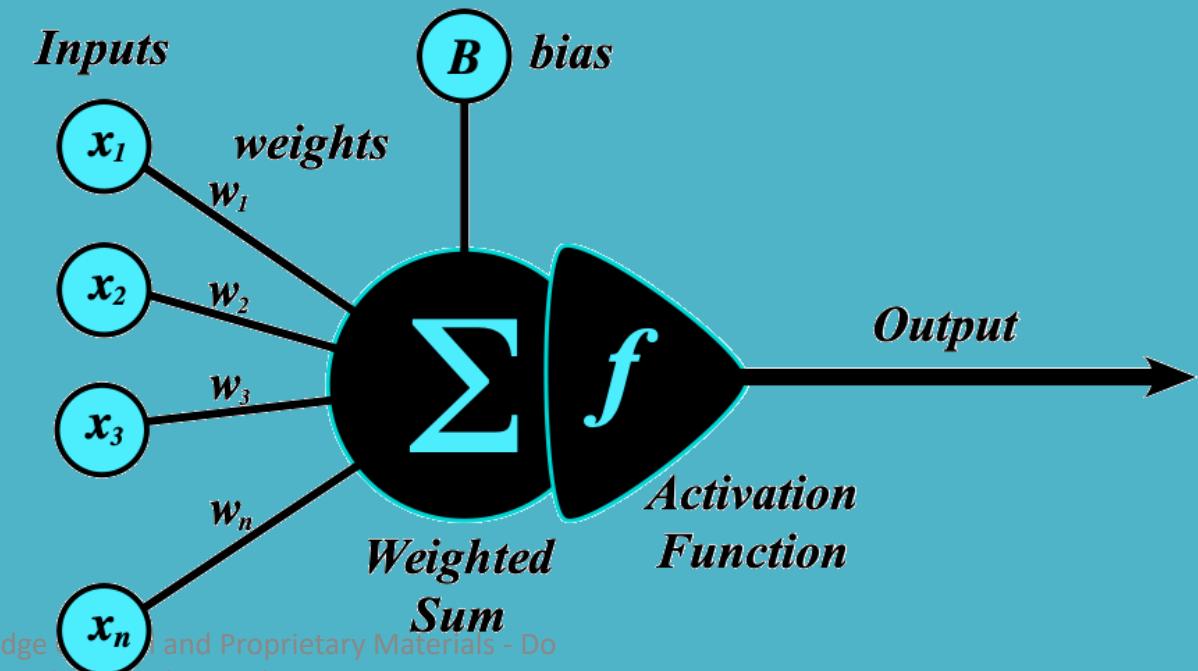
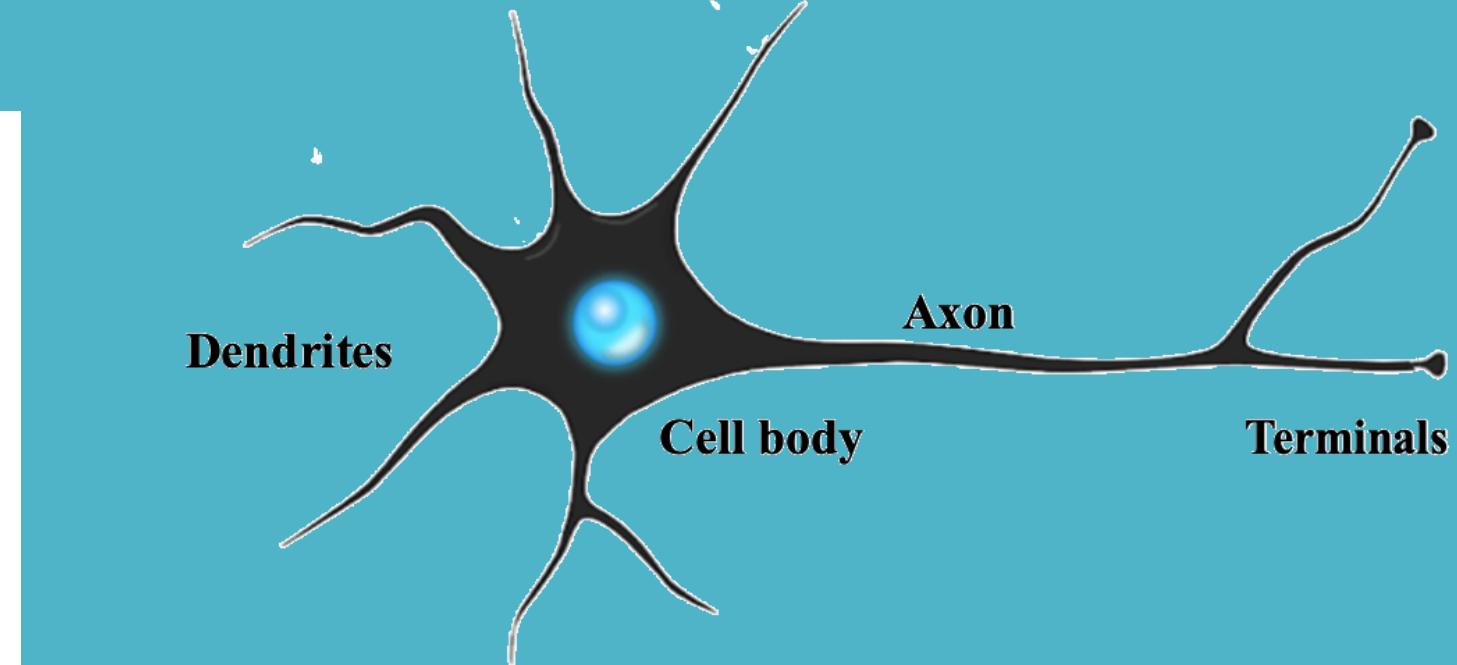
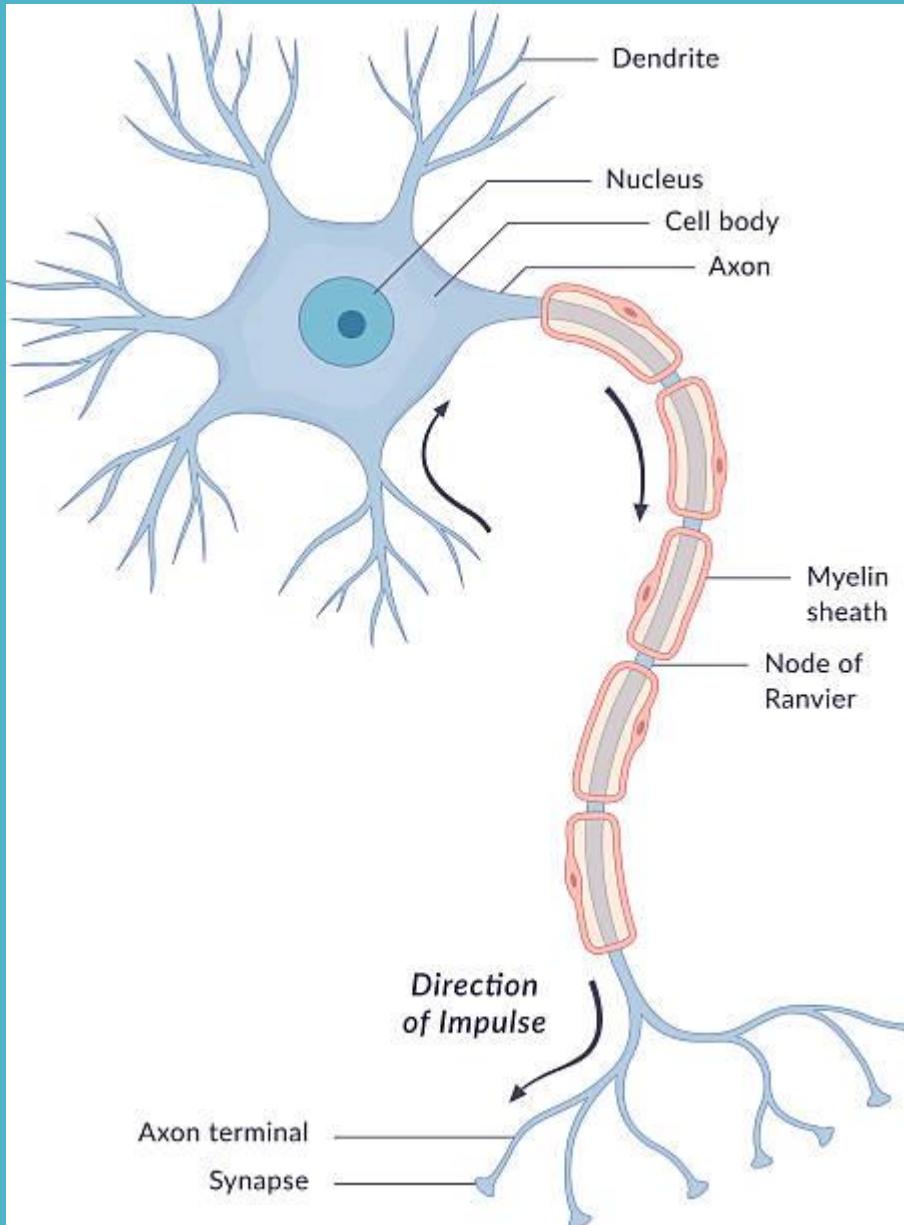
DW - 7361

37.4856



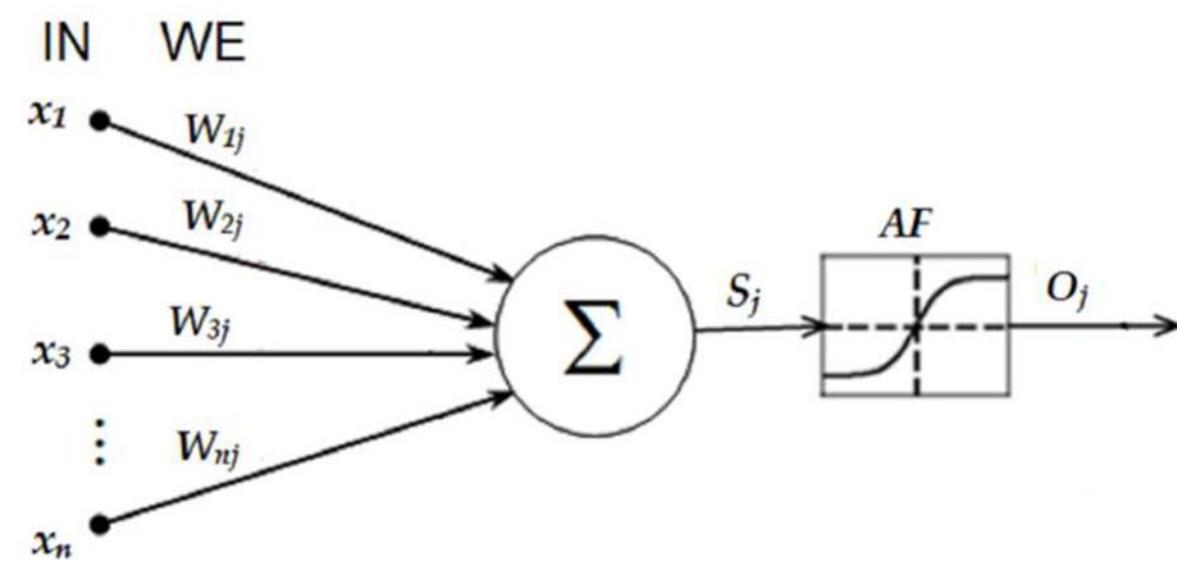
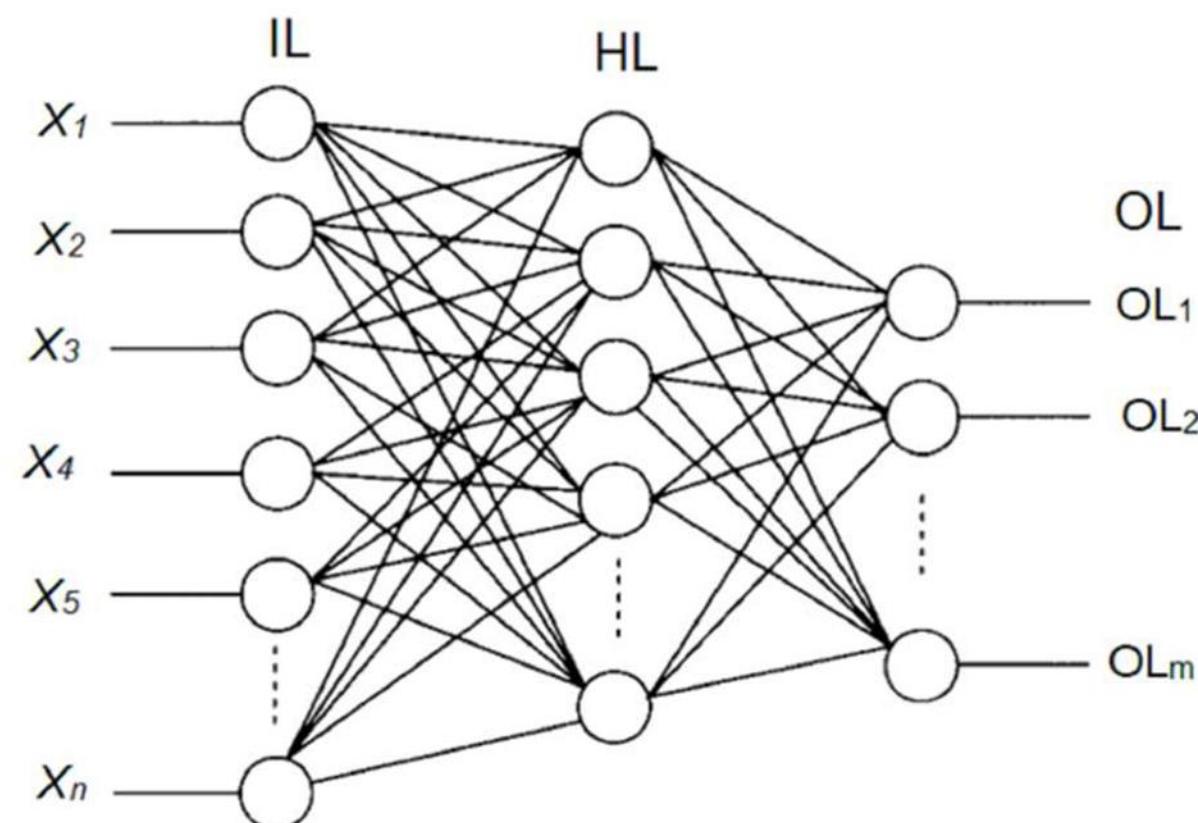
Neurones biologiques

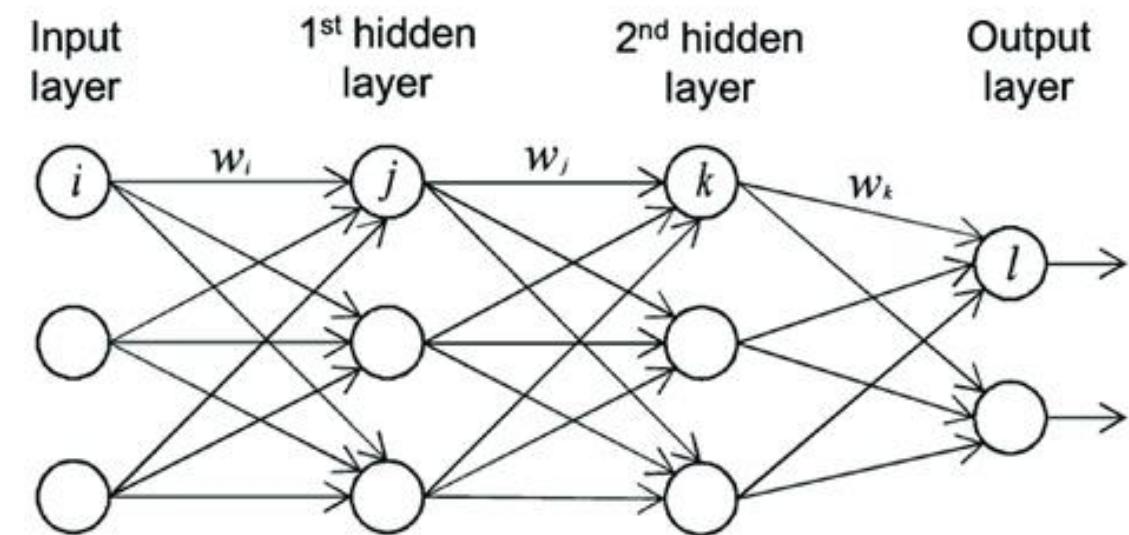
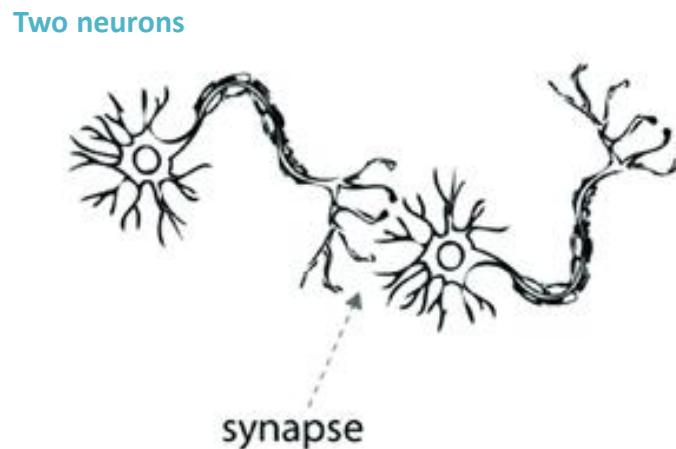
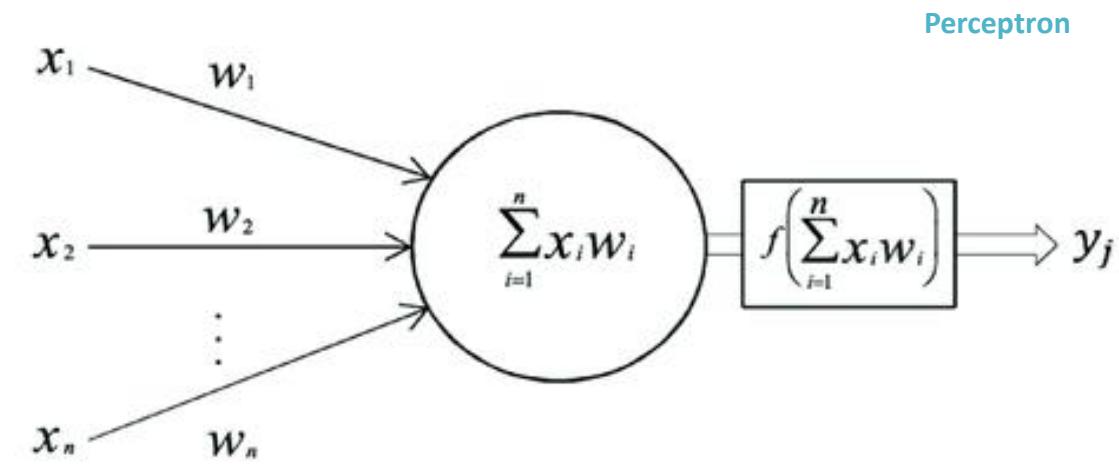
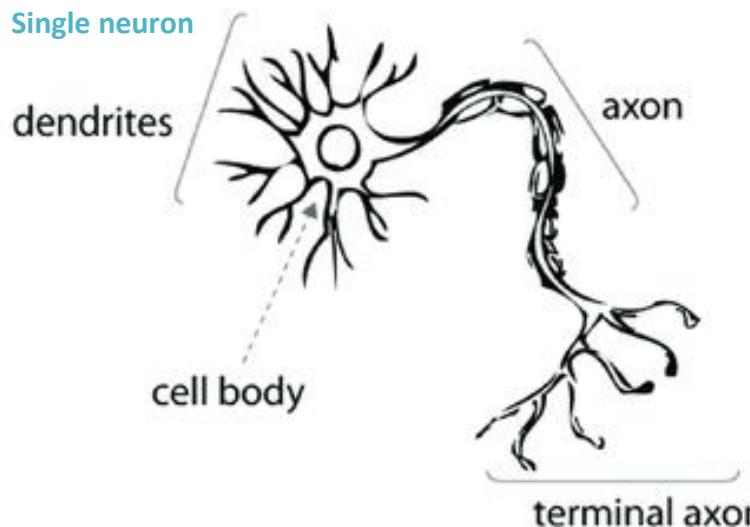
- Les réseaux neuronaux s'inspirent de la structure et de la fonction du cerveau humain et sont conçus pour modéliser le comportement des neurones biologiques et leurs interactions.
- Comme les neurones biologiques, les neurones artificiels d'un réseau neuronal reçoivent des entrées d'autres neurones ou de sources externes et produisent une sortie qui est transmise à d'autres neurones ou couches de sortie du réseau.
- Dans les neurones biologiques, les entrées sont reçues par des dendrites, qui transmettent des signaux électriques au corps cellulaire, où les entrées sont intégrées et traitées.
- De même, les neurones artificiels d'un réseau neuronal reçoivent des entrées via des connexions pondérées, qui sont additionnées et transmises par une fonction d'activation pour produire une sortie.
- La fonction d'activation dans les neurones artificiels est souvent modelée d'après le comportement de déclenchement des neurones biologiques, tels que les fonctions sigmoïdes ou ReLU.
- La sortie d'un neurone artificiel peut être considérée comme une approximation de la vitesse de déclenchement d'un neurone biologique.
- Dans les neurones biologiques, la force des connexions entre les neurones est déterminée par les propriétés des synapses, qui peuvent être renforcées ou affaiblies par un processus connu sous le nom de plasticité synaptique.
- De même, la force des connexions entre les neurones artificiels dans un réseau neuronal est déterminée par le poids des connexions, qui sont ajustées pendant l'entraînement pour optimiser les performances du réseau.
- Bien que les réseaux de neurones artificiels ne soient pas des répliques exactes de la structure et de la fonction du cerveau humain, l'analogie fournit un cadre utile pour comprendre et concevoir ces modèles, et a conduit à de nombreuses percées importantes dans le domaine de l'apprentissage automatique.



Perceptron

- Le perceptron est le type de réseau neuronal le plus simple, composé d'une seule couche de neurones avec des connexions d'entrée et une connexion de sortie unique.
- Chaque entrée est multipliée par un poids, et ces entrées pondérées sont additionnées.
- La somme est ensuite transmise par une fonction d'activation, qui détermine si le neurone doit se déclencher ou non en fonction de son entrée.
- La sortie du perceptron est la sortie de la fonction d'activation.

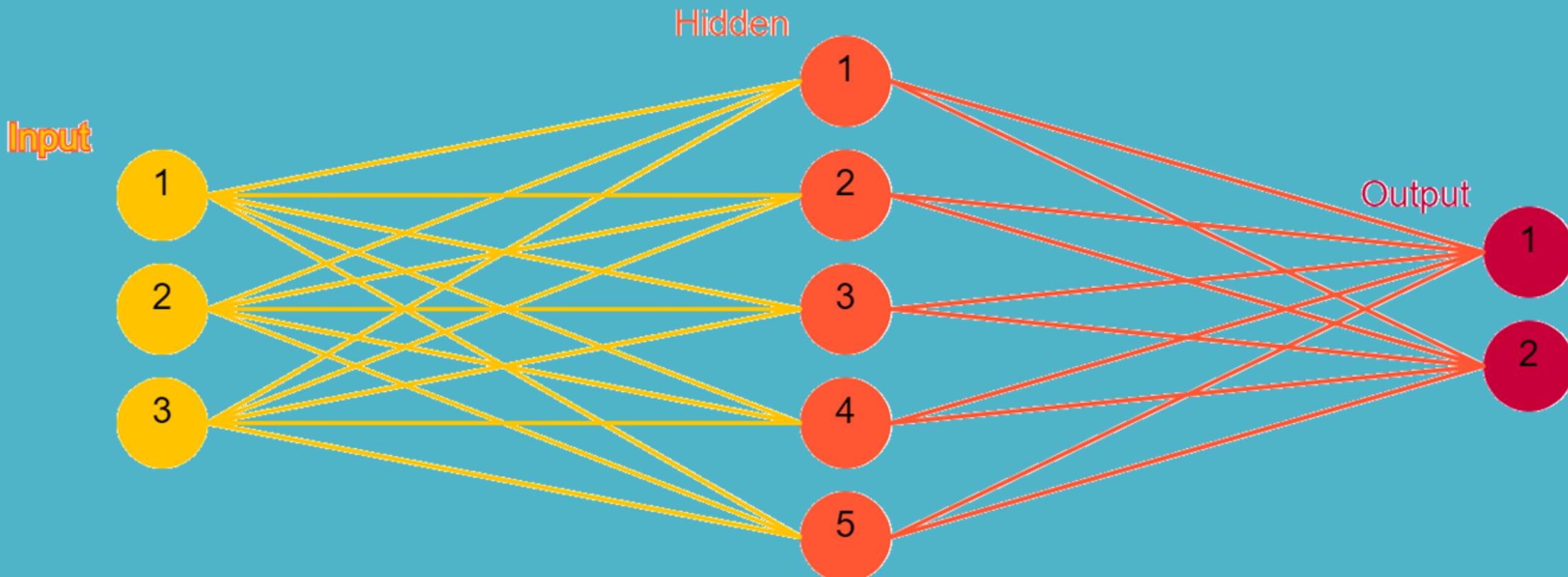




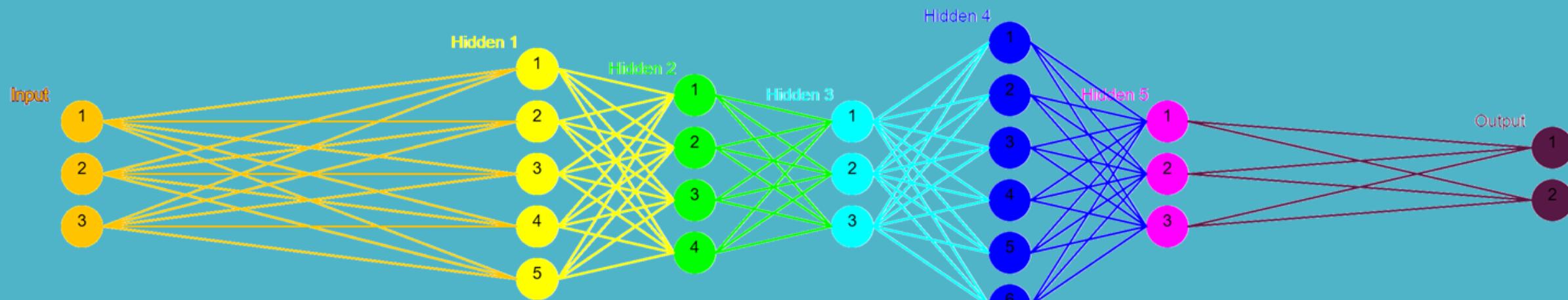
Perceptron multicouche

- Le perceptron multicouche (MLP) est un type de réseau neuronal qui contient plusieurs couches de neurones, y compris une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie.
- Dans un MLP, chaque neurone d'une couche est connecté à chaque neurone de la couche précédente, et chaque connexion a un poids qui lui est associé.
- Les poids de toutes les connexions du réseau sont stockés dans une matrice appelée matrice de poids.
- La fonction d'activation est appliquée à la sortie de chaque neurone du réseau, et la sortie résultante est transmise à la couche suivante.

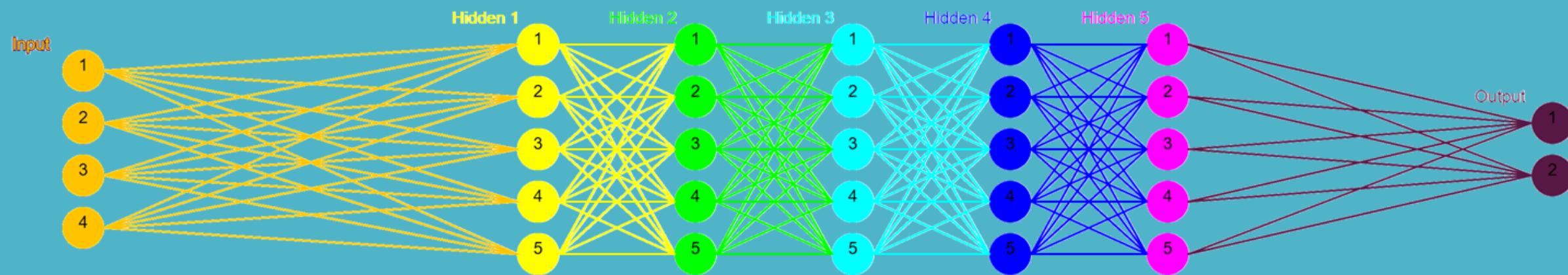
SLP perceptron – neural network



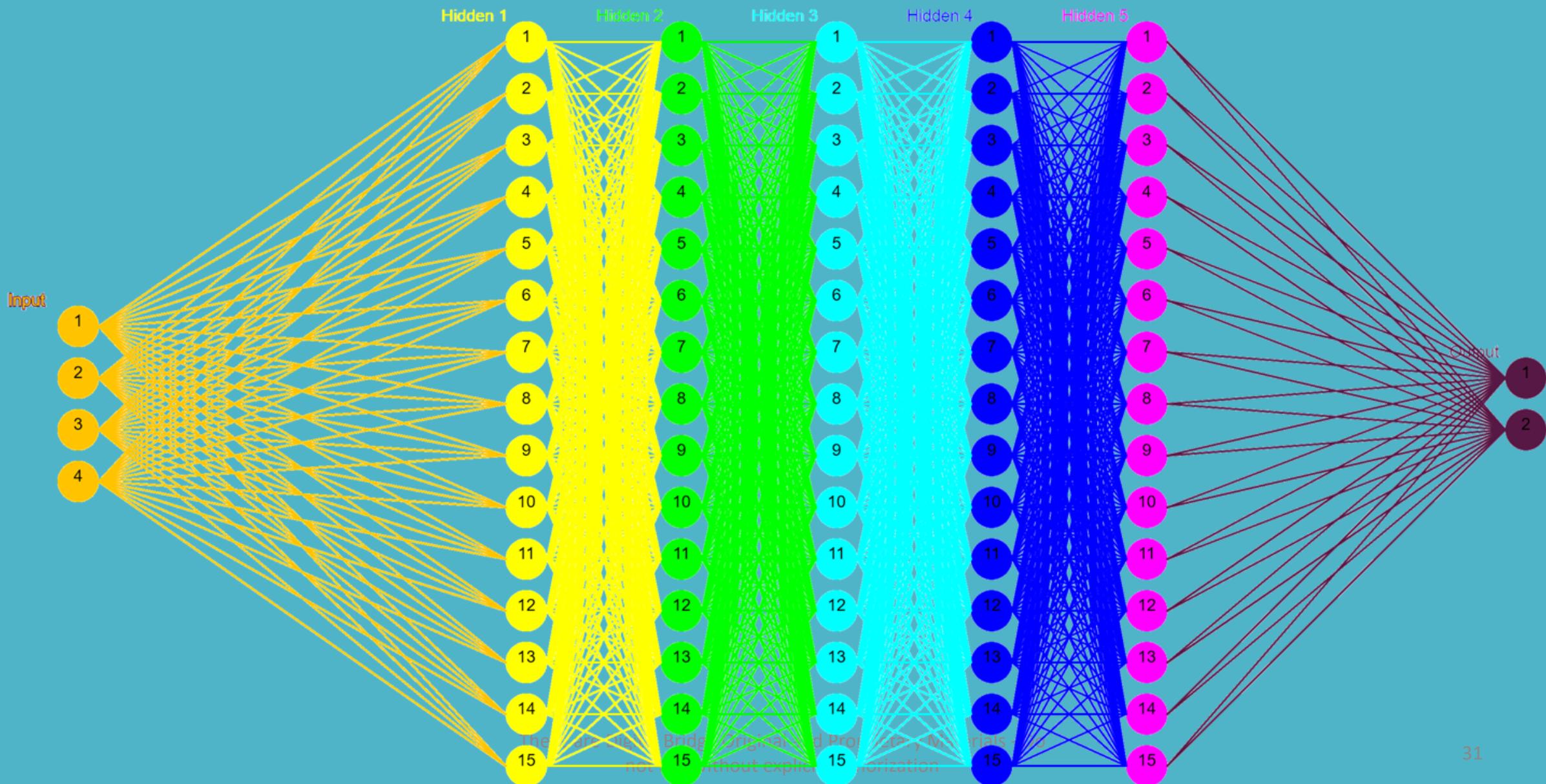
MLP perceptron – neural network



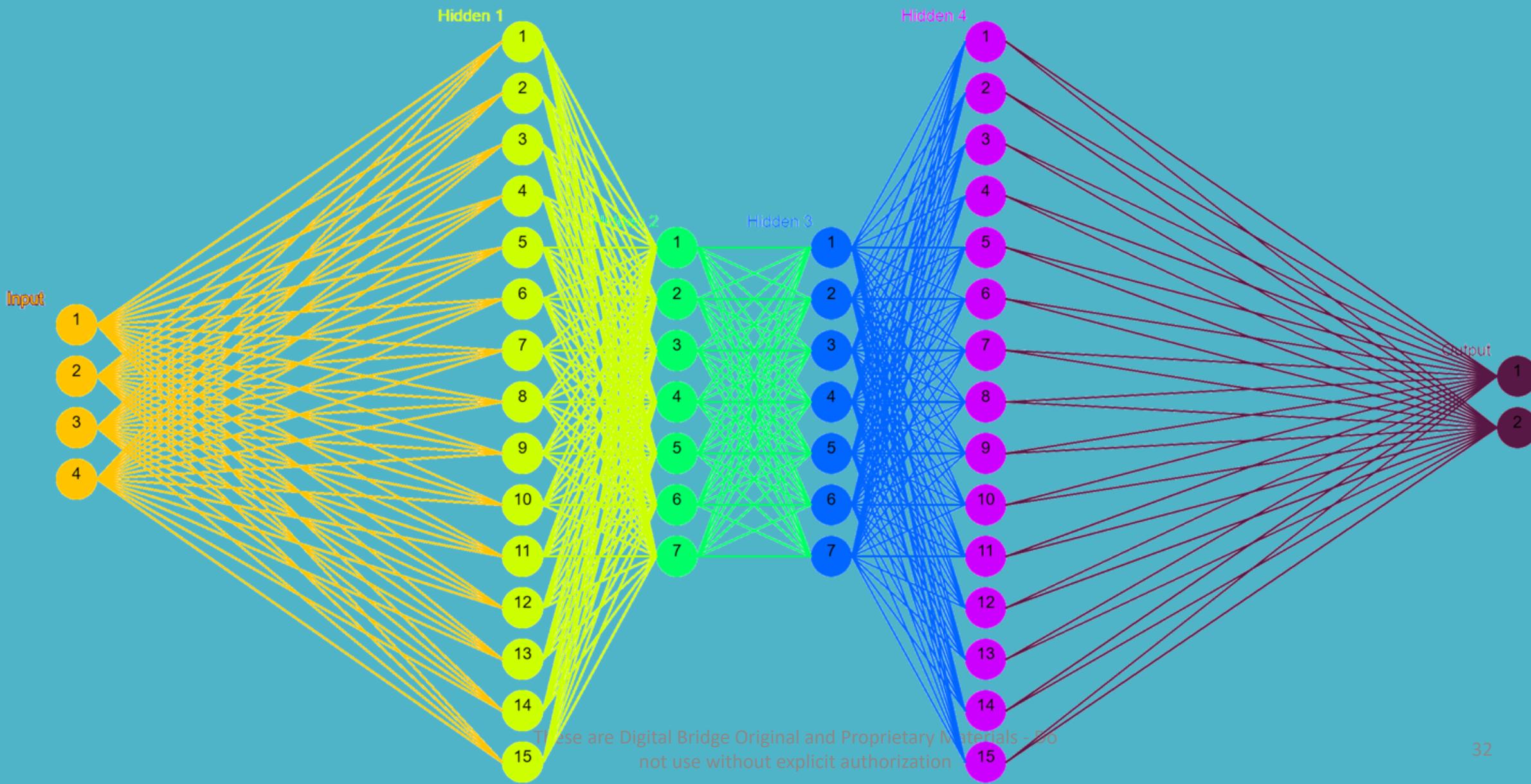
MLP perceptron – neural network



MLP perceptron – neural network



MLP perceptron – neural network



Mathématiques matricielles

- La multiplication matricielle est une opération clé dans les réseaux neuronaux et est utilisée pour calculer la somme pondérée des entrées à chaque couche du réseau.
- La matrice de poids de chaque couche est multipliée par la sortie de la couche précédente pour produire l'entrée de la couche suivante.
- Ce processus se poursuit jusqu'à ce que la couche de sortie soit atteinte et que la sortie finale du réseau soit produite.
- L'opération de multiplication matricielle est effectuée efficacement à l'aide de bibliothèques d'algèbre linéaire telles que NumPy.

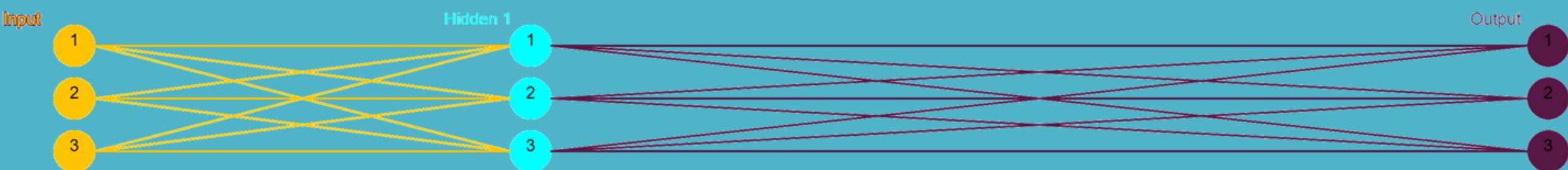
```
import numpy as np

# Define input data and weight matrix
input_data = np.array([1, 2, 3]) # shape: (3,)
weight_matrix = np.array([[0.1, 0.2, 0.3], [0.4, 0.5, 0.6], [0.7, 0.8, 0.9]]) # shape: (3, 3)

# Perform matrix multiplication and apply activation function
output_data = np.dot(weight_matrix, input_data)
output_data = 1 / (1 + np.exp(-output_data))

print(input_data)
print(output_data)
```

```
[0.80218389 0.96083428 0.99330715]
```



Matrix math

Matrix product 3 by 3 matrices

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$c_{ij} = \sum_{k=1}^3 a_{ik}b_{kj}$$

$$c_{ij} = \sum_{k=1}^3 a_{ik}b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j}$$

Matrix math

Matrix product of 3x5 by 5x5 matrices

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{pmatrix}$$

The formula for calculating the entries of the resulting matrix is:

$$c_{ij} = \sum_{k=1}^5 a_{ik}b_{kj}$$

In this case, the matrices are 3 by 5 and 5 by 5, so the resulting product matrix is 3 by 5. The subscript indices for the matrix entries and summation variable have been adjusted accordingly.

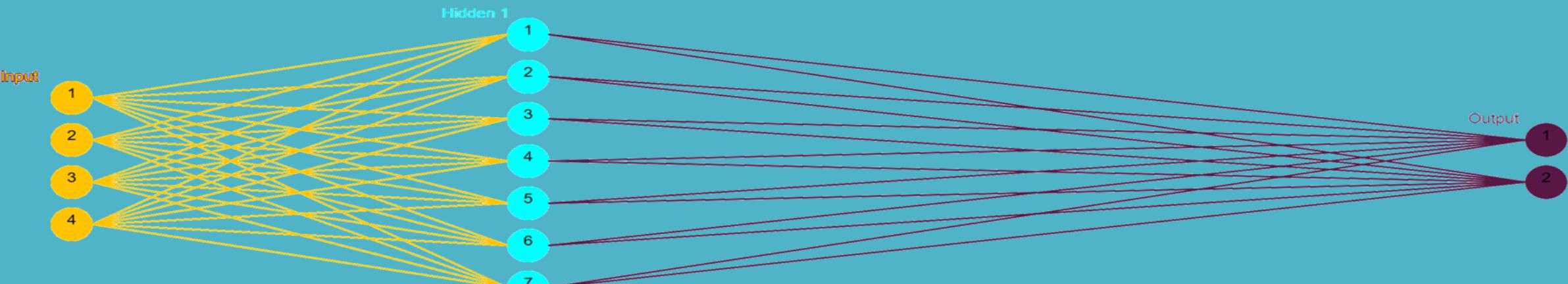
Matrix math

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \\ b_{51} & b_{52} \\ b_{61} & b_{62} \\ b_{71} & b_{72} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \\ c_{41} & c_{42} \end{pmatrix}$$

The formula for calculating the entries of the resulting matrix is:

$$c_{ij} = \sum_{k=1}^7 a_{ik}b_{kj}$$

In this case, the matrices are 4 by 7 and 7 by 2, so the resulting product matrix is 4 by 2. The subscript indices for the matrix entries and summation variable have been adjusted accordingly.



Biais

- Le biais est un paramètre supplémentaire utilisé pour ajuster la sortie de chaque neurone dans un réseau neuronal.
- Il est ajouté à la somme pondérée des entrées à chaque neurone avant que la fonction d'activation ne soit appliquée.
- Le biais est une valeur d'entrée supplémentaire qui est toujours égale à 1, et il est associé à un poids tout comme les autres entrées.
- Le terme de biais est utilisé pour déplacer la fonction d'activation vers la gauche ou la droite, donnant au réseau neuronal plus de flexibilité pour s'adapter aux données.
- Sans biais, la fonction d'activation ne pourrait passer par l'origine (0,0) que sur le plan entrée-sortie.
- Le biais permet à la fonction d'activation de se déplacer le long de l'axe y, ce qui peut être utile pour modéliser des données qui peuvent avoir une moyenne non nulle.
- La sortie d'un neurone avec biais peut être représentée comme une somme pondérée d'entrées plus un terme de biais, passé par une fonction d'activation.
- Le biais est un paramètre apprenant qui est initialisé avec une certaine valeur et mis à jour pendant le processus de formation pour optimiser les performances du réseau neuronal.
- Le poids de biais peut être ajusté à l'aide de techniques telles que la descente de gradient stochastique, tout comme les autres poids du réseau.

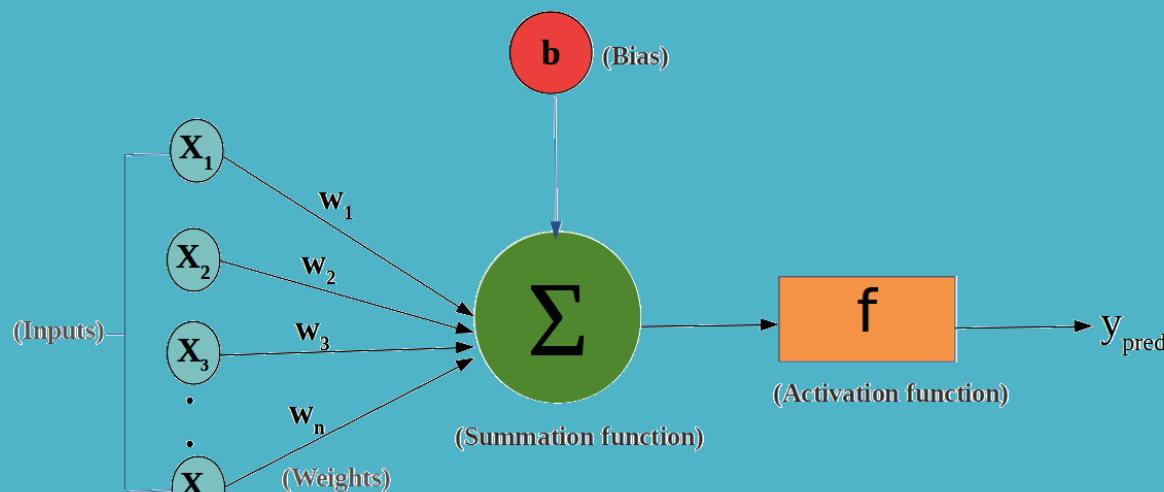
Poids (1)

- Les poids sont les paramètres d'un réseau neuronal qui sont appris pendant l'entraînement afin de faire des prédictions précises.
- Dans un réseau neuronal feedforward, chaque neurone d'une couche est connecté à chaque neurone de la couche précédente par un ensemble de poids, qui déterminent la force et la direction de la connexion.
- Pendant l'entraînement, les poids sont mis à jour par un algorithme d'optimisation, tel que la descente de gradient, afin de minimiser la fonction d'erreur ou de perte du réseau.
- Les poids sont initialisés aléatoirement au début de l'entraînement, et leurs valeurs sont mises à jour par petits incrément lors de chaque itération de l'algorithme d'optimisation.
- Les valeurs des poids sont essentielles à la performance du réseau et peuvent avoir un impact considérable sur la précision et la rapidité du processus d'entraînement.
- Le nombre de poids dans un réseau neuronal dépend du nombre de neurones dans chaque couche, ainsi que de la connectivité entre les couches.

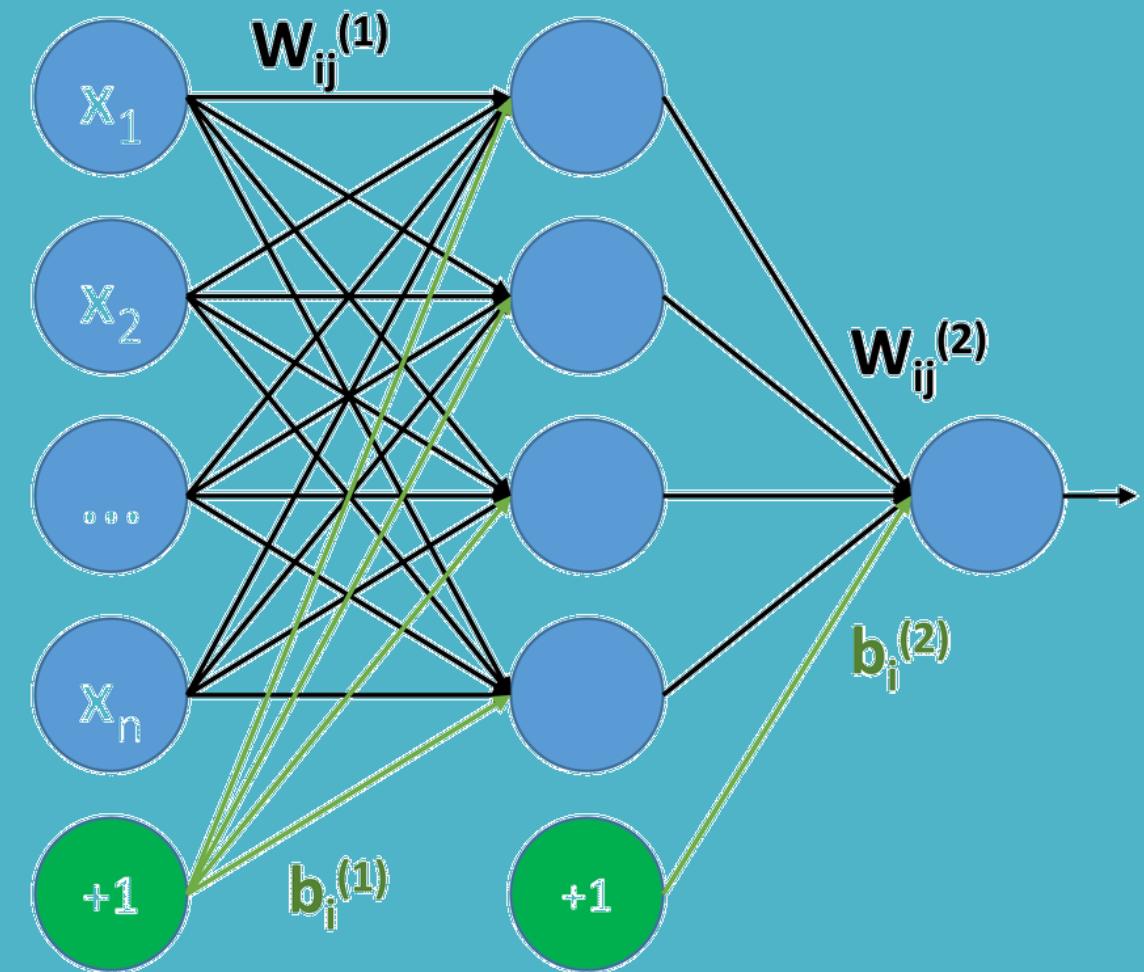
Poids (2)

- Dans les réseaux neuronaux profonds avec de nombreuses couches, le nombre de poids peut rapidement devenir très important, ce qui rend difficile l'optimisation efficace du réseau.
- Les techniques de régularisation, telles que la régularisation L1 et L2, l'abandon et la perte de poids, peuvent être utilisées pour prévenir le surajustement et améliorer les performances de généralisation du réseau.
- Dans les réseaux de neurones convolutifs, les poids sont généralement disposés dans un tenseur à 3 dimensions, où chaque tranche correspond à une seule carte d'entités dans la sortie d'une couche convolutionnelle.
- Dans les réseaux neuronaux récurrents, les poids sont utilisés pour propager l'état caché du réseau à travers les pas de temps, permettant au réseau de modéliser des séquences de longueur variable.
- Les poids d'un réseau neuronal peuvent être visualisés et analysés de différentes manières, par exemple en traçant leurs distributions, en visualisant leurs représentations apprises ou en calculant leurs scores d'importance à l'aide de techniques telles que les cartes de saillance ou l'importance des caractéristiques.

[



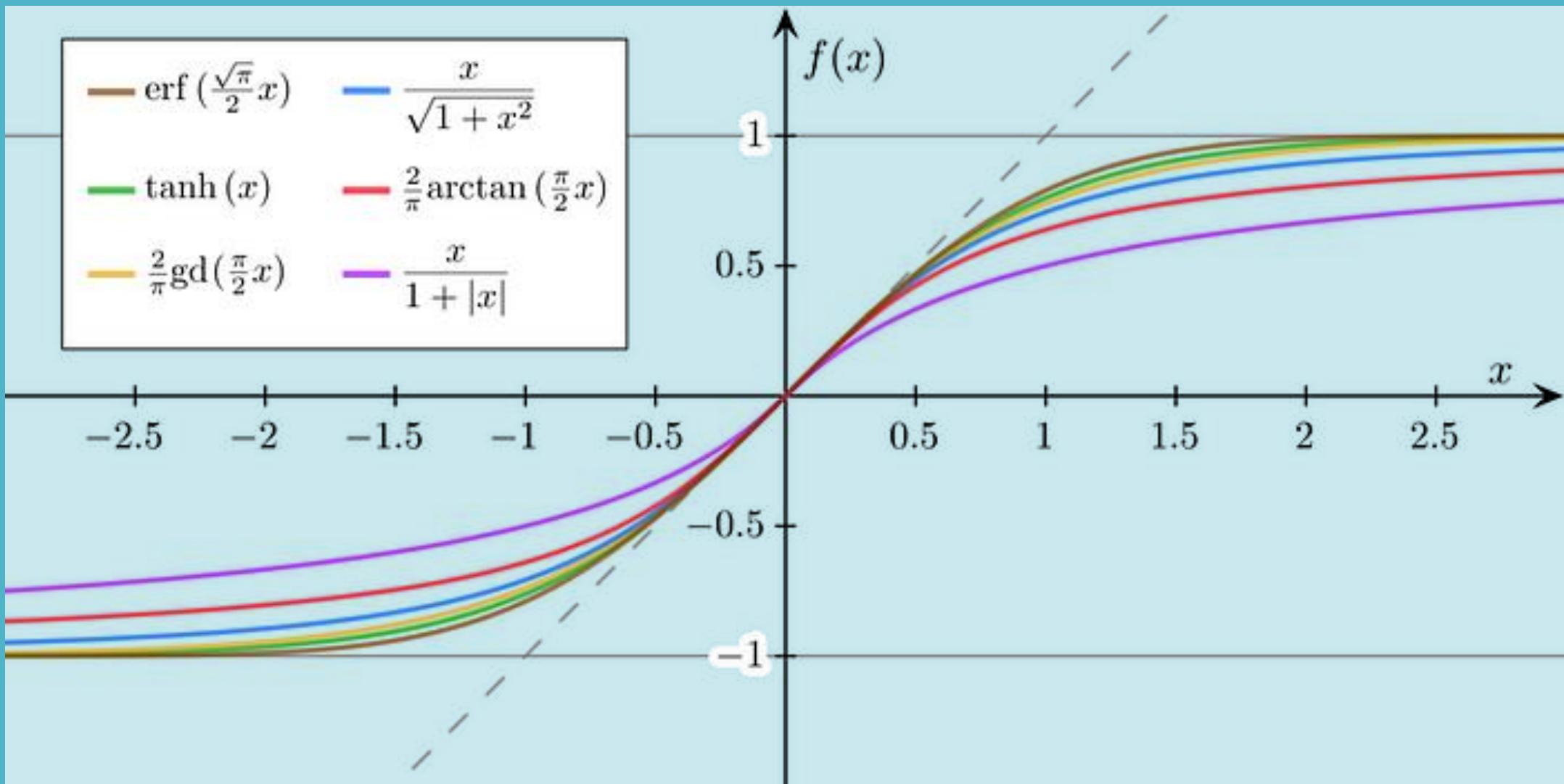
Layer 1 Layer 2 Layer 3



Fonctions d'activation

- Les fonctions d'activation sont des fonctions mathématiques qui introduisent la non-linéarité dans un réseau neuronal.
- Ils transforment la somme pondérée des entrées plus le terme de biais en une sortie qui peut être utilisée comme entrée à la couche suivante de neurones ou comme sortie finale du réseau neuronal.
- Les fonctions d'activation permettent au réseau neuronal d'apprendre des relations complexes et non linéaires entre l'entrée et la sortie.
- Le choix de la fonction d'activation dépend du type de problème résolu et de l'architecture du réseau neuronal.
- Certaines fonctions d'activation couramment utilisées incluent sigmoïde, ReLU et tanh.
- La fonction sigmoïde mappe l'entrée à une valeur comprise entre 0 et 1 et est utilisée dans les problèmes de classification binaire.
- La fonction ReLU renvoie l'entrée si elle est positive et 0 si elle est négative, et est largement utilisée dans l'apprentissage profond.
- La fonction tanh est similaire à la fonction sigmoïde mais mappe l'entrée à une valeur comprise entre -1 et 1, et est couramment utilisée dans les réseaux neuronaux récurrents.
- Les fonctions d'activation peuvent être modifiées ou personnalisées pour répondre aux exigences d'un problème spécifique ou d'une architecture réseau.
- Choisir la bonne fonction d'activation est important pour obtenir de bonnes performances et une bonne convergence dans un réseau neuronal.

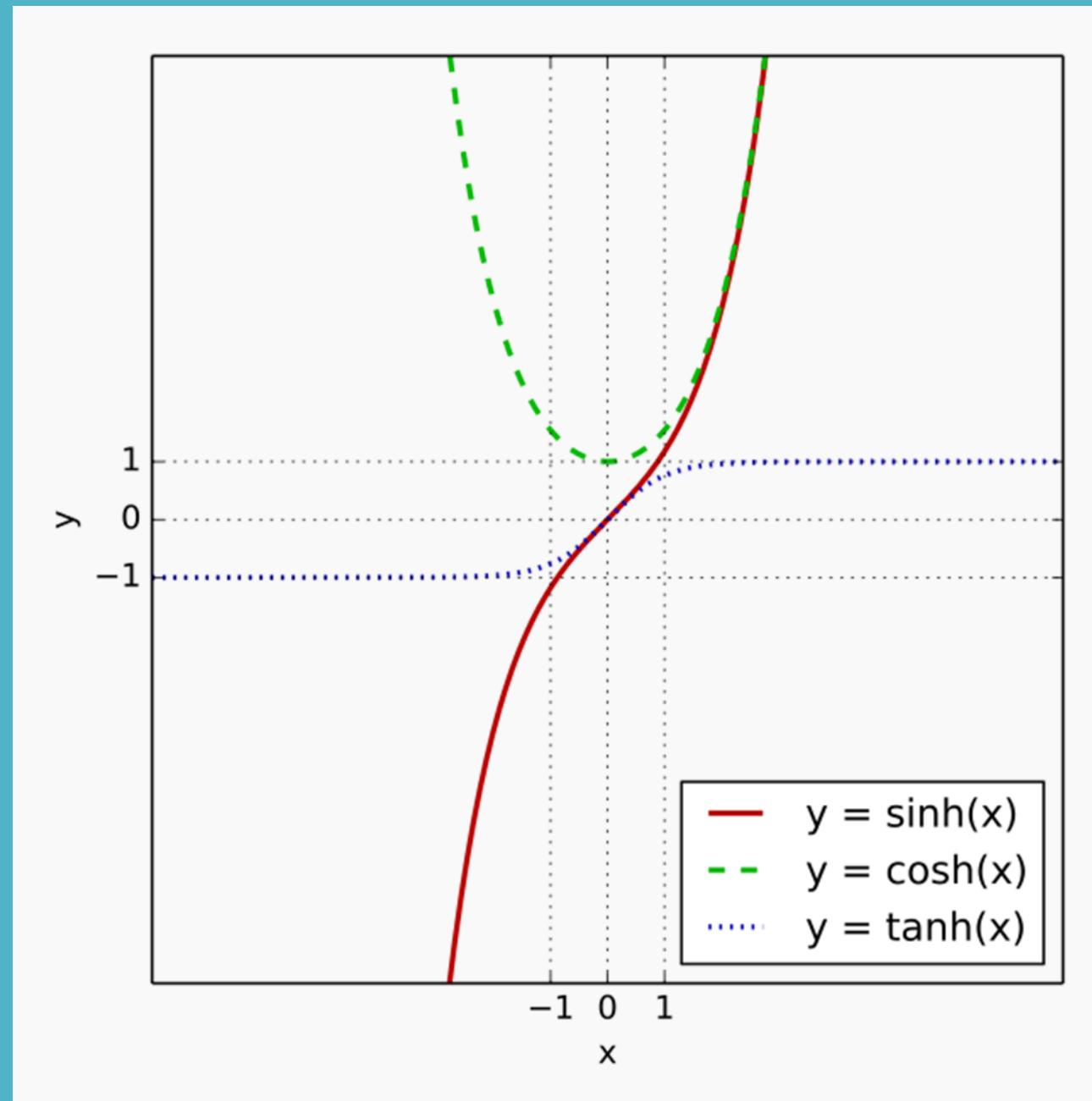
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



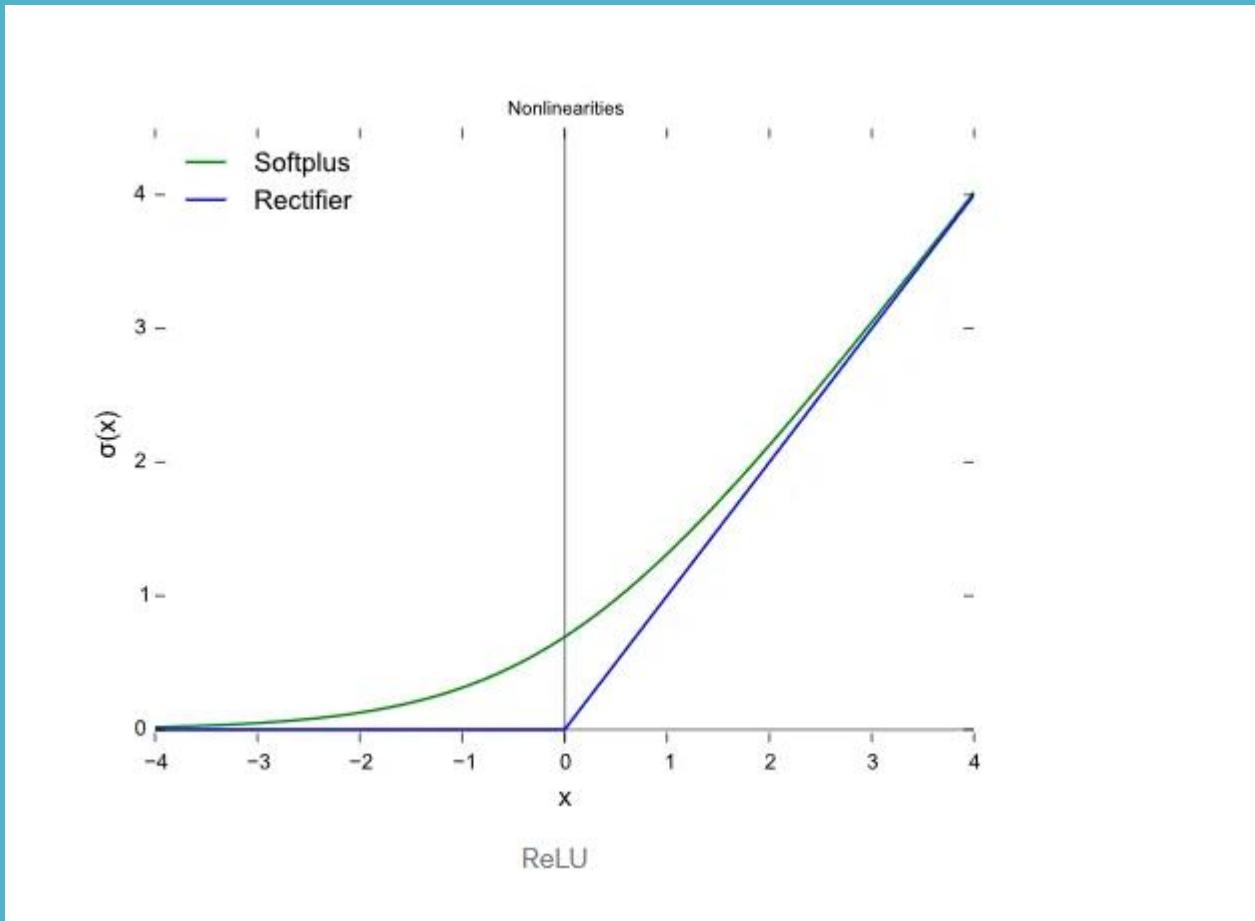
$$f'(x) = f(x)(1 - f(x)) \quad f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic tangent (tanh)

$$\frac{e^{2x} - 1}{e^{2x} + 1}$$

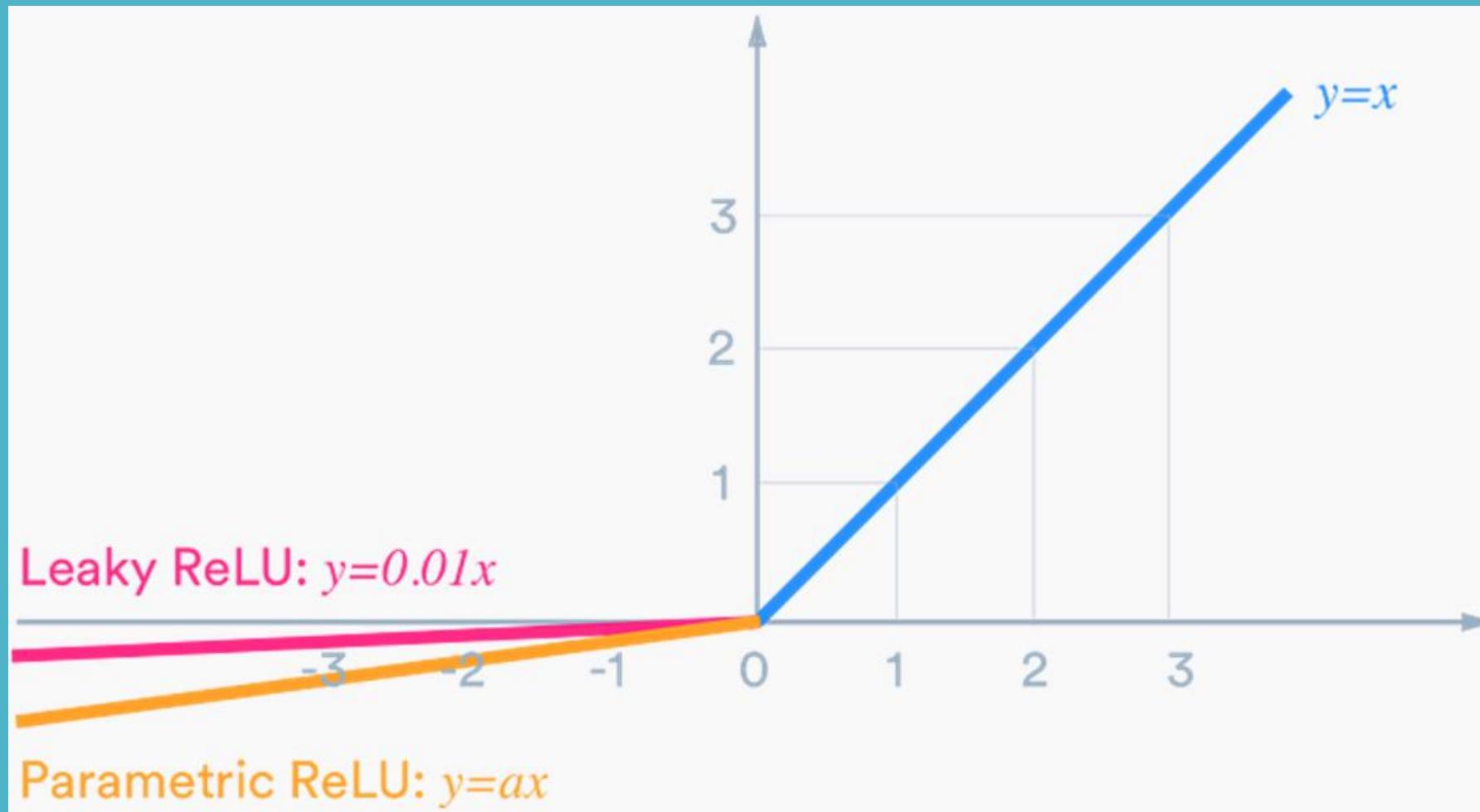


ReLU and Softplus

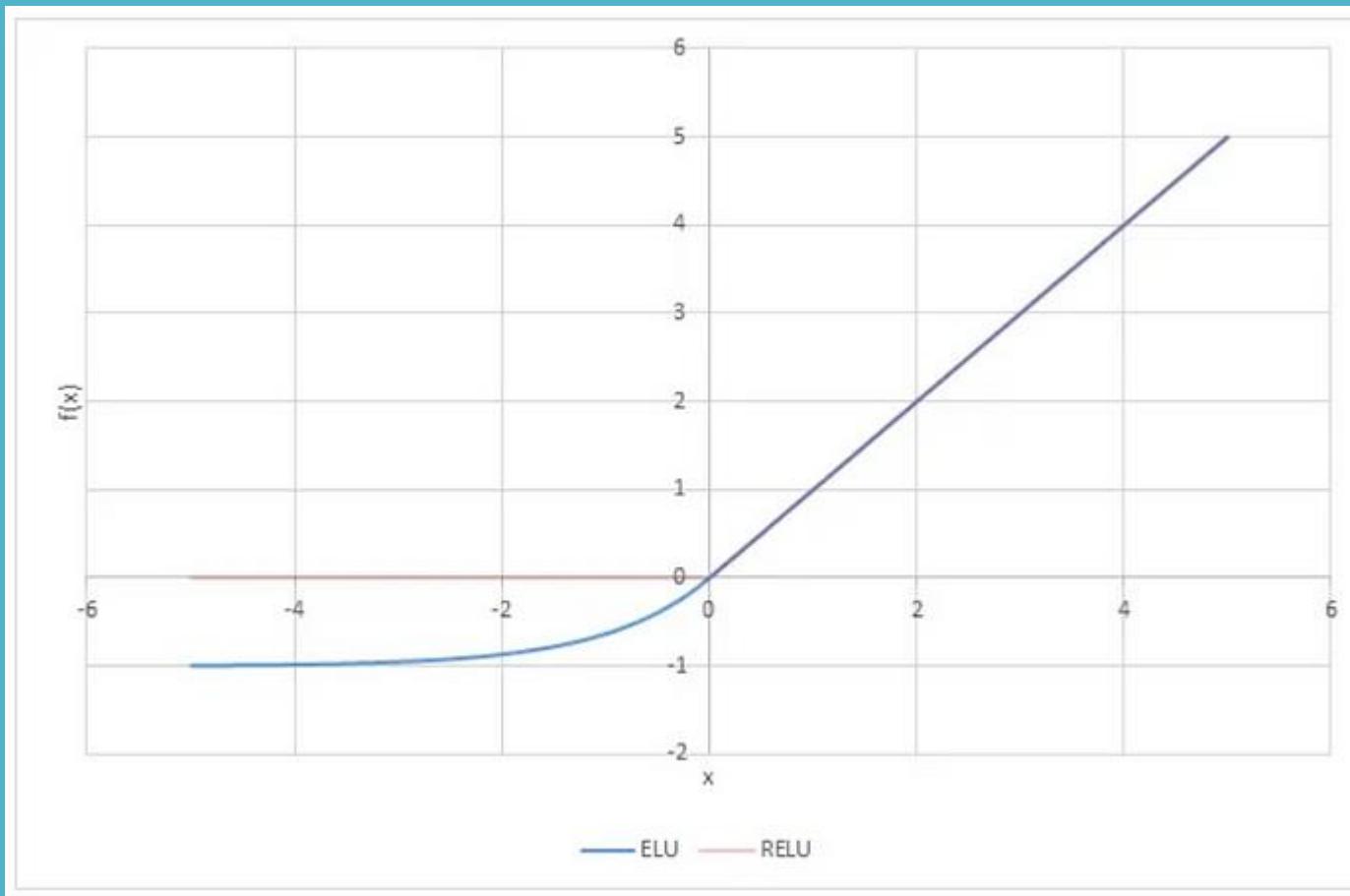


$$f(x) = \max(x, 0)$$

Leaky ReLu

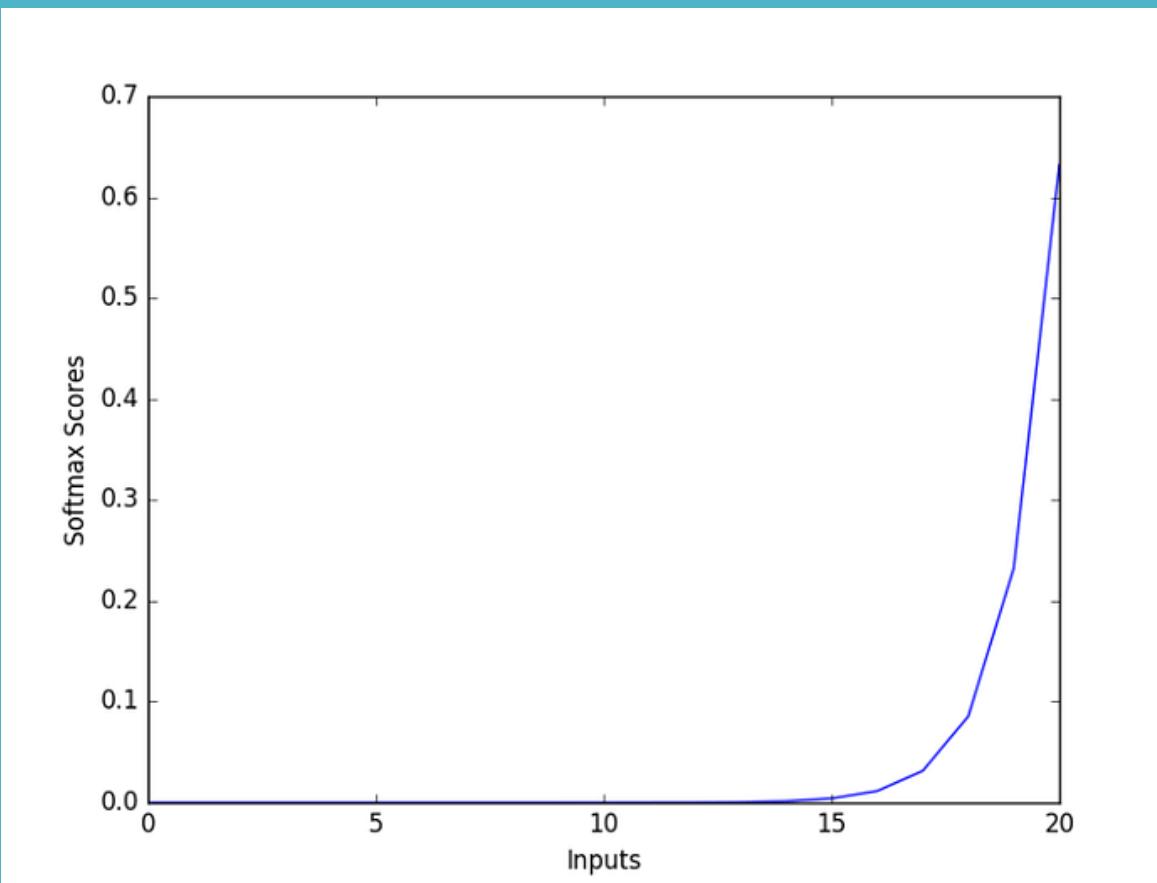


ELU Exponential LU



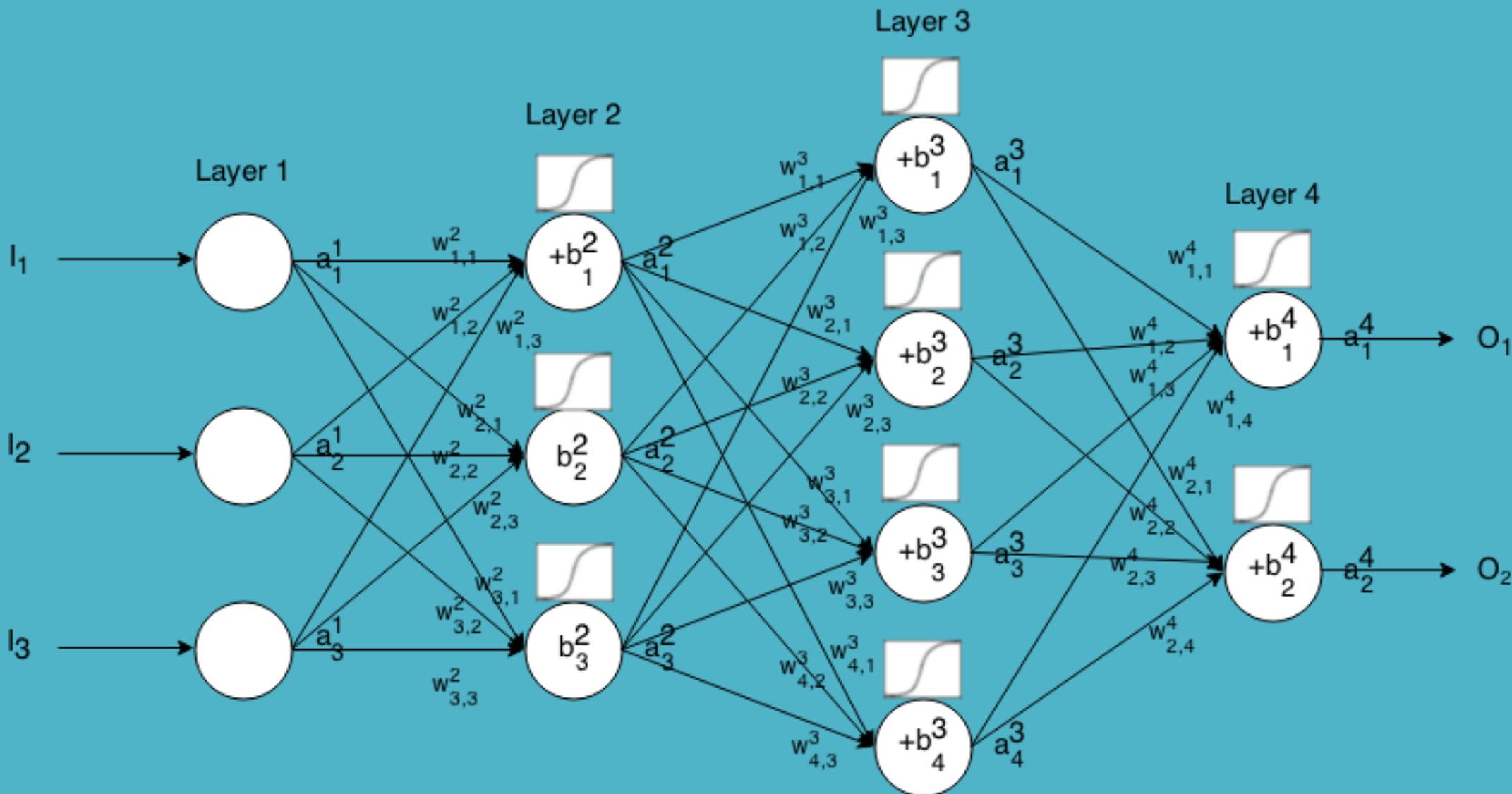
$$f(x) = \begin{cases} x & \text{if } x > 0, \\ a(e^x - 1) & \text{otherwise,} \end{cases}$$

Softmax



(for output layer)

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$



Fonctions de coût (1)

- Les fonctions de coût ou de perte sont utilisées pour mesurer la différence entre la sortie prévue d'un réseau neuronal et la sortie réelle pour une entrée donnée.
- L'objectif d'un réseau neuronal est de minimiser le coût ou la fonction de perte pour améliorer la précision des prédictions.
- Le choix de la fonction de coût ou de perte dépend du type de problème résolu et de la sortie du réseau neuronal.
- Certaines fonctions de coût ou de perte couramment utilisées incluent l'erreur quadratique moyenne, l'entropie croisée et l'entropie croisée binaire.
- Les fonctions de coût ou de perte peuvent être modifiées ou personnalisées pour répondre aux exigences d'un problème spécifique ou d'une architecture réseau.
- Choisir la bonne fonction de coût ou de perte est important pour obtenir de bonnes performances et une bonne convergence dans un réseau neuronal.
- En plus de la fonction de coût ou de perte, les réseaux neuronaux utilisent également des algorithmes d'optimisation tels que la descente de gradient stochastique pour mettre à jour les poids et les biais du réseau pendant l'entraînement.

Fonctions de coût (2)

- L'erreur quadratique moyenne est utilisée pour les problèmes de régression, où la sortie du réseau neuronal est une valeur continue.
- L'entropie croisée et l'entropie croisée binaire sont utilisées pour les problèmes de classification, où la sortie du réseau neuronal représente la probabilité que l'entrée appartienne à une classe particulière.
- La fonction d'entropie croisée mesure la différence entre la sortie prédictive et la sortie réelle en termes de probabilités des classes correctes et incorrectes.
- L'entropie croisée binaire est similaire à l'entropie croisée mais est utilisée pour les problèmes de classification binaire.

R Means Square Error

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where n is the number of data points, y_i is the actual output for the i^{th} input, and \hat{y}_i is the predicted output for the i^{th} input.

Cross Entropy

$$\text{Cross-entropy} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij})$$

where n is the number of data points, C is the number of classes,
 y_{ij} is the actual output for the i^{th} input and j^{th} class,
and \hat{y}_{ij} is the predicted probability of the input belonging to the j^{th} class.

Binary Cross Entropy

$$\text{Binary cross-entropy} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where n is the number of data points, y_i is the actual output for the i^{th} input,
and \hat{y}_i is the predicted probability of the input belonging to the positive class.

Feed-forward (1)

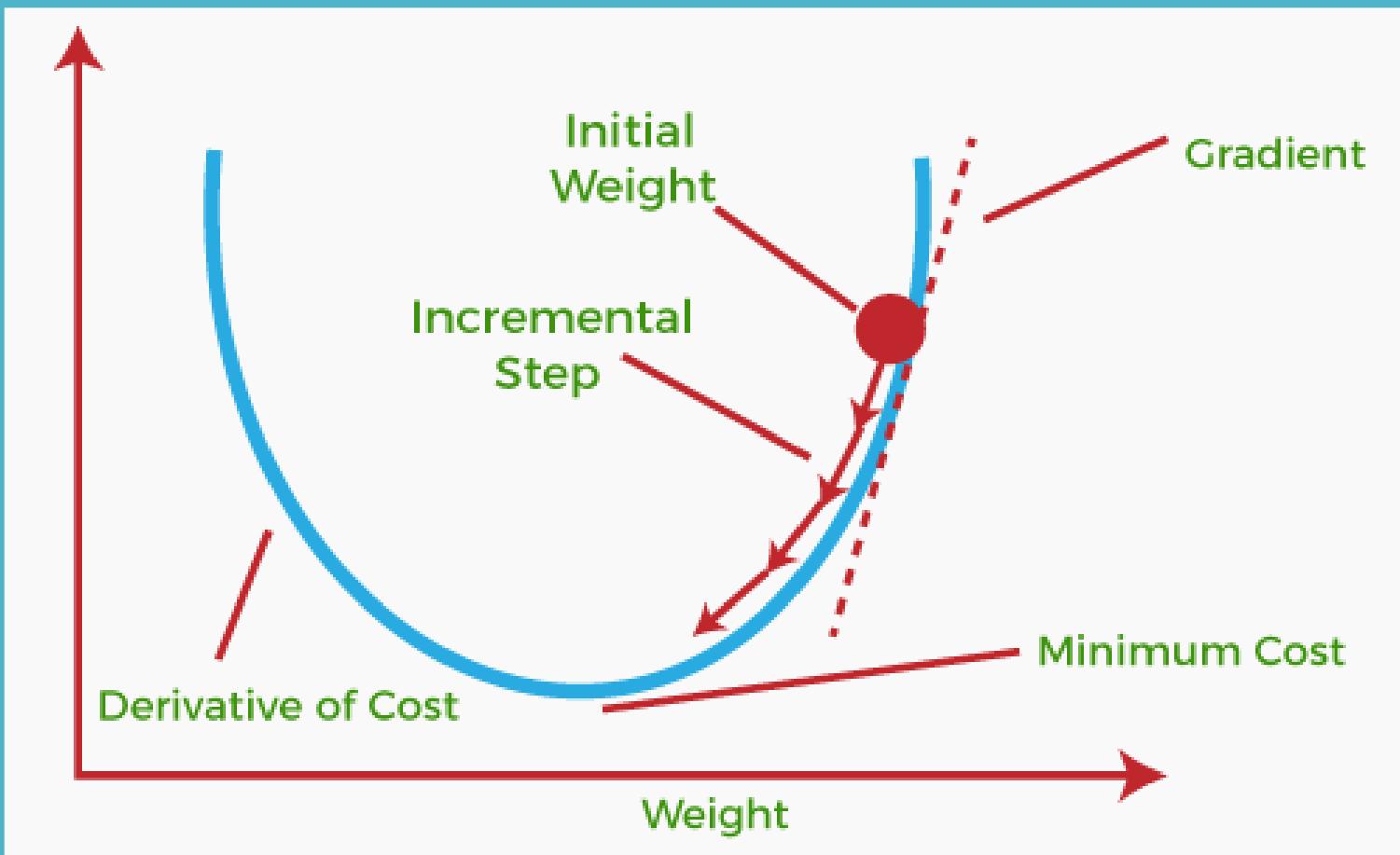
- Feed-forward est le type le plus élémentaire d'architecture de réseau neuronal, également connu sous le nom de perceptron multicouche (MLP).
- Dans un réseau neuronal feed-forward, l'information circule uniquement dans une direction, de la couche d'entrée à la couche de sortie à travers une ou plusieurs couches cachées.
- Chaque neurone du réseau est connecté à tous les neurones de la couche précédente, et les poids sur ces connexions sont appris pendant l'entraînement.
- La sortie d'un neurone dans un réseau feed-forward est déterminée par la somme pondérée de ses entrées, passées par une fonction d'activation.
- La sortie de chaque neurone d'une couche sert d'entrée à la couche suivante jusqu'à ce que la couche de sortie soit atteinte, ce qui produit la sortie finale du réseau.
- Les réseaux de feedforward peuvent être utilisés à la fois pour les problèmes de régression et de classification, selon le type de sortie produit par la couche finale.

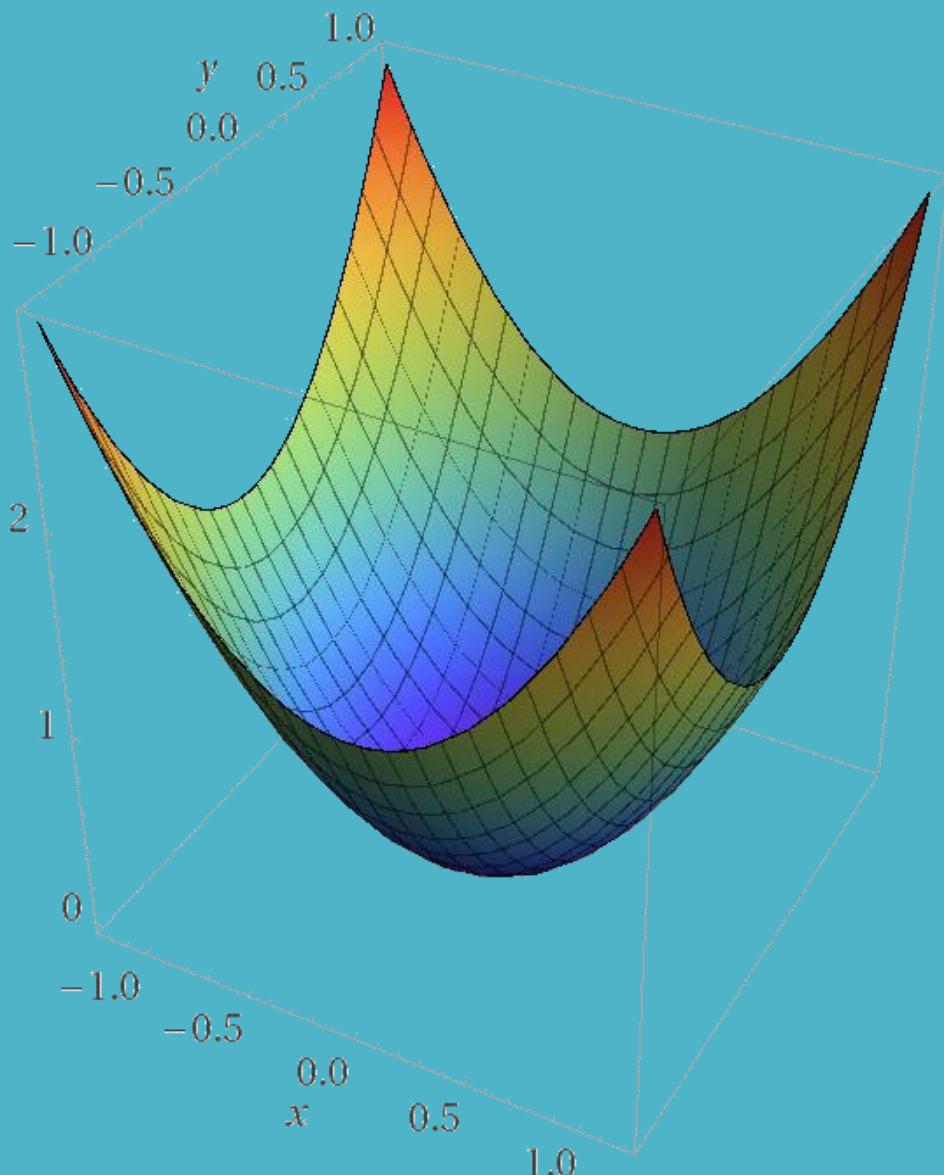
Feed-forward (2)

- L'entraînement d'un réseau de feed-forward implique d'ajuster les poids pour minimiser l'erreur entre la sortie prévue et la sortie réelle, en utilisant des techniques telles que la rétropropagation.
- L'architecture et les hyperparamètres d'un réseau feed-forward, tels que le nombre de couches, le nombre de neurones par couche et le choix de la fonction d'activation, peuvent être réglés pour obtenir des performances optimales sur une tâche donnée.
- Malgré leur simplicité, les réseaux de feed-forward peuvent toujours être efficaces pour résoudre des problèmes complexes, mais peuvent nécessiter de plus grandes quantités de données d'apprentissage et des temps de formation plus longs par rapport à d'autres architectures de réseaux neuronaux.
- Les réseaux Feed-forward sont encore largement utilisés dans diverses applications et servent de base à des architectures de réseaux neuronaux plus avancées telles que les réseaux neuronaux convolutifs et les réseaux neuronaux récurrents.

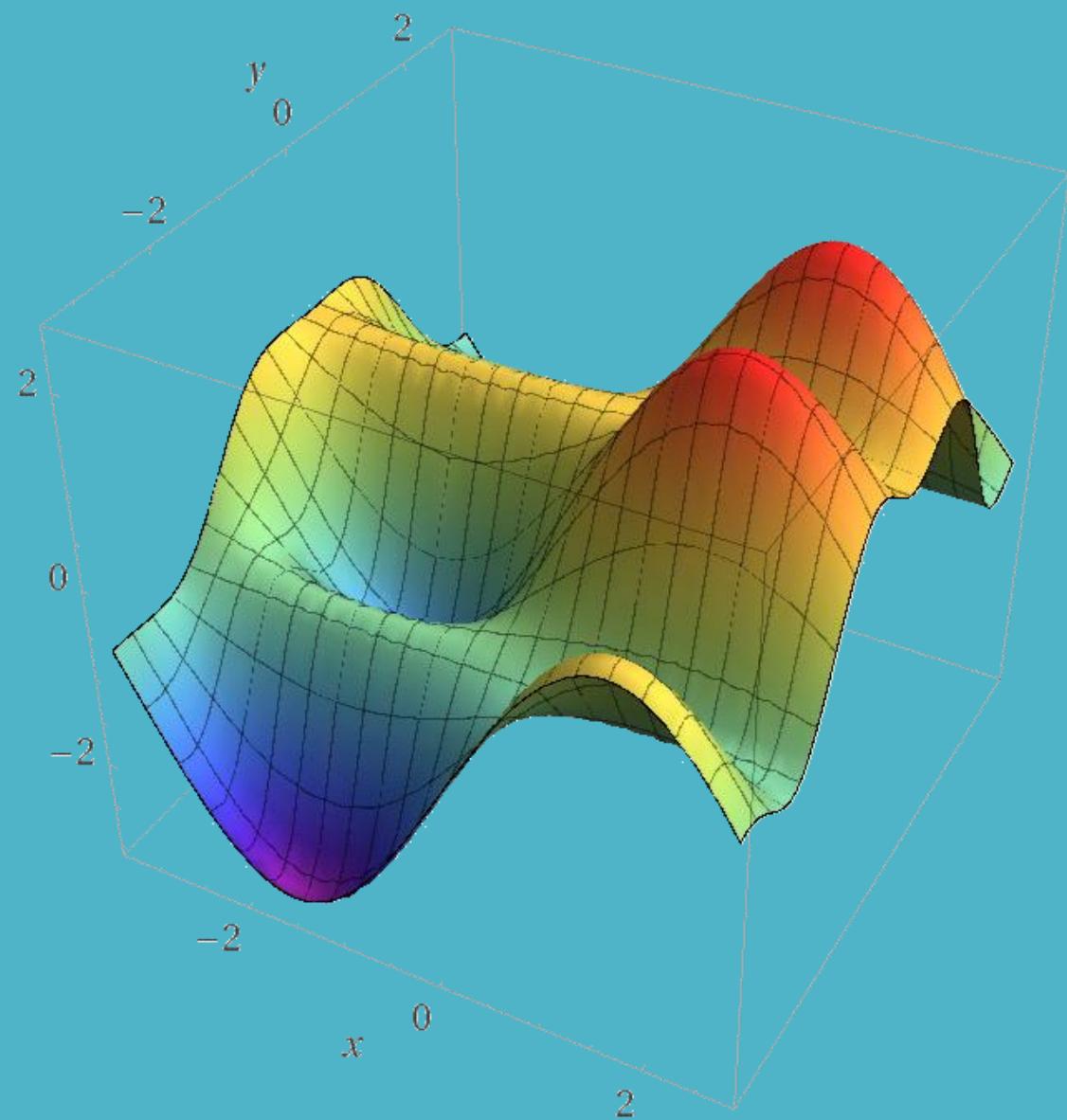
Descente de pente

- La descente de gradient est un algorithme d'optimisation utilisé pour trouver le minimum d'une fonction en ajustant itérativement les paramètres d'entrée dans la direction de la descente la plus raide du gradient.
- Dans les réseaux de neurones, la descente de gradient est utilisée pour mettre à jour le poids des connexions entre les neurones pendant l'entraînement, afin de minimiser la fonction d'erreur ou de perte.
- Il existe deux principaux types de descente de gradient : la descente de gradient par lots et la descente de gradient stochastique (SGD).
- La descente de gradient par lots calcule le gradient de la fonction de perte sur l'ensemble du jeu de données d'apprentissage avant de mettre à jour les poids, tandis que SGD met à jour les poids après avoir calculé le gradient sur un seul exemple d'entraînement sélectionné au hasard.
- La descente de gradient par mini-lots est un compromis entre la descente de gradient par lots et SGD, où le gradient est calculé sur un petit lot d'exemples d'entraînement sélectionnés au hasard avant de mettre à jour les poids.
- La descente de gradient peut souffrir du problème de rester coincé dans les minima locaux ou les points de selle, et diverses modifications ont été proposées pour atténuer cela, telles que l'élan, les taux d'apprentissage adaptatif et les méthodes de second ordre.

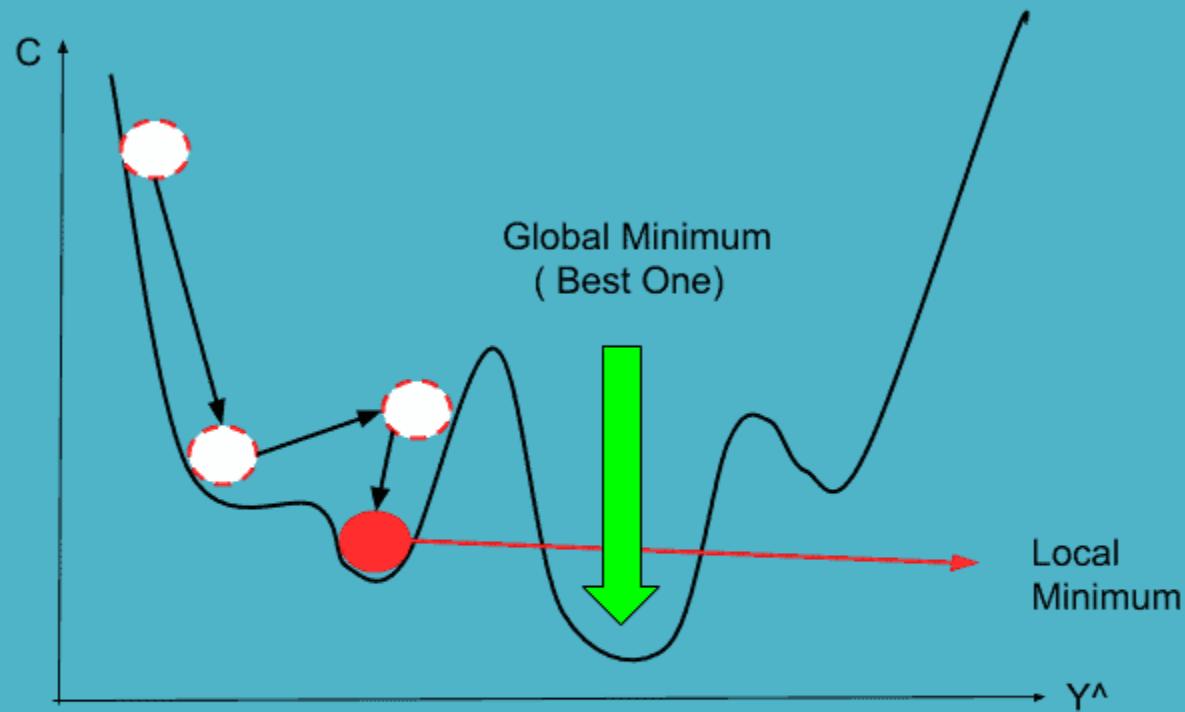




Computed by Wolfram Alpha



These are Digital Bridge Original and Proprietary Materials - Do
not use without explicit authorization



Gradient descent for MSE

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j) \mathbf{x}_{ij}$$

$$\nabla_{\mathbf{w}_i} \text{MSE} = -\frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j) \mathbf{x}_{ij}$$

Gradient Descent for Cross Entropy

$$w_{i+1} = w_i - \alpha \frac{1}{N} \sum_{j=1}^N \sum_{k=1}^C (y_{jk} - \hat{y}_{jk}) x_{ij}$$

$$\nabla_{w_i} \text{Cross-entropy} = -\frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j) \mathbf{x}_j$$

Gradient descent for Binary Cross Entropy

$$w_{i+1} = w_i - \alpha \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j) x_{ij}$$

$$\nabla_{w_i} \text{Binary cross-entropy} = -\frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j) x_{ij}$$

Autres optimiseurs

- D'autres algorithmes d'optimisation ont été proposés pour améliorer les limitations de la descente de gradient, tels que Adagrad, Adadelta, RMSprop et Adam.
- Ces optimiseurs utilisent des taux d'apprentissage adaptatif qui ajustent la taille des pas de la règle de mise à jour en fonction de l'historique des valeurs de dégradé, afin de converger plus rapidement et d'éviter les oscillations.
- Adagrad utilise un taux d'apprentissage par paramètre qui diminue au fil du temps pour les paramètres qui reçoivent des mises à jour fréquentes, tandis qu'Adadelta et RMSprop utilisent des moyennes mobiles à décroissance exponentielle des dégradés pour normaliser l'étape de mise à jour.
- Adam est un optimiseur populaire qui combine les avantages de la quantité de mouvement et des taux d'apprentissage adaptatif, et utilise la correction des biais pour tenir compte des estimations initiales des gradients.
- Choisir le bon optimiseur et régler ses hyperparamètres peut avoir un impact significatif sur les performances et la vitesse de convergence d'un réseau neuronal, et constitue un aspect important de la formation de modèles d'apprentissage profond.

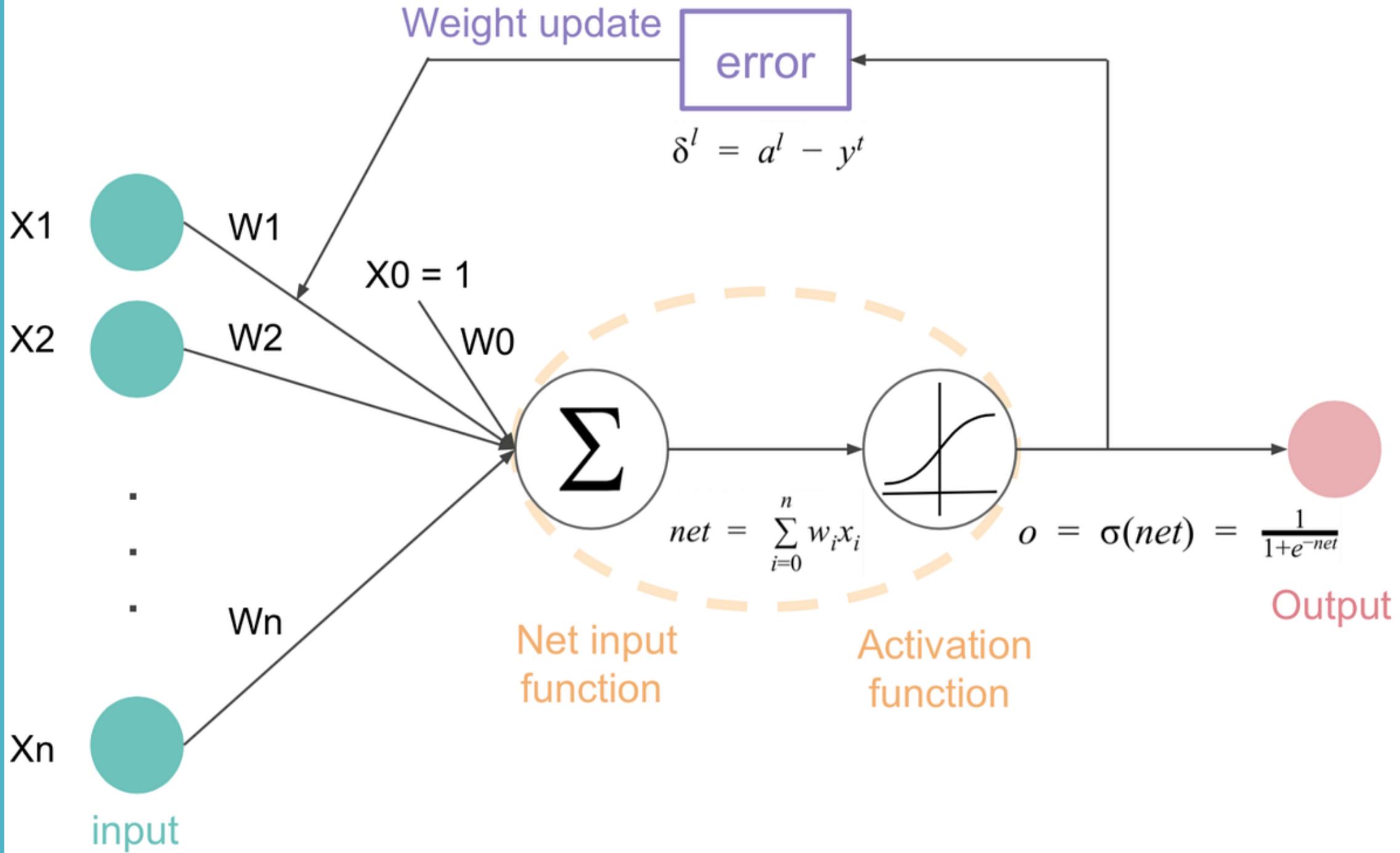
Name	Update Rule
SGD	$\Delta\theta_t = -\alpha g_t$
Momentum	$m_t = \gamma m_{t-1} + (1 - \gamma)g_t,$ $\Delta\theta_t = -\alpha m_t$
Adagrad	$G_t = G_{t-1} + g_t^2,$ $\Delta\theta_t = -\alpha g_t G_t^{-1/2}$
Adadelta	$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2} D_{t-1}^{1/2},$ $D_t = \beta_1 D_{t-1} + (1 - \beta_1)(\Delta\theta_t/\alpha)^2$
RMSprop	$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2}$
Adam	$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t,$ $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2,$ $\hat{m}_t = m_t / (1 - \beta_1^t),$ $\hat{v}_t = v_t / (1 - \beta_2^t),$ $\Delta\theta_t = -\alpha \hat{m}_t \hat{v}_t^{-1/2}$

Rétropropagation (1)

- La rétropropagation est un algorithme utilisé pour former des réseaux neuronaux artificiels, en particulier des réseaux neuronaux feedforward.
- Il est basé sur l'idée de propager l'erreur ou la perte vers l'arrière à travers le réseau, de la couche de sortie à la couche d'entrée, afin d'ajuster les poids des connexions et de minimiser l'erreur.
- L'algorithme est basé sur la règle de chaîne du calcul, qui permet de calculer récursivement le gradient de la fonction de perte par rapport à chaque poids.
- Pendant la passe directe, les données d'entrée sont introduites dans le réseau et la sortie est calculée en propageant les activations de chaque couche à travers le réseau.
- Pendant la passe en arrière, l'erreur ou la perte entre la sortie prévue et la sortie réelle est calculée, puis propagée vers l'arrière à travers les couches du réseau.

Rétropropagation (2)

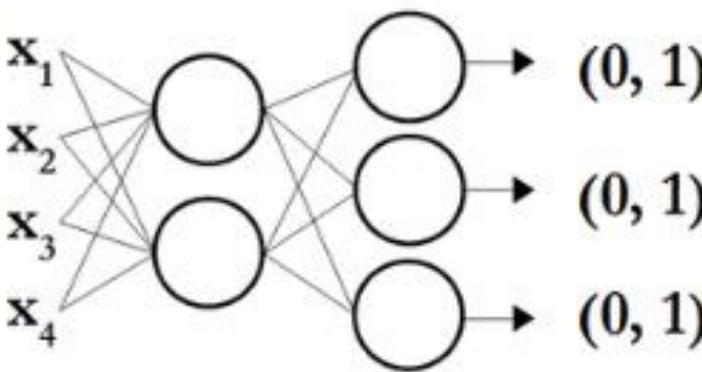
- Les gradients de la fonction de perte par rapport aux poids de chaque couche sont calculés à l'aide de la règle de chaîne, puis utilisés pour mettre à jour les poids par un processus connu sous le nom de descente de gradient.
- La rétropropagation peut être utilisée avec une variété de fonctions de perte, y compris l'erreur quadratique moyenne, l'entropie croisée et l'entropie croisée binaire.
- L'algorithme peut être sujet à un surajustement si le réseau est trop grand ou si les données d'entraînement sont limitées, et diverses techniques de régularisation telles que l'abandon et la perte de poids peuvent être utilisées pour atténuer cela.
- La rétropropagation peut être étendue pour fonctionner avec d'autres types de réseaux neuronaux, tels que les réseaux de neurones convolutifs et les réseaux de neurones récurrents, en modifiant les calculs des gradients pour les architectures respectives.
- Malgré ses limites, la rétropropagation reste l'un des algorithmes les plus utilisés pour la formation de réseaux de neurones et a joué un rôle déterminant dans le succès de l'apprentissage profond.



Règle de chaîne pour la rétropropagation

- La règle de chaîne est un concept fondamental en calcul qui permet de calculer la dérivée d'une fonction composite en termes de dérivées de ses composants individuels.
- Dans le contexte de la rétropropagation, la règle de la chaîne est utilisée pour calculer le gradient de la fonction de perte par rapport aux poids de chaque neurone du réseau, en multipliant récursivement les gradients de chaque couche par les poids des connexions entre elles.
- Plus précisément, la règle de la chaîne stipule que la dérivée de la composition de deux fonctions, f et g , peut être calculée comme le produit de leurs dérivées individuelles, c'est-à-dire $(f \circ g)' = (f' \circ g) * g'$.
- Dans le cas des réseaux de neurones, la sortie de chaque neurone d'une couche est fonction de la somme pondérée des sorties de la couche précédente, passées par une fonction d'activation.
- Au cours de la rétropropagation, le gradient de la fonction de perte par rapport à la sortie de chaque neurone est calculé en premier, puis le gradient de la fonction de perte par rapport à l'entrée pondérée de chaque neurone est calculé en multipliant le gradient précédent par la dérivée de la fonction d'activation.
- Les gradients de la fonction de perte par rapport aux poids de chaque neurone d'une couche peuvent ensuite être calculés en multipliant le gradient de l'entrée pondérée par la sortie de la couche précédente, en utilisant la règle de la chaîne.
- La rétropropagation parcourt les couches du réseau dans l'ordre inverse, en commençant par la couche de sortie et en reculant vers la couche d'entrée, en calculant les gradients et les mises à jour de poids à chaque étape.
- La règle de chaîne permet un calcul efficace des gradients dans des réseaux complexes avec plusieurs couches et fonctions d'activation, et est un élément clé de nombreux algorithmes d'apprentissage profond.

Feedforward



$$\begin{aligned}H_{in} &= XW_1 \\H_{out} &= \text{sigmoid}(H_{in}) \\O_{in} &= H_{out}W_2 \\O_{out} &= \text{sigmoid}(O_{in})\end{aligned}$$

Cost function/Loss function:

$$\text{Mean Squared Error} = \frac{1}{2N} \sum_e \sum_n (O_{out,ei} - Y_{ei})^2$$

Gradient Descent:

$$W_i := W_i - \alpha \cdot \frac{\partial \text{MSE}}{\partial W_i}(W)$$

Chain Rule:

$$\frac{\partial Z}{\partial X} = \frac{\partial Z}{\partial Y} \cdot \frac{\partial Y}{\partial X}$$

Backpropagation

$$\frac{\partial \text{MSE}}{\partial W_1}(W_1) = \frac{\partial \text{MSE}}{\partial O_{out}} \frac{\partial O_{out}}{\partial O_{in}} \frac{\partial O_{in}}{\partial H_{out}} \frac{\partial H_{out}}{\partial H_{in}} \frac{\partial H_{in}}{\partial W_1}$$

$$\frac{\partial \text{MSE}}{\partial W_2}(W_2) = \frac{\partial \text{MSE}}{\partial O_{out}} \frac{\partial O_{out}}{\partial O_{in}} \frac{\partial O_{in}}{\partial W_2}$$

Model training

- **NEXT**

Model evaluation

- NEXT

Model prediction

- NEXT

Practical examples

- NEXT

Saving and reusing a model

- **NEXT**