FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Réseaux de neurones pour l'apprentissage automatique

B. HILALI - brahill79@gmail.com / T. SABRI – sabritarik@gmail.com
Filière: Ingénierie Logicielle et Intégration des Systèmes Informatiques (ILISI) 24-25

Définition

- Définition : Un réseau de neurones artificiels (RNA) est un modèle inspiré du fonctionnement du cerveau humain, composé de neurones interconnectés capables d'apprendre à partir de données.

- Il est utilisé pour résoudre des problèmes complexes en apprentissage automatique, tels que la **classification**, la **régression** et la **génération de données**.

- Il est utilisé dans de nombreux domaines : vision par ordinateur, traitement du langage, reconnaissance vocale, etc.

Types de réseaux de neurones

- Types courants :

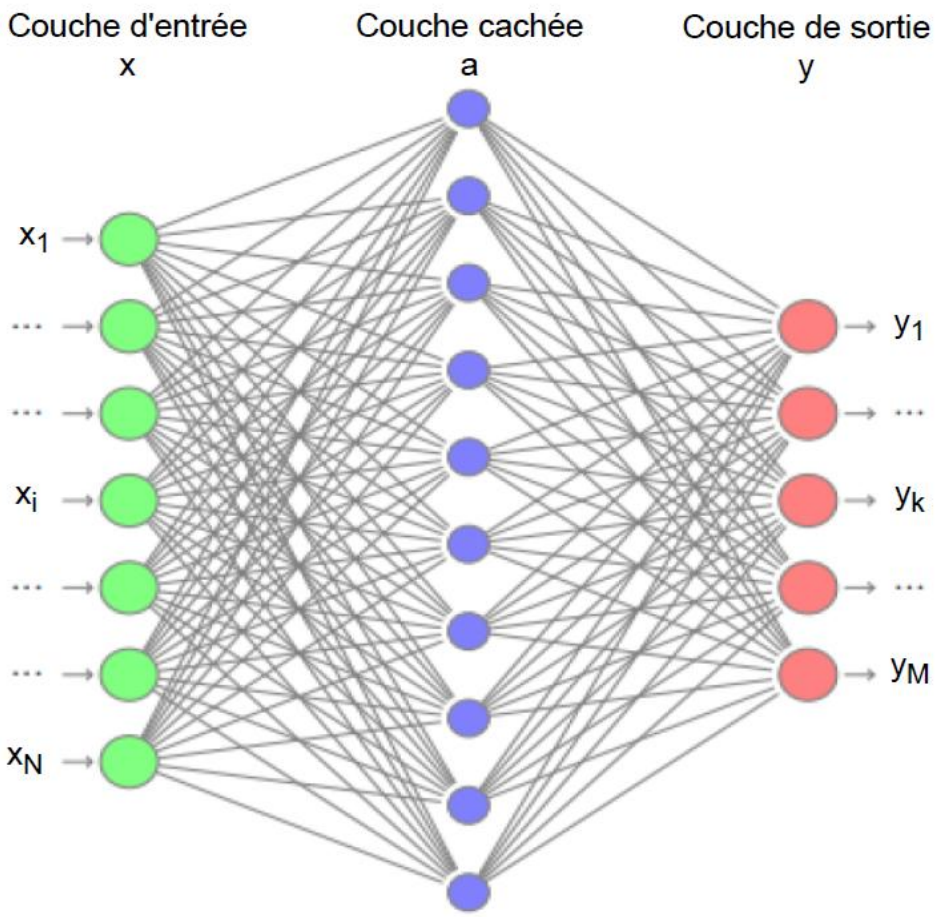
✓Perceptron multicouche (MLP) : Réseau feedforward (FFNN).
✓Réseaux convolutifs (CNN) : Pour le traitement d'images.
✓Réseaux récurrents (RNN) : Pour les séquences temporelles

Fonctionnement des Réseaux Feedforward (FFNN)

- Les réseaux feedforward (Feedforward Neural Networks, FFNN) sont les architectures les plus simples. Les données circulent uniquement dans une direction : de l'entrée vers la sortie, sans boucle ou retour en arrière.
- Les perceptrons multicouches (MLP) sont un type de réseau neuronal de type feedforward.

Perceptron multicouche (MLP)

- Couches :
  - ✓Entrée : reçoit les données.
  - ✓Cachées : appliquent des transformations non linéaires.
  - ✓Sortie : fournit la prédiction.



Couche d’entrée

La couche d'entrée se compose de nœuds ou de neurones qui reçoivent les données d'entrée initiales. Chaque neurone représente une caractéristique ou une dimension des données d'entrée. Le nombre de neurones dans la couche d'entrée est déterminé par la dimensionnalité des données d'entrée.

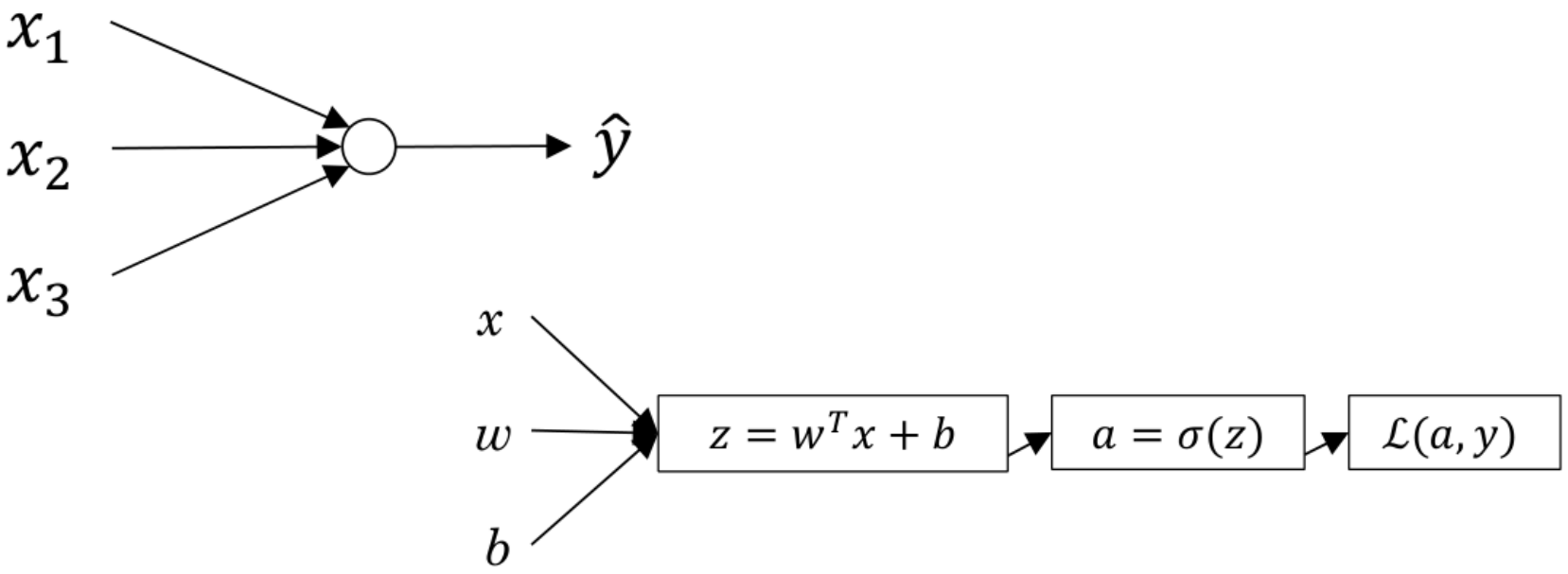
Couches cachées

- il peut y avoir une ou plusieurs couches cachées.
- Chaque neurone d'une couche cachée reçoit des entrées de tous les neurones de la couche précédente et produit une sortie qui est transmise à la couche suivante.
- Le nombre de couches cachées et le nombre de neurones dans chaque couche cachée sont des hyperparamètres qui doivent être déterminés pendant la phase de conception du modèle.

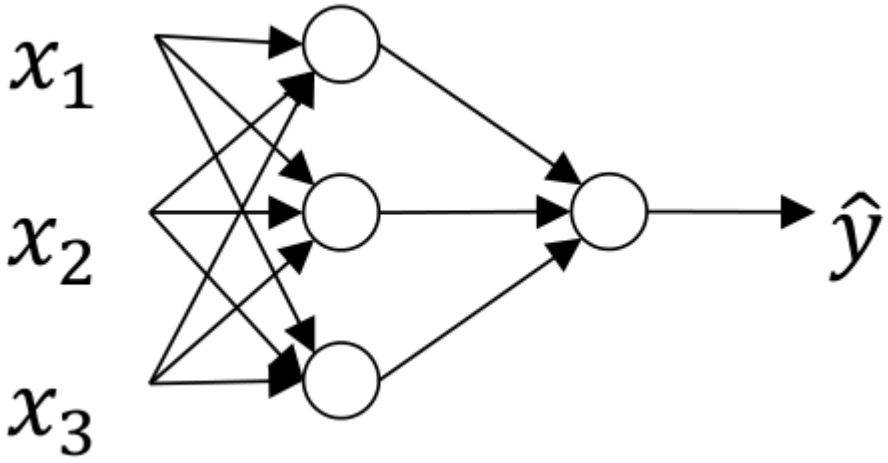
Couches de sorties

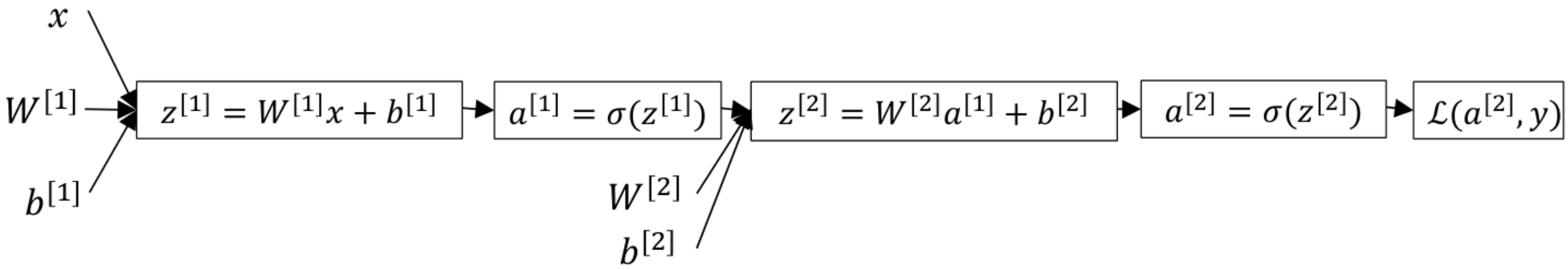
- Cette couche est constituée de neurones qui produisent la sortie finale du réseau.
- Le nombre de neurones dans la couche de sortie dépend de la nature de la tâche.

Réseau d’une seule couche

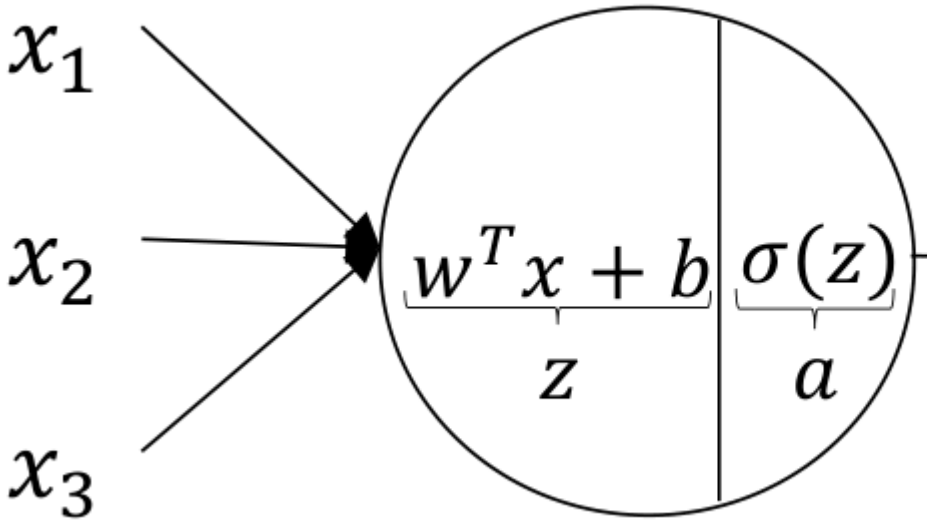


Réseau de plusieurs couches



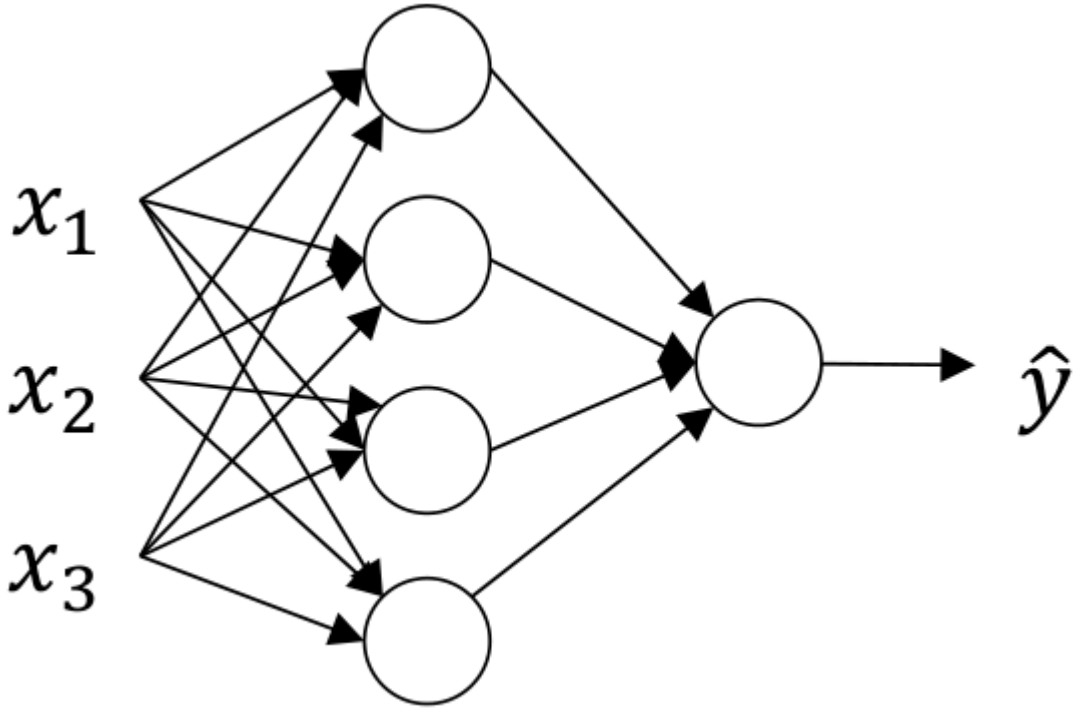


Réseau de plusieurs couches

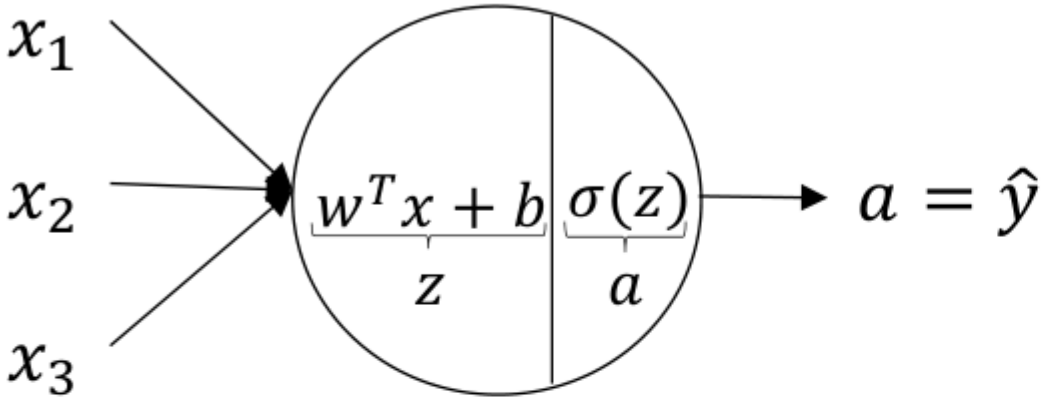


$$z = w^T x + b$$

$$a = \sigma(z)$$

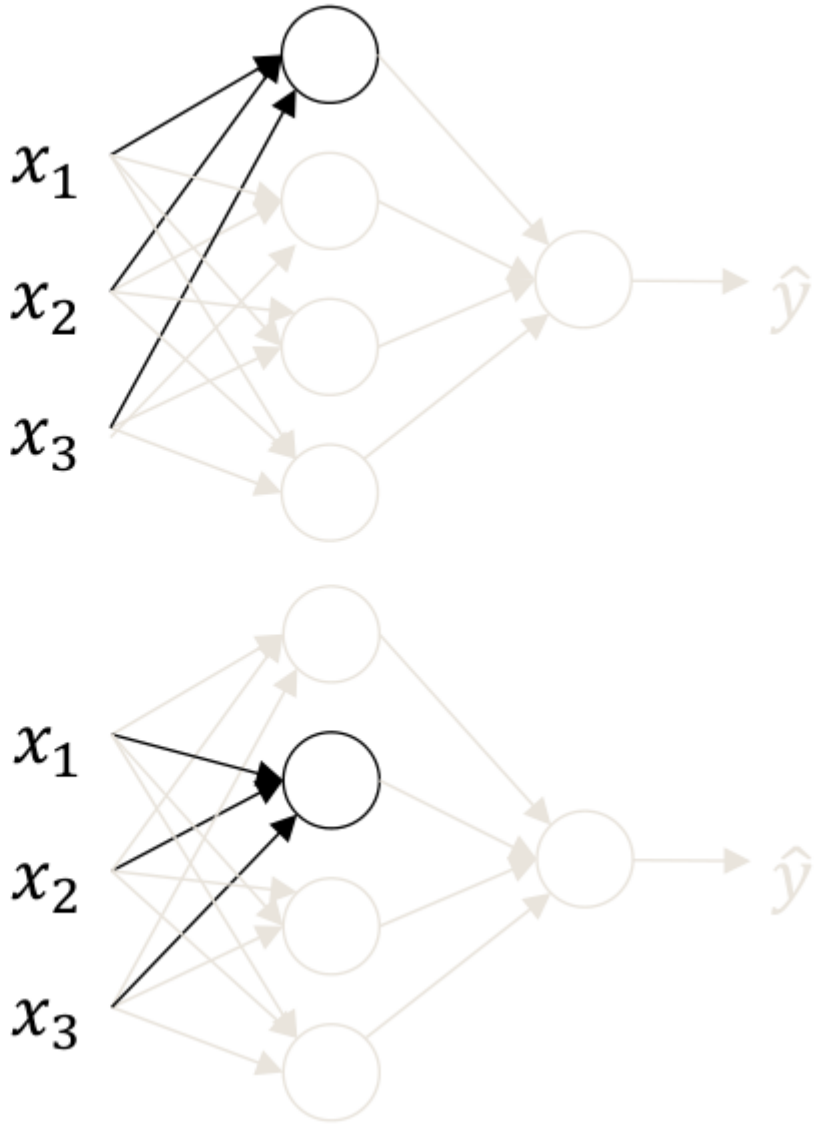


Réseau de plusieurs couches

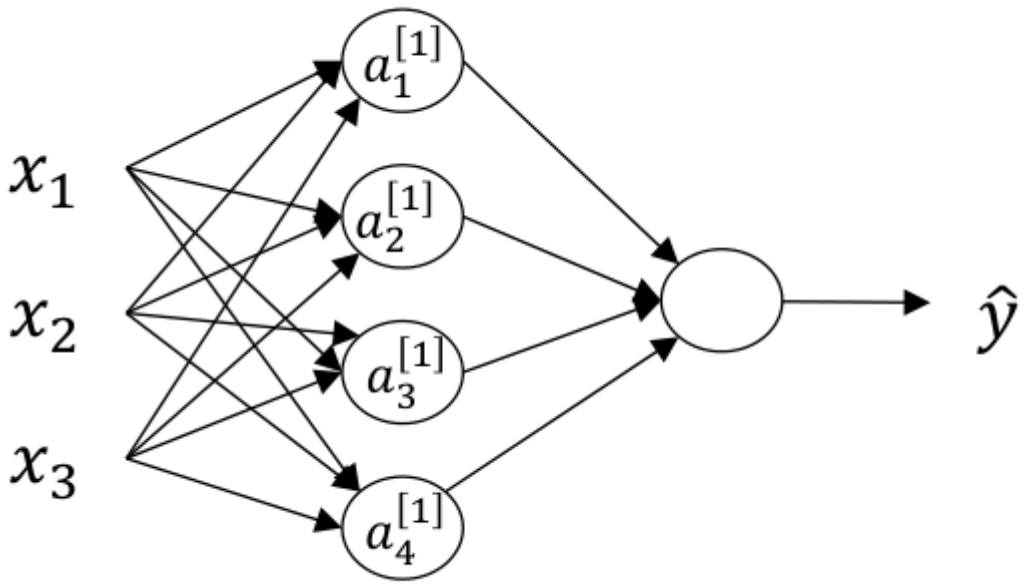


$$z = w^T x + b$$

$$a = \sigma(z)$$



Réseau de plusieurs couches



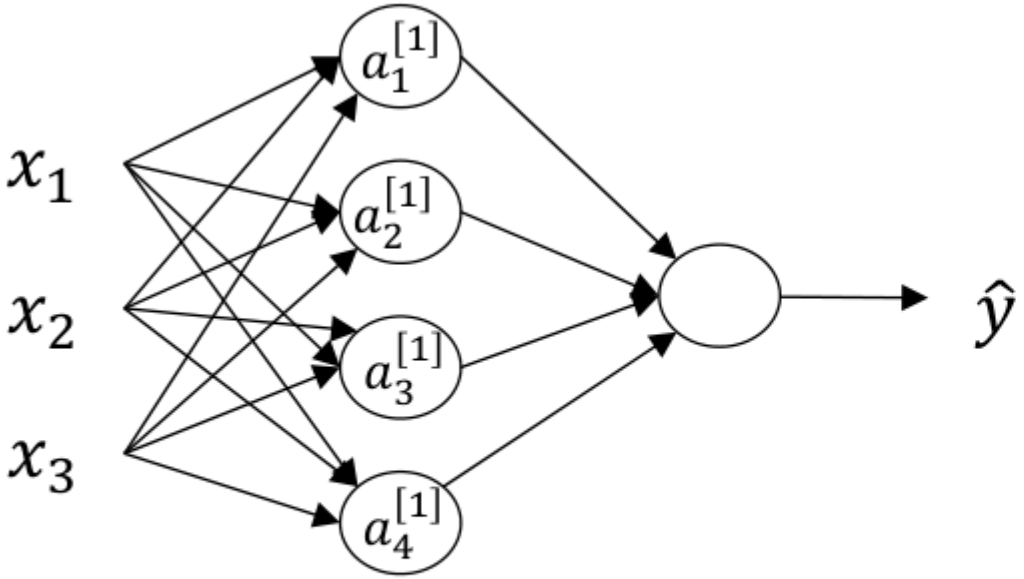
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

Réseau de plusieurs couches



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

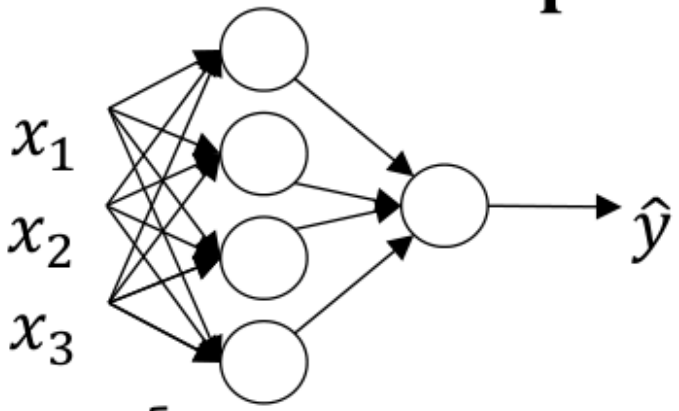
$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Réseau de plusieurs couches

Vectorisation pour exemples multiple



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Réseau de plusieurs couches

Paramètres: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyper-paramètres: le taux d'apprentissage α

Itérations

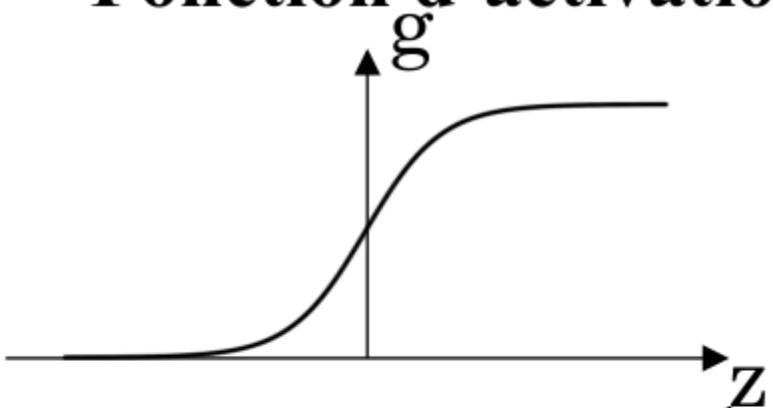
Choix de la fonction d'activation

Les couches cachées

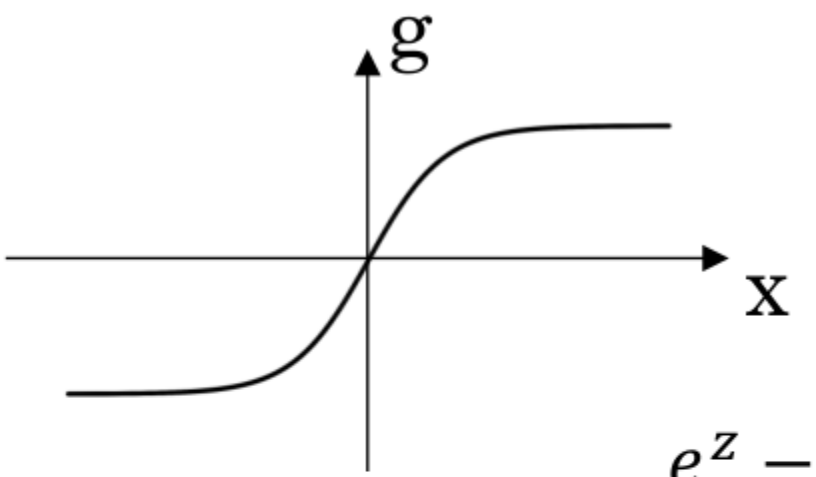
Mini batch size

Réseau de plusieurs couches

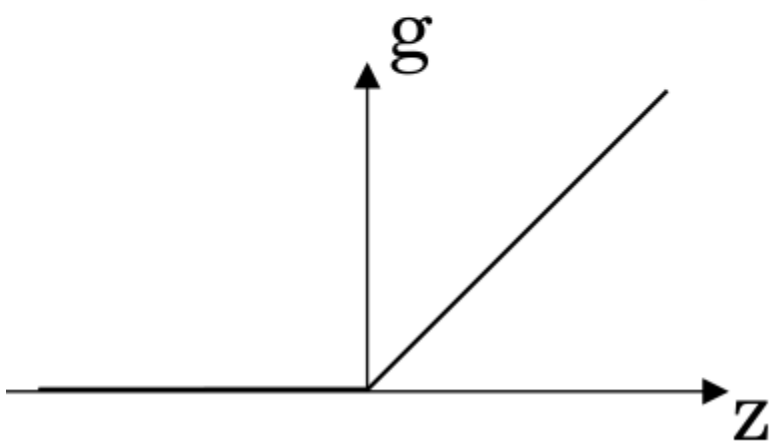
Fonction d'activation



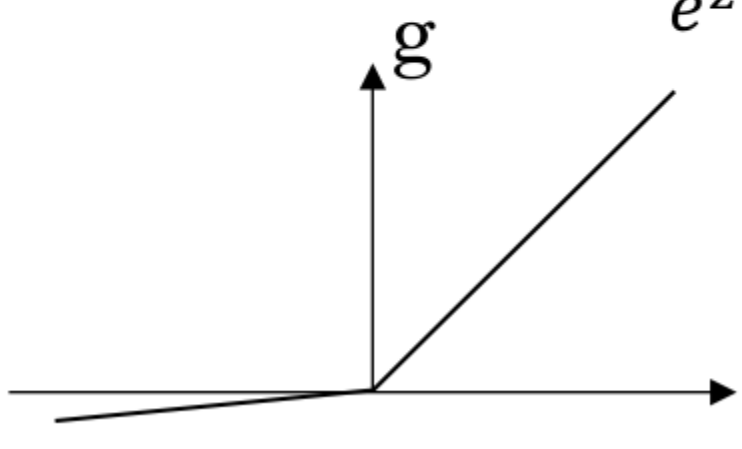
sigmoid: $g(z) = \dfrac{1}{1+e^{-z}}$



tanh: $g(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



ReLu: $g(z) = \max(0, z)$



Leaky ReLu: $g(z) = \max(0{,}01z, z)$

Réseau de plusieurs couches

Fonction d'erreur (Régression):

- MSE, RMSE, R-squared

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - \hat{y}_i)^2 \qquad RMSE = \sqrt{MSE} \qquad R^2 = 1 - \frac{MSE}{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

- MAE $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \widehat{y_i}|$
- MSPE, MAPE $\frac{100\%}{N}\sum_{i=1}^{N}\left(\frac{y_i - \widehat{y_i}}{y_i}\right)^2$ $\frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \widehat{y_i}}{y_i}\right|^2$
- MSLE $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(log(y_i + 1)log(1 + \widehat{y_i})\right)^2}$

Réseau de plusieurs couches

Fonction d'erreur (Classification):

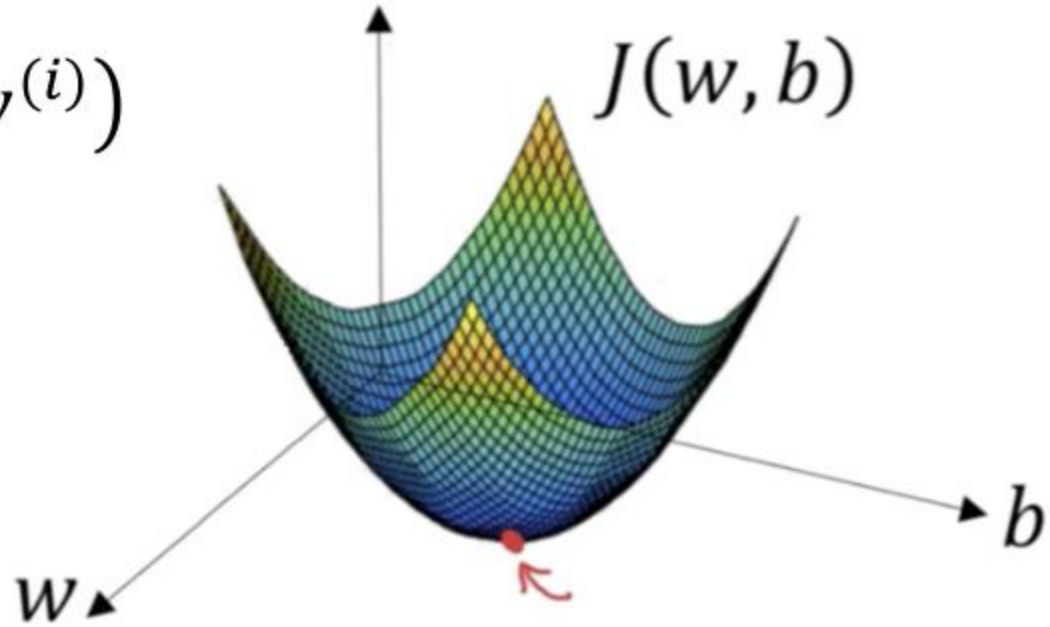
- Logloss

$$LogLoss_{binary} = -\frac{1}{N}\sum_{i=1}^{N} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

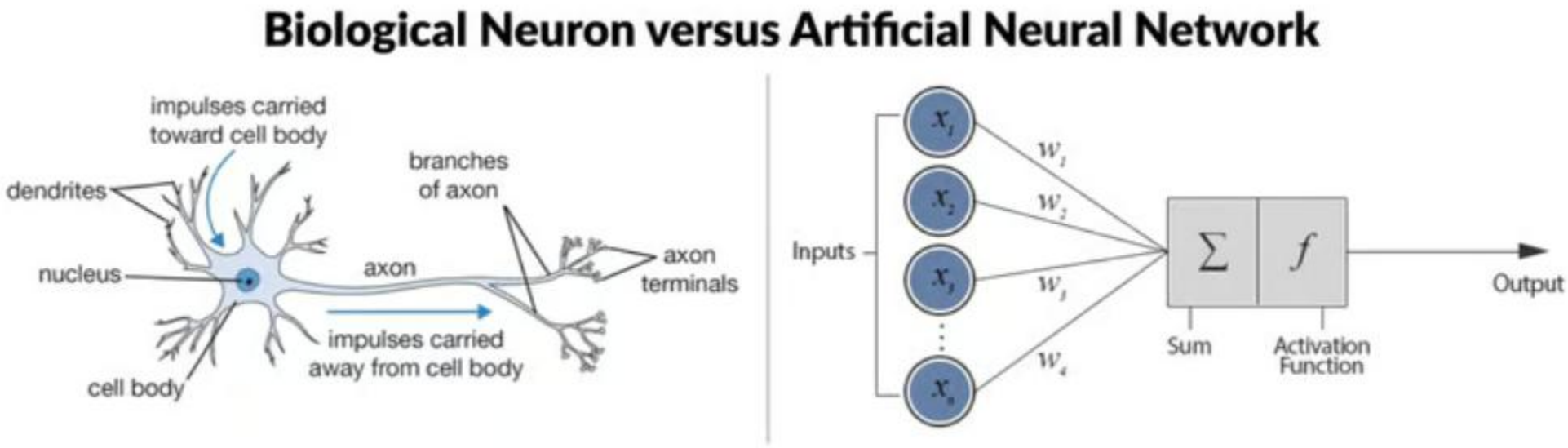
$$LogLoss_{multi-class} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L} y_{il}\, log(\hat{y}_{il_i})$$

Réseau de plusieurs couches

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$



Neurones


Biological Neuron versus Artificial Neural Network
impulses carried toward cell body
dendrites
nucleus
cell body
axon
branches of axon
axon terminals
impulses carried away from cell body
Inputs
Sum
Activation Function
Output

Importer les bibliothèques nécessaires

```
# Importing the necessary functionality
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Conv2D
from tensorflow.keras.layers import Flatten, MaxPooling2D
```

Bibliothèques nécessaires

- Keras est une interface vers la bibliothèque de machine learning TensorFlow.

- TensorFlow est une bibliothèque open source pour le calcul numérique et l'apprentissage automatique, qui permet une exécution efficace des calculs sur les CPU et les GPU.

- Construire un réseau neuronal avec Keras commence par la définition du modèle. Le type de modèle le plus courant est Séquentiel, qui permet la création de modèles couche par couche de manière séquentielle.

modèle = Séquentiel()

Créer des réseaux de neurones avec Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

modèle = Séquentiel()
model.add(Dense(units=64, activation='relu’, input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

Création d'un modèle de type séquentiel

```
# Creating the model
DNN_Model = Sequential()


Compiler le modèle
model.compile(optimizer='adam',          loss='categorical_crossentropy',
metrics=['accuracy'])
```

Application

- Charger les données
- Définir le modèle Keras
- Compiler le modèle Keras
- Ajuster le modèle Keras
- Évaluer le modèle Keras
- Faire des prédictions

Application

```
# Charger les données
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# divisé en variables d'entrée (X) et de sortie (y)
X = dataset[:,0:8]
```

- y = dataset[:,8]

Application

- Définir le modèle Keras

```
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Application

```
# compiler le modèle keras
model.compile(loss='binary_crossentropy’, optimizer='adam', metrics=['accuracy'])
```

Application

```
# ajuster le modèle keras sur le jeu de données
model.fit(X, y, epochs=150, batch_size=10)
```

Application

```
# évaluer le modèle keras
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Application

```
# faire des prévisions de probabilité avec le modèle
predictions = model.predict(X)
# round predictions
rounded = [round(x[0]) for x in predictions]
```

Ou bien:

```
# faire des prédictions de classe à l'aide du modèle
predictions = (model.predict(X) > 0.5).astype(int)
```