Deep Learning

Introduction

Thomas Ranvier

Ynov

Sommaire

Contexte

Introduction au Deep Learning

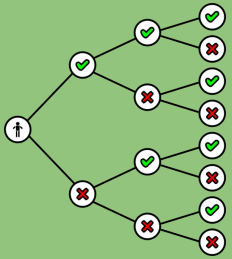
Qu'est ce que le Deep Learning ?

Artificial Intelligence

Méthodes capable d'imiter un comportement humain

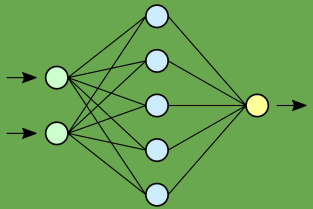
Machine Learning

Méthodes capable d'apprendre automatiquement à partir de données



Deep Learning

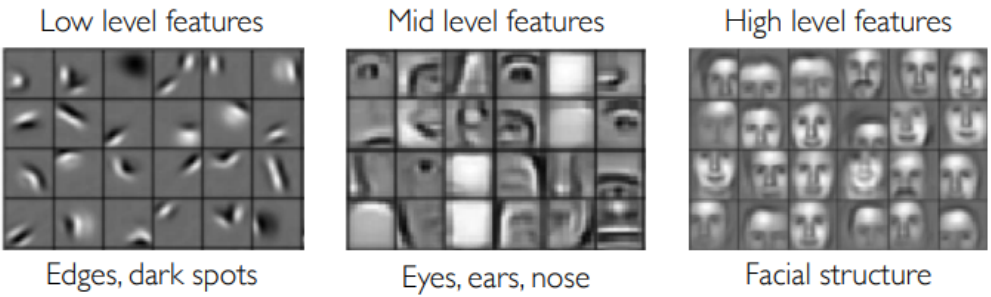
Utilisation de réseaux de neurones profonds



L'intérêt du Deep Learning

Problématique

Coder manuellement un algorithme à reconnaître des features est très laborieux, long et peu efficace.



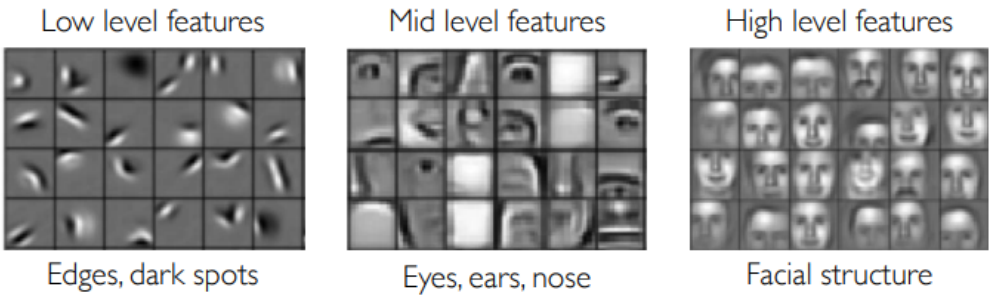
L'intérêt du Deep Learning

Problématique

Coder manuellement un algorithme à reconnaître des features est très laborieux, long et peu efficace.

Solution

Le Deep Learning rend possible l'apprentissage automatique de ces features.



Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.
- Robotique

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.
- Robotique
- Sécurité

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.
- Robotique
- Sécurité
- Santé

Domaines d'application du Deep Learning

Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.
- Robotique
- Sécurité
- Santé
- Traduction

Domaines d'application du Deep Learning

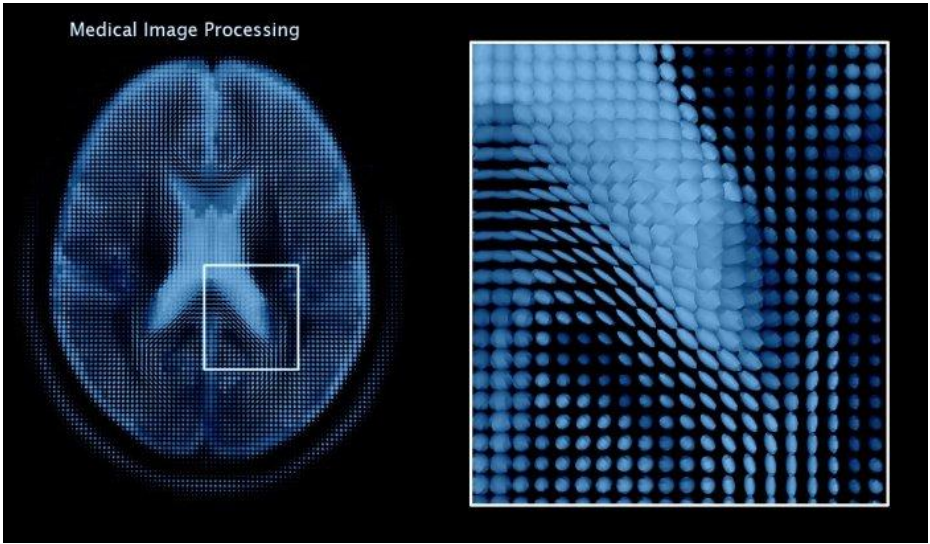
Domaines d'application

- Reconnaissance visuelle
  - Reconnaissance faciale
  - Reconnaissance de formes
  - Reconnaissance de texte
  - Etc.
- Robotique
- Sécurité
- Santé
- Traduction
- Etc.

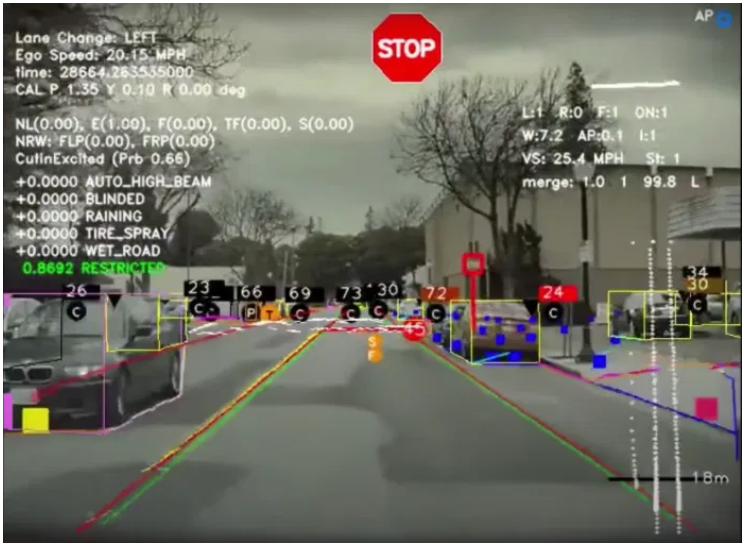
Exemples d'applications du Deep Learning

Exemples d'applications du Deep Learning

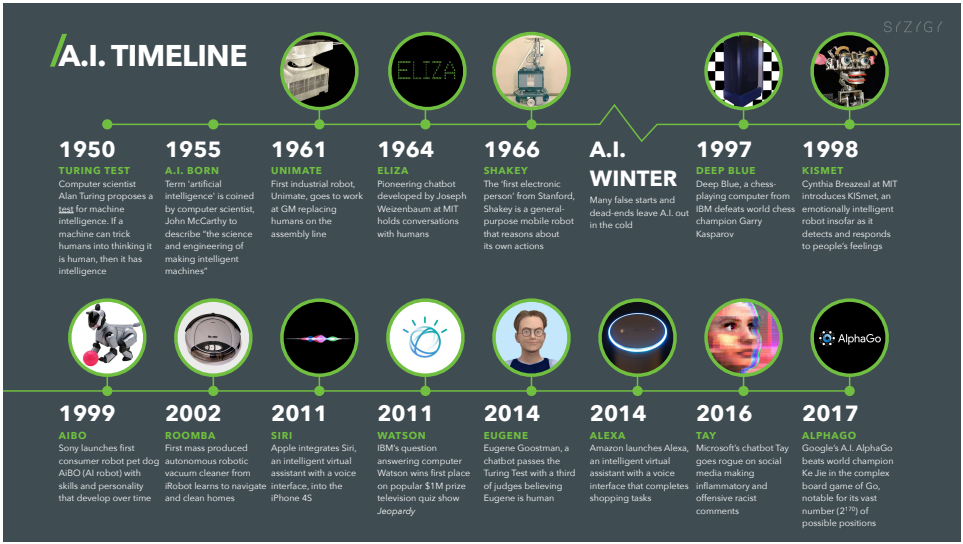
Exemples d'applications du Deep Learning



Exemples d'applications du Deep Learning



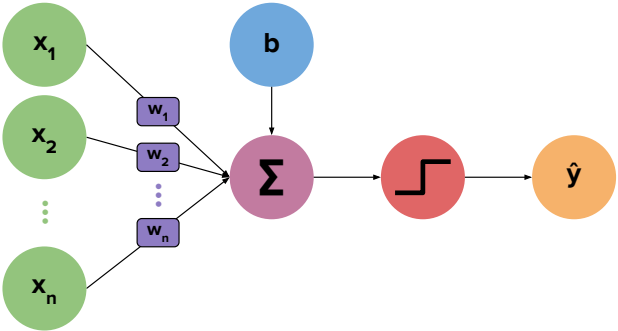
Pourquoi le Deep Learning se développe-t-il maintenant ?



Le Perceptron

L'élément de base pour construire des réseaux de neurones

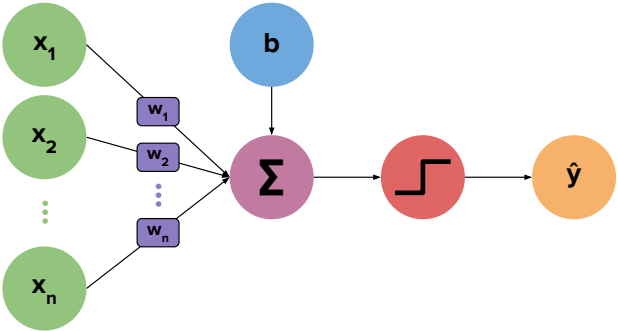
Le perceptron - forward propagation



Formalisation mathématique

$$\hat{y} = g(b + \sum_{i=1}^{m} x_i w_i)$$

Le perceptron - forward propagation



Formalisation mathématique

$$\hat{y} = g(b + \sum_{i=1}^{m} x_i w_i)$$

$$\Leftrightarrow \hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$$

Où $\boldsymbol{X}^T = [x_1 \dots x_m]$ et $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)

Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)
- L'intérêt des fonctions d'activation est d'introduire une non-linéarité dans le modèle

Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)
- L'intérêt des fonctions d'activation est d'introduire une non-linéarité dans le modèle
- Exemples de fonctions d'activations courantes :

Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)
- L'intérêt des fonctions d'activation est d'introduire une non-linéarité dans le modèle
- Exemples de fonctions d'activations courantes :
  - Sigmoid

Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)
- L'intérêt des fonctions d'activation est d'introduire une non-linéarité dans le modèle
- Exemples de fonctions d'activations courantes :
  - Sigmoid
  - Tanh

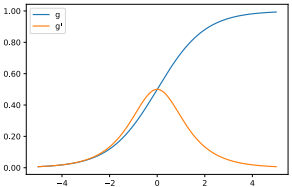
Le perceptron - fonction d'activation

Les fonctions d'activation

- Dans $\hat{y} = g(b + \boldsymbol{X}^T \boldsymbol{W})$, g est une fonction d'activation non linéaire (activation function / nonlinearity)
- L'intérêt des fonctions d'activation est d'introduire une non-linéarité dans le modèle
- Exemples de fonctions d'activations courantes :
  - Sigmoid
  - Tanh
  - ReLU

Exemples de fonctions d'activation

Sigmoid

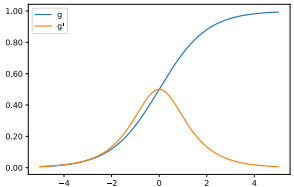


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z) \cdot (1 - g(z))$$

Exemples de fonctions d'activation

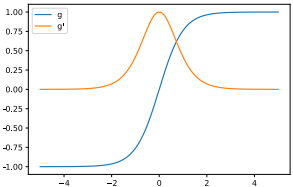
Sigmoid



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z) \cdot (1 - g(z))$$

Hyperbolic Tangent (tanh)

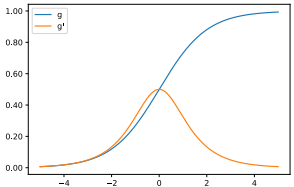


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

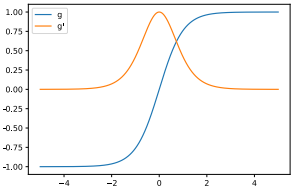
Exemples de fonctions d'activation

Sigmoid



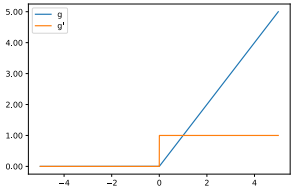
$$g(z) = \frac{1}{1 + e^{-z}}$$
$$g'(z) = g(z) \cdot (1 - g(z))$$

Hyperbolic Tangent (tanh)



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$
$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = max(0, z)$$
$$g'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Visualisation de l'intérêt des fonctions d'activation

Fonctionnement d'un réseau de neurones

- Sans fonction d'activation un réseau de neurones ne peut apprendre qu'une séparation linéaire des données

Visualisation de l'intérêt des fonctions d'activation

Fonctionnement d'un réseau de neurones

- Sans fonction d'activation un réseau de neurones ne peut apprendre qu'une séparation linéaire des données
- Pour apprendre à séparer des données non linéairement séparable on doit donc utiliser des fonctions d'activation non linéaires

Visualisation de l'intérêt des fonctions d'activation

Fonctionnement d'un réseau de neurones

- Sans fonction d'activation un réseau de neurones ne peut apprendre qu'une séparation linéaire des données
- Pour apprendre à séparer des données non linéairement séparable on doit donc utiliser des fonctions d'activation non linéaires

Outil de visualisation

- On va utiliser un outil permettant de visualiser le fonctionnement d'un réseau de neurones couche par couche : `https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`

Visualisation de l'intérêt des fonctions d'activation

Fonctionnement d'un réseau de neurones

- Sans fonction d'activation un réseau de neurones ne peut apprendre qu'une séparation linéaire des données
- Pour apprendre à séparer des données non linéairement séparable on doit donc utiliser des fonctions d'activation non linéaires

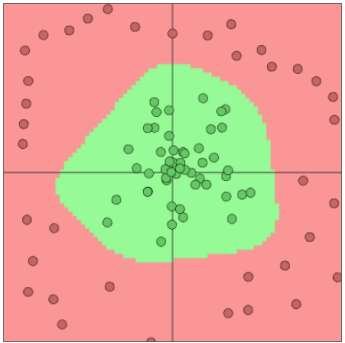
Outil de visualisation

- On va utiliser un outil permettant de visualiser le fonctionnement d'un réseau de neurones couche par couche : `https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`
- Cet outil permet de visualiser la frontière apprise par le modèle, ainsi que les opérations effectuées à chaque étape du modèle

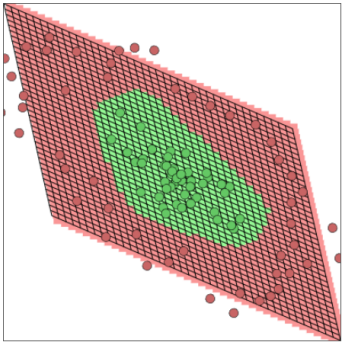
Visualisation de l'intérêt des fonctions d'activation

Visualisation

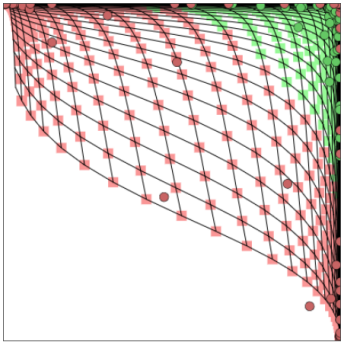
Avec fonction d'activation, première couche



(a) Données d'entrée



(b) Première couche

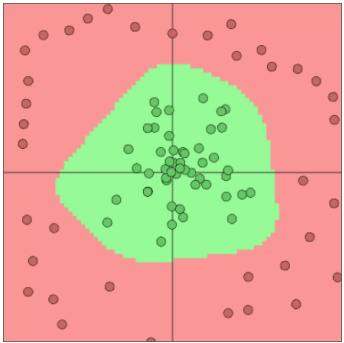


(c) Première fonction d'activation

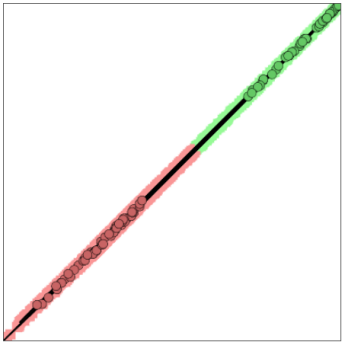
Visualisation de l'intérêt des fonctions d'activation

Visualisation

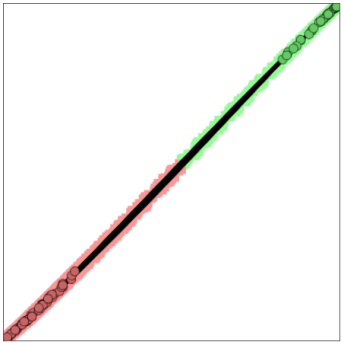
Avec fonction d'activation, seconde couche : données linéairement séparables



(a) Données d'entrée



(b) Seconde couche

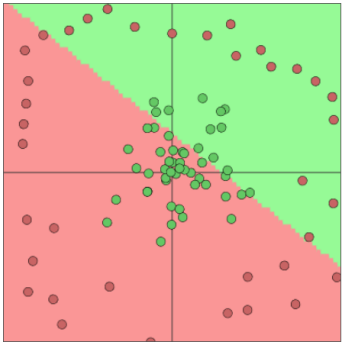


(c) Seconde fonction d'activation

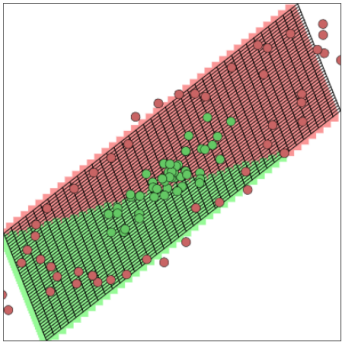
Visualisation de l'intérêt des fonctions d'activation

Visualisation

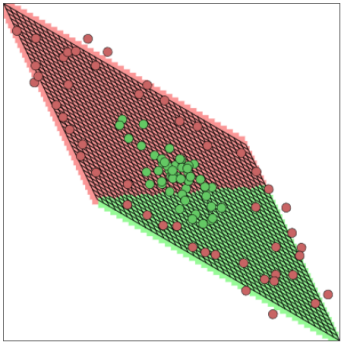
Sans fonction d'activation, première et seconde couches : données non linéairement séparables



(a) Données d'entrée



(b) Première couche

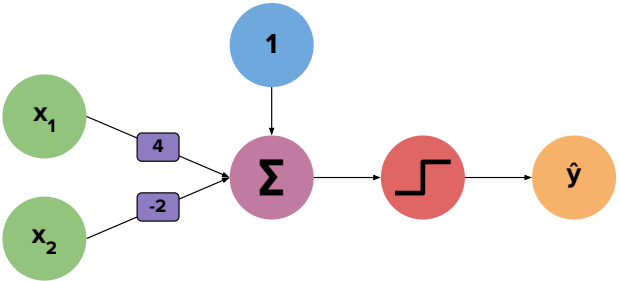


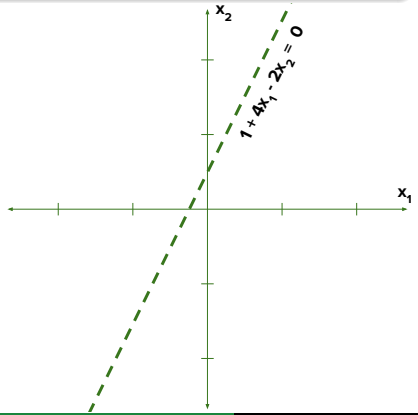
(c) Seconde couche

Le perceptron - exercice

Paramètres

$$W = \begin{bmatrix} 4 \\ -2 \end{bmatrix}, b = 1, \text{donc } \hat{y} = g(X^T W + b) = g(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 4 \\ -2 \end{bmatrix} + 1), \text{avec } g(z) = \frac{1}{1 + e^{-}z}$$

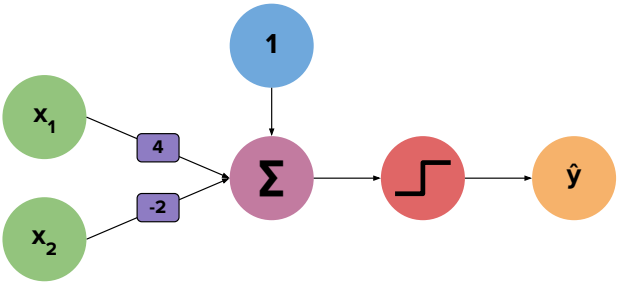


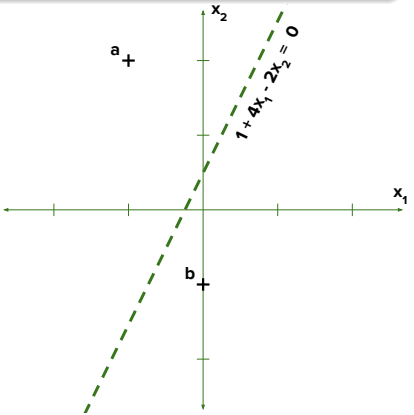


Le perceptron - exercice

Points

Calculer $\hat{y}$ pour les points $a = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ et $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

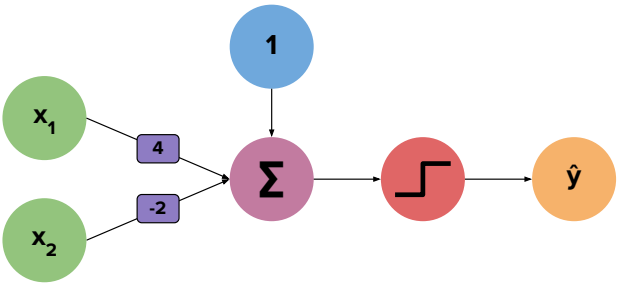


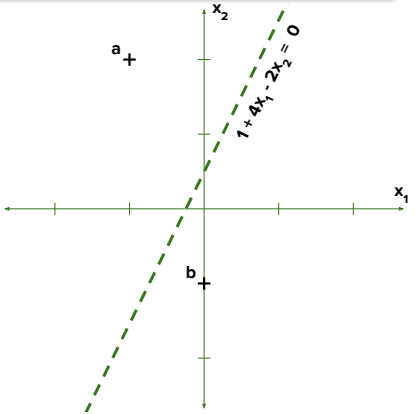


Le perceptron - exercice

Points

Calculer $\hat{y}$ pour les points $a = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ et $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, réponse : $\hat{y}_a \approx 0.00091$ et $\hat{y}_b \approx 0.95257$

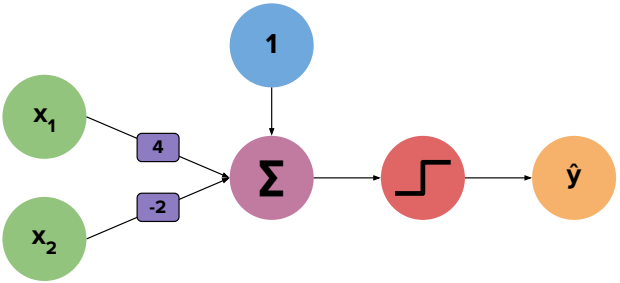


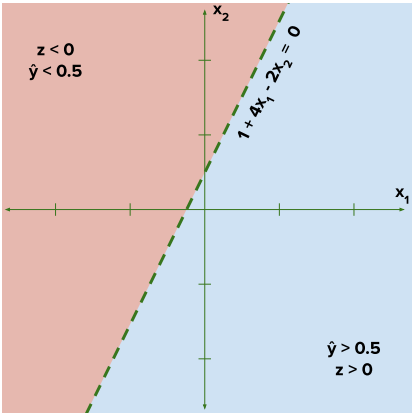


Le perceptron - exercice

Visualisation de la frontière

On peut tracer la frontière de décision du perceptron se situant sur $\hat{y} = 0.5$





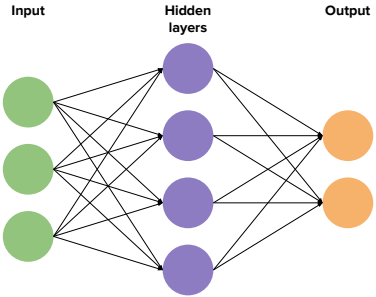
Réseaux de neurones

Organisation de perceptrons en une architecture permettant l'apprentissage

Composition d'un réseau de neurones

Réseau de neurones

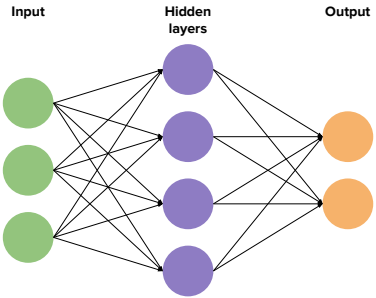
- Il est composé de perceptrons, organisés par couches : Perceptron Multicouche (MLP)



Composition d'un réseau de neurones

Réseau de neurones

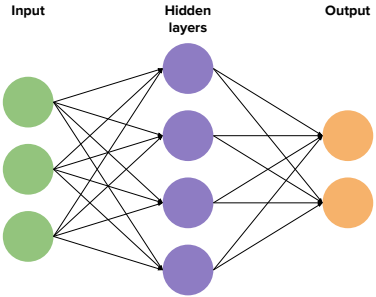
- Il est composé de perceptrons, organisés par couches : Perceptron Multicouche (MLP)
- La couche d'entrée correspond aux données en entrée, la couche de sortie correspond aux résultats donnés par le modèle



Composition d'un réseau de neurones

Réseau de neurones

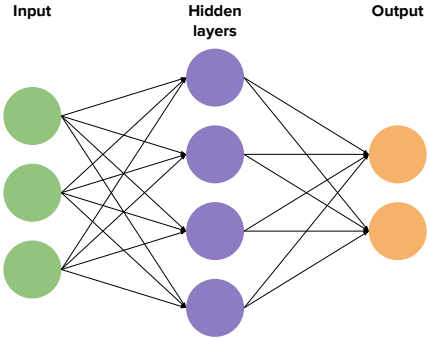
- Il est composé de perceptrons, organisés par couches : Perceptron Multicouche (MLP)
- La couche d'entrée correspond aux données en entrée, la couche de sortie correspond aux résultats donnés par le modèle
- Les couches entre les deux sont des couches dites cachées, c'est là que se passe l'essentiel de l'apprentissage



Fonctionnement d'un réseau de neurones

Réseau de neurones

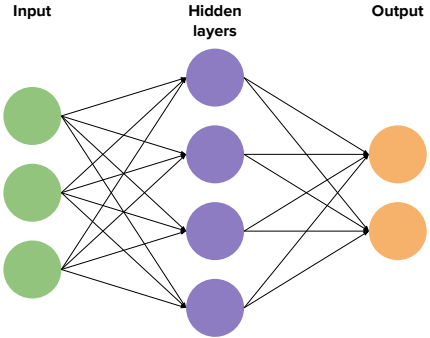
- Il prend des données en entrées



Fonctionnement d'un réseau de neurones

Réseau de neurones

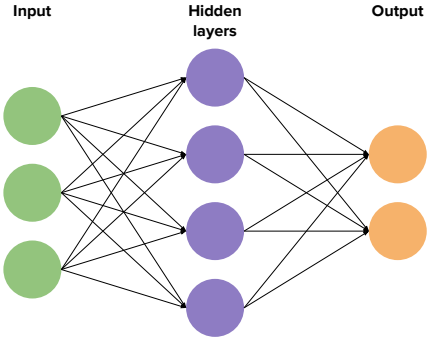
- Il prend des données en entrées
- On l'entraîne pour qu'il découvre et apprenne des patterns au sein de données connues



Fonctionnement d'un réseau de neurones

Réseau de neurones

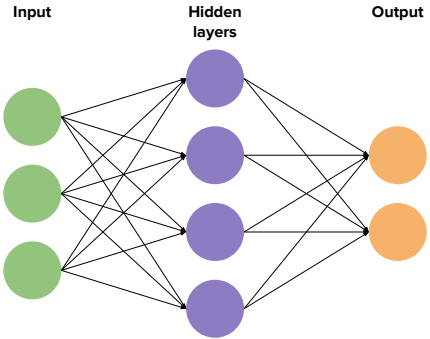
- Il prend des données en entrées
- On l'entraîne pour qu'il découvre et apprenne des patterns au sein de données connues
- On l'utilise ensuite pour générer des prédictions sur de nouvelles données inconnues



Fonctionnement d'un réseau de neurones

Réseau de neurones

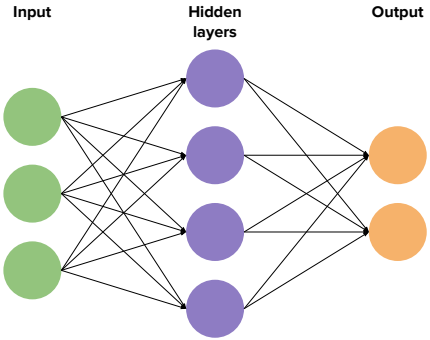
- Il prend des données en entrées



Fonctionnement d'un réseau de neurones

Réseau de neurones

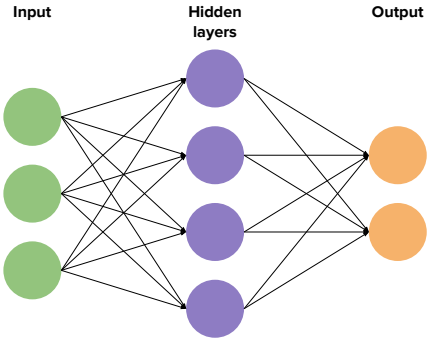
- Il prend des données en entrées
- On l'entraîne pour qu'il découvre et apprenne des patterns au sein de données connues



Fonctionnement d'un réseau de neurones

Réseau de neurones

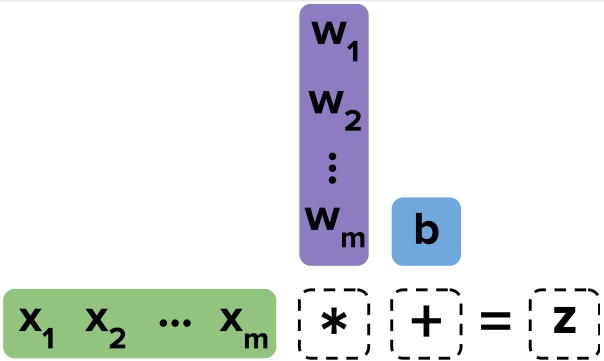
- Il prend des données en entrées
- On l'entraîne pour qu'il découvre et apprenne des patterns au sein de données connues
- On l'utilise ensuite pour générer des prédictions sur de nouvelles données inconnues



Modéliser une couche de perceptrons de manière efficiente

Forward propagation d'une donnée à m dimensions dans un perceptron seul

$\boldsymbol{X}$ et $\boldsymbol{W}$ sont des vecteurs, b est une valeur unique, donc $\hat{y}$ est une valeur unique également



Modéliser une couche de perceptrons de manière efficiente

Forward propagation d'une donnée à *m* dimensions dans un perceptron seul

X et ***B*** sont des vecteurs, ***W*** est une matrice, donc $\hat{Y}$ est un vecteur

$$\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{pmatrix}$$

$$\begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix}$$

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_m \end{pmatrix} \quad * \quad + \quad = \quad Z$$

Avec g(Z) = $\hat{Y}$

Modéliser une couche de perceptrons de manière efficiente

Forward propagation d'une donnée à *m* dimensions dans un perceptron seul

B est un vecteur, ***X*** et ***W*** sont des matrices, donc $\hat{Y}$ est une matrice

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \quad \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & \dots & x_{2,m} \\ \vdots & \vdots & & \vdots \\ x_{d,1} & x_{d,2} & \dots & x_{d,m} \end{bmatrix} \quad * \quad + \quad = \quad Z$$

Avec g(Z) = $\hat{Y}$

Programmer une couche de perceptrons sous TensorFlow

Coder manuellement une couche “dense”

```python
class CustomDense(tf.keras.layers.Layer):
    def __init__(self, input_dim, units):
        super(CustomDense, self).__init__()
        # Create weights and biases
        self.weights = self.add_weight(shape=(input_dim, units))
        self.biases = self.add_weight(shape=(units, 1))

    def call(self, x):
        # Forward propagation
        z = tf.matmul(x, self.weights) + self.biases
        # Add non-linearity using an activation function
        y = tf.keras.activations.relu(z)
        return y
```

Programmer une couche de perceptrons sous TensorFlow

Coder manuellement une couche “dense”

```
class CustomDense(tf.keras.layers.Layer):
    def __init__(self, input_dim, units):
        super(CustomDense, self).__init__()
        # Create weights and biases
        self.weights = self.add_weight(shape=(input_dim, units))
        self.biases = self.add_weight(shape=(units, 1))

    def call(self, x):
        # Forward propagation
        z = tf.matmul(x, self.weights) + self.biases
        # Add non-linearity using an activation function
        y = tf.keras.activations.relu(z)
        return y
```

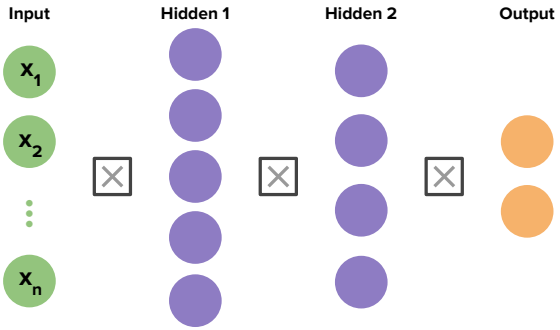
Créer une couche de 10 neurones en utilisant simplement le code déjà existant sous TensorFlow

```
dense_layer = tf.keras.layers.Dense(units=10, activation='relu')
```

Programmer un réseau de neurones de plusieurs couches sous TensorFlow

Créer un modèle de dimensions (5, 4, 2)

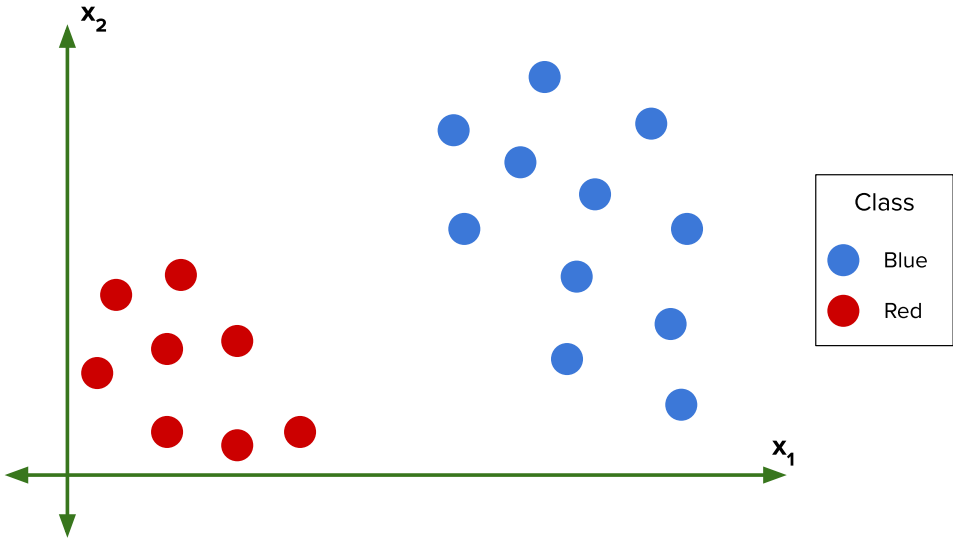
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=5, activation='relu'),
    tf.keras.layers.Dense(units=4, activation='relu'),
    tf.keras.layers.Dense(units=2, activation='softmax'),
])
```



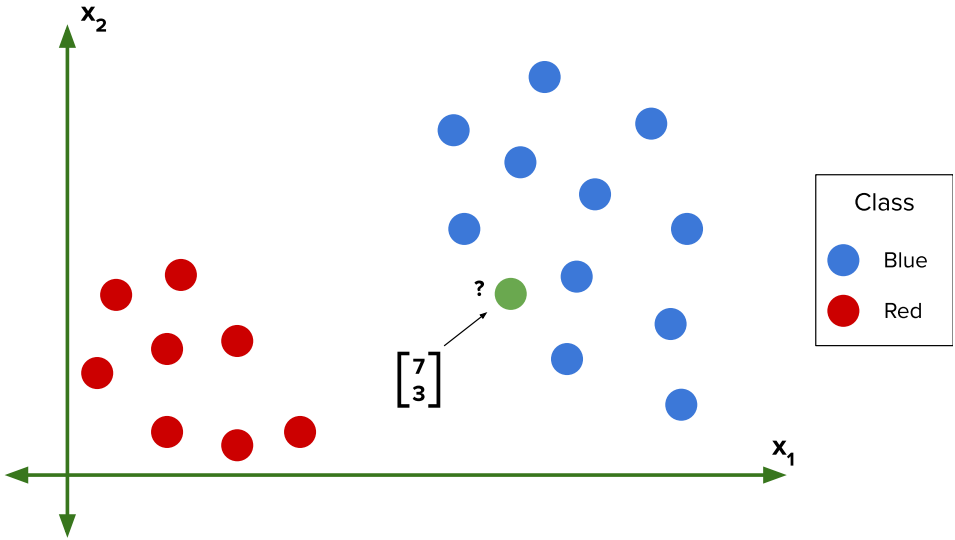
Apprentissage en théorie

Apprendre une tâche à un réseau de neurones en théorie

Exemple de problème de classification



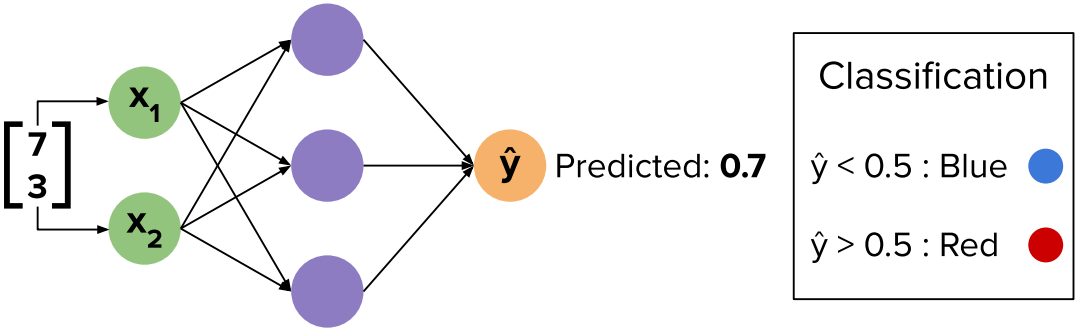
Exemple de problème de classification



Exemple de problème de classification

Forward propagation

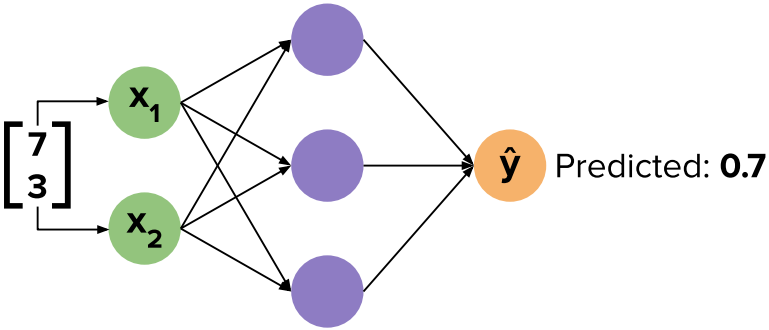
- On associe la valeur **0** pour la classe **Blue** et **1** pour la classe **Red**.



Exemple de problème de classification

Forward propagation

- On associe la valeur **0** pour la classe **Blue** et **1** pour la classe **Red**.
- On passe le point $\begin{bmatrix} 7 \\ 3 \end{bmatrix}$ en entrée de notre modèle.



Classification

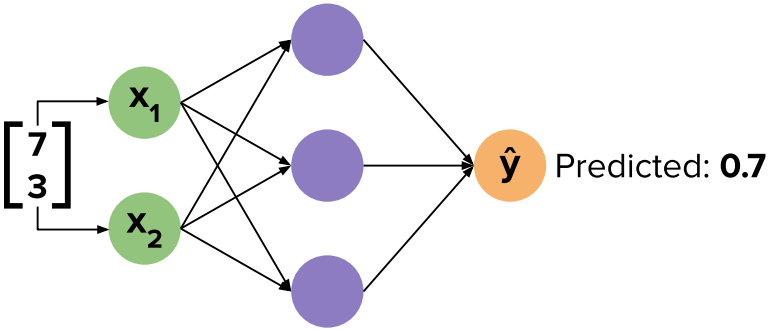
ŷ < 0.5 : Blue

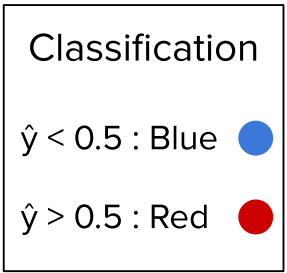
ŷ > 0.5 : Red

Exemple de problème de classification

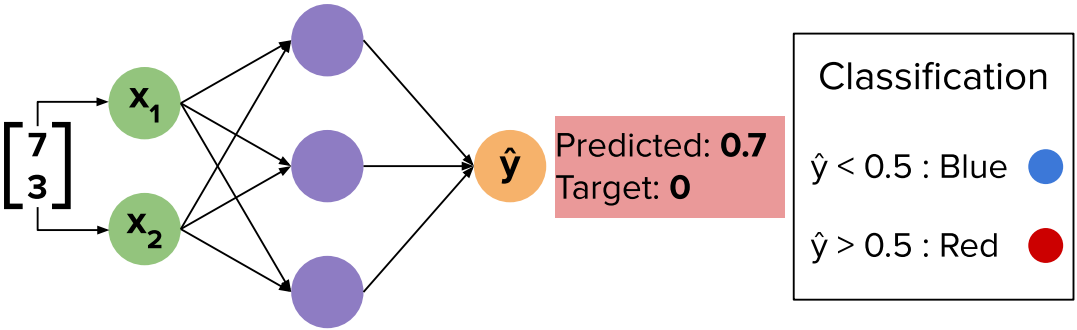
Forward propagation

- On associe la valeur **0** pour la classe **Blue** et **1** pour la classe **Red**.
- On passe le point $\begin{bmatrix} 7 \\ 3 \end{bmatrix}$ en entrée de notre modèle.
- Le modèle nous renvoi une valeur : **0.7**.





Exemple de problème de classification



Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.
  - `tf.keras.losses.BinaryCrossentropy()` ou `tf.keras.losses.CategoricalCrossentropy()`.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.
  - `tf.keras.losses.BinaryCrossentropy()` ou `tf.keras.losses.CategoricalCrossentropy()`.
- Problèmes de régression : Mean Squared Error (MSE)

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.
  - `tf.keras.losses.BinaryCrossentropy()` ou `tf.keras.losses.CategoricalCrossentropy()`.
- Problèmes de régression : Mean Squared Error (MSE)
  - $\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$.

Loss functions

Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.
  - `tf.keras.losses.BinaryCrossentropy()` ou `tf.keras.losses.CategoricalCrossentropy()`.
- Problèmes de régression : Mean Squared Error (MSE)
  - $\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$.
  - `tf.keras.losses.MeanSquaredError()`.

Loss functions

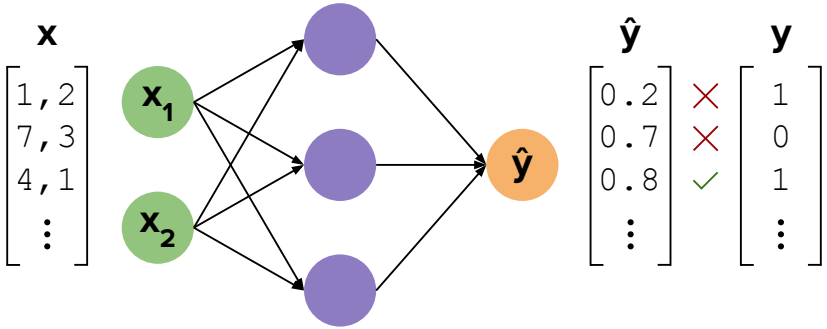
Intérêt d'une loss function

- Elle permet de mesurer l'écart entre la prédiction faite par le modèle et la cible qu'on lui a donné.
- Le modèle est entraîné à minimiser la valeur de cette fonction, c'est ce qui lui permet d'apprendre la tâche souhaitée.

Exemples de fonctions adaptées selon le problème

- Pour permettre un bon apprentissage il faut choisir une loss function adaptée à la tâche attendue.
- Problèmes de classification : Cross-entropy
  - $\mathcal{L}(\hat{y}, y) = y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})$.
  - `tf.keras.losses.BinaryCrossentropy()` ou `tf.keras.losses.CategoricalCrossentropy()`.
- Problèmes de régression : Mean Squared Error (MSE)
  - $\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$.
  - `tf.keras.losses.MeanSquaredError()`.
- On notera $\mathcal{L}(f(x, \boldsymbol{W}), y)$ les fonctions de perte pour le reste du cours, avec f le réseau de neurones dans lequel on envoie les données x et W les paramètres (poids et biais) du modèle.

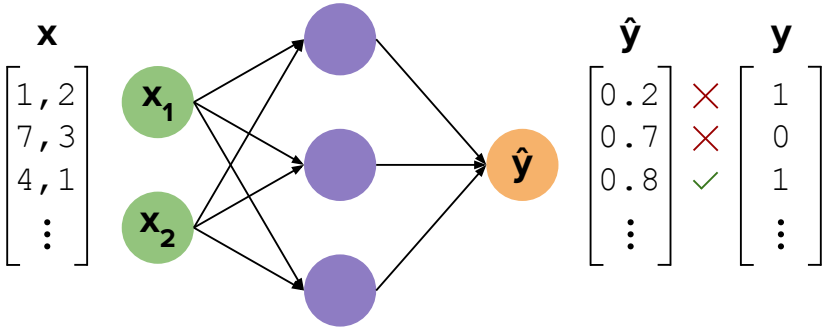
Fonction objectif



Intérêt d'une fonction objectif

- La fonction objectif mesure la moyenne de la loss sur l'ensemble du jeu de données.

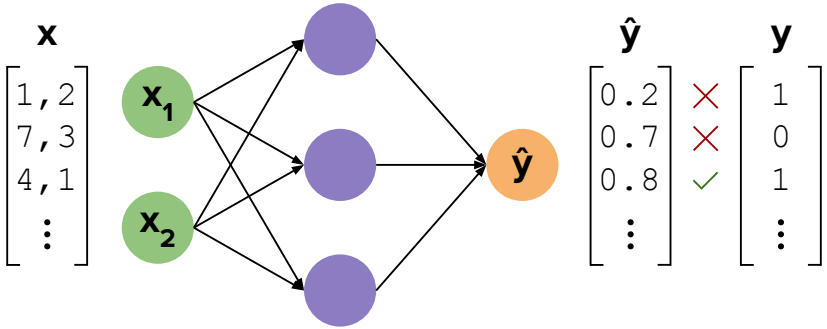
Fonction objectif



Intérêt d'une fonction objectif

- La fonction objectif mesure la moyenne de la loss sur l'ensemble du jeu de données.
- Elle est notée $J(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x_i, \boldsymbol{W}), y_i)$

Fonction objectif



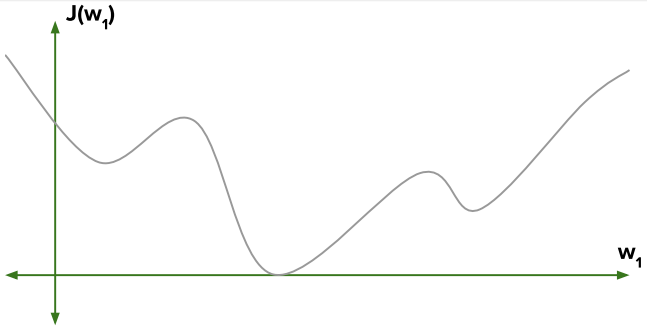
Intérêt d'une fonction objectif

- La fonction objectif mesure la moyenne de la loss sur l'ensemble du jeu de données.
- Elle est notée $J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x_i, \boldsymbol{W}), y_i)$
- On cherche les poids et biais qui permettent d'atteindre la loss la plus basse possible :
  $W^* = \underset{\boldsymbol{W}}{argmin}\, J(\boldsymbol{W}) = \underset{\boldsymbol{W}}{argmin}\, \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x_i, \boldsymbol{W}), y_i)$

Optimisation d'une fonction objectif

Optimisation d'une fonction objectif

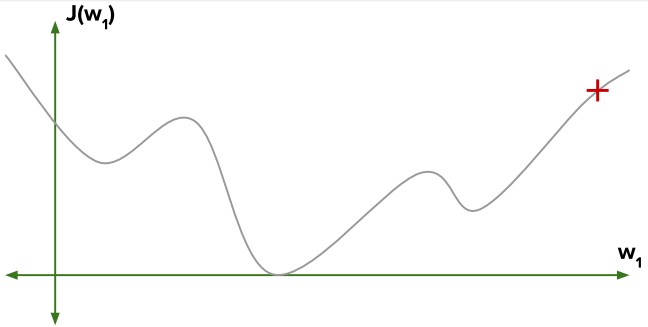
- On cherche les paramètres permettant de minimiser $J(\boldsymbol{W})$



Optimisation d'une fonction objectif

Optimisation d'une fonction objectif

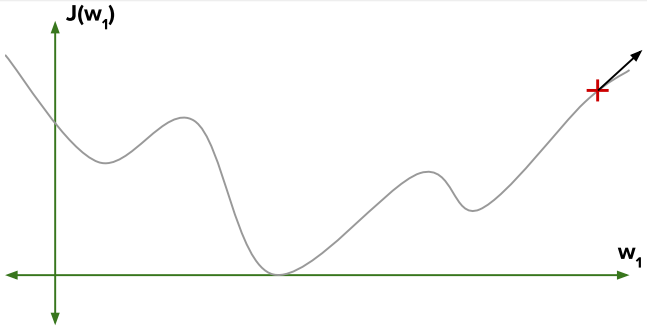
- On cherche les paramètres permettant de minimiser $J(\boldsymbol{W})$
- On initialise les poids du modèle à des valeurs aléatoires



Optimisation d'une fonction objectif

Optimisation d'une fonction objectif

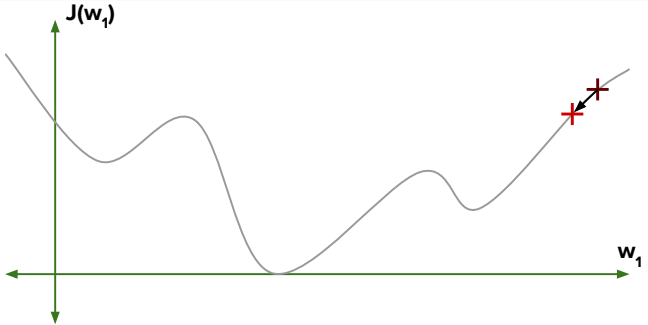
- On cherche les paramètres permettant de minimiser $J(\boldsymbol{W})$
- On initialise les poids du modèle à des valeurs aléatoires
- On calcule le gradient à notre emplacement actuel $\frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$



Optimisation d'une fonction objectif

Optimisation d'une fonction objectif

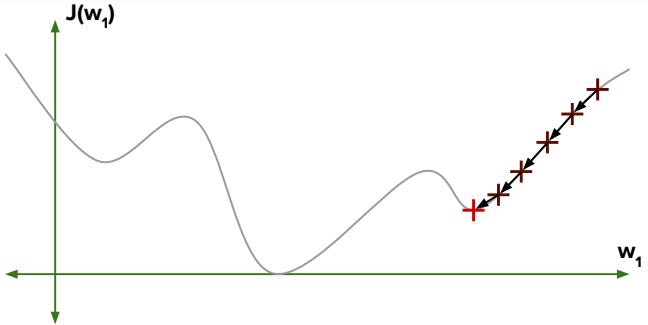
- On cherche les paramètres permettant de minimiser $J(\boldsymbol{W})$
- On initialise les poids du modèle à des valeurs aléatoires
- On calcule le gradient à notre emplacement actuel $\frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$
- On se déplace dans la direction opposée au gradient



Optimisation d'une fonction objectif

Optimisation d'une fonction objectif

- On cherche les paramètres permettant de minimiser $J(\boldsymbol{W})$
- On initialise les poids du modèle à des valeurs aléatoires
- On calcule le gradient à notre emplacement actuel $\frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$
- On se déplace dans la direction opposée au gradient



Pseudocode pour l'optimisation d'une fonction objectif

Gradient Descent algorithme

1. Randomly initialize model parameters $\boldsymbol{W}$

Pseudocode pour l'optimisation d'une fonction objectif

Gradient Descent algorithme

1. Randomly initialize model parameters $\boldsymbol{W}$
2. Loop until convergence :

Pseudocode pour l'optimisation d'une fonction objectif

Gradient Descent algorithme

1. Randomly initialize model parameters $\boldsymbol{W}$
2. Loop until convergence :
3. Compute gradient $\frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$

Pseudocode pour l'optimisation d'une fonction objectif

Gradient Descent algorithme

1. Randomly initialize model parameters $\boldsymbol{W}$
2. Loop until convergence :
3. Compute gradient $\frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$
4. Update model parameters $W = W - \eta \frac{\delta J(\boldsymbol{W})}{\delta \boldsymbol{W}}$

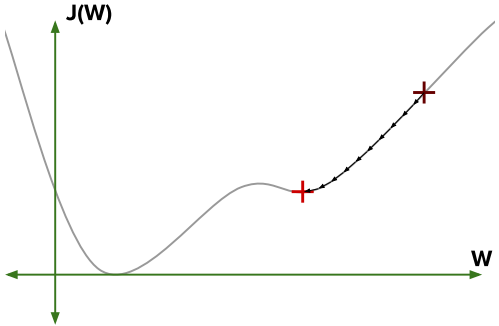
Apprentissage en pratique

Apprendre une tâche à un réseau de neurones en pratique

Choisir un taux d'apprentissage adapté

Choisir un taux d'apprentissage adapté

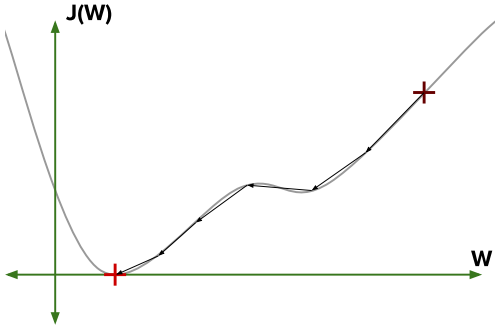
- Un learning rate trop petit va converger très lentement et a de grandes chances de rester bloqué sur un minimum local



Choisir un taux d'apprentissage adapté

Choisir un taux d'apprentissage adapté

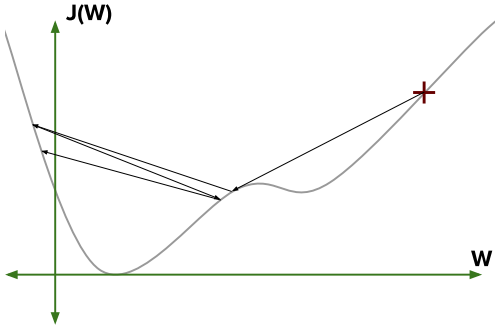
- Un learning rate trop petit va converger très lentement et a de grandes chances de rester bloqué sur un minimum local
- Un learning rate bien adapté permettra de converger rapidement vers un minimum intéressant



Choisir un taux d'apprentissage adapté

Choisir un taux d'apprentissage adapté

- Un learning rate trop petit va converger très lentement et a de grandes chances de rester bloqué sur un minimum local
- Un learning rate bien adapté permettra de converger rapidement vers un minimum intéressant
- Un learning rate trop large risque de ne pas parvenir a converger vers un minimum



Optimiseurs

Choisir un optimiseur

- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`

Optimiseurs

Choisir un optimiseur

- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`
- **Adagrad :** Diminue le learning rate petit à petit au fur et à mesure de l'optimisation
  - `tf.keras.optimizers.Adagrad()`

Optimiseurs

Choisir un optimiseur

- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`
- **Adagrad :** Diminue le learning rate petit à petit au fur et à mesure de l'optimisation
  - `tf.keras.optimizers.Adagrad()`
- **RMSProp :** Peut être considéré comme une version plus avancée d'Adagrad
  - `tf.keras.optimizers.RMSProp()`

Optimiseurs

Choisir un optimiseur

- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`
- **Adagrad :** Diminue le learning rate petit à petit au fur et à mesure de l'optimisation
  - `tf.keras.optimizers.Adagrad()`
- **RMSProp :** Peut être considéré comme une version plus avancée d'Adagrad
  - `tf.keras.optimizers.RMSProp()`
- **Adadelta :** Vise à ne pas avoir besoin de fixer soit même un learning rate manuellement
  - `tf.keras.optimizers.Adadelta()`

Optimiseurs

Choisir un optimiseur

- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`
- **Adagrad :** Diminue le learning rate petit à petit au fur et à mesure de l'optimisation
  - `tf.keras.optimizers.Adagrad()`
- **RMSProp :** Peut être considéré comme une version plus avancée d'Adagrad
  - `tf.keras.optimizers.RMSProp()`
- **Adadelta :** Vise à ne pas avoir besoin de fixer soit même un learning rate manuellement
  - `tf.keras.optimizers.Adadelta()`
- **Adam :** Adapte automatiquement la valeur du learning rate, offre des résultats consistants en général
  - `tf.keras.optimizers.Adam()`

Optimiseurs

Choisir un optimiseur

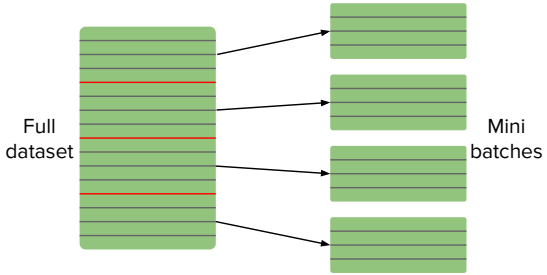
- **SGD :** Version plus rapide de l'algorithme de Gradient Descent, sert de base à la majorité des autres algorithmes d'optimisation
  - `tf.keras.optimizers.SGD()`
- **Adagrad :** Diminue le learning rate petit à petit au fur et à mesure de l'optimisation
  - `tf.keras.optimizers.Adagrad()`
- **RMSProp :** Peut être considéré comme une version plus avancée d'Adagrad
  - `tf.keras.optimizers.RMSProp()`
- **Adadelta :** Vise à ne pas avoir besoin de fixer soit même un learning rate manuellement
  - `tf.keras.optimizers.Adadelta()`
- **Adam :** Adapte automatiquement la valeur du learning rate, offre des résultats consistants en général
  - `tf.keras.optimizers.Adam()`
- Plus d'algorithmes d'optimisation et détails sur les hyperparamètres de chacun : `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers`

Implémentation générique d'une optimisation

Implémentation générique d'une optimisation

```
model = tf.keras.Sequential([...])
# Choisir l'optimiseur de son choix
optimizer = tf.keras.optimizers.Adam()
# Choisir la loss de son choix
loss_function = tf.keras.losses.BinaryCrossentropy()
# Entraîner le modèle pour un nombre fixé d'epochs
for _ in range(nb_epochs):
    with tf.GradientTape() as tape:
        # Forward propagation des données x dans le modèle pour obtenir les prédictions
        predictions = model(x)
        # Calcul de la loss entre les targets y et les prédictions obtenues
        loss = loss_function(y, predictions)
    # Calcul automatique du gradient par TensorFlow sur les paramètres du modèle
    # (model.trainable_variables)
    gradients = tape.gradient(loss, model.trainable_variables)
    # Mise à jour des paramètres en utilisant l'algorithme d'optimisation
    # (backward propagation)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

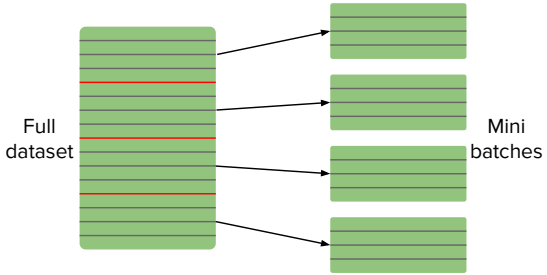
L'avantage d'utiliser des mini-batches



L'intérêt de séparer les données en mini-batches

- Le but est de séparer les données en mini-batches durant la phase d'optimisation

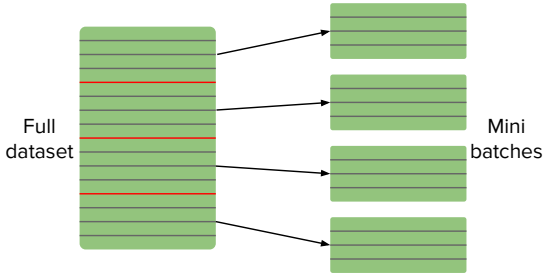
L'avantage d'utiliser des mini-batches



L'intérêt de séparer les données en mini-batches

- Le but est de séparer les données en mini-batches durant la phase d'optimisation
- Utiliser des mini-batches permet d'accélérer largement la vitesse de l'optimisation, les gradients étant basés sur moins de données sont plus rapides à calculer

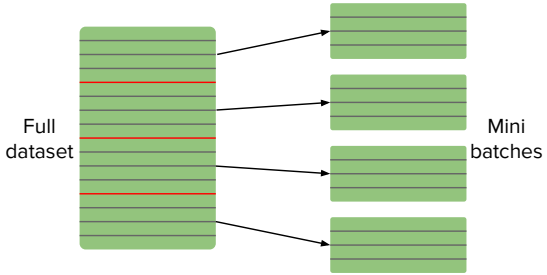
L'avantage d'utiliser des mini-batches



L'intérêt de séparer les données en mini-batches

- Le but est de séparer les données en mini-batches durant la phase d'optimisation
- Utiliser des mini-batches permet d'accélérer largement la vitesse de l'optimisation, les gradients étant basés sur moins de données sont plus rapides à calculer
- Cela permet aussi d'économiser de la RAM puisqu'il n'est pas nécessaire de charger l'ensemble du jeu de données d'un coup

L'avantage d'utiliser des mini-batches



L'intérêt de séparer les données en mini-batches

- Le but est de séparer les données en mini-batches durant la phase d'optimisation
- Utiliser des mini-batches permet d'accélérer largement la vitesse de l'optimisation, les gradients étant basés sur moins de données sont plus rapides à calculer
- Cela permet aussi d'économiser de la RAM puisqu'il n'est pas nécessaire de charger l'ensemble du jeu de données d'un coup
- Enfin, cela permet globalement d'améliorer les résultats en limitant le sur-apprentissage

Implémentation générique d'une optimisation avec des mini-batches

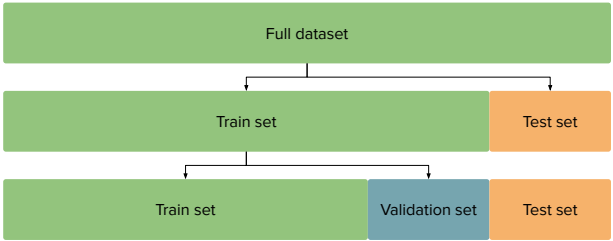
Implémentation générique d'une optimisation avec des mini-batches

```
model = tf.keras.Sequential([...])
optimizer = tf.keras.optimizers.Adam()
loss_function = tf.keras.losses.BinaryCrossentropy()
# Séparer les données x en un nombre fixé de mini-batches
split_ds = tf.data.Dataset.from_tensor_slices((x, y)).batch(batch_size)
for _ in range(nb_epochs):
    # Parcours toutes les mini-batches à chaque epoch
    for x_batch, y_batch in split_ds:
        with tf.GradientTape() as tape:
            # Obtenir les prédictions pour la mini-batch courante
            predictions = model(x_batch)
            # Calcul de la loss pour la mini-batch courante
            loss = loss_function(y_batch, predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

Jeux d'entraînement, de validation et de test

En pratique lorsqu'on veut entraîner un réseau de neurones on divise nos données en trois sets

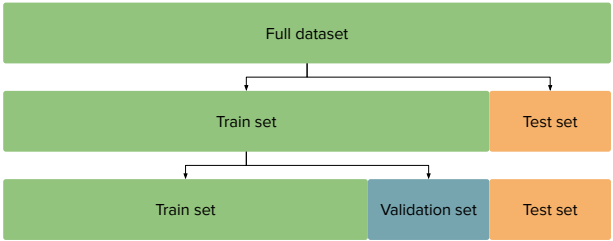
1. **Jeu d'entraînement :** Ce sont les données que le modèle utilise pour apprendre, il doit être assez large et présenter des données assez variées pour que le modèle soit capable d'apprendre des généralités qu'il pourra retrouver dans des données inconnues par la suite



Jeux d'entraînement, de validation et de test

En pratique lorsqu'on veut entraîner un réseau de neurones on divise nos données en trois sets

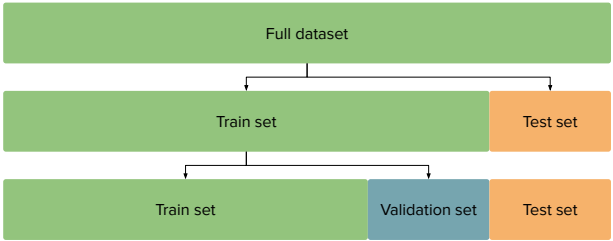
1. **Jeu d'entraînement :** Ce sont les données que le modèle utilise pour apprendre, il doit être assez large et présenter des données assez variées pour que le modèle soit capable d'apprendre des généralités qu'il pourra retrouver dans des données inconnues par la suite
2. **Jeu de validation :** Ce sont des données utilisées pour évaluer et vérifier le bon déroulement de la phase d'apprentissage, ce jeu ne nécessite pas de contenir un nombre très large de données



Jeux d'entraînement, de validation et de test

En pratique lorsqu'on veut entraîner un réseau de neurones on divise nos données en trois sets

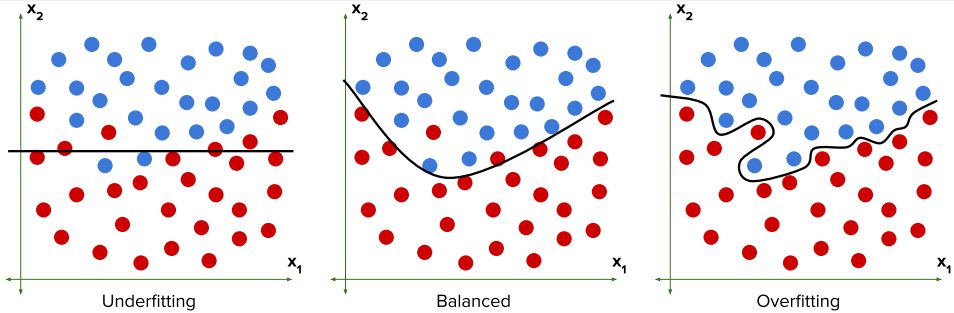
1. **Jeu d'entraînement :** Ce sont les données que le modèle utilise pour apprendre, il doit être assez large et présenter des données assez variées pour que le modèle soit capable d'apprendre des généralités qu'il pourra retrouver dans des données inconnues par la suite
2. **Jeu de validation :** Ce sont des données utilisées pour évaluer et vérifier le bon déroulement de la phase d'apprentissage, ce jeu ne nécessite pas de contenir un nombre très large de données
3. **Jeu de test :** Ce sont les données que le modèle n'aura jamais vu durant son apprentissage, ce jeu est utilisé pour tester et évaluer les performances du modèle en situation réelle



Les problèmes de sous- et sur-apprentissage

Les différentes phases d'apprentissage

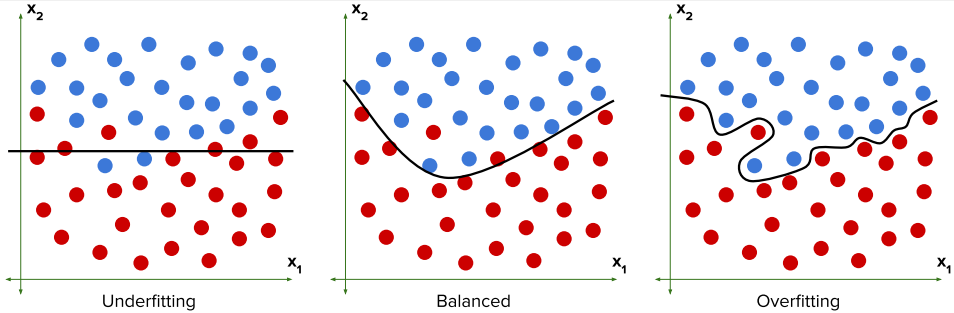
- On parle de sous-apprentissage lorsque le modèle n'a pas assez appris et n'est donc pas capable de répondre à la tâche demandée



Les problèmes de sous- et sur-apprentissage

Les différentes phases d'apprentissage

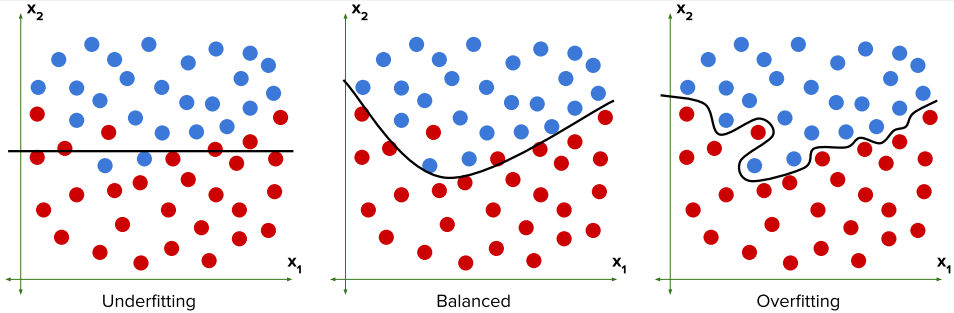
- On parle de sous-apprentissage lorsque le modèle n'a pas assez appris et n'est donc pas capable de répondre à la tâche demandée
- On parle de sur-apprentissage lorsque le modèle apprend "par coeur" les données d'entraînement et n'est donc pas capable de généraliser sur des données qu'il n'a jamais vu



Les problèmes de sous- et sur-apprentissage

Les différentes phases d'apprentissage

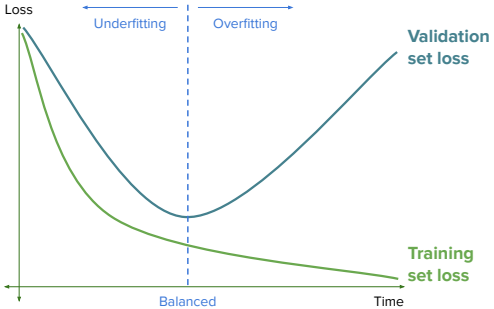
- On parle de sous-apprentissage lorsque le modèle n'a pas assez appris et n'est donc pas capable de répondre à la tâche demandée
- On parle de sur-apprentissage lorsque le modèle apprend "par coeur" les données d'entraînement et n'est donc pas capable de généraliser sur des données qu'il n'a jamais vu
- Entre ces deux phases l'apprentissage du modèle est correct



Les problèmes de sous- et sur-apprentissage

Les différentes phases d'apprentissage

- On parle de sous-apprentissage lorsque le modèle n'a pas assez appris et n'est donc pas capable de répondre à la tâche demandée
- On parle de sur-apprentissage lorsque le modèle apprend "par coeur" les données d'entraînement et n'est donc pas capable de généraliser sur des données qu'il n'a jamais vu
- Entre ces deux phases l'apprentissage du modèle est correct



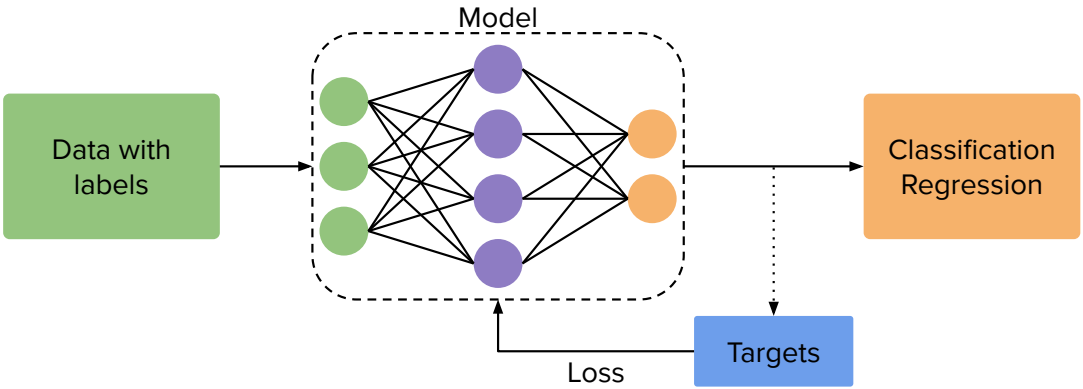
Types d'apprentissage

Différentes manière d'apprendre

Apprentissage supervisé

Objectif

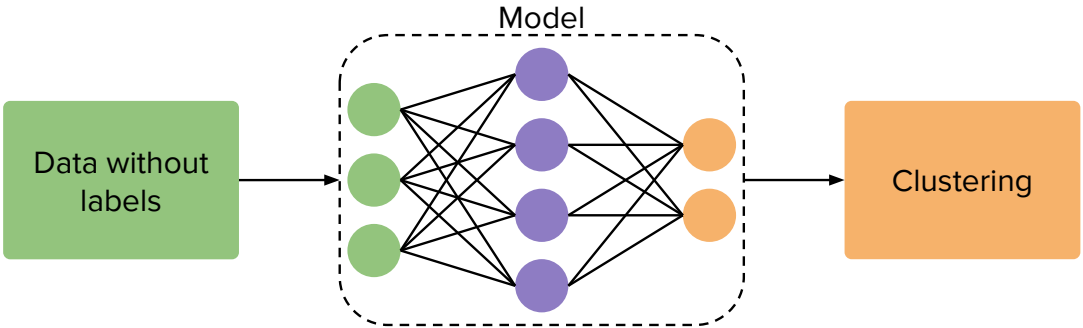
Prédire les labels de données inconnues en apprenant à partir de données connues et labelisées



Apprentissage non supervisé

Objectif

Organiser des données par groupes en apprenant à partir de données non labélisées



Apprentissage par renforcement

Objectif

Obtenir un modèle capable de réaliser l'action la mieux adaptée selon sa situation en apprenant de ses propres expériences et erreurs