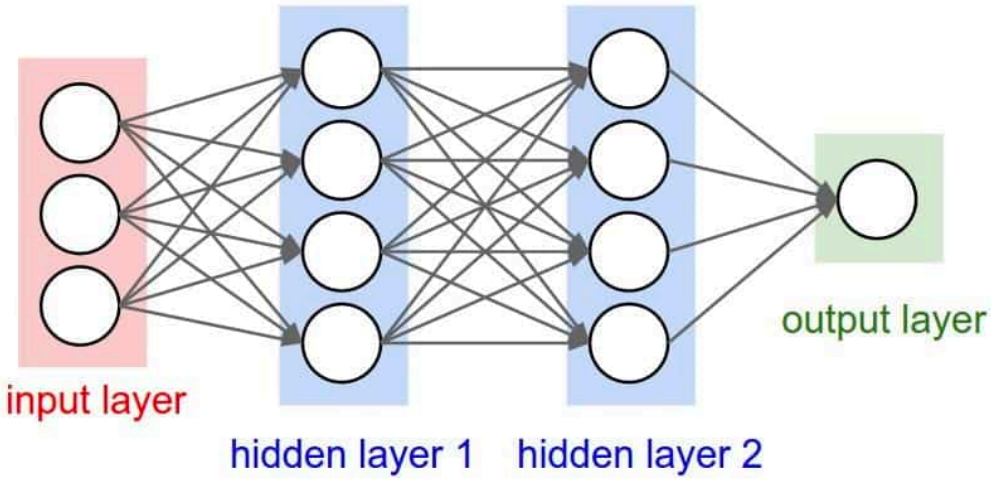
Réseau de Neurones

Les différentes parties

Un réseau de neurones est composé de 3 parties:
- La/les couche(s) d'input (input layer): qui intègre les données sous forme de matrices. C'est données, qu'elles soient des images, de la vidéo, du texte ou autre doivent donc être transformé en nombres et en matrice de différentes dimensions.
- La/les couche(s) caché(s) (hidden layer): qui vont intégré les informations venant de la couche d'input. Cette couche n'a pas de lien avec le monde extérieur. C'est pourquoi il est parfois compliqué d'expliquer les résultats de réseaux de neurones et sont appelés "black box"
- La/les couche(s) d'output (output layer): qui utilise les données venant des couches cachés pour donner la valeur à prédire et la comparé à la valeur connue lors de l'entraînement



Les connections (Weights)

- Les inputs **I** sont sous forme de matrices (exemple: (4x1) si l'on veut utiliser 4 données)
- Les connections (weights) **W** sont également sous forme de matrices et correspondent à **WI** le nombre de neurones de la couche précédente x **WI+1** le nombre de neurones de la couche suivante. Dans l'exemple au dessus les premiers **W1** sont donc égal à (3x4), les second **W2** sont égal à (4x4) et enfin les derniers **W3** à (4x1).
- L'output peut prendre différente forme en fonction du nombre de valeurs que l'on veut prédire, pour rester simple nous allons rester sur un output unique

Feedforward Propagation

- La feedforward propagation correspond au passage de l'information de la couche d'input vers la couche d'output
- L'input est d'abords multiplier par les connections W1 (matrice = (3x4))
- La somme de ces connections est ensuite réalisé par neurones (matrice = (4x1))
- Ces sommes passent par une fonction d'activation. Il y en a plusieurs comme tangente, ReLu on a donc une matrice = (4x1)
- Puis on continue ce cheminement jusqu'à l'output

Calculer l'erreur (Cost function)

- A la fin de la feedforward propagation on obtient une valeur unique de l'output pour notre réseau
- Il faut donc la comparer en faisant **Erreur = Valeur connue - Valeur prédite**, il existe différent type de calcul d'erreur. Ce dernier calcul s'appelle la **Cost or Loss function** la plus connu est la mean square error (MSE). L'objectif est de diminué cette erreur au cours de l'entrainement.
- On utilise ensuite la dérivative de l'erreur qui correspond à l'ampleur du changement de la valeur de la fonction (valeur de sortie) par rapport à un petit changement de son argument (valeur d'entrée). Elle se calcule de la façon suivante:

Erreur derivative = output * (1.0 - output) x Erreur

Back-propagation

- La back-propagation correspond au passage de l'erreur depuis la couche d'output vers la couche d'input
- Elle se calcule de la façon suivante:
  - Erreur de la couche = Erreur connu - Erreur prédite
  - Dérivative de l'erreur de la couche (Erreur de la couche x (1 - Erreur de la couche)
  - Dot product de la derivative et des connections (w-1)

On obtient alors une matrice **dW** de même dimension pour toutes les connections (W1=4x1), etc. En général on normalise cette valeur par N, N étant le nombre d'exemple que l'on a dans notre training set.

Mise à jour des connections (Weight update)

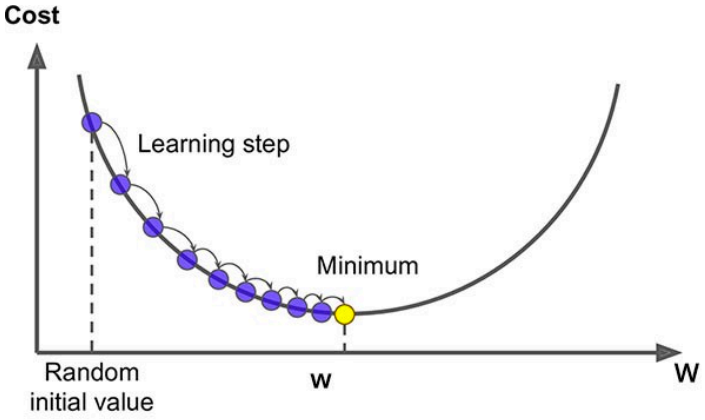
Pour faire la mise à jour des connections il suffit de suivre la formule suivante

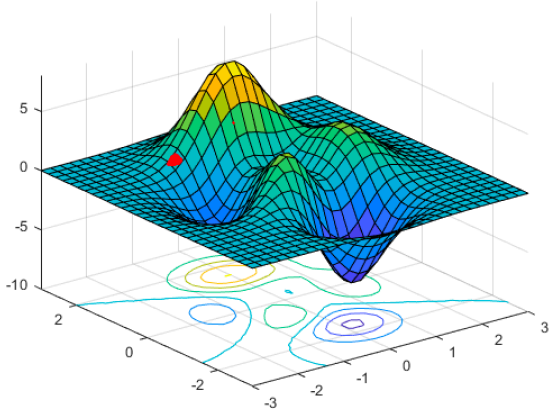
$$W\_update = W - learning\_rate * dW$$

W étant nos weights (connections)
Le learning rate est une valeur qui va mesurer avec quel force on va faire la mise à jours des connections
dW représente la matrice calculé durant la back propagation

Gradient descent

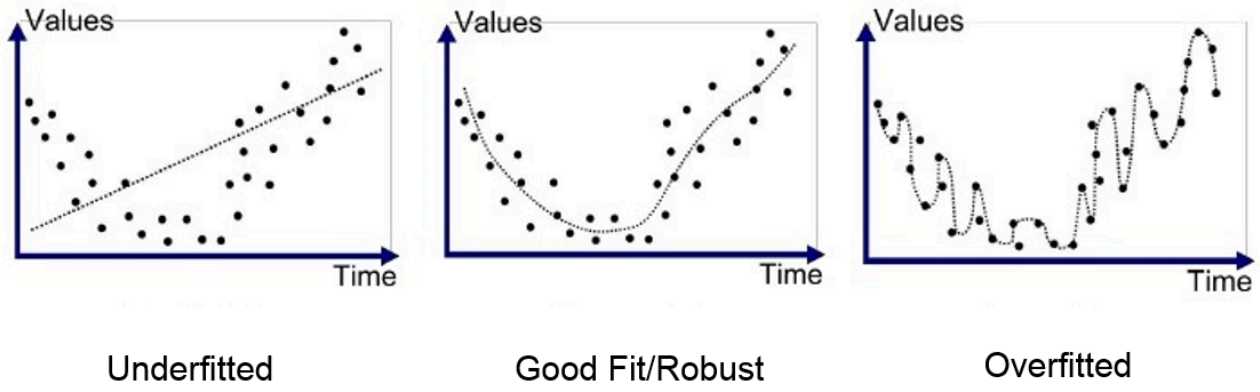
Ainsi au cours de l'entrainement, la mise à jour des connections va permettre de réduire l'erreur. On appelle cette méthode le gradient descent qui est donc l'optimisation par minimisation de l'erreur.





Learning Curve

Comme on entraine notre réseau de neurones sur le training set et qu'il est testé sur notre training set, on peut calculer l'erreur sur les deux en fonctions des époques.
Il est possible que le réseau apprenne trop sur le training set et qu'il mémorise la totalité de la variance. Dans ce cas il préformera faiblement sur le testing set, on appelle cela **overfitting**.
De même s'il n'apprends pas assez bien sur le training set, il performera aussi faiblement sur le testing set, on appelle cela **underfitting.**
Il est donc indispensable de trouver un compromis entre trop d'entrainement et pas assez.



Underfitted | Good Fit/Robust | Overfitted

Limiter l'overfitting

Il existe plusieurs technique pour limité l'overfitting:

- Avoir le maximum de données. Plus on a de données, plus on a en général de variance et donc moins le réseau de neurones ou tout autre technique de machine learning pourra apprendre la totalité des données. Il est également possible de créer de nouvelles données par exemple avec le GAN.
- Utiliser un dropout. Comme nous l'avons vu un réseau de neurones est composé de plusieurs neurones. Il est possible de rendre certain neurones non fonctionnellement de façon aléatoire durant l'entrainement. Si cela paraît étrange, cela permet qu'un neurone ne soit pas prépondérant dans la compréhension d'un pattern. A chaque époque le neurone peut donc être ou ne pas être fonctionnel
- Batch Normalization. Durant l'entrainement d'un réseau de neurones on peut avoir un update des connections après chaque exemple, ou après un paquet d'exemple appelé batch. La valeur du batch est donc comprise entre 1 si on update après chaque exemple, et le nombre total du nombre d'exemple que l'on a. La normalisation de ces batches est donc tout simplement que la mise à jour des connections sera comprise entre des valeurs standard (normalisées) du batch proposé.