😀

Nous allons aborder les concepts de manière d'un point de vue théorique, puis appliqué et finirons sur leur rôle dans le réseau de convolutions. Nous verrons ainsi de la structure globale de ce type de réseaux à l'opération la plus basique sur laquelle il est construit, la convolution.

Definitions:

- **Neurone** - Il s'agit d'un élément de base d'un réseau national. Il prend des valeurs pondérées, effectue des calculs mathématiques et produit des résultats. Il est également appelé unité, nœud ou perceptron.
- **Entrée** - Il s'agit des données/valeurs transmises aux neurones.
  Réseau neuronal profond (RNP) - Il s'agit d'un RNA comportant de nombreuses couches cachées (couches situées entre la couche d'entrée (première) et la couche de sortie (dernière)).
- ***Poids*** - Des paramètres internes accomplissant l'analyse. Ils symbolisent la compréhension de l'information.Ces valeurs expliquent la force (degré d'importance) de la connexion entre deux neurones.
- ***Biais*** - Il s'agit d'une valeur constante ajoutée à la somme du produit entre les valeurs d'entrée et les poids respectifs. Il est utilisé pour accélérer ou retarder l'activation d'un nœud donné.
- ***Fonction d'activation*** - fonction utilisée pour introduire le phénomène de non-linéarité dans le système NN. Cette propriété permettra au réseau d'apprendre des modèles plus complexes.
- ***Backword Propagation*** - une méthode d'apprentissage supervisé utilisée par les réseaux nationaux pour mettre à jour les paramètres afin de rendre les prédictions du réseau plus précises. Le processus d'optimisation des paramètres est réalisé à l'aide d'un algorithme d'optimisation appelé descente de gradient (ce concept sera très clair au fur et à mesure de votre lecture).

Dans quel cas utiliser un réseau de neuronne?

Il est admit que **si le problème est statistique, alors un réseau neuronal pourra le résoudre**. Pour des problèmes plus complexes, on combinera cette technique avec d’autres algorithmes.

1. Classification : Les réseaux de neurones peuvent être utilisés pour la classification de données. Par exemple, pour classer des images en différentes catégories (par exemple, chat ou chien), ou pour classer des courriels en spam ou en non-spam.
2. Reconnaissance de motifs : Les réseaux de neurones sont largement utilisés pour la reconnaissance de motifs dans les données. Cela peut inclure la reconnaissance d'images, la reconnaissance de la parole, la reconnaissance d'écriture manuscrite, etc.
3. Prévision et séries temporelles : Les réseaux de neurones sont souvent utilisés pour la prévision et l'analyse des séries temporelles, où l'objectif est de prédire les valeurs futures d'une série de données temporelles. Cela peut être utilisé pour la prévision de la demande, la prévision des prix, la prévision des ventes, etc.
4. Traitement du langage naturel : Les réseaux de neurones sont largement utilisés dans le traitement du langage naturel pour des tâches telles que la traduction automatique, la génération de texte, la classification de texte, la reconnaissance d'entités nommées, etc.
5. Optimisation et recherche opérationnelle : Les réseaux de neurones peuvent également être utilisés pour résoudre des problèmes d'optimisation et de recherche opérationnelle, tels que la planification des ressources, la logistique, l'affectation de tâches, etc.

I - Neuronne

a) Explication globale

Un neurone possède un **potentiel d’action**. Il s’agit d’un pique d’énergie, s’il est au-dessus d’un certain seuil, alors il va déclencher une succession de déclenchement vers d’autres neurones.

Certains ingénieurs se sont inspirés de ce système pour nos ordinateurs pour créer **des réseaux neuraux artificiels**. L’idée est de créer **des noeuds qui sont des connexions similaires aux neurones dans notre cerveau.**

Maintenant, faisons un zoom sur un seul neurone.

Il existe deux types de neurones, concentrons-nous sur un seul type : **le neurone produit scalaire**
. L'autre type est le neurone de distance.Pour simplifier, un **neurone est un joli cercle, un "Perceptron"**.

Pour que l'IA puisse comprendre la valeur du neurone, il a besoin de traduire la réalité en chiffre.

Or, u**n neurone doit retenir un chiffre pour pouvoir se multiplier aux autres**. Jusque là, c'est logique, on a des cercles avec des chiffres qu'on additionne et multiplie entre eux pour obtenir des probabilités sur un problème donné. **Un neurone possède une valeur et les connexions entre chaque neurone ont une valeur nommée "poids"** *(ou "weight" dans la littérature)*. Cela veut donc dire qu'on va pouvoir utiliser ces nombres.

b) ***Neurone artificiel - Explication mathématique sur un neurone***

Disons qu'un neurone représente un nombre entre 0 et 1. Par exemple, le réseau commence par un packet de neurones qui correspondent chacun 28 pixels et 28 pixels. Cela fait 784 neurones au total et chacun a un nombre qui correspond à une valeur de gris correspondant au pixel.

Le nombre à l'intérieur du neurone s'appelle son activation and l'image que nous pouvons avoir en tête c'est que chaque neuronne s'allume.

Un neurone artificiel prend des valeurs d'entrées (il peut y en avoir plusieurs) auxquelles sont attribués des poids. À l'intérieur du nœud, les entrées pondérées sont additionnées et une fonction d'activation est appliquée pour obtenir les résultats. La sortie du nœud est transmise aux autres nœuds ou, dans le cas de la dernière couche du réseau, la sortie est la sortie globale du réseau.

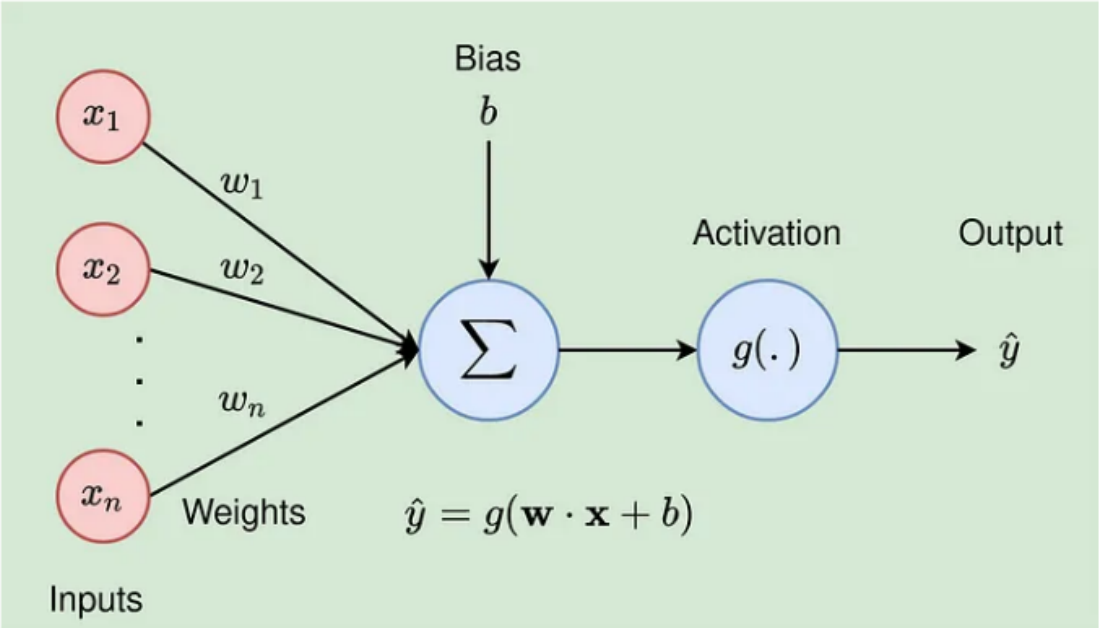


Figure 1: An artificial neuron with n input values (Source: Author).

II - Réseaux de neurones

A) Théorie

Le réseau neuronal est un système composé de nombreux neurones empilés en couches. Nous lui fournissons des données en entrée, et il en sort une analyse.

Il est constitué de couches interconnectées de petites unités appelées nœuds qui effectuent des opérations mathématiques pour détecter des modèles dans les données. Les algorithmes des réseaux neuronaux artificiels sont conçus de manière à imiter le fonctionnement des neurones humains.

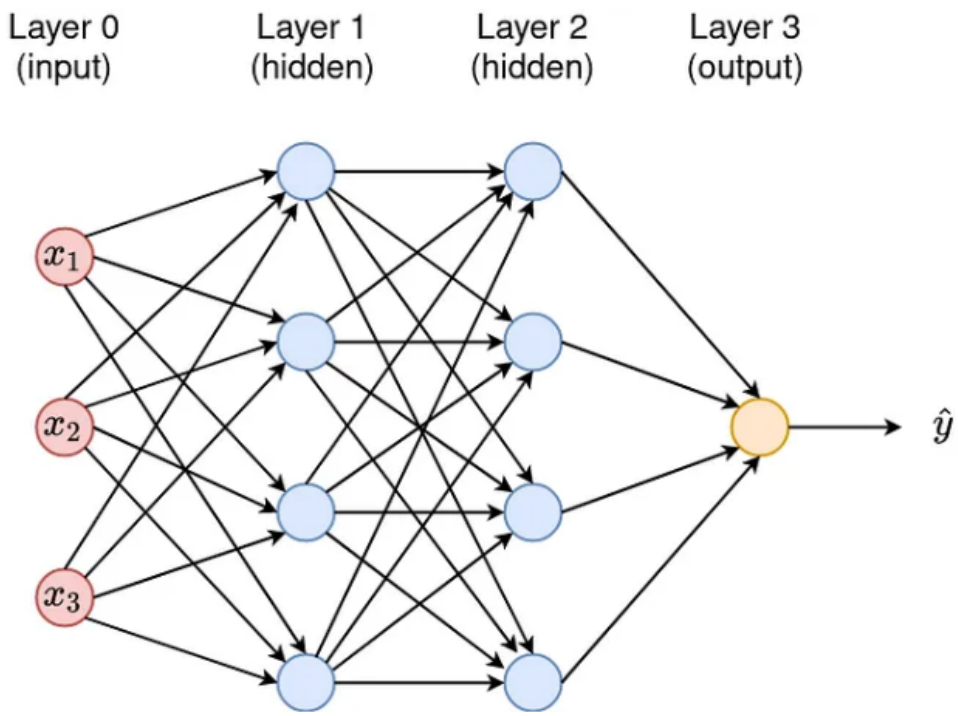


Figure 2: A Neural network with 3 input features, two hidden layers with 4 nodes each and one-value output. The nodes are densely connected- each node is connected to all the neurons in the immediate previous layer. Each connection has weights expressing the strength of the connection between any two nodes. Every node performs the computation described in Equation 1 except the nodes at the input layer (Source: Author).

La première couche, la couche d'entrée (que nous appellerons couche 0), n'effectue aucun calcul, si ce n'est qu'elle transmet les valeurs d'entrée. C'est pourquoi, lorsque l'on compte le nombre de couches d'un réseau national, on ne tient pas compte de la couche d'entrée. La figure 1 ci-dessous est donc un réseau à 2 couches.

La couche de sortie calcule la sortie finale du réseau. La ou les couches situées entre la couche d'entrée et la couche de sortie sont appelées ***hidden layers***. Le réseau national de la figure 1 ci-dessous est décrit comme un réseau 3-4-1 avec 3 unités dans la couche d'entrée, 4 unités dans chacunes des hidden layers et une sortie à valeur unique.

Nous entraînons notre réseau de neurones en lui indiquant ce qu'il doit répondre à chaque fois qu'on lui donne quelque chose en entrée. Ce qui lui permet de s'adapter et de "comprendre", ce sont ses **poids.** Comme indiqué pécédemment, ce sont des paramètres internes accomplissant l'analyse. Ils symbolisent la compréhension de l'information. Cela permet d'obtenir de meilleures prédictions.

B) Architecture

La conception du réseau neuronal est souvent appelée architecture du réseau neuronal. Dans la suite de la série, nous utiliserons ces termes (conception du réseau neuronal et architecture du réseau neuronal) de manière interchangeable.

La structure des données influence l'architecture choisie pour la modélisation. En effet, **le nombre de caractéristiques de l'ensemble de données est égal au nombre de neurones de la couche d'entrée**. Dans notre exemple, illustré dans la figure ci-dessous, nous avons 3 caractéristiques et la couche d'entrée de l'architecture doit donc comporter 3 neurones.

Le nombre de couches cachées affecte le processus d'apprentissage et est donc choisi en fonction de l'application. Un réseau comportant plusieurs couches cachées est appelé réseau neuronal profond (DNN). Dans cet exemple, nous avons affaire à un réseau neuronal peu profond, pour ainsi dire. Nous avons 3 neurones dans la couche d'entrée, 4 neurones dans la couche cachée et 1 neurone dans la sortie - un réseau neuronal 3-4-1.

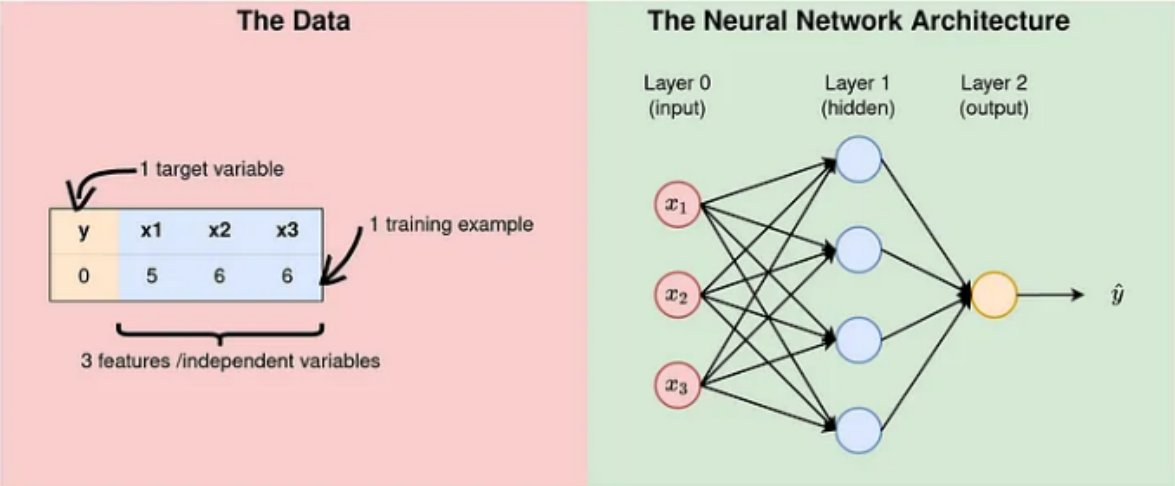
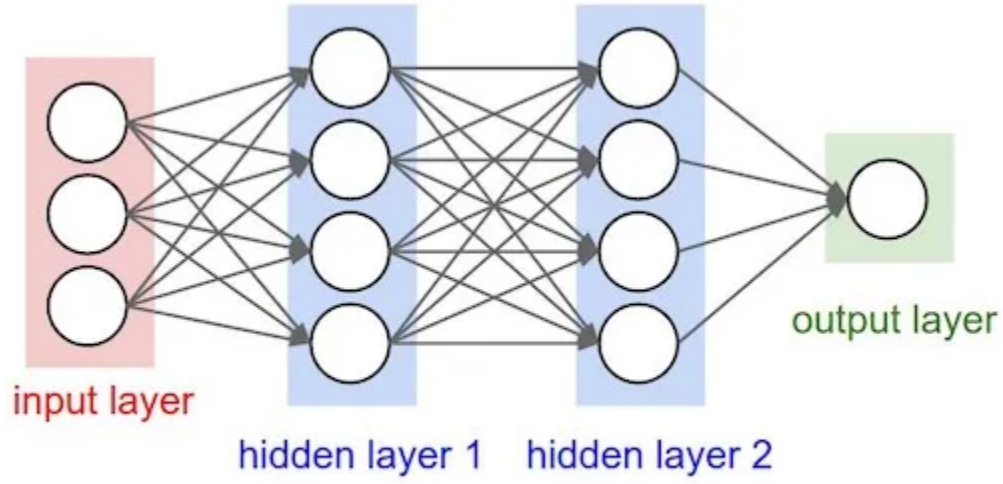


Figure 1 — Left: the data, right: the neural network architecture (Source: Author).

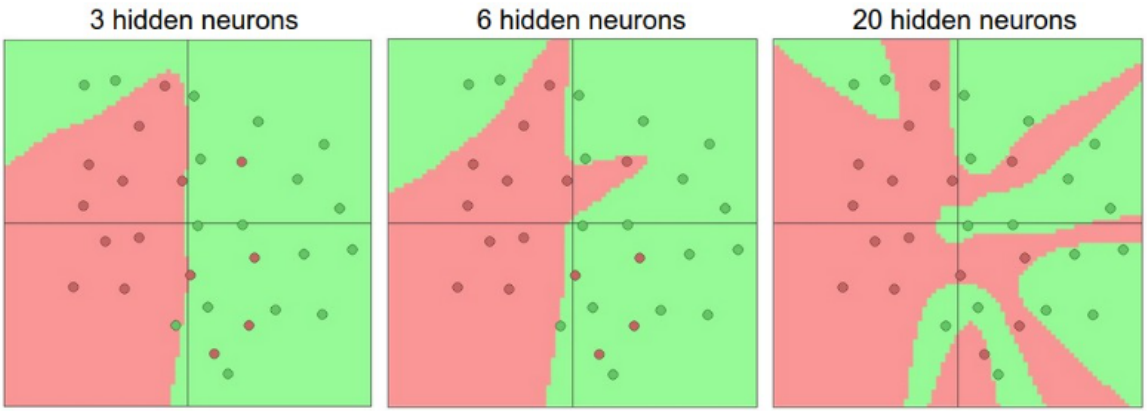
C) Hidden Layers

Rendons maintenant notre réseau un peu plus complexe. Nous en avons dorénavant deux. Plus le nombre de couches est élevé, plus la prise de décision peut être nuancée.



(Source: Stanford CS231n)

Les réseaux portent souvent des noms différents : réseaux feedforward profonds, réseaux neuronaux feedforward ou perceptrons multicouches (MLP). On les appelle réseaux feedforward parce que l'information circule dans une direction générale (vers l'avant), où des fonctions mathématiques sont appliquées à chaque étape. En fait, ils sont appelés "réseaux" en raison de cette chaîne de fonctions. La longueur de cette chaîne donne la profondeur du modèle, et c'est en fait de là que vient le terme "profond" dans ***l'apprentissage profond !*** Le nombre de couches d'un NN détermine la ***profondeur du réseau***. Sur cette base, les réseaux neuronaux comportant de nombreuses couches cachées sont appelés réseaux neuronaux profonds (RNP).



Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.

Le diagramme ci-dessus montre que les réseaux neuronaux comportant davantage de neurones peuvent exprimer des fonctions plus complexes. Toutefois, il s'agit à la fois d'une bénédiction (puisque nous pouvons apprendre à classer des données plus compliquées) et d'une malédiction (puisqu'il est plus facile de suradapter les données d'apprentissage). Il y a suradaptation lorsqu'un modèle à forte capacité s'adapte au bruit des données plutôt qu'à la relation sous-jacente (supposée). Par exemple, le modèle à 20 neurones cachés s'adapte à toutes les données d'apprentissage, mais au prix d'une segmentation de l'espace en de nombreuses régions de décision rouges et vertes disjointes. Le modèle à 3 neurones cachés n'a que le pouvoir de représentation nécessaire pour classer les données dans les grandes lignes. Il modélise les données comme deux taches et interprète les quelques points rouges à l'intérieur de la grappe verte comme des valeurs aberrantes (bruit). Dans la pratique, cela pourrait conduire à une meilleure généralisation sur l'ensemble de test.

D) Stratégie

Sur la base de notre discussion ci-dessus, il semble que l'on puisse préférer des réseaux neuronaux plus petits si les données ne sont pas assez complexes pour éviter le surajustement. Il existe en effet de nombreux autres moyens d'éviter le surajustement dans les réseaux neuronaux, que nous aborderons ultérieurement (régularisation L2, abandon, bruit d'entrée, etc.). Dans la pratique, il est toujours préférable d'utiliser ces méthodes pour contrôler le surajustement plutôt que le nombre de neurones.

La raison subtile en est qu'il est plus difficile d'entraîner des réseaux plus petits avec des méthodes locales telles que la descente de gradient : Il est clair que leurs fonctions de perte présentent relativement peu de minima locaux, mais il s'avère que nombre de ces minima sont plus faciles à atteindre et qu'ils sont mauvais (c'est-à-dire qu'ils entraînent une perte élevée). Inversement, les réseaux neuronaux plus importants contiennent beaucoup plus de minima locaux, mais ces minima s'avèrent bien meilleurs en termes de perte réelle. Les réseaux neuronaux n'étant pas convexes, il est difficile d'étudier ces propriétés mathématiquement, mais certaines tentatives de compréhension de ces fonctions objectives ont été faites, par exemple dans un article récent intitulé The Loss Surfaces of Multilayer Networks (Surfaces de perte des réseaux multicouches). Dans la pratique, on constate que si l'on forme un petit réseau, la perte finale peut présenter une grande variance - dans certains cas, on a de la chance et on converge vers un bon endroit, mais dans d'autres cas, on se retrouve piégé dans l'un des mauvais minima. En revanche, si vous entraînez un grand réseau, vous commencerez à trouver de nombreuses solutions différentes, mais la variance de la

perte finale obtenue sera beaucoup plus faible. En d'autres termes, toutes les solutions sont à peu près aussi bonnes les unes que les autres et dépendent moins de la chance d'une initialisation aléatoire.

Utilisez la fonction loss

En apprentissage supervisé, la notion principale est celle de **perte d'information** (*loss* en anglais) due à l'approximation dont je viens de parler. Elle détermine à quel point notre modélisation du phénomène, qui est une approximation de la réalité, perd de l'information par rapport à la réalité observée à travers les données d'exemple.

La grande majorité des algorithmes d'apprentissage supervisé utilisent cette fonction de perte ! L'apprentissage se résume en fait souvent à une méthode itérative qui **converge vers un minimum** de cette fonction. *Plus la perte d'information diminue, plus on se rapproche de la réalité et meilleur est notre modèle.*

On peut retrouver beaucoup de types de perte. Je vais vous en présenter deux ici, pour vous donner une idée des types d'approche qu'on peut avoir.

d) Exemple

1)

Par exemple, nous pouvons utiliser un réseau de neurones pour détecter les chutes à vélo. Avec des capteurs dans le casque, nous lui donnons en entrée la vitesse, l'accélération et l'orientation du cycliste, et le réseau prédit si oui ou non il y a une chute. Plus la valeur de la sortie est proche de 1 plus la probabilité d'une chute avérée est forte.

2)

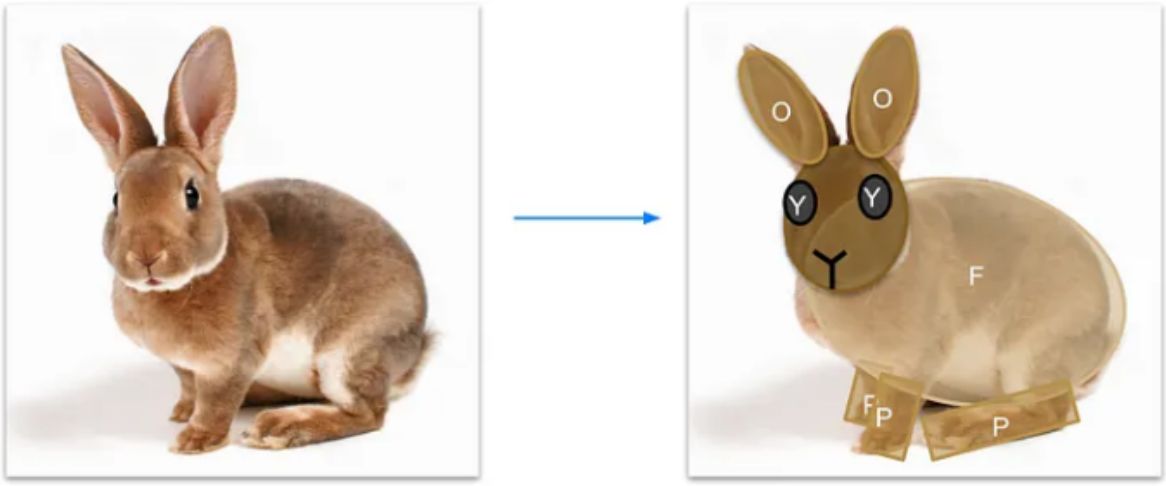
Imaginons qu'on veuille déduire la préférence politique d'un futur électeur ! On aurait déjà plein d'informations, mais pour celles dont il manque un paramètre, que fait-on ? On utilise un réseau de neurones ! 🙂

3)

Le nombre de paramètres rend donc l'analyse pixel par pixel totalement inefficace du fait de la complexité de l'ensemble. Notre objectif va être de convertir l'image en un format condensé qui sera bien plus digeste. Nous passons par une étape de "*preprocessing.*

L’idée est d’analyser l’image en **caractéristiques** (appelées également ***features***). Si je vous demandais ce qui caractérise un lapin, vous me répondriez probablement : *Deux grandes oreilles, une petite truffe et deux yeux globuleux, quatre pattes, un corps rondelet et plein de fourrure, et un pompon en guise de queue !*

C’est exactement notre but : détecter dans l’image ce qui fait le lapin (avec ses caractéristiques : les deux oreilles, la truffe, les deux yeux …).



Description de l’image en termes de caractéristiques par zones.

Description de l’image en termes de caractéristique associée à chaque pixel

L’opération de Pooling

Pour réaliser ses prédictions sur une image, le réseau de neurones n’a pas besoin de connaître tous les pixels qui concernent une information, mais plutôt le “ratio” d’importance et la localisation de celle-ci. Par exemple la grande quantité de pixels “*patte*” ou “*fourrure*” ne sont pas nécessaires. Pour réduire ceci, nous utilisons l’opération de **Max Pooling.**

.
C’est un sous-échantillonnage. Elle consiste à réduire la dimension de l’image tout en conservant l’information la plus importante. Comme le montre l’exemple suivant, l’image est découpée en “tuiles” de *2x2* pixels dans lesquelles seul le pixel de plus grande valeur est conservé.

Deux étapes de pooling avec un rectangle de taille 2×2.

En enchaînant deux **Max Pooling** l’image *32x3*est transformée en *8x8*, soit 16 fois moins d’informations.

La décomposition en caractéristiques

L’analyse met en corrélation des pixels proches pour obtenir des caractéristiques. Par exemple, des pixels adjacents à la frontière entre deux couleurs très différentes sont interprétables en tant que “bord”. En continuant, nous allons mettre en corrélation des caractéristiques proches pour en déterminer de nouvelles plus complexes. De fait, les ensembles de bords forment des contours, l’ensemble de couleur unie forme un intérieur, …Nous interposons des phases de **Pooling**
qui nous permettent de simplifier le trop-plein d’informations.

La caractéristique “Oreille” est en vérité l’assemblage de toutes les caractéristiques à des niveaux d’analyses différents
. En effet elle est le résultat des caractéristiques “Contours” et “Intérieurs”, avec la caractéristique “Contours” étant elle aussi résultat des caractéristiques “Bords Droit, Gauche et Haut”, elles-mêmes venant de la première analyse des pixels. On part donc d’un ensemble de caractéristiques très simples relevant des pixels même pour arriver à

une ou plusieurs caractéristiques bien plus complexe. **Chaque caractéristique est issue d'un ensemble de sous-caractéristiques, dont la racine est le pixel.**

III - Le produit de Convolution

Tout ce que nous venons de voir va être résumé ici grâce à une opération très simple qui est le support de l'ensemble de ces transformations. C'est la convolution.

a) Definition

Dans les réseaux neuronaux, les réseaux neuronaux convolutifs CNNs constituent l'une des principales catégories pour la reconnaissance et la classification d'images. La détection d'objets, la reconnaissance de visages, etc. sont quelques-uns des domaines dans lesquels les CNN sont largement utilisés.

La convolution est la première couche qui permet d'extraire les caractéristiques d'une image d'entrée. La convolution préserve la relation entre les pixels en apprenant les caractéristiques de l'image à l'aide de petits carrés de données d'entrée. Il s'agit d'une opération mathématique qui prend deux entrées telles que la matrice de l'image et un filtre ou un noyau.

La convolution préserve la relation entre les pixels en apprenant les caractéristiques de l'image à l'aide de petits carrés de données d'entrée. Il s'agit d'une opération mathématique qui prend deux entrées telles qu'une matrice d'image et un filtre ou un noyau.

La convolution d'une image avec différents filtres permet d'effectuer des opérations telles que la détection des contours, le flou et la netteté en appliquant des filtres. L'exemple ci-dessous montre diverses images de convolution après l'application de différents types de filtres (noyaux).

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

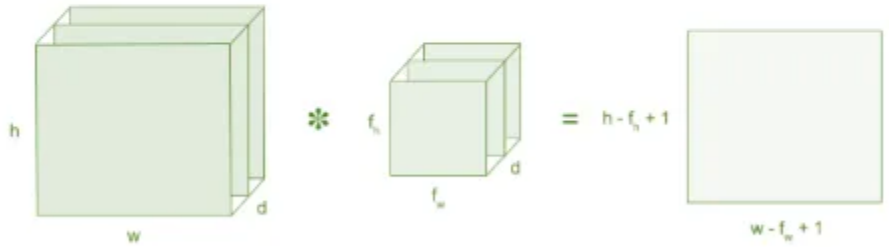


Figure 3: Image matrix multiplies kernel or filter matrix

Exemple Technique

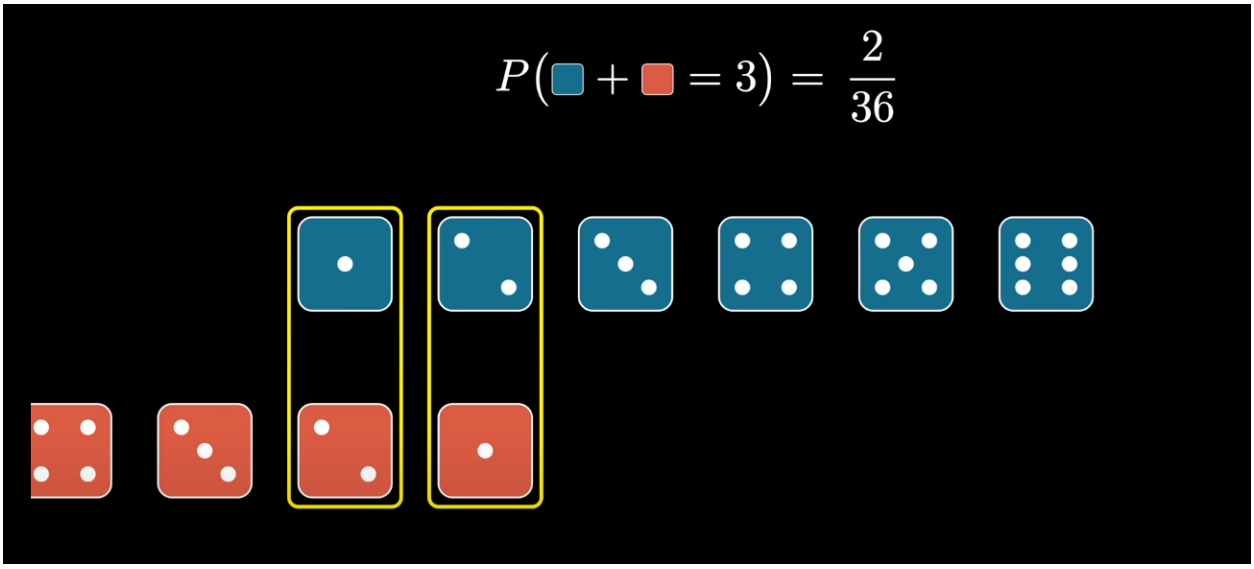
Les classifications d'images NN prennent une image en entrée, la traitent et la classent dans certaines catégories (par exemple, chien, chat, tigre, lion). Les ordinateurs voient une image d'entrée comme un tableau de pixels et cela dépend de la résolution de l'image. En fonction de la résolution de l'image, il verra h x w x d(h = hauteur, w = largeur, d = dimension). Par exemple, une image de 6 x 6 x 3 matrice de RVB (3 correspond aux valeurs RVB) et une image de 4 x 4 x 1 matrice d'image en niveaux de gris.

Techniquement, les modèles d'apprentissage profond CNN à former et à tester, chaque image d'entrée passera par une série de couches de convolution avec des filtres (Kernals), la mise en commun, les couches entièrement connectées (FC) et l'application de la fonction Softmax pour classer un objet avec des valeurs probabilistes entre 0 et 1. La figure ci-dessous est un flux complet de CNN pour traiter une image d'entrée et classer les objets sur la base des valeurs.

Pourquoi passer par la convolution et pas une autre opération?

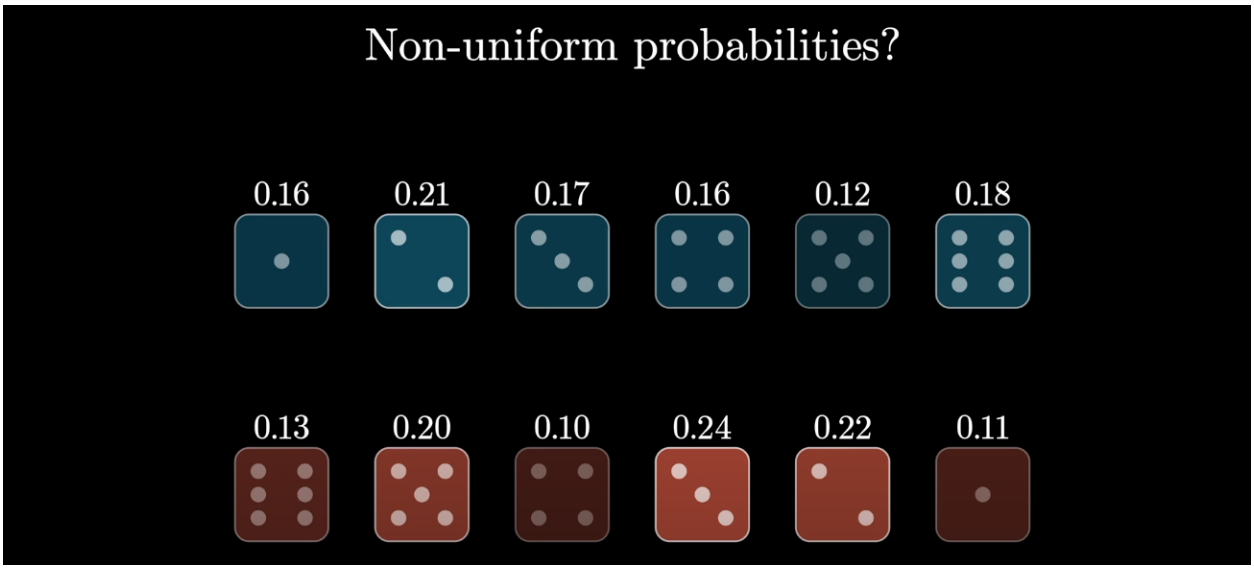
Supposons que je donne deux listes différentes de nombres, disons a = [1,2,3,4] et b = [5,6,7,8] et réfléchissons à toutes les façons de combiner ces deux listes pour obtenir une nouvelle liste de nombres. Nous pourrions les additionner, les multiplier par exemple.

Contrairement aux idées précédentes, la convolution n'est pas quelque chose de simplement hérité d'une opération que l'on peut faire sur les nombres. C'est quelque chose de nouveau dans le contexte des listes de nombres ou des fonctions de combinaison. Ils apparaissent partout, ils sont omniprésents. Il s'agit d'une construction essentielle dans la théorie des probabilités. On les résout souvent en résolvant des équations différentielles.



Nous imaginons deux séries de possibilités différentes, chacune sur une ligne. Nous retournons ensuite la deuxième rangée, de manière à ce que toutes les paires différentes soient alignées verticalement. Si nous faisons glisser le tout vers la droite, la paire unique dont la somme est égale à deux est la seule à s'additionner. En général, les différentes valeurs de décalage de ce tableau inférieur révèlent toutes les paires différentes qui ont une somme distincte. En ce qui concerne les probabilités, ce n'est pas intéressant car nous comptons simplement le nombre de résultats dans chacune de ces catégories. Mais cela repose sur l'hypothèse implicite qu'il y a une chance égale pour que ces visages apparaissent.

Mais qu'en est-il si les probabilités ne sont pas uniformes pour chaque ensemble ?



Pour calculer la probabilité d'en obtenir trois, il faut examiner les deux paires distinctes où cela est possible. Et on multiplie les probabilités correspondantes et ces deux produits ensemble.

Et bien sûr, en informatique, nous pensons aux couleurs en termes de couleurs en termes de vecteurs.

$$\frac{1}{9}\begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}$$

Lorsque je multiplie toutes ces valeurs par 1/9e et que je les additionne, cela donne une moyenne de chaque canal de couleur pour le pixel correspondant, qui est définie comme étant cette somme. L'effet global, comme nous le faisons pour chaque pixel de l'image, est que chacun d'entre eux se fond dans ses voisins.

On dit que l'image créée est une convolution de la grille de valeurs.

Si nous modifions les lots pour les porter à 5 et que nous changeons les valeurs de la grille, la différence n'est pas tant la taille, mais plutôt la valeur au milieu qui est beaucoup plus grande que la valeur vers les bords. Nous pouvons observer qu'il s'agit d'échantillons d'une courbe de Bell, connue sous le nom de distribution gaussienne.

***Ainsi, lorsque nous multiplions toutes ces valeurs par les valeurs correspondantes, nous donnons beaucoup plus de poids au pixel central et beaucoup moins à ceux qui se trouvent sur les bords.

b) Kernel

L'opération de convolution consiste à appliquer un filtre sur une image. Le filtre est une petite matrice de taille variable appelée "*kernel*" (ou "*kernel de convolution*").

Nous appliquons ce *kernel* sur chaque portion de l'image comme si elle représentait une fenêtre que l'on déplaçait sur chacun des pixels. L'application fonctionne de la même manière que le **Pooling** (la seule différence est dans la quantité de pixels de décalage de la fenêtre au cours de son déplacement).

Le *kernel* positionne son centre sur le pixel que l'on souhaite modifier, et applique un simple **produit scalaire** entre la fenêtre de l'image induite et le *kernel* lui-même. On obtient une image de taille légèrement réduite (pour une taille de kernel de *3x3*, l'image est logiquement réduite de 1 sur chaque côté, donc de 2 sur chaque axe, voir animation précédente). Le produit scalaire entre ces deux parties a la propriété d'être maximal lorsque les deux sont identiques.

Pour calculer un Perceptron, voici trois étapes toute simples :

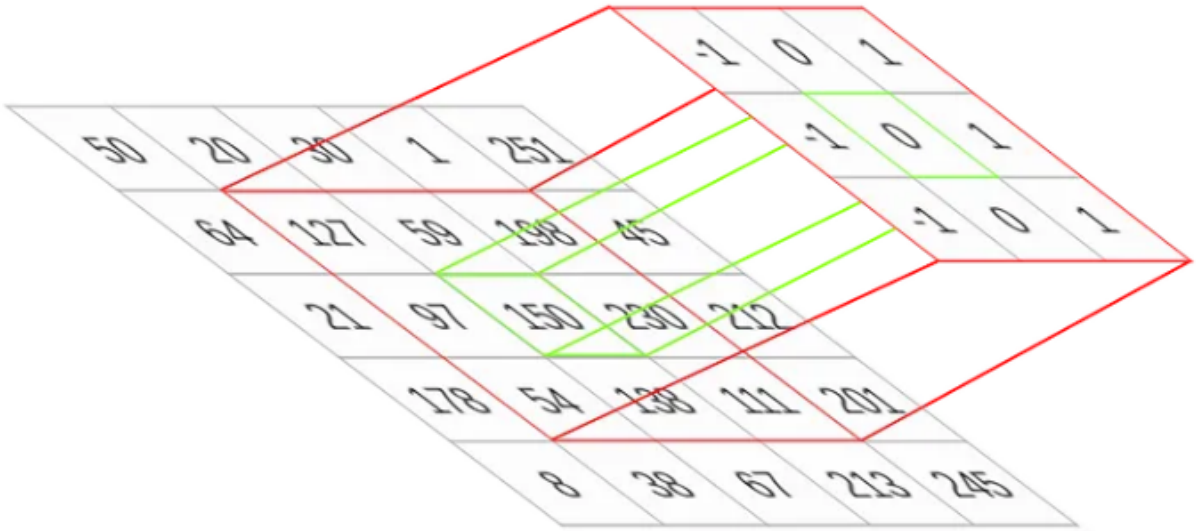
- On multiplie un Perceptron et son poids en l'additionnant aux autres :

Cela donne : (0.10 x 4.0) + (0.20 x -5.0) + (0.30 x 6.0) = 1.20

- Chaque Perceptron a un "faux neurone", c'est-à-dire une constante qui est importante pour l'apprentissage.

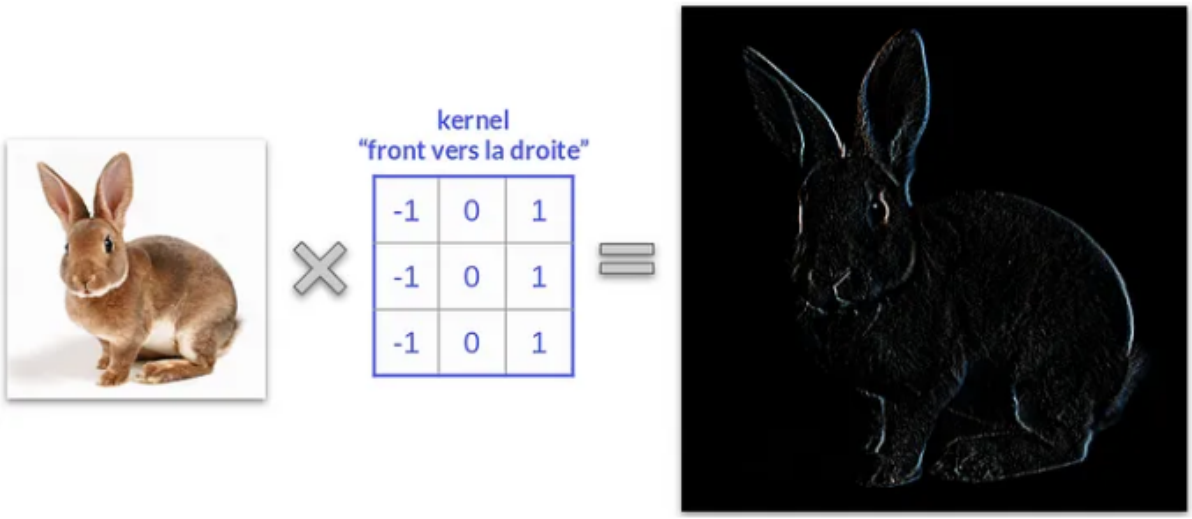
Cela donne : 2.0 + 1.2 = 3.2

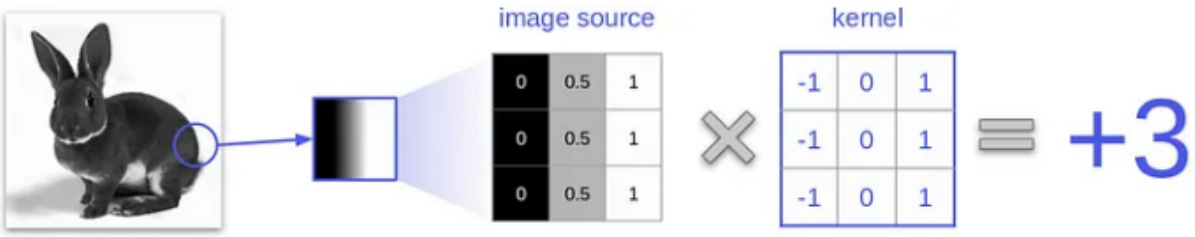
Sur l'exemple de l'image ci-contre, notre produit scalaire sera donc :(**-1***127)+(**0***59)+(**1***198)+(**-1***97)+(**0***150)+(**1***230)+(**-1***54)+(**0***138)+(**1***111) = **261**



Exemple d'application d'un kernel de convolution sur une matrice.

Comme ce sont des couleurs RGB, nos valeurs sont comprises entre 0 et 255 : le résultat 261 se transforme donc en 255. L'emplacement vert dans la nouvelle image n'aura donc plus la valeur 150 mais **255**. Nous allons nommer ce kernel "front vers la droite". Une fois appliqué à l'ensemble de l'image, il donne le résultat ci-dessous :





Nous venons donc de voir comment la convolution transforme l’image en mettant en valeur certaines de ses composantes : les caractéristiques. L’image résultante est nommée **carte de caractéristiques**(en anglais ***feature map***).

Ensuite, la convolution de la matrice d'image mario est multipliée par la matrice de filtre 3 x 3, appelée "feature mapping", comme indiqué ci-dessous.

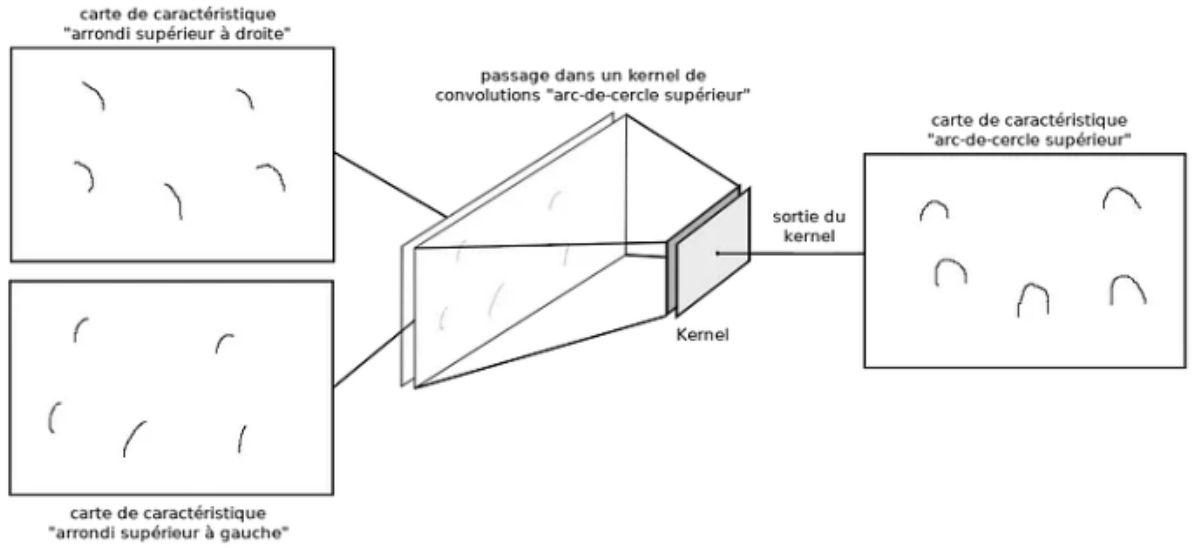
Des cartes de convolutions à une nouvelle caractéristique

Le but serait d’avoir un mécanisme qui pourrait combiner les pixel en bords ou en pattern puis en digits. Disons que le but est qu’un neuronne particulier dans la seconde couche pour savoir si oui ou non l’image a les bords dasn cette région. Ici, la question est de savoir quels paramètres est-ce que le réseau a quel neuronne devrait être capable de peaufiner pour que ce soit assez expressif pour saisir le pattern.

Ce qu’on va faire c’est d’assigner un poid à chaque connection entre le neuronne et les neuronnes du premier layer. On prend chaque activation du premier layer pour calculer la somme des poids en fonction de chacun des poids.

Pourquoi les couches ? Quand nous reconnaissons un chiffre, on assemble un ensemble de rond au top et un trait en bas ppur constituer un neuf. Un 8 est composé de deux ronds. Dans un monde idéal, nous esperrons que chaque neuronne dans la secodne couche corresponde avec l’un des composants du chiffre. A chaque fois que nous alimentons une image avec un rond comme dans un 9 ou 8. Des neuronnes spécifiques dont l’activation va être proche de 1 vont passé d’un layer à l’autre.

Mettons que nous ayons deux cartes de convolutions issues d’une image composée de ronds : “arrondi supérieur à droite” et “arrondi supérieur à gauche”. Pour continuer à analyser, nous voulons mettre en évidence la caractéristique “arc-de-cercle supérieur”. Nous allons donc utiliser un kernel capable d’unifier ces deux caractéristiques.



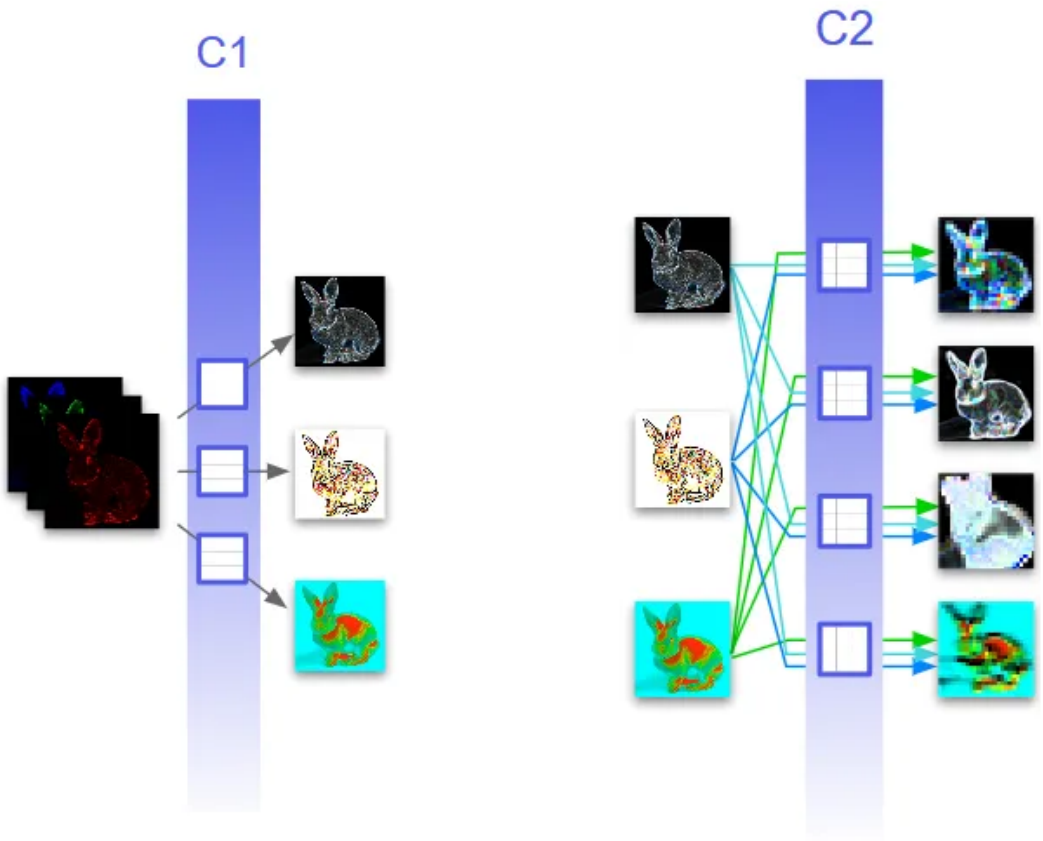
L’entrée de notre kernel est une image constituée à partir de l’ensemble des cartes de caractéristiques empilées. Le kernel a alors pour profondeur le nombre de ces cartes.

Il est possible de voir une image RGB comme 3 cartes de caractéristiques distinctes, une pour chaque couleur. Chaque sous-kernel a son propre ensemble de valeurs et est indépendant des autres.

Commençons avec une première couche “C1” qui prend en entrée l’image et lui applique trois kernels. **Chacun produit en sortie une carte de convolutions** , et nous nous retrouvons donc avec trois cartes de convolutions en sortie. Chacune met en évidence une caractéristique particulière de l’image.

Réseaux de neurones de convolutions

Vous l’aurez compris, le nom de ces réseaux vient de l’opération de convolution. Pour le moment nous n’avons vu qu’une seule application de convolution à l’image. Nous allons maintenant chaîner les convolutions pour faire gagner en complexité les caractéristiques.

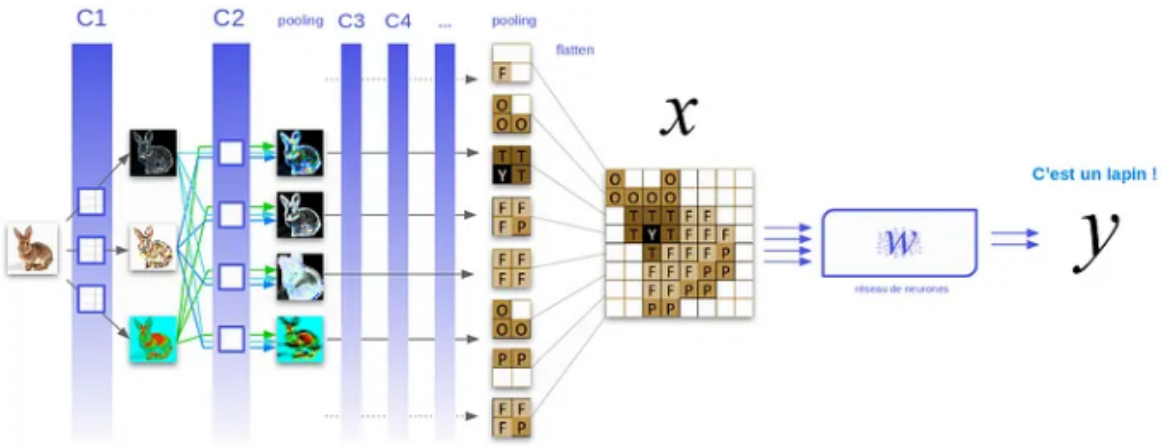


Nous allons continuer ainsi les couches de convolutions, augmentant la complexité des caractéristiques interprétées. Cependant il est possible et même fortement recommandé dans un réseau de convolutions d’ajouter une opération intermédiaire

nommée **Max Pooling** que nous avons déjà entrevue, permettant de réduire la quantité d'information tout en conservant son sens.

Synthèse du réseau de neurones de convolutions

Lorsque nous avons atteint un certain nombre de convolutions et donc de complexité, nous utilisons une opération de **Flatten** (aplatir) chargée d'agréger les données dans un tableau linéaire. En effet, à force de chaîner les convolutions et les **Poolings**, nous finissons avec beaucoup de cartes de convolutions de petites tailles. Le **Flatten** a pour objectif de passer d'un ensemble de matrices à un vecteur, en mettant toutes les valeurs sur le même plan. Nos données optimisées peuvent maintenant être placées en entrée de notre réseau de neurones.



Ensemble formant le réseau de neurones de convolutions (la reconstruction de l'image sur le flatten n'est qu'à but illustratif).

Qui détermine les valeurs des kernels ?

Il n'y a pas plus simple: le réseau fait tout pour vous. C'est d'ailleurs exactement pour cela qu'on l'appelle réseau de neurones de convolutions, parce que ses convolutions font partie intégrante du réseau de neurones. C'est le réseau qui détermine la valeur des kernels de convolutions.

Chaque valeur de chaque kernel (ainsi que de ses sous-kernels) est en réalité un paramètre du réseau ! Ainsi au début de l'entraînement du réseau les kernels ont tous des valeurs aléatoires.

Puis au fur et à mesure de l’apprentissage chacun des poids s’adapte pour apprendre à faire ressortir les caractéristiques qui permettent au réseau de neurones d’interpréter au mieux l’image. La tâche d’apprentissage est donc double : le réseau doit apprendre à convenablement pré-formater l’image puis à correctement analyser son propre pré-formatage.

Conclusion

Pour résumer, nous venons de voir qu’un réseau de neurones de convolutions se décompose en deux parties : une de mise en évidence des caractéristiques d’une image, l’autre d’analyse et d’interprétation de l’agencement de ces caractéristiques.

Tout ceci est rendu possible grâce à l’opération de convolution, qui permet d’analyser des images et d’en sortir des cartes de caractéristiques, mettant en valeur l’information importante de l’image.

Une fois le réseau de convolution construit, il restera à l’entraîner sur une grande quantité d’images pour qu’il soit capable de reconnaître par exemple des objets de la vie courante. Il pourra même en l’améliorant en reconnaître plusieurs en même temps ainsi que donner leurs emplacements dans une photo !

Comment le training fonctionne?

Linear Regression Simplified - Ordinary Least Square vs Gradient Descent

Gradient Descend

Qu'est-ce que la régression linéaire ?

La régression linéaire est une méthode statistique qui permet de trouver la relation entre les variables indépendantes et les variables dépendantes. Prenons un ensemble de données simple pour expliquer le modèle de régression linéaire.

Il se résume à trouver le minimum dans une fonction. N’oubliez pas que conceptuellement on pense à chaque neuronne comme étant connecté à tous les uatres neuronnes de la couche précédente. De plus, les poids dans la somme des poids sont comme la puissance de ces connections. Le biais est une indication si oui et non tendent à être actifs ou inactifs pour démarrer.

L'objectif principal de l'algorithme de descente de gradient est de minimiser la fonction de coût. C'est l'un des meilleurs algorithmes d'optimisation pour minimiser les erreurs (différence entre la valeur réelle et la valeur prédite).

Somme totale des carrés (SST):
La SST est la somme de toutes les différences au carré entre la moyenne d'un échantillon et les valeurs individuelles de cet échantillon. Elle est représentée mathématiquement par la formule suivante.

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$y_i = Dependent\ Variables\ (Salary)$

$\bar{y} = Average\ of\ Dependent\ Variables$

Pour obtenir le meilleur ajustement de l'ordonnée à la ligne, nous devons appliquer un modèle de régression linéaire afin de réduire au minimum la valeur de l'ESS. Pour identifier l'ordonnée à l'origine d'une pente, nous utilisons l'équation suivante.

$$y = mx + b,$$

- 'm' is the slope
- 'x' → independent variables
- 'b' is intercept

L'objectif principal de l'algorithme de descente de gradient est de minimiser la fonction de coût. C'est l'un des meilleurs algorithmes d'optimisation pour minimiser les erreurs (différence entre la valeur réelle et la valeur prédite).

Cost function

Le cost function est une couche de complexité au-dessus qui le prend comme un input. Cela prend tous les biais et poids et les recrache comme un numéro singulier qui décrit dans quelle mesure ces poids et biaias sont mauvais.

On a envie de dire à la fonction comment ces poids et biais peuvent changer pour devenir meilleurs. Pour simplifier l'opération, au lieu d'imaginer 13 000 inputs. Il faut imaginer un fonction qui a un nombre comme paramètre et un nombre comme sortie. Comment trouver un paramètre qui minimize la valeur de cette fonction? On peut commencer par savoir quelle direction prendre pour faire en sorte que l'output soit plus bas. Plus pécifiquement, on peut se demander quelle est la tangeante où on se trouve. On se déplace là où ça devient négatif n fois jusqu'à trouver un minimum.

Gradient Descent in action

L'objectif est similaire à l'opération ci-dessus que nous avons effectuée pour trouver le meilleur ajustement de la ligne d'ordonnée à l'origine "y" à la pente "m". En utilisant également l'algorithme de descente de gradient, nous trouverons une fonction de coût minimal en appliquant divers paramètres pour thêta 0 et thêta 1 et nous verrons l'ordonnée à l'origine de la pente jusqu'à ce qu'elle atteigne la convergence.

Dans un exemple concret, il s'agit de trouver la meilleure direction pour descendre une pente.

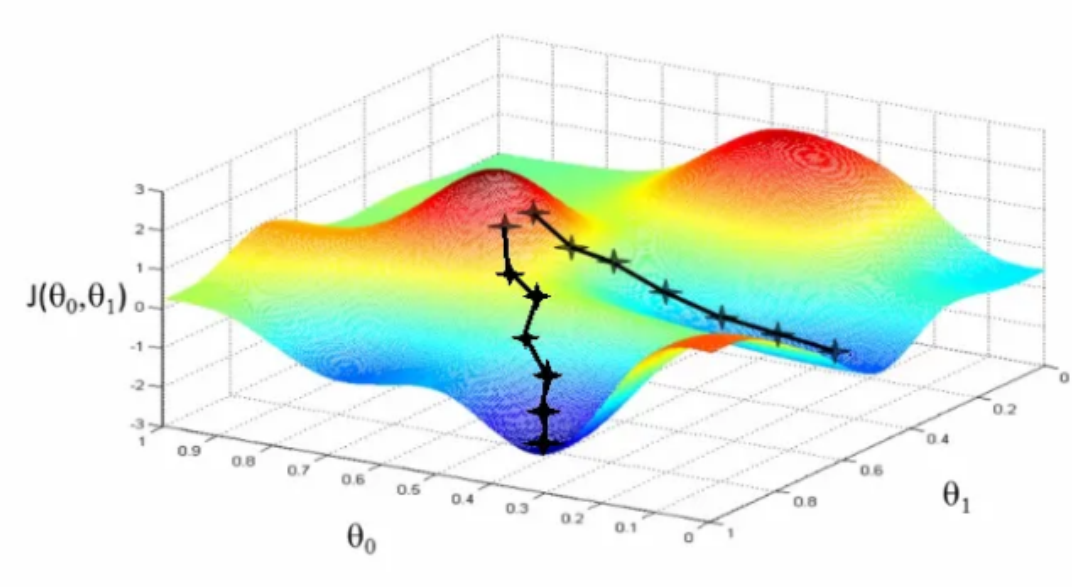


Figure 3: Gradient Descent 3D diagram. Source: Coursera — Andrew Ng

On fait un pas vers la direction pour descendre. Après chaque pas, on regarde à nouveau la direction pour descendre plus vite et plus rapidement. Cet algorithme utilise

une approche similaire pour minimiser la fonction de coût.

Nous pouvons mesurer la précision de notre fonction d'hypothèse en utilisant une fonction de coût dont la formule est la suivante

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Pourquoi utiliser la dérivée partielle dans l'équation ? Les dérivées partielles représentent le taux de variation des fonctions lorsque la variable change. Dans notre cas, nous changeons les valeurs de thêta 0 et thêta 1 et identifions le taux de variation. Pour appliquer les valeurs de taux de variation pour thêta 0 et thêta 1, voici les équations pour thêta 0 et thêta 1 à appliquer à chaque époch.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

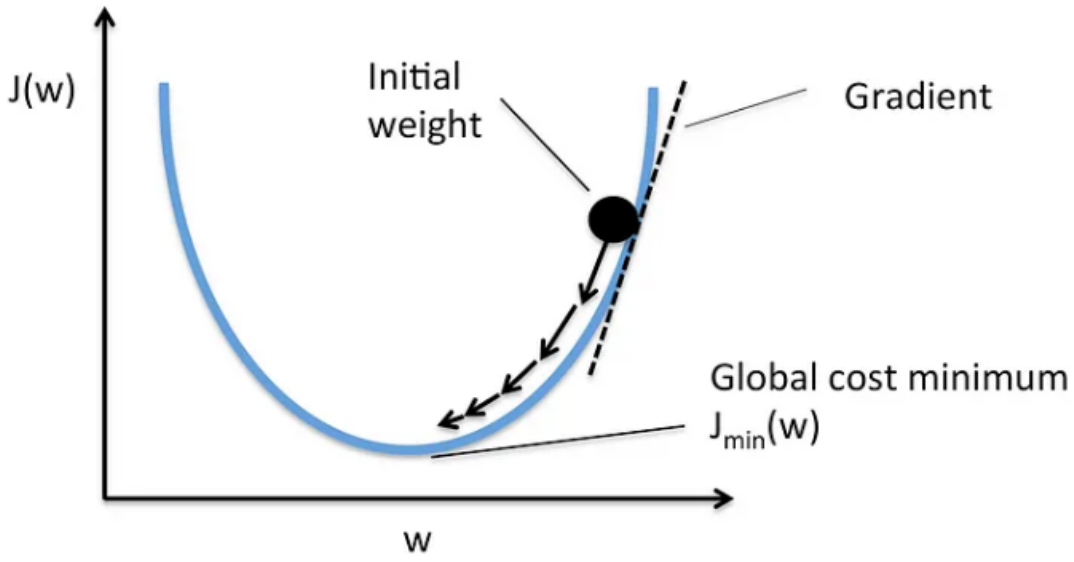


Figure 4: Gradient Descent for linear regression. Source: http://sebastianraschka.com/ — Python Machine Learning, 2nd Edition

Comment fonctionne la rétropropagation dans les réseaux neuronaux ?

Pour permettre un meilleur apprentissage d'un ANN, il faut un mécanisme capable de corriger **une erreur de prédiction**. Il s'agit là de la pièce manquante de notre exemple puisqu'on a vu à quoi ressemble un ANN, puis comment calculer un neurone, mais **comment un ANN peut évoluer s'il se trompe sur le résultat final ?**

Pour cela, on utilise un algorithme qui s'appelle la **rétropropagation *(du gradient)***. Cette propagation démarre de la dernière couche ***(Sortie)*** vers la première ***(Entrée)***. **L'objectif est de modifier les poids de l'ensemble des connexions entre les neurones.**

Sans rentrer dans le détail mathématique, prenons l'exemple *d'un enfant que l'on gronde afin de lui faire comprendre que son comportement n'est pas approprié. Dans cet exemple, la rétroprogapagation transmet à son réseau neural l'apprentissage suivant : qu'il ne faut plus casser les vases de mamie.*

Sur le réseau neural, durant la phase d'apprentissage, on va vérifier si le parti politique de notre exemple est correct. **S'il ne l'est pas, alors on va effectuer un calcul pour modifier le poids des connexions et la valeur des neurones précédents.** Ainsi, le neurone qui est le plus responsable de l'erreur sera susceptible d'être modifié davantage.

Au fur et à mesure de l'apprentissage, le nombre d'erreurs diminue : **l'apprentissage est donc fini.** On peut enfin utiliser notre ANN et l'utiliser pour calculer des probabilités sur un problème donné.

Les réseaux neuronaux apprennent par réglage itératif des paramètres (poids et biais) au cours de la phase de formation. Au départ, les paramètres sont initialisés par des poids générés de manière aléatoire et les biais sont fixés à zéro. Ensuite, les données sont transmises au réseau pour obtenir la sortie du modèle. Enfin, la rétro-propagation est effectuée. Le processus d'apprentissage du modèle comprend généralement plusieurs itérations d'une passe avant, d'une rétropropagation et d'une mise à jour des paramètres.

How Does Back-Propagation Work in Neural Networks?

Les réseaux neuronaux apprennent par réglage itératif des paramètres (poids et biais) au cours de la phase de formation. Au départ, les paramètres sont initialisés par des poids générés de manière aléatoire et les biais sont fixés à zéro. Ensuite, les données sont transmises au réseau pour obtenir la sortie du modèle. Enfin, la rétro-propagation est effectuée. Le processus d'apprentissage du modèle comprend généralement plusieurs itérations d'une passe avant, d'une rétropropagation et d'une mise à jour des paramètres.

Cet article se concentre sur la façon dont la rétropropagation met à jour les paramètres après un passage vers l'avant (nous avons déjà couvert la propagation vers l'avant dans l'article précédent). Nous travaillerons sur un exemple simple mais détaillé de rétropropagation. Avant de poursuivre, voyons les données et l'architecture que nous utiliserons dans cet article.

Definition:

Back-propagation is a method for supervised learning used by NN to update parameters to make the network's predictions more accurate. The parameter optimization process is achieved using an optimization algorithm called **gradient descent (**this concept will be very clear as you read along).

Une forward pass permet de prédire (yhat) la cible (y) moyennant une perte qui est saisie par une fonction de coût (E) définie comme suit :

$$E(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(y, yhat)$$

Equation 1: Cost function

où m est le nombre d'exemples d'apprentissage et L est l'erreur/la perte encourue lorsque le modèle prédit yhat au lieu de la valeur réelle y. L'objectif est de minimiser le coût E. Pour ce faire, on différencie E par rapport aux paramètres (poids et paramètres) et on ajuste les paramètres dans la direction opposée du gradient (c'est la raison pour laquelle l'algorithme d'optimisation est appelé descente de gradient).

Mathématiques

Non Linearity (ReLU)

ReLU signifie Rectified Linear Unit (unité linéaire rectifiée) pour une opération non linéaire. La sortie est $f(x) = \max(0,x)$.

Pourquoi ReLU est-il important ? L'objectif de ReLU est d'introduire la non-linéarité dans notre ConvNet. En effet, les données du monde réel que notre ConvNet devrait apprendre sont des valeurs linéaires non négatives.

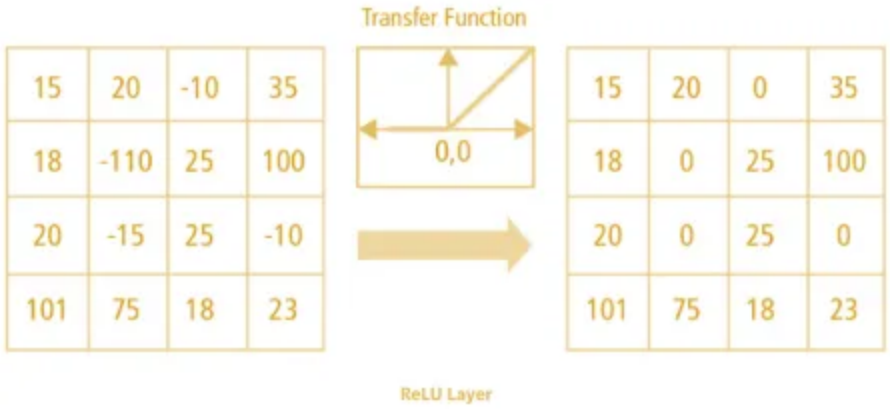


Figure 7 : ReLU operation

- (+) Il a été constaté qu'elles accélèrent considérablement (par exemple, un facteur de 6 dans Krizhevsky et al.) la convergence de la descente de gradient stochastique par rapport aux fonctions sigmoïde/tanh. L'argument avancé est que cela est dû à sa forme linéaire, non saturante.
- (+) Par rapport aux neurones tanh/sigmoïde qui impliquent des opérations coûteuses (exponentielles, etc.), la ReLU peut être mise en œuvre en seuillant simplement une matrice d'activations à zéro.
- (-) Malheureusement, les unités ReLU peuvent être fragiles pendant l'apprentissage et peuvent "mourir". Par exemple, un gradient important circulant dans un neurone ReLU peut entraîner une mise à jour des poids telle que le neurone ne s'activera plus jamais sur aucun point de données. Dans ce cas, le gradient traversant l'unité sera toujours nul à partir de ce point. En d'autres termes, les unités ReLU peuvent mourir de manière irréversible au cours de la formation, puisqu'elles peuvent être exclues du collecteur de données. Par exemple, vous pouvez constater que jusqu'à 40 % de votre réseau peut être "mort" (c'est-à-dire des neurones qui ne s'activent jamais sur l'ensemble des données d'apprentissage) si le taux d'apprentissage est trop élevé. Avec un réglage correct du taux d'apprentissage, ce problème est moins fréquent.

Pour simplifier un peu les choses en vue de l'entraînement ultérieur, nous allons procéder à un petit réajustement de la formule ci-dessus. Déplaçons le seuil de l'autre côté de l'inégalité et remplaçons-le par ce que l'on appelle le biais du neurone. Nous pouvons maintenant réécrire l'équation comme suit :

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i \ + \ bias < 0 \\ 1 \text{ if } \sum_i w_i x_i \ + \ bias \geqslant 0 \end{cases}$$

En effet, biais = - threshold.
Vous pouvez considérer le biais comme la facilité avec laquelle le neurone produit un 1 - avec un biais très important, il est très facile pour le neurone de produire un 1, mais si le biais est très négatif, c'est difficile.

Une fonction qui transforme les valeurs ou énonce les conditions de la décision du neurone de sortie est appelée fonction d'activation. La formule mathématique ci-dessus n'est qu'une des nombreuses fonctions d'activation (et la plus simple) utilisées dans l'apprentissage profond, et elle s'appelle la fonction d'échelon de Heaviside.

Fonctions :

Sigmoid Function

Sans rentrer dans le détail, il existe de nombreuses fonctions d'activations, celle qui est assumée par défaut est la **fonction sigmoïde.**

Cette fonction a pour but d'établir un seuil d'une valeur où le neurone est stimulé ou pas. *C'est comme un signal électrique, plus il est fort et plus on peut stimuler la lumière d'une ampoule.*

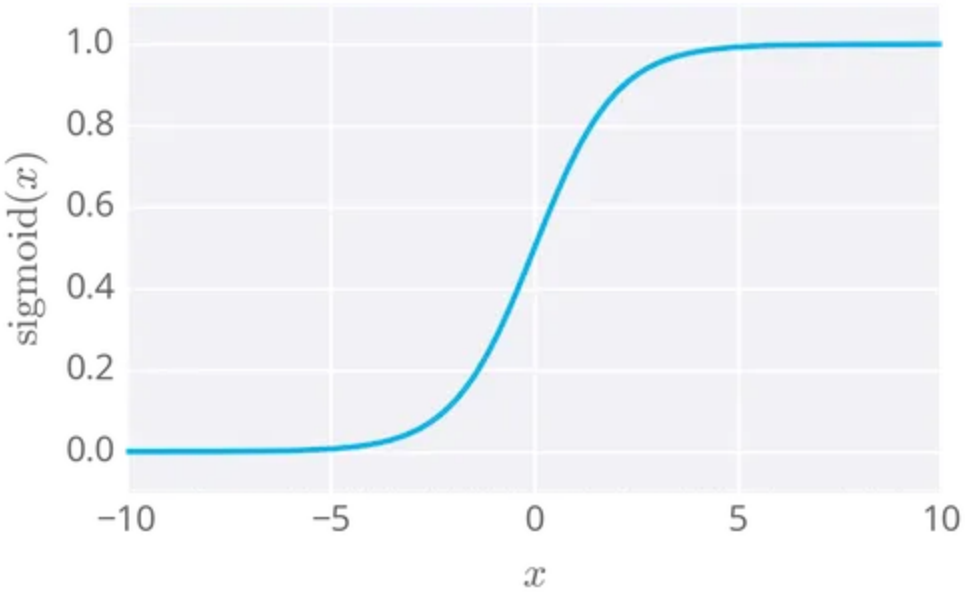
Cette fonction d'activation va alors stimuler le potentiel d'action d'un neurone et, s'il est au-dessus d'un certain seuil il pourra propager son potentiel.

Le processus n'est pas si compliqué quand on admet certains postulats, lorsqu'on débute dans le sujet il vaut mieux éviter de se poser trop de questions. Il existe

différents types de neurones, notamment un qui offert une mémoire à long terme permettant une plus grande précision dans de nombreux cas d'intelligence artificielle.

La forme mathématique du modèle de calcul direct du neurone peut vous sembler familière. Comme nous l'avons vu avec les classificateurs linéaires, un neurone a la capacité d'"aimer" (activation proche de un) ou de "ne pas aimer" (activation proche de zéro) certaines régions linéaires de son espace d'entrée. Par conséquent, avec une fonction de perte appropriée sur la sortie du neurone, nous pouvons transformer un seul neurone en un classificateur linéaire :

Même lorsqu'il s'agit d'absolus (1 et 0, oui et non), il est utile que la sortie donne une valeur intermédiaire. C'est un peu comme répondre "peut-être" à une question de type oui ou non dont vous n'avez aucune idée, au lieu de deviner. C'est essentiellement l'avantage de la fonction sigmoïde sur la fonction de Heaviside.



$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The graph and formula of the sigmoid function. Source: Udacity

la sigmoïde a des valeurs comprises entre 0 et 1. Mais cette fois, il n'y a pas d'échelon ; elle a été lissée pour créer une ligne continue. Ainsi, la sortie peut être considérée comme une probabilité de succès (1) ou de oui. En termes courants, un résultat de 0,5 signifie que le réseau ne sait pas s'il s'agit d'un oui ou d'un non, tandis qu'un résultat de 0,8 signifie que le réseau est "presque sûr" qu'il s'agit d'un oui.

Cette caractéristique est particulièrement importante pour les capacités d'apprentissage du réseau, que nous verrons dans un prochain article. Pour l'instant, il suffit de penser qu'il est plus facile d'apprendre à un réseau à se rapprocher progressivement de la bonne réponse, plutôt que de passer directement d'un 0 à un 1 (ou vice-versa).

Matrices and Matrix Operations

Pour calculer efficacement les opérations des neurones, nous pouvons utiliser des matrices et des vecteurs pour l'entrée, les poids et les biais pour une couche donnée. Nous aurons donc les matrices/vecteurs et les opérations suivantes :

- entrée sous forme de x,
- les poids arrivant à la couche l sous la forme de la matrice w^l
- les biais dans la couche l sous forme de vecteur b^l,
- $z^l = w^l \cdot f^{(l-1)}$- est l'entrée pondérée pour tous les neurones de la couche l. Medium.com a des limitations dans le formatage. f est à la puissance (l-1) pour l'entrée de la couche précédente.
- Dot product - Pour deux vecteurs a et b de même longueur, le produit de points a·b est défini comme suit :

$$a \cdot b = \sum_{i=1}^{d} a_i * b_i$$

Equation 1: Dot product a·b (Source: Author).

$a_i * b_i$ est la multiplication scalaire entre l'élément i^{th} du vecteur a et l'élément i^{th} du vecteur b.

- $f^l = g^l(z^l+b^l)$ sera la sortie de la couche l.

Matrix Multiplication

Deux matrices ne peuvent être multipliées que si elles sont compatibles. Deux matrices sont dites compatibles pour la multiplication si et seulement si le nombre de colonnes de la première matrice est égal au nombre de lignes de la seconde matrice. Ainsi, pour A(m,n) et B(n,p), alors A·B est d'ordre (m,p) (voir figure 2 ci-dessous).

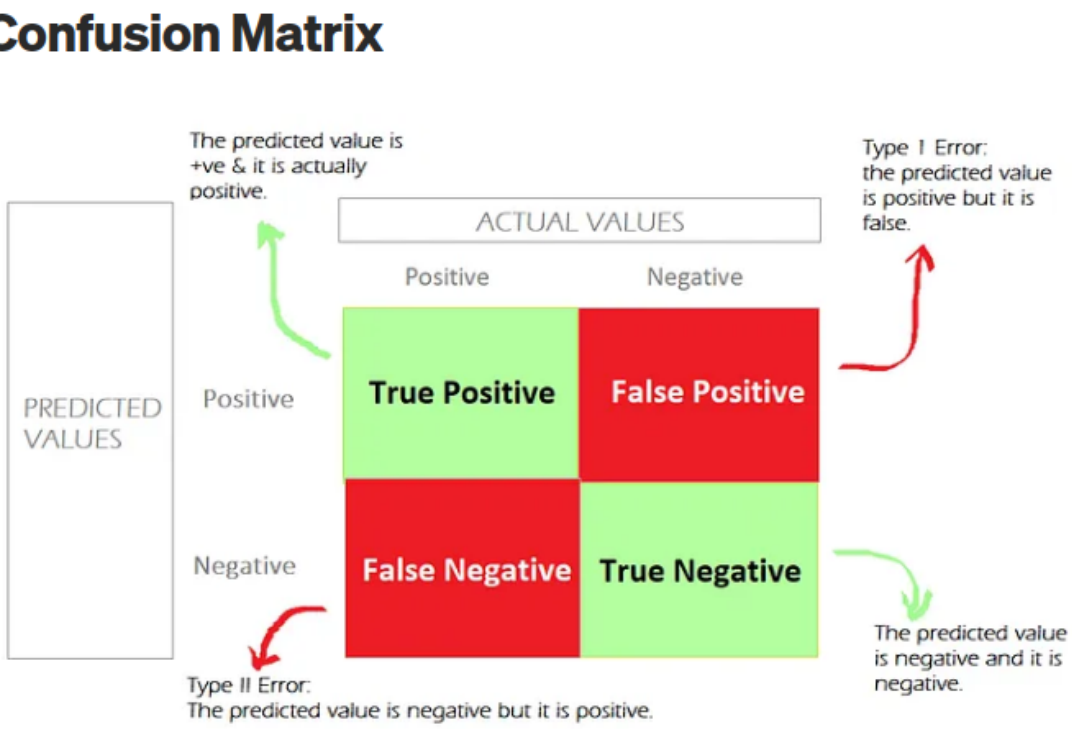
Passons à la dernière couche. Il y a 10 neuronnes, chacun représentant un des digits. Le degré d'activation de ces neuronnes dépend encore une fois du nombre entre 0 et 1.

Comme je l'ai mentionné dans la première partie, nous attribuons des nombres aléatoires aux poids (nous ne connaissons pas les bons à l'avance) et, au fur et à mesure que le réseau neuronal s'entraîne, il modifie progressivement ces poids pour produire des résultats plus précis. De même, nous ne connaissons pas le bon seuil et, comme pour les poids, le réseau devra modifier le seuil pour produire des résultats plus précis. Avec le biais, nous n'avons plus qu'à modifier le côté gauche de l'équation, tandis que le côté droit peut rester constant à zéro, et vous comprendrez bientôt pourquoi cela est utile.

Le coeur du réseau considéré comme étant un process/mécanisme d'information se réduit à

Le pattern d'activation cause des patterns très spécifiques dans la couche qui la suit. Cela donne un pattern pour la couche de sortie. Les neuronnes les plus claires/lumineux pour cette couche est le choix du réseau pour décider le digit qui est représenté.

VI - Matrice de confusion



Une matrice de confusion est un tableau utilisé pour décrire les performances d'un modèle de classification sur un ensemble de données dont les vraies valeurs sont connues. Elle évalue l'efficacité d'un modèle d'apprentissage automatique et aide à comprendre les types et les quantités d'erreurs commises par le modèle, et donne un aperçu des forces et des faiblesses du modèle.

B) Metriques de performances

Choix de la métrique

Métriques	modèle mathématiques	Description
Accuracy	$\frac{\text{Correct Predictions}}{\text{Total Predictions}}$	Il calcule le rapport entre le nombre de prédictions correctes et le nombre total de prédictions.
Précision	$\frac{\text{Correct predicted positive}}{\text{Total predicted positives}}$	Mesure de la précision avec laquelle un modèle prédit la classe positive ou la classe d'intérêt.
Recall	$\frac{\text{Correct predicted positives}}{\text{Total actual positives}}$	C'est une mesure des observations réelles qui sont prédites correctement, c'est-à-dire le nombre d'observations de la classe positive qui sont effectivement prédites comme positives.
F1-Score	$\frac{\text{Correctly predicted negatives}}{\text{Total actual negatives}}$	The F1-score is the **harmonic mean of precision and recall**, and provides a balanced measure that combines both metrics.
Corrélation de Matthew	$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	MCC est une meilleure mesure de classification à valeur unique qui aide à résumer la matrice de confusion ou une matrice d'erreur
Coefficient Kappa	$\kappa = \frac{P[A] - P[H]}{1 - P[H]}$	Le coefficient de kappa est une mesure statistique utilisée pour évaluer l'accord entre deux ou plusieurs évaluateurs lorsqu'ils attribuent des catégories ou des

Coefficient Kappa de Cohen (K)

Indice statistique variant entre 0 et 1 (théoriquement, car dans certains cas il peut assumer des valeurs négatives), utilisé notamment pour évaluer le degré d'accord (de concordance) entre deux juges, évaluateurs ou observateurs quant à la manière de

classer un ensemble d'individus ou d'objets dans un certain nombre de catégories définissant les modalités d'une variable nominale (catégories non ordonnées).

Il est souvent utilisé pour mesurer la fiabilité ou la validité des mesures subjectives, telles que les évaluations humaines dans les études de recherche, les évaluations médicales ou les évaluations de performances.

Pourquoi choisir Kappa de Cohen au lieu de accuracy par exemple?

La métrique d'exactitude (accuracy) est le rapport du nombre de prédictions correctes sur le nombre total de prédictions, exprimé en pourcentage. Bien que l'exactitude soit une métrique couramment utilisée pour évaluer la performance d'un modèle de machine learning, elle peut ne pas être adaptée à des ensembles de données déséquilibrés.

Lorsqu'un ensemble de données est déséquilibré, cela signifie que les classes d'étiquettes ont des fréquences différentes, ce qui peut entraîner des biais dans l'évaluation de la performance du modèle. Prenons un exemple concret : supposons que vous avez un ensemble de données de détection de fraude avec 99% d'exemples négatifs (non frauduleux) et seulement 1% d'exemples positifs (frauduleux). Un modèle qui prédit toujours la classe majoritaire (non frauduleux) aura une exactitude de 99%, ce qui semble excellent, mais en réalité, il ne détecte pas du tout les fraudes, ce qui est l'objectif principal de la tâche.

Le coefficient de Kappa, en revanche, est une métrique qui tient compte du déséquilibre des classes. Il mesure l'accord entre les prédictions du modèle et les étiquettes réelles, en tenant compte de la probabilité d'accord aléatoire. Le coefficient de Kappa prend en compte à la fois la précision (la proportion d'accord entre les prédictions et les étiquettes réelles) et l'accord au hasard (la probabilité d'accord entre les prédictions et les étiquettes réelles simplement par chance). Ainsi, le coefficient de Kappa offre une meilleure évaluation de la performance du modèle dans le cas d'un ensemble de données déséquilibré, car il prend en compte à la fois la précision et le hasard.

En somme, bien que l'exactitude puisse être trompeuse dans le cas d'un ensemble de données déséquilibré, le coefficient de Kappa est une métrique plus appropriée pour évaluer la performance d'un modèle de machine learning dans cette situation, car il prend en compte le déséquilibre des classes. Cependant, il est toujours important de considérer le contexte spécifique de la tâche et de choisir les métriques d'évaluation en fonction des objectifs et des caractéristiques de l'ensemble de données.