Probabilités

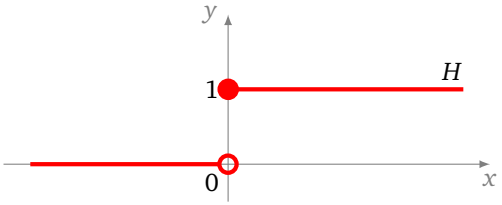
Vidéo ■ partie 16.1. Activation softmax
Vidéo ■ partie 16.2. Fonctions d'erreur
Vidéo ■ partie 16.3. Loi normale

Nous présentons quelques thèmes probabilistes qui interviennent dans les réseaux de neurones.

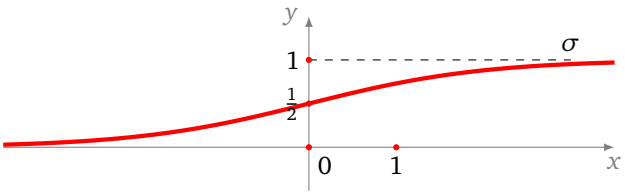
1. Activation softmax

1.1. La fonction σ

On souhaite pouvoir distinguer une image d'un chat (valeur 0) de celle d'un chien (valeur 1). La première idée serait d'utiliser la fonction marche de Heaviside qui répondrait très clairement à la question, mais nous allons voir que la fonction σ est plus appropriée.



$$\begin{cases} H(x) = 0 & \text{si } x < 0 \\ H(x) = 1 & \text{si } x \geqslant 0 \end{cases}$$



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

On rappelle d'abord les qualités de la fonction σ par rapport à la fonction marche de Heaviside :

- La fonction σ est dérivable en tout point avec une dérivée non identiquement nulle, ce qui est important pour la descente de gradient. A contrario la fonction H est de dérivée partout nulle (sauf en 0 où elle n'est pas dérivable).
- La fonction σ prend toutes les valeurs entre 0 et 1, ce qui permet de nuancer le résultat. Par exemple si la valeur est 0.92, on peut affirmer que l'image est plutôt celle d'un chien.

On interprète donc une valeur p entre 0 et 1 renvoyée par σ comme une probabilité, autrement dit un degré de certitude. Évidemment, si l'on souhaite à chaque fois se positionner, on peut répondre : 0 si $\sigma(x) < \frac{1}{2}$ et 1 sinon. Cela revient à composer σ par la fonction $H(x - \frac{1}{2})$.

1.2. Fonction softmax

Comment construire une fonction qui permette de décider non plus entre deux choix possibles, mais parmi un nombre fixé, comme par exemple lorsqu'il faut classer des images en 10 catégories ? Une solution est d'obtenir k valeurs $(x_1, \ldots, x_k)$. On rattache $(x_1, \ldots, x_k)$ à l'une des k catégories données par un vecteur de longueur k :

- $(1,0,0,\dots,0)$ première catégorie,
- $(0,1,0,\dots,0)$ deuxième catégorie,
- $\dots$
- $(0,0,0,\dots,1)$ k-ème et dernière catégorie.

Argmax. La fonction argmax renvoie le rang pour lequel le maximum est atteint. Par exemple si $X = (x_1,x_2,x_3,x_4,x_5) = (3,1,8,6,0)$ alors argmax(X) vaut 3, car le maximum $x_3 = 8$ est atteint au rang 3. Cela correspond donc à la catégorie $(0,0,1,0,0)$.

Softmax. Pour $X = (x_1,x_2,\dots,x_k)$ on note

$$\sigma_i(X) = \frac{e^{x_i}}{e^{x_1}+e^{x_2}+\cdots+e^{x_k}}.$$

Proposition 1.
Pour chaque $i = 1,\dots,k$ on a $0 < \sigma_i(X) \leqslant 1$ et

$$\sigma_1(X)+\sigma_2(X)+\cdots+\sigma_k(X) = 1.$$

Comme la somme des $\sigma_i(X)$ vaut 1, on interprète la valeur $\sigma_i(X)$ comme une probabilité p_i. Par exemple avec $X = (3,1,8,6,0)$ alors

$$p_1 = \sigma_1(X) \simeq 0.0059 \quad p_2 = \sigma_2(X) \simeq 0.0008 \quad p_3 \simeq 0.8746 \quad p_4 \simeq 0.1184 \quad p_5 \simeq 0.0003$$

On note que la valeur maximale est obtenue au rang 3. On obtient aussi ce rang 3 en composant par la fonction argmax.

2. Fonctions d'erreur

Prenons un jeu de N données (x_i,y_i) et un réseau de neurones, nous avons pour $i = 1,\dots,N$:

- la sortie attendue y_i issue des données,
- à comparer avec la sortie $\widetilde{y}_i$ produite par le réseau (calculée comme un $F(x_i)$).

Nous allons voir différentes formules d'erreur. La valeur de l'erreur n'est pas très importante, ce qui est important c'est que cette erreur soit adaptée au problème et tende vers 0 par descente de gradient.

2.1. Erreur quadratique moyenne

L'*erreur quadratique moyenne* calcule une moyenne des carrés des distances entre données et prédictions :

$$E = \frac{1}{N}\sum_{i=1}^{N}(y_i-\widetilde{y}_i)^2.$$

Si l'erreur vaut 0, c'est que toutes les valeurs prédites sont exactement les valeurs attendues.

> **Exemple.**
> Avec les données et les prédictions suivantes :
>
> $$(y_i) = [1,2,5,1,3] \qquad \text{et} \qquad (\widetilde{y}_i) = [1.2,1.9,4.5,1.1,2.7],$$
>
> l'erreur quadratique moyenne est $(N = 5)$:
>
> $$E = \frac{1}{5}\big((1-1.2)^2+(2-1.9)^2+\cdots\big) = 0.08.$$

2.2. Erreur absolue moyenne

L'*erreur absolue moyenne* calcule une moyenne des distances entre données et prédictions :

$$E = \frac{1}{N}\sum_{i=1}^{N} |y_i - \tilde{y}_i|.$$

Intuitivement, cette erreur est plus naturelle car elle mesure une distance (et non le carré d'une distance), mais l'usage de la valeur absolue la rend d'utilisation moins aisée que la précédente.

> **Exemple.**
> Avec les mêmes données et prédictions que dans l'exemple précédent, cette fois :
> $$E = \frac{1}{5}\big(|1-1.2| + |2-1.9| + \cdots\big) = 0.24.$$

2.3. Erreur logarithmique moyenne

L'*erreur logarithmique moyenne* est :

$$E = \frac{1}{N}\sum_{i=1}^{N} \big(\ln(y_i+1) - \ln(\tilde{y}_i+1)\big)^2.$$

L'usage du logarithme est adapté à des données de différentes tailles. En effet, elle prend en compte de façon identique une petite erreur sur une petite valeur et une grande erreur sur une grande valeur.

> **Exemple.**
> Avec les données et prédictions suivantes :
> $$(y_i) = [1, 5, 10, 100] \qquad \text{et} \qquad (\tilde{y}_i) = [2, 4, 8, 105],$$
> et $N = 4$, l'erreur est :
> $$E = \frac{1}{4}\big((\ln(1+1) - \ln(2+1))^2 + (\ln(5+1) - \ln(4+1))^2 + \cdots + (\ln(100+1) - \ln(105+1))^2\big) \simeq 0.060.$$
> Pour illustrer les propos précédents, il est à noter que l'erreur absolue moyenne vaut $E' \simeq 2.25$ et que l'écart entre $y_4 = 100$ et $\tilde{y}_4 = 105$ contribue à 55% de cette erreur. Tandis que pour l'erreur logarithmique moyenne, l'écart entre $y_4 = 100$ et $\tilde{y}_4 = 105$ contribue à 15% de l'erreur E. Selon l'origine des données, on peut considérer normal d'avoir une plus grande erreur sur de plus grandes valeurs.

2.4. Entropie croisée binaire

L'*entropie croisée binaire* est adaptée lorsque la réponse attendue est soit $y_i = 0$ soit $y_i = 1$ et que la sortie produite (c'est-à-dire la prédiction) est une probabilité $\tilde{y}_i$ avec $0 < \tilde{y}_i < 1$:

$$E = -\frac{1}{N}\sum_{i=1}^{N} \big(y_i \ln(\tilde{y}_i) + (1-y_i)\ln(1-\tilde{y}_i)\big).$$

Chaque terme de la somme n'est en fait constitué que d'un seul élément : si $y_i = 1$ c'est $\ln(\tilde{y}_i)$ et si $y_i = 0$ c'est $\ln(1-\tilde{y}_i)$.

> **Exemple.**
> Avec les données et prédictions suivantes :
> $$(y_i) = [1, 0, 1, 1, 0, 1] \qquad \text{et} \qquad (\tilde{y}_i) = [0.9, 0.2, 0.8, 1.0, 0.1, 0.7].$$

et $N = 6$, l'entropie croisée binaire est :

$$E = -\frac{1}{6}\big(\ln(0.9) + \ln(1-0.2) + \ln(0.8) + \ln(1.0) + \ln(1-0.1) + \ln(0.7)\big) \simeq 0.17.$$

Les couleurs sont à mettre en correspondance avec les valeurs des données :

$$[1, 0, 1, 1, 0, 1].$$

2.5. Entropie croisée pour catégories

L'*entropie croisée pour catégories* est adaptée lorsque la réponse attendue est un vecteur du type $y_i = (0, 0, \ldots, 1, \ldots, 0)$ (avec un seul 1 à la place qui désigne le rang de la catégorie attendue) et que la sortie produite par la prédiction est une liste de probabilités $\widetilde{y}_i = (p_1, p_2, \ldots)$ (voir l'exemple de la reconnaissance de chiffres).

3. Normalisation

3.1. Motivation

Les données brutes qui servent à l'apprentissage peuvent avoir des valeurs diverses. Afin de standardiser les procédures, il est préférable de ramener les valeurs des données dans un intervalle déterminé par exemple $[0, 1]$ ou bien $[-1, 1]$. C'est plus efficace car au départ de l'apprentissage les valeurs initiales des poids du réseau sont fixées indépendamment des données, par exemple au hasard entre $[-1, 1]$. Cela permet également de partir de poids qui ont été calculés pour une autre série de données au lieu de valeurs aléatoires.

Prenons l'exemple d'images en niveau de gris, pour lesquelles chaque pixel est codé par un entier n_{ij} entre 0 et 255. On « normalise » l'image en remplaçant la matrice des pixels (n_{ij}) par la matrice $(n_{ij}/255)$. Ainsi les valeurs sont maintenant des réels entre 0 et 1.

Plus généralement, si les données sont comprises entre $y_{\min}$ et $y_{\max}$ alors on redéfinit les données à l'aide de la fonction affine suivante dont l'ensemble d'arrivée est l'intervalle $[0, 1]$:

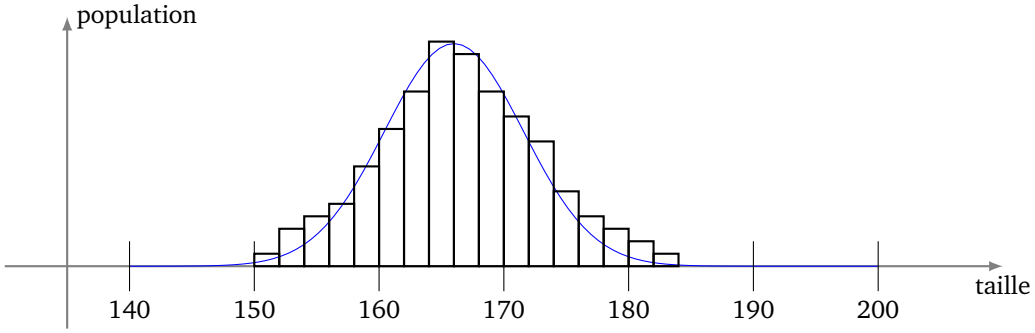
$$y \mapsto \frac{y - y_{\min}}{y_{\max} - y_{\min}}.$$

Pour les ramener dans l'intervalle $[a, b]$, on utiliserait la fonction :

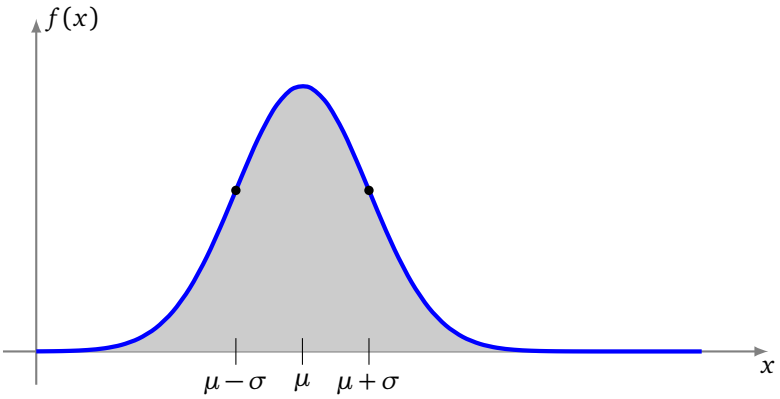
$$y \mapsto \frac{y - y_{\min}}{y_{\max} - y_{\min}}(a - b) + b.$$

3.2. Loi de Gauss

Cependant pour certaines données, nous n'avons pas connaissance de leurs valeurs extrêmes. Prenons l'exemple de la taille de personnes. Voici la répartition du nombre de femmes en fonction de leur taille.



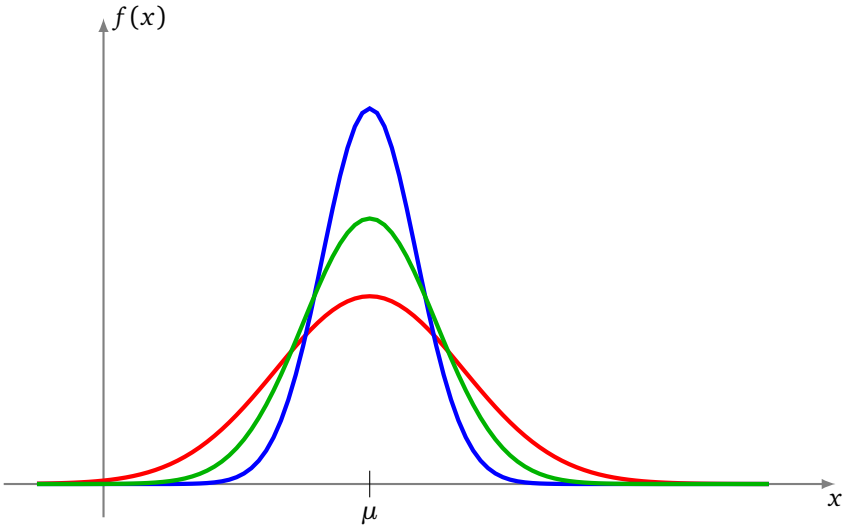
Une telle répartition se fait suivant la « loi de Gauss » que l'on nomme aussi « courbe en cloche ». Elle dépend de deux paramètres : l'*espérance* μ et l'*écart-type* σ.



L'aire sous la courbe vaut 1, la courbe est symétrique par rapport à l'axe ($x = \mu$) et elle possède deux points d'inflexion en $x = \mu - \sigma$ et $x = \mu + \sigma$. L'équation est la suivante :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

La variable μ définit l'axe de symétrie, tandis que la variable σ détermine l'étalement de la courbe. Voici différentes courbes pour la même valeur de μ mais pour différentes valeurs de σ.



3.3. Espérance, écart-type

On se donne une série de données $(x_i)_{1 \leqslant i \leqslant n}$. On souhaite retrouver les paramètres μ et σ correspondant à cette série.

- L'*espérance* est la moyenne des données et se calcule par :

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

- On note $\mathrm{Var}(x)$ la *variance* des $(x_i)_{1 \leqslant i \leqslant n}$:

$$\mathrm{Var}(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2.$$

  La variance mesure l'écart des valeurs avec la moyenne.

- L'*écart-type* est la racine carrée de la variance :

$$\sigma = \sqrt{\mathrm{Var}(x)}.$$

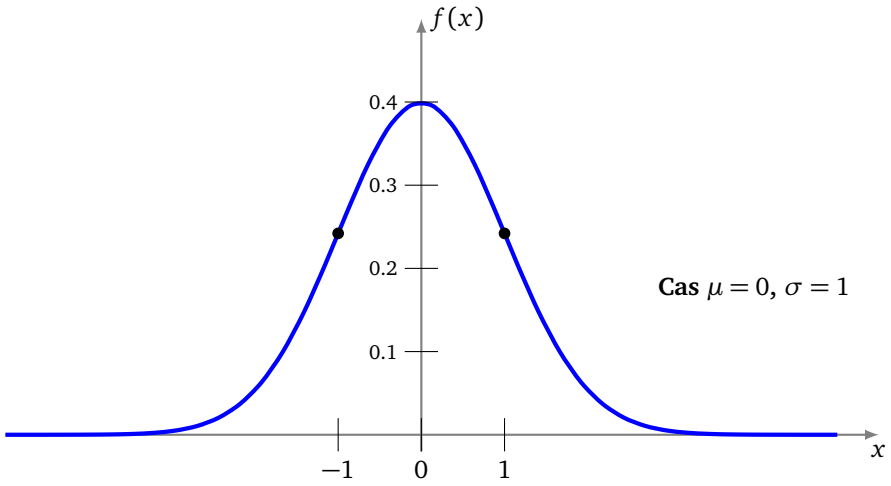
La ***loi normale*** ou ***loi de Gauss*** $\mathcal{N}(\mu,\sigma)$ est alors la loi associée à la densité de probabilité :

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

3.4. Loi normale centrée et réduite

On souhaite normaliser nos données en se ramenant à une loi normale $\mathcal{N}(0,1)$ qui est donc centrée ($\mu=0$) et réduite ($\sigma=1$). La ***loi de Gauss centrée réduite*** est donc définie par :

$$f(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}.$$



Si les données $(x_i)_{1\leqslant i\leqslant n}$ ont pour espérance μ et écart-type σ, alors les données $(x'_i)_{1\leqslant i\leqslant n}$ définies par

$$x'_i=\frac{x_i-\mu}{\sigma}$$

ont pour espérance $\mu'=0$ et écart-type $\sigma'=1$.

3.5. Exemple

Voici un échantillon de tailles d'hommes (en cm) :

$$[172,165,187,181,167,184,168,174,180,186].$$

Pour déterminer une loi de Gauss à partir de cet échantillon, on calcule l'espérance :

$$\mu=\frac{172+165+\cdots+186}{10}=176.4$$

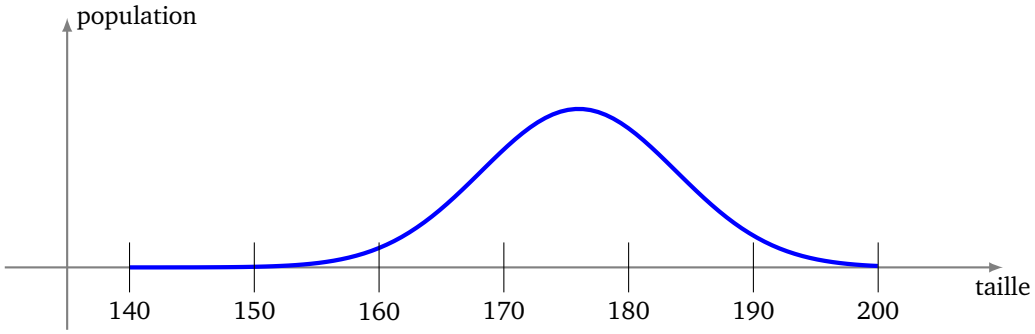
puis la variance :

$$\text{Var}=\frac{(172-\mu)^2+(165-\mu)^2+\cdots+(186-\mu)^2}{10}=61.04$$

qui donne l'écart-type

$$\sigma=\sqrt{\text{Var}}\simeq 7.8.$$

Ainsi on modélise la taille des hommes par une loi normale $\mathcal{N}(\mu,\sigma)$ avec $\mu\simeq 176$ et $\sigma\simeq 7.8$.



On en déduit la répartition attendue des hommes en fonction de leur taille, par exemple 95% des hommes ont une taille comprise entre $\mu - 2\sigma$ et $\mu + 2\sigma$, c'est-à-dire dans l'intervalle $[161, 192]$.
Pour obtenir une loi normale centrée réduite on utilise la transformation :

$$x_i' = \frac{x_i - \mu}{\sigma}$$

où x_i est la hauteur (en cm) et x_i' est la donnée transformée (sans unité ni signification physique). Ce qui nous donne des données transformées x_i' (arrondies) :

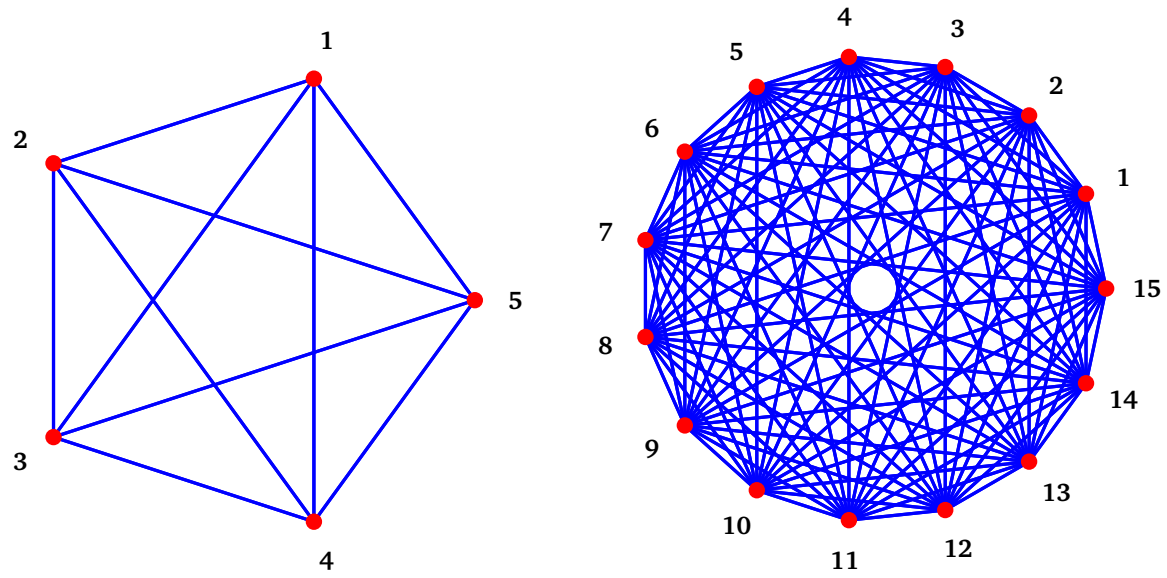
$$[-0.56, -1.46, 1.36, 0.59, -1.20, 0.97, -1.07, -0.31, 0.46, 1.23].$$

4. Dropout

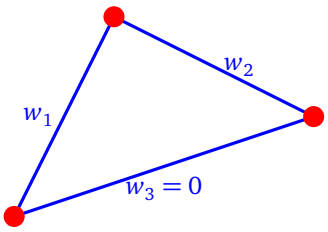
Le dropout est une technique qui simule différentes configurations de liens entre les neurones et limite le sur-apprentissage.

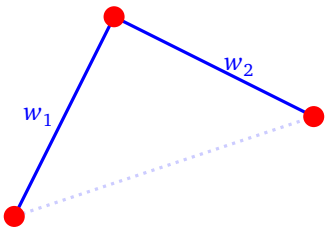
4.1. Motivation

Imaginons n neurones. Quelle est la meilleure architecture pour les relier entre eux ? Bien sûr la réponse dépend du problème, ainsi on ne peut pas le savoir à l'avance.
Une première façon de faire est de relier tous les neurones entre eux.

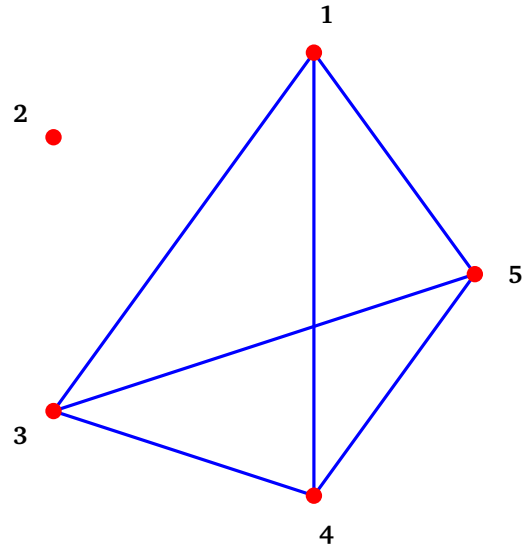


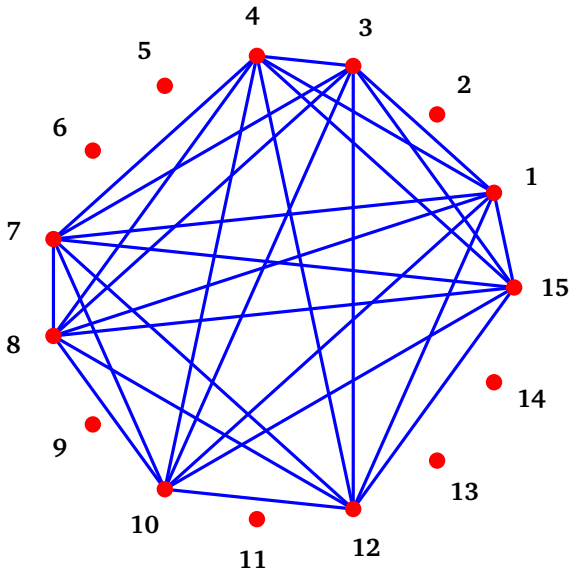
Lors de l'apprentissage certains poids vont peut-être devenir nuls (ou presque) ce qui revient à faire disparaître l'arête reliant deux neurones.





Une autre façon de procéder est de désactiver dès le départ certains neurones avant de commencer l'apprentissage. Cela revient à imposer un poids 0 immuable au cours de l'apprentissage à toutes les arêtes reliées à ces neurones. Mais quelle configuration choisir ? Avec n neurones, il existe $N = 2^n$ configurations possibles. Voici deux exemples :





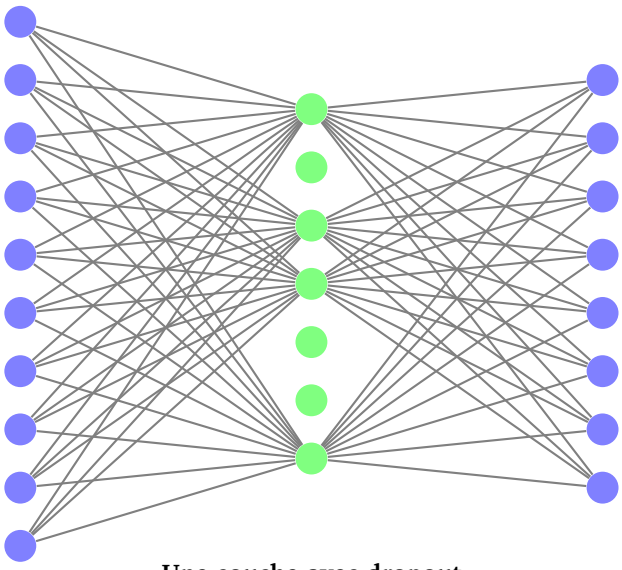
Pour $n = 5$ cela fait $N = 32$ configurations, pour $n = 15$ cela donne plus de 32 768 possibilités ! Pour chacune de ces configurations, il faudrait appliquer la descente de gradient. C'est beaucoup trop de travail, même pour un ordinateur.
Voici l'idée du dropout (avec paramètre $p = 0.5$). On part d'un réseau de n neurones, tous reliés les uns aux autres. Avant la première étape d'apprentissage, on décide de désactiver certains neurones. Cette décision est prise au hasard. Pour chaque neurone on lance une pièce de monnaie, si c'est « pile » on conserve le neurone, si c'est « face » on le désactive. Ensuite on effectue une étape de la descente de gradient, avec seulement une partie de nos neurones activés. Avant la deuxième étape de la descente de gradient, on reprend notre pièce et on choisit au hasard les neurones à désactiver, etc.

4.2. Loi de Bernoulli

Une variable aléatoire suit la ***loi de Bernoulli de paramètre*** p si le résultat est 1 avec une probabilité p et 0 avec une probabilité $1 - p$.
Par exemple, le lancer d'une pièce de monnaie non truquée suit une loi de Bernoulli de paramètre $p = 0.5$. Désactiver les neurones suivant un loi de Bernoulli de paramètre $p = 0.8$ signifie que chaque neurone à 8 chances sur 10 d'être conservé (et donc 2 chances sur 10 d'être désactivé). Sur un grand nombre de neurones, on peut estimer qu'environ 80% sont conservés, mais il peut y avoir des tirages exceptionnels pour lesquels ce n'est pas le cas.

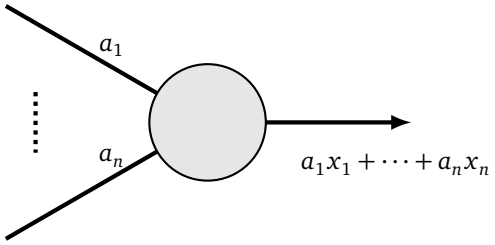
4.3. Dropout

Revenons à nos réseaux de neurones ou plus exactement à une couche de n neurones. Appliquer un ***dropout*** à cette couche, c'est désactiver chaque neurone suivant une loi de Bernoulli de paramètre p, où $0 < p \leqslant 1$ est un réel fixé. On rappelle que désactiver un neurone signifie mettre les poids à 0 pour toutes les arêtes entrantes et sortantes de ce neurone.



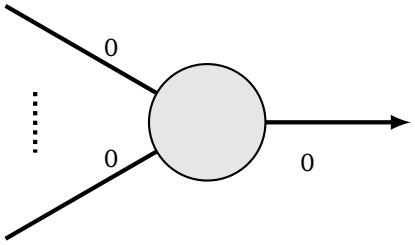
Une couche avec dropout

Neurone « normal ».
Partons d'un neurone classique (sans fonction d'activation) et voyons comment le modifier pour l'apprentissage avec dropout.

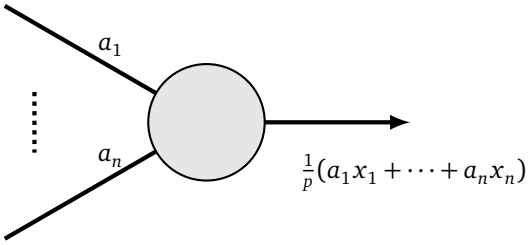


Neurone normal

Apprentissage. Lors de la phase d'apprentissage, si le neurone est désactivé (avec une probabilité $1-p$) alors tous les poids sont nuls, et bien sûr la valeur renvoyée est nulle. Par contre si le neurone est activé (avec une probabilité p), alors il se comporte comme un neurone « presque normal », il faut juste pondérer la sortie d'un facteur $\frac{1}{p}$ (comme il y a moins de neurones chaque neurone doit contribuer de façon plus importante).

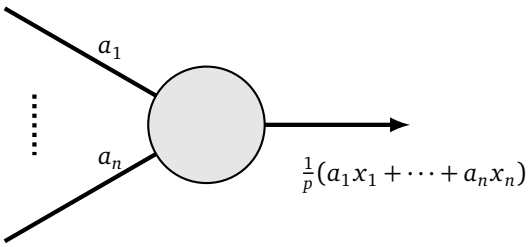


Neurone désactivé, probabilité $1-p$

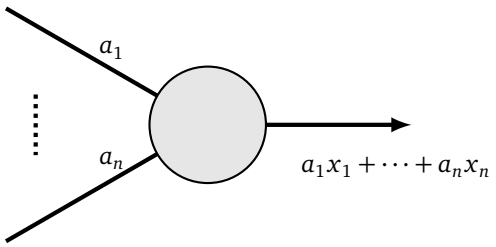


Neurone actif, probabilité p

Test. Lors de la phase de test, tous les neurones sont actifs. La sortie est calculée comme un neurone normal.



Phase apprentissage
Actif avec probabilité p



Phase de test