Les mathématiques des réseaux de neurones

Gabriel Peyré
CNRS & DMA
PSL, École Normale Supérieure
gabriel.peyre@ens.fr

Chapitre 1

Les réseaux discriminatifs

Depuis 2012, les réseaux de neurones profonds ont révolutionné l'apprentissage automatique. Bien que relativement ancienne, cette technique a permis ces dernières années des avancées très spectaculaires pour la reconnaissance de textes, de sons, d'images et de vidéos. Comprendre les enjeux de ces méthodes soulève des questions à l'interfaces entre les mathématiques et l'algorithmique. Dans cet article, je vais expliquer la structure de ces réseaux ainsi que les concepts clefs de leur apprentissage supervisé.

1.1 L'algorithmique et les mathématiques de l'apprentissage

Les réseaux de neurones sont des algorithmes, qui permettent à partir d'une entrée x (par exemple une image) de calculer une sortie y. Comme montré à la figure 1.1, cette sortie est le plus souvent un ensemble de probabilités : par exemple la première sortie est la probabilité que l'image contienne un chat (plus ce nombre est proche de 100%, plus cela signifie que l'algorithme est sûr de lui), la deuxième est la probabilité que l'image contienne un chien, etc. Pour simplifier, on ne considèrera dans nos exemples que deux classes : les chats et les chiens, mais en pratique on peut considérer une sortie y avec plusieurs milliers de classes. On se restreint également à l'exemple des images, mais les réseaux de neurones sont également très performants pour reconnaitre des textes ou des vidéos.

Mathématiquement, un tel algorithme définit une fonction f_w (c'est-à-dire que $y = f_w(x)$). Le programme informatique qui permet de calculer cette fonction est très simple : il est composé d'un enchainement de plusieurs étapes, et chaque étape effectue des calculs élémentaires (des additions, des multiplications, et un maximum). En comparaison, les programmes informatiques que l'on trouve dans le système d'exploitation d'un ordinateur sont beaucoup plus complexes. Mais ce qui fait l'énorme différence entre un algorithme « classique » et un réseau de neurones, c'est que ce dernier dépend de paramètres, qui sont les poids des neurones. Avant d'utiliser un réseau de neurones, il faut modifier ces poids pour que l'algorithme puisse résoudre le mieux possible la tâche demandée. Ceci s'effectue à l'aide de méthodes mathématiques et algorithmiques que l'on va expliquer dans les sections suivantes. C'est ce que l'on appelle « entrainer » un réseau de neurone, et ceci nécessite beaucoup de temps, de calculs machine et d'énergie.

Utiliser à bon escient de tels algorithmes nécessite donc des compétences en informatique et en mathématiques. Il faut ainsi manipuler les concepts clefs de l'algorithmique (méthodes itératives, temps de calcul, espace mémoire, implémentation efficace, ...) et des mathématiques (algèbre

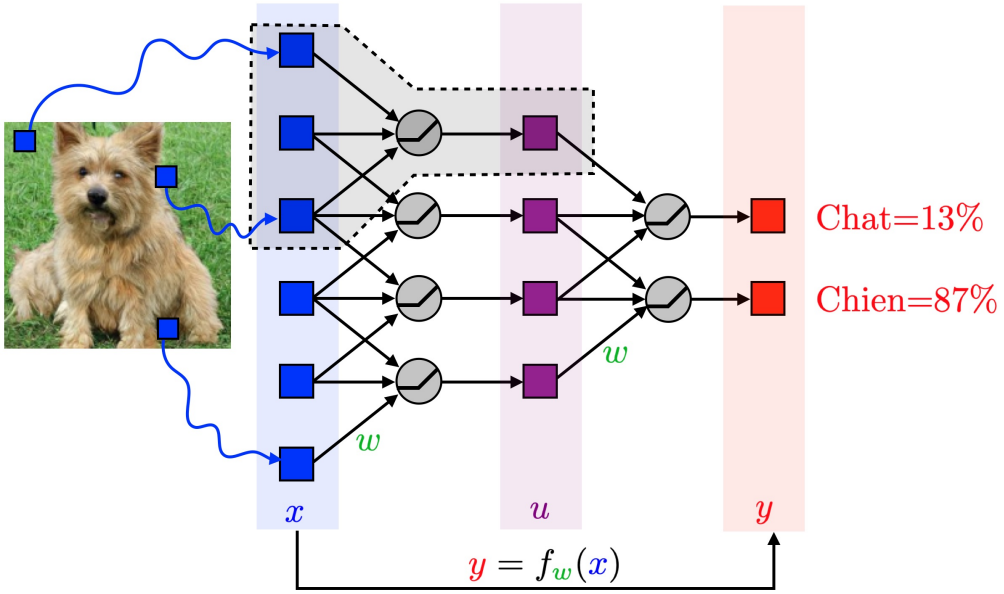

FIGURE 1.1 – Exemple d'un réseau de neurones discriminatif avec deux couches.

linéaire, optimisation, statistiques, ...).

1.2 Réseaux de neurones discriminatifs

Un réseau de neurones artificiel est construit autour d'une métaphore biologique. On connait relativement bien la structure du cortex visuel primaire, et la découverte en 1962 de l'organisation des neurones dans les premières couches [8] a valu le prix Nobel en physiologie à Hubel et Wiesel. Ainsi, dans une vision extrêmement simplifiée du fonctionnement du cerveau, les neurones sont organisés en couches, chaque neurone récupère de l'information d'une couche précédente, effectue un calcul très simple, et communique son résultat à des neurones de la couche suivante. Il faut cependant garder à l'esprit qu'il ne s'agit que d'une métaphore et une source d'inspiration : les réseaux biologiques ont des connexions beaucoup plus complexes et les équations mathématiques qui les régissent sont également très complexes (elles ont été découvertes par Alan Hodgkin et Andrew Huxley en 1952 [7] et ils ont eu le prix Nobel). Il reste ainsi difficile de mettre précisément en relation les performances parfois surprenantes des neurones artificiels et les capacités cognitives du cerveau. Par exemple, les techniques d'entrainement des réseaux artificiels que l'on va maintenant expliquer sont très différentes de la façon dont un enfant apprend.

La figure 1.1 détaille un exemple d'un tel réseau artificiel. Ce type de neuron a été introduit en 1943 par McCulloch et Pitts [14]. Pour simplifier, il est ici composé de seulement deux couches de neurones (la première couche entre x et u, la deuxième entre u et y), mais les réseaux actuels les plus performants peuvent comporter plusieurs dizaines de couches, on dit qu'ils sont plus profonds. Dans notre exemple, les entrées x sont les pixels d'une image. Une image contient typiquement des millions de pixels, et la figure n'en représente volontairement qu'un petit nombre : un réseau réaliste est très complexe. De plus, chaque pixel qui compose x est en fait composé de 3 valeurs (une pour chaque couleur primaire rouge, vert et bleu).

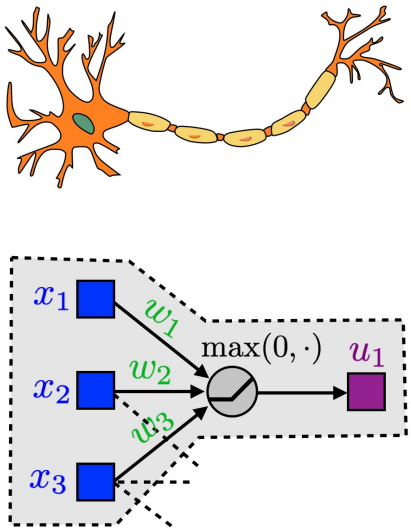

FIGURE 1.2 – Neurones biologique et artificiel.

Le passage d'une couche (par exemple la couche x des entrées) à une autre (par exemple la deuxième couche u, qui est une couche « cachée » au milieu du réseau) se fait par l'intermédiaire d'un ensemble de neurones artificiels. Un neurone est représenté sur la figure 1.2. C'est le premier neurone, celui qui calcule la première valeur u_1 qui compose la couche u. Ce neurone connecte un certain nombre d'éléments de la première couche (ici trois : x_1, x_2, x_3, mais il peut y en avoir plus) à un seul élément de la deuxième, donc ici u_1. La formule calculée par le neurone est

$$u_1 = \max(w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3 + w_4, 0).$$

Le neurone effectue ainsi une somme pondérée des trois entrées, avec trois poids w_1, w_2, w_3, et on ajoute également w_4, qui est un biais. Puis le neurone calcule le maximum entre cette somme et zero. On peut également utiliser une autre fonction que la fonction maximum, mais celle-ci est la plus populaire. Il s'agit d'une opération de seuillage. On peut la comparer aux neurones biologiques qui laissent ou non passer l'information suivant s'ils sont suffisamment excités ou pas. Ainsi, si la somme pondérée $w_1x_1 + w_2x_2 + w_3x_3 + w_4$ est plus petite que 0, alors le neurone renvoie la valeur $u_1 = 0$, sinon il renvoie la valeur de cette somme et la place dans u_1.

De tels réseaux de neurones ont été introduits par Rosenblatt [20] en 1957, qui les a appelés « perceptrons ». Les premiers perceptrons ne contenaient qu'une seule couche. De telles architectures avec une seule couche sont trop simples pour pouvoir effectuer des tâches complexes. C'est en rajoutant plusieurs couches que l'on peut calculer des fonctions plus complexes. Les réseaux de neurones profonds utilisent ainsi un très grand nombre de couches. Depuis quelques années, ces architectures ont permis d'obtenir des résultats très impressionnants pour faire de la reconnaissance d'images et de vidéos ainsi que pour la traduction automatique de textes. Ce sont ces recherches sur les réseaux profonds qui ont permises au chercheur français Yann Le Cun ainsi qu'à Geoffrey Hinton et Yoshua Bengio [11] d'obtenir le prix Turing en 2019. Ce prix est considéré comme étant l'équivalent du prix Nobel en informatique. Pour se familiariser avec ces réseaux multi-couches, on peut utiliser l'application interactive `https://playground.tensorflow.org`.

FIGURE 1.3 – Exemples d'images issues de la base de données ImageNet [5] utilisées pour l'apprentissage.

1.3 L'apprentissage supervisé d'un réseau de neurones

L'entrainement d'un réseau de neurones consiste à choisir les « meilleurs » poids possibles de l'ensemble des neurones qui compose un réseau (par exemple en particulier les poids w_1, w_2 et w_3 du neurone montré à la figure 1.2). Il faut ainsi choisir les valeurs de ces poids afin de résoudre le mieux possible la tâche étudiée, et ceci sur un ensemble de données d'apprentissage. Pour la reconnaissance d'objets dans les images, il s'agit d'un problème d'apprentissage supervisé : on dispose à la fois des images x et des y (les probabilités de présence d'un chat et/ou d'un chien dans l'image). La figure 1.3 montre quelques exemples d'images utilisées pour entrainer un réseau, pour lesquelles on sait ce qu'elles contiennent (la classe des chats et la classe des chiens). Il faut donc, en amont de la phase d'apprentissage, que des humains fasse un long et fastidieux travail d'étiquetage de milliers voir de millions d'images.

La procédure d'entrainement consiste ainsi à modifier les poids w tels que pour chaque x, le réseau f_w prédise aussi précisément que possible le y associé, c'est-à-dire que l'on souhaite à la fin de l'entrainement que $y \approx f_w(x)$. Un choix simple est de minimiser la somme $E(w)$ des carrés des erreurs, ce qu'on écrit mathématiquement comme

$$\min_w E(w) = \sum_{(x,y)} (f_w(x) - y)^2.$$

Ceci correspond à un problème d'optimisation, car il faut trouver un jeu de paramètres qui optimise une certaine quantité d'intérêt. C'est un problème difficile, car il y a beaucoup de paramètres, et ces paramètres, surtout ceux des couches cachées, influencent de façon très subtile le résultat. Heureusement, il existe des méthodes mathématiques et algorithmiques performantes pour résoudre de façon satisfaisante ce type de problème d'optimisation. Elles ne sont pas encore totalement comprises sur le plan théorique et c'est un domaine de recherche en pleine explosion. Ces méthodes d'optimisation modifient les poids w du réseau pour l'améliorer et diminuer l'erreur d'entrainement $E(w)$. La règle mathématique pour décider de la stratégie de mise à jour des poids s'appelle la rétro-propagation [21] et est une merveille d'ingéniosité, c'est un cas particulier d'une méthode mathématique et algorithmique qui s'appelle la différentiation automatique à l'envers [13].

Ces techniques d'apprentissage supervisé datent pour l'essentiel des année 1980. Mais c'est seulement en 2012 qu'un papier de Krizhevsky, Sutskever et Hinton [10] crée un coup de tonnerre en

montrant que les réseaux profonds permettent de résoudre efficacement des problème de reconnaissance d'images. Cette révolution a été possible grâce à la combinaison de trois ingrédients : des nouvelles bases de données beaucoup plus grandes qu'auparavant [5] ; des grosses puissances de calcul grâce aux processeurs graphiques (les « GPUs » , qui étaient auparavant cantonnés aux jeux vidéo) ; l'introduction de plusieurs techniques d'optimisation qui stabilisent l'apprentissage [22].

1.4 L'efficacité des réseaux de neurones

George Cybenko a démontré en 1989 [3] qu'un réseau de neurones f_w avec deux couches peut approcher aussi précisément que l'on veut n'importe quelle fonction continue $f^\star$ (donc en quelque sorte résoudre n'importe quelle tâche, représentée par la fonction $f^\star$ inconnue, qui serait capable de reconnaitre des objets dans n'importe quelle image) pour peu que la taille de la couche interne u (donc le nombre de neurones) soit arbitrairement grande. Ce n'est pas pour autant qu'un tel réseau f_w avec seulement deux couches marche bien en pratique. Pour appliquer le théorème de Cybenko, il faut pouvoir disposer d'un nombre de donnée d'apprentissage potentiellement infini, ce qui est très loin d'être le cas en pratique. Le but final de l'apprentissage n'est pas de minimiser l'erreur d'apprentissage $E(w)$, mais de pouvoir prédire aussi précisément que possible sur des nouvelles données. Si l'on dispose de peu de données, on risque de ne pas pouvoir apprendre assez précisément, et donc de faire des mauvaises prédictions : la fonction f_w sera en réalité très loin de la fonction $f^\star$ idéale que l'on voudrait apprendre si on avait une infinité d'exemples.

Afin d'effectuer les meilleures prédictions possibles avec un nombre limité de données d'entrainement, on cherche donc les architectures de réseaux les plus adaptées, qui peuvent capter efficacement l'information présente dans les données. Les réseaux de neurones profonds (avec de nombreuses couches) mais avec relativement peu de connexions entre les couches se sont avérés très efficaces sur les données très « structurées » comme les textes, les sons et les images. Par exemple, pour une image, les pixels ont des relations de voisinage, et on peut imposer des connexions spécifiques (une architecture) et ne pas connecter un neurone avec tous les autres mais seulement avec ses voisins (sinon il y aurait trop de connexions). De plus, on peut imposer que les poids associés à un neurone soit les mêmes que ceux associés à un autre neurone. On appelle ce type de réseaux les réseaux convolutifs [12]. Pour l'instant, il n'y a pas d'analyse mathématique qui explique cette efficacité des réseaux convolutifs profonds. Il y a donc besoin de nouvelles avancées mathématiques pour comprendre les comportements et les limitations de ces réseaux profonds.

Chapitre 2

Les réseaux génératifs

Dans l'article précédent, nous avons vu comment entrainer de façon supervisée des réseaux de neurones. Ceci permet de résoudre efficacement des problèmes de classification, par exemple de reconnaissance d'images. Ce qui est peut être encore plus surprenant, c'est que ces réseaux de neurones sont également utilisés de façon non-supervisée afin de générer automatiquement des textes ou des images « virtuelles », ce que l'on appelle souvent des « deep fakes ». Dans ce second article, je tisserai un lien entre l'apprentissage de réseaux de neurones génératifs et la théorie du transport optimal. Ce problème a été posé par Gaspard Monge au 18e siècle, puis il a été reformulé par Leonid Kantorovitch au milieu du 20e siècle. Il est maintenant devenu un outil de choix pour aborder l'explosion récente de la science des données.

2.1 Réseaux de neurones génératifs

Au lieu d'utiliser des réseaux de neurones pour analyser des images, des travaux de 2014 [6] ont montré qu'on pouvait les utiliser « à l'envers » afin de générer des images. Ces réseaux de neurones génératifs trouvent par exemple des applications pour les effets spéciaux, les jeux vidéo et la création artistique. On retrouve des questions similaires dans l'apprentissage des voitures autonomes et la résolution de jeux de stratégie. La figure 2.1 montre la structure d'un tel réseau g_w, qui dépend de poids w. Les couches jouent en quelque sorte des rôles miroirs par rapport à l'architecture des réseaux de neurones discriminatifs exposés dans l'article précédent. A partir d'une entrée y composée d'un petit nombre de valeurs, qui sont typiquement tirées aléatoirement, on génère une image $x = g_w(y)$.

Le problème d'apprentissage de tels réseaux est non-supervisé : on dispose uniquement d'un grand nombre d'images d'apprentissage, sans indication sur ce qu'elles contiennent. Il n'y a plus besoin d'intervention humaine pour indiquer au réseau le contenu des images qu'il doit reconnaitre. La collecte de données est ainsi plus simple que pour l'entrainement de réseaux discriminatifs. De plus, ce principe d'apprentissage non supervisé est proche de la façon dont les enfants apprennent, principalement en observant et manipulant le monde qui les entoure. Le but est alors de sélectionner les poids w des neurones du réseau g_w de sorte que les images aléatoires générées (les images fausses, « fakes » en anglais) ressemblent le plus possible aux images d'apprentissage.

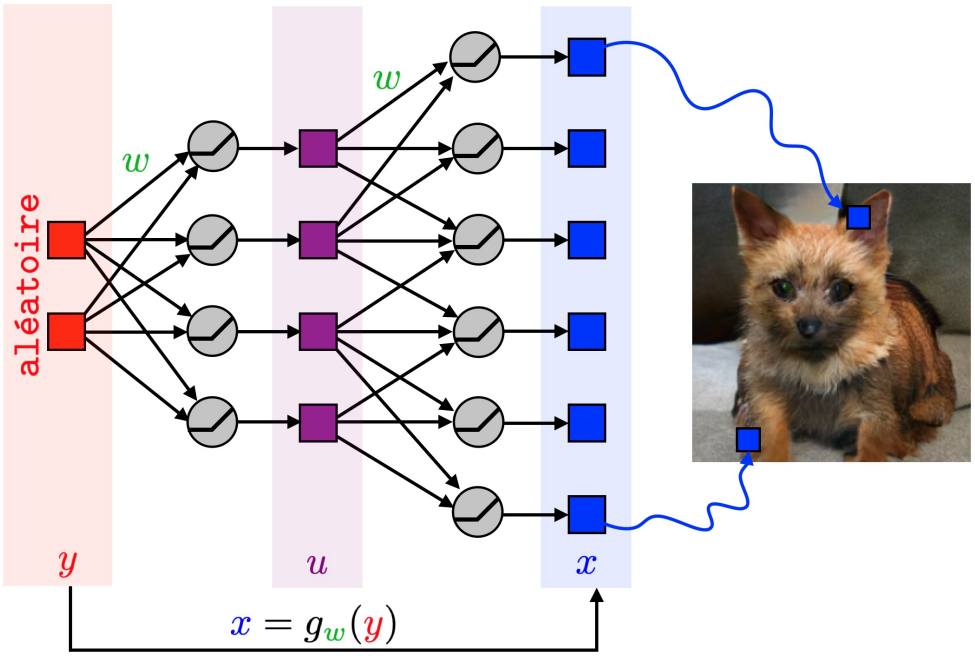

FIGURE 2.1 – Exemple d'un réseau de neurones génératif simplifié (un réseau permettant de générer des images aussi complexes possède plus de couches).

2.2 L'apprentissage non supervisé de réseaux génératifs

Le but des réseaux de neurones génératifs n'est pas de résoudre une tâche telle que la reconnaissance d'objets dans les images. Afin d'entrainer les poids w du réseau, il faut formaliser mathématiquement le problème. Il s'agit de générer un ensemble d'images « virtuelles » (les fausses) qui ressemblent aux images réelles d'une base de données. Il ne s'agit pas simplement qu'une image générée ressemble à une image réelle, il faut mettre en correspondance les deux ensembles d'images. Par exemple, si dans la base de données il y a la moitié d'images de chiens et la moitié d'images de chats, il faut que le réseau génère également pour moitié des chiens et pour moitié des chats.

On va noter $\{z_1, z_2, \ldots, z_n\}$ l'ensemble des n images de la base de données. Le nombre n d'images est très grand, de l'ordre de plusieurs milliers ou millions. Etant donné un réseau de neurones génératif g_w, qui est paramètré par ses poids w, on note $\{x_1, x_2, \ldots, x_n\}$ un ensemble de n images « fausses » générées aléatoirement par le réseau. Pour générer la première image fausse x_1, ceci signifie que l'on tire aléatoirement les valeurs d'entrée y_1 et qu'on applique le réseau à ces entrées, pour obtenir l'image virtuelle $x_1 = g_w(y_1)$. On a ensuite fait la même chose avec $x_2 = g_w(y_2)$ et ainsi de suite.

Le but de l'apprentissage non supervisé est donc de trouver des poids w de sorte que l'ensemble des fausses images $\{x_1, \ldots, x_n\}$ soit le plus proche de l'ensemble des images $\{z_1, \ldots, z_n\}$ de la base de données. Le problème d'optimisation s'écrit ainsi

$$\min_w \text{Distance}(\{x_1, \ldots, x_n\}, \{z_1, \ldots, z_n\}).$$

Il faut ici se rappeler que les images générées $\{z_1, \ldots, z_n\}$ dépendent du réseau g_w et donc des poids

w. On peut reformuler le problème précédent comme

$$\min_{w} \text{Distance}(\{g_w(y_1), \ldots, g_w(y_n)\}, \{z_1, \ldots, z_n\}).$$

La question mathématique qui se pose est donc de définir une notion de distance entre deux ensembles de points. Il existe de nombreuses façons de le faire, et nous allons en expliquer une qui est bien adaptée à ce problème d'apprentissage. Elle exploite la théorie du transport optimal.

2.3 Le transport optimal de Monge

Le problème de transport optimal est formulé par Gaspard Monge [16] en 1781, pour des applications militaires. La question posée est de déterminer la façon la plus économe de transférer des objets depuis un ensemble de sources $\{x_1, \ldots, x_n\}$ vers un ensemble de destinations $\{z_1, \ldots, z_n\}$. Pour Monge, il s'agit de transférer de la terre depuis des déblais pour créer des remblais. Mais cette question trouve une multitude d'applications. Pour le problème d'entrainement de réseaux génératifs, les sources sont les images fausses générées par le réseau et les destinations sont les images de la base de données.

Il faut ainsi relier chaque source, par exemple x_1 vers un unique point de destination, que l'on va noter z_{s_1}, où s_1 est un entier entre 1 et n. De manière similaire, x_2 est relié à z_{s_2} et ainsi de suite. Par exemple, à la figure 2.2, on a relié x_2 à z_5, ce qui signifie que $s_2 = 5$. Il faut également que chacune des n destinations soit approvisionnée par une source. Ceci signifie par exemple que x_1 et x_2 ne peuvent pas être reliés à la même destination, il faut relier toutes les sources à des destinations différentes. Ceci signifie que $\{s_1, \ldots, s_n\}$ doit être une permutation des n premiers nombres entiers. Par exemple, sur la figure 2.2, sur un exemple simple avec $n = 6$ éléments, on a choisit sur la gauche la permutation

$$(s_1 = 1, s_2 = 5, s_3 = 4, s_4 = 6, s_5 = 3, s_6 = 2).$$

Le problème de Monge consiste alors à trouver la permutation qui minimise la somme des coûts de transport. Monge a décidé que le coût de transport entre une source x et une destination z est égal à la distance euclidienne $\|x - z\|$ entre les deux points, mais on peut choisir un autre coût : par exemple un temps de trajet ou bien le prix nécessaire en essence si on utilise des camions, etc. On doit ainsi résoudre le problème

$$\min_{\text{permutation } s} \|x_1 - z_{s_1}\| + \|x_2 - z_{s_2}\| + \ldots + \|x_n - z_{s_n}\|.$$

Une fois que l'on a calculé une permutation $s^\star = (s_1^\star, \ldots, s_n^\star)$ optimale (c'est-à-dire qui est solution du problème précédent), on définit la distance entre les ensembles de points comme la valeur du cout total de transport

$$\text{Distance}(\{x_1, \ldots, x_n\}, \{z_1, \ldots, z_n\}) \overset{\text{def.}}{=} \|x_1 - z_{s_1^\star}\| + \|x_2 - z_{s_2^\star}\| + \ldots + \|x_n - z_{s_n^\star}\|.$$

La difficulté pour calculer cette distance est que le nombre total de permutations à tester est très grand. En effet, pour le choix de s_1 il y a n possibilités, pour celui de s_2 il en y a $n-1$ (puisque la valeur de s_1 est prise), pour s_2 il y en a $n-2$, etc. Donc le nombre total de permutations est égal à $n!$, la factorielle du nombre n, qui est définie comme

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1.$$

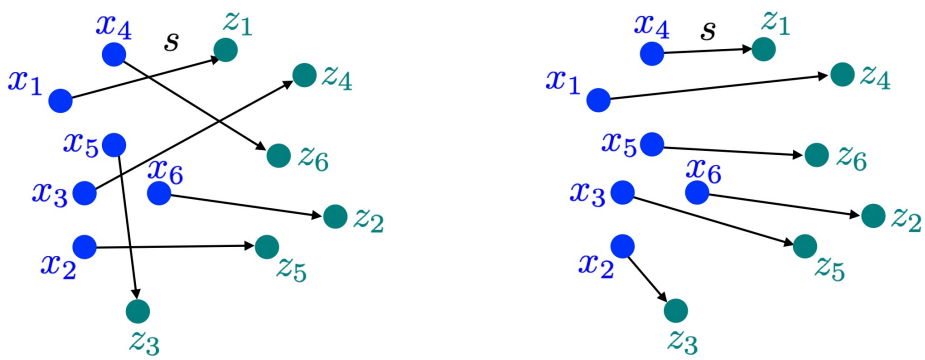

FIGURE 2.2 – Exemple à gauche d'une permutation s non-optimale et à droite de la permutation optimale, dans le cas de 6 points en dimension 2.

Pour $n = 6$ comme à la figure 2.2, il y a donc

$$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720 \text{ permutations possibles.}$$

Dans ce cas simple, on peut toutes les tester et choisir la meilleure, qui est, comme montré à la droite de la figure 2.2,

$$(s_1 = 4, s_2 = 3, s_3 = 5, s_4 = 1, s_5 = 6, s_6 = 2).$$

La difficulté est que pour $n = 70$, il y a plus de 10^{100} possibilités, ce qui est à comparer aux 10^{79} atomes dans l'univers ... Et pour entrainer des réseaux de neurones, n est encore beaucoup plus grand ! Il a donc fallu attendre plusieurs révolutions mathématiques et algorithmiques pour pouvoir obtenir une méthode permettant de résoudre ce problème.

2.4 Le transport optimal de Kantorovitch

Monge a remarqué que les solutions de son problème ont des structures très particulières. Par exemple, on peut observer sur la figure 2.2, à droite, que les trajets optimaux ne se croisent pas, et Monge l'avait prouvé dans son article [16]. Mais ceci n'est pas suffisant pour résoudre le problème, car il existe encore énormément de trajectoires sans croisement. Il a fallu plus de 200 ans pour comprendre comment obtenir plus d'information sur les solutions afin de les calculer efficacement. C'est Leonid Kantorovitch qui a trouvé en, 1942 [9], une nouvelle formulation du problème de transport optimal calculable rapidement. Il a autorisé chaque source à se diviser en plusieurs parties, par exemple deux parties égales avec une pondération de $1/2$ chacune. Cette division de la production est intéressante car elle simplifie le problème d'optimisation. Elle est également naturelle pour les préoccupations de Kantorovitch qui était de modéliser et planifier la production en économie. Il a d'ailleurs obtenu le prix Nobel d'économie pour cette idée. Conjointement à ces travaux pionniers de Kantorovitch, George Dantzig a trouvé en 1947 l'algorithme du simplexe [4], qui permet de résoudre efficacement des problèmes de transport de grande taille. Sa complexité numérique pour résoudre un problème de transport optimal entre n points est de l'ordre de $n^3 = n \times n \times n$, ce qui est beaucoup plus faible que $n! = n \times (n-1) \times \ldots \times 2 \times 1$. Il est au cœur d'un très grand nombre de systèmes industriels qui doivent optimiser l'adéquation entre des moyens de production et de consommation. Et on peut du coup également l'utiliser pour entrainer des réseaux de neurones génératifs ! On pourra

FIGURE 2.3 – Deux exemples de « deep fakes » qui sont des images virtuelles interpolant entre chats et chiens.

regarder [19] pour plus de détails sur la théorie du transport optimal, les algorithmes efficaces et ses applications à la science des données.

2.5 Les réseaux antagonistes

Une difficulté pour appliquer le transport optimal pour entrainer des réseaux génératifs est qu'il faut choisir le coût de transport entre deux images. On pourrait calculer la distance euclidienne entre les pixels des images, mais ceci ne marche pas bien, car cela ne prend pas en compte la géométrie des objets présents dans les images. Une idée très fructueuse a été introduite en 2014 par Ian Goodfellow et ses collaborateurs [6]. Elle consiste à utiliser un second réseau de neurones pour déterminer ce coût de transport [1]. Ce second réseau f, nommé réseau adversaire, joue un rôle de discriminateur. Alors que le but du générateur g est de générer des images fausses très ressemblantes, le but de f est au contraire de faire de son mieux pour reconnaitre les vraies et les fausses images. Ces deux réseaux sont entrainés conjointement, c'est pour cela que l'on parle de réseaux antagonistes. L'entrainement de ces deux réseaux correspond à ce que l'on appelle un jeu à somme nulle, introduit par John Von Neumann en 1944 [17] et généralisé ensuite par John Nash en 1950 [18], qui a obtenu tout comme Kantorovitch le prix Nobel d'économie.

Ces avancées récentes [6] ont ainsi permis d'obtenir des résultats excellents pour la génération d'images. La figure 2.3 montre des résultats obtenus avec la méthode expliquée dans [2] et son utilisation pour calculer des « chemins » d'images entre chiens et chats.

Remerciements

Je remercie Gwenn Guichaoua pour sa relecture attentive, ainsi que Sébastien Racanière et Vincent Barra pour leurs corrections.

Bibliographie

[1] Martin Arjovsky and Leon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*, 2017.

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *Proc. ICLR 2019*, 2019.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4) :303–314, 1989.

[4] George B Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4) :500–544, 1952.

[8] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1) :106–154, 1962.

[9] Leonid Kantorovich. On the transfer of masses (in russian). *Doklady Akademii Nauk*, 37(2) :227–229, 1942.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553) :436–444, 2015.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[13] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2) :146–160, 1976.

[14] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133, 1943.

[15] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences*, pages 666–704, 1781.

[16] Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton university press, 1953.

[17] John F Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1) :48–49, 1950.

[18] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5–6) :355–607, 2019.

[19] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088) :533–536, 1986.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958, 2014.