Eduardo M. Cabana D., Hakim Bouguettaya, Marianne Déry, Jérémie Provencher
Sciences de la nature

Analyse de différentes intelligences artificielles

Rapport présenté à
Olivier Rousseau
Dans le cadre du cours
Projet en mathématiques
Groupe 1

Cégep de l'Outaouais
Campus Gabrielle-Roy
Date 22 mars 2019

Table des matières

Abstract

Our project is an investigation on the nature of neural networks in the wider context of artificial intelligence and on its inner workings in image recognition technology. We described, analyzed and tested two methods we used to train neural networks specifically for numbers' images recognition: back propagation and genetic algorithms. In our report you will find a detailed mathematical and conceptual description of what is artificial intelligence in general and what constitutes a neural network more specifically. While there are many more ways to train neural networks, we only explain in our report the mathematics and functioning behind back propagation and genetic algorithms. Since we have put into practice both training methods by programming our own neural networks with Python, the report includes our results for the efficiency of both methods.

Notre projet est une investigation sur la nature des réseaux de neurones dans le cadre de l'intelligence artificielle et sur leurs fonctionnements dans la technologie de reconnaissance d'image. Nous avons décrit, analysé et testé deux méthodes utilisées pour entrainer des réseaux de neurones spécifiquement pour la reconnaissance d'images de nombres : la rétro propagation et l'algorithme génétique. Dans notre rapport, vous trouverez une description mathématique et conceptuelle de ce qu'est l'intelligence artificielle en général et de ce que constitue un réseau de neurones plus spécifiquement. Même s'il existe plusieurs autres façons d'entrainer des réseaux de neurones, nous n'abordons dans notre rapport que les mathématiques et le fonctionnement derrière la rétropropagation et l'algorithme génétique. Étant donné que nous avons mis en pratique les deux méthodes d'entrainement en programmant nos propres réseaux de neurones avec Python, le rapport inclus nos résultats sur l'efficacité des deux méthodes.

Introduction

Mise en contexte

Plusieurs tâches qu'un humain doit effectuer sont monotones et lentes à réaliser. Prenons par exemple le journaliste passionné qui doit remettre un rapport des résultats des derniers matchs sportifs ou des derniers résultats électoraux [1], ou encore le travailleur qui doit vérifier les vidéos

d'une plateforme web pour tenter d'identifier les *deepfake* [2]. L'intelligence artificielle, qui prend de plus en plus d'ampleur ces dernières années, est là pour répondre à ce genre de besoins. En effet, cette technologie permet de remplacer l'humain dans certaines tâches en étant même plus efficace que ce dernier. Tobi, une intelligence artificielle qui génère un texte, a ainsi permit de transmettre les résultats de chacune des 2222 municipalités de l'Allemagne, et ce en allemand et en français [1]. L'intelligence artificielle permet également l'analyse de données récoltées, tel que l'identification de différents types de roches souterraines. [3]

L'intelligence artificielle permet également de résoudre plusieurs situations en lien avec les problèmes actuels, tel que les changements climatiques. En effet, cette technologie permet à certains groupes de résoudre plusieurs problèmes en lien avec la recherche de solutions de développements durables que ce soit pour la préservation des océans ou pour diagnostiquer la santé environnementale. Entre autres, les chercheurs du MIT veulent rendre un robot déguisé en poisson (le *Soft Robotic Fish*) autonome grâce à l'intelligence artificielle. Ce robot, pour le moment contrôlé à distance, permet d'étudier le comportement de diverses espèces marines dans leurs habitats naturels. Il permet également de voir comment leur comportement change selon la pollution des eaux. [4]

Théorie

Gradient

Le gradient (∇f) d'une fonction à plusieurs variables indépendantes est une matrice verticale qui regroupe l'ensemble des dérivées partielles de la fonction selon chacune des variables.

Par exemple, le gradient de la fonction $y = f(x_1, x_2, \dots, x_n)$ serait :

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \dots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Le produit d'Hadamard

Le produit d'Hadamard ($s \odot t$) est une opération matricielle qui consiste à faire le produit de chacun des termes de deux matrices de même dimension qui occupent la même position.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \end{bmatrix}$$

La dérivée directionnelle

La dérivée directionnelle indique, dans le cas d'une fonction en trois dimensions, les changements relatifs en z pour un vecteur de direction donné $\vec{s}(\overrightarrow{x_1, y_1})$ à un point donné P(x_0, y_0). L'équation de la dérivée directionnelle est :

$$\frac{df}{ds} = \frac{\nabla f \cdot \vec{s}}{||s||} = \big||\nabla f|\big| \cos \phi$$

où ϕ est l'angle entre le gradient de la fonction et la direction choisie

Ainsi, pour avoir une augmentation maximale de la fonction$f$, l'angle ϕ doit être égal à 0. En d'autres mots, la direction $\vec{s}$ choisie doit être équivalente au gradient de la fonction.

Description du rapport

Ce rapport commence par présenter les problèmes d'intelligence artificielle en général. Il vulgarise le but et le fonctionnement général des intelligences artificielles ainsi que la division de celles-ci en différentes catégories, soit les intelligences faibles et les fortes. Par la suite, les

problèmes en intelligences artificielles faibles sont décrits en plus amples détails. Le rapport présente ensuite une méthode qui permet de bien décortiquer un problème en intelligence artificielle en différentes composantes. De plus, la structure d'un réseau de neurones, puisque c'est sur technique d'apprentissage que nous nous sommes attardés, est présentée exhaustivement. Le rapport détaille par la suite le fonctionnement d'un réseau de neurone, c'est-à-dire comment celui-ci traite les données fournies afin de trouver une réponse.

De plus, deux méthodes d'apprentissage adaptées aux réseaux de neurones sont décrites, soit la rétropropagation du gradient et les algorithmes génétiques. Dans la section sur la rétropropagation, le rapport commence par expliquer comment la performance du réseau est évaluée à l'aide de la fonction de coût. Ensuite, il explique comment cette fonction est minimisée à l'aide du gradient, trouvé grâce à l'algorithme de rétropropagation. Afin de conclure cette section, différentes améliorations pouvant être implémentées à un réseau de neurones sont présentés. Ces améliorations sont le choix de la fonction de coût, l'agrandissement artificielle des données d'entraînement et quatre techniques servant à contrer le sur-apprentissage d'un réseau, soit l'arrêt prématuré de l'apprentissage, la régulation L1 et L2, et le « dropout ».

Une troisième partie du rapport est consacrée à l'explication de l'apprentissage par algorithme génétique. Suite à une brève introduction, les six étapes de fonctionnement de l'algorithme génétique sont expliquées, soit l'initialisation aléatoire de l'algorithme, l'évaluation de la performance avec la fonction « fitness », la sélection des individus en fonction de probabilités, le croisement des individus, les mutations apportées et l'émergence d'une nouvelle population. Dans cette section du rapport, il est décrit comment l'algorithme génétique utilise la fonction de coût pour optimiser un réseau de neurones à partir d'un modèle évolutif.

Une quatrième partie du rapport porte sur la mise en pratique des méthodes décrites et expliquées plus haut. Cette partie décrit les réseaux de neurone construits par notre équipe sous forme informatique et décrits leurs résultats pour résoudre divers problèmes avec divers types d'apprentissages. Il décrit aussi les méthodes informatiques utilisées pour réaliser ces réseaux et les problèmes lors de la création des réseaux. Cette section comprend aussi une comparaison des deux méthodes en présentant les avantages et les inconvénients de chacune des méthodes d'apprentissage.

Intelligence artificielle

Qu'est-ce qu'une intelligence artificielle ? Il faut tout d'abord savoir ce qu'est l'intelligence avant d'aborder le sujet. Selon le dictionnaire Larousse, l'intelligence est l'«aptitude d'un être humain à s'adapter à une situation, à choisir des moyens d'action en fonction des circonstances ». Durant une conférence avec le chercheur en intelligence artificielle Jean Ponce, l'écrivain et économiste français, Jacques Attali, a défini l'intelligence comme étant «la faculté du cerveau humain à faire des liens entre des choses qui n'en avaient apparemment pas» [5]. Ainsi, l'intelligence artificielle ne fait que reproduire cette capacité d'adaptation et cette capacité à faire des liens grâce aux ordinateurs et aux mathématiques.

Les intelligences artificielles sont divisées en deux grandes familles : les IA faibles et les IA fortes. Pour ce qui est des IA faibles, elles sont conçues afin de répondre à des problèmes spécifiques. Ainsi, ces intelligences s'adaptent et évoluent aux problèmes en question. Alors que, pour les AI fortes, elles évoluent sans se limiter à un problème. Celles-ci sont dotées de conscience, de créativité, d'anticipation et de toute autre aptitude que l'intelligence humaine possède. Les IA fortes sont complexes et encore mal comprises. Elles sont en processus de recherche et au cœur de grands débats, alors elles ne seront pas élaborées dans ce rapport. À partir de maintenant, quand l'intelligence artificielle sera mentionnée, elle fera référence aux IA faibles [5].

Problèmes en intelligence artificielle

Avant toutes choses, il est important de comprendre ce qu'est un problème en intelligence artificielle. Le but de l'IA est de trouver la ou les réponses au problème pour lequel il a été créé. Il y a une panoplie de problèmes différents qui existent : reconnaître différents numéros, reconnaître s'il y a ou non un chat sur des photos, suggérer des chansons que vous pourriez aimer ou diriger une automobile pour qu'elle ne fasse pas de collision. Cependant, pour que l'IA trouve les réponses au problème, le concepteur doit changer de point de vue. Par exemple, le problème n'est plus de reconnaître des numéros. Le problème devient : comment l'IA va-t-il reconnaître les numéros ? Or, l'IA doit faire des liens et ces liens se font par entraînement. L'entraînement consiste à fournir une énorme quantité de données initiales, aussi appelées données d'entrée, pour que l'IA fasse des liens entre ces données initiales et les réponses, aussi appelé données de sortie. Il y a plusieurs techniques différentes pour faire ces liens et elles seront expliquées prochainement. Ce qui est important à savoir est qu'il y a, dans tous les problèmes, des données d'entrée et de sortie qui

dépendent du problème en question. Ces données doivent être numériques pour pouvoir être utilisées dans un programme qui utilise des équations mathématiques, donc les données initiales et les réponses du problème sont associées à des données d'entrée et de sortie numériques. Pour déterminer les données d'entrée et de sortie, il faut se mettre dans la peau de l'IA. Pour les données d'entrée, il faut se demander : qu'est-ce qu'on connait de façon générale et qui est présent à chaque situation du problème ? Ensuite, ce qui varie d'une situation à l'autre sera représenté sous forme de valeurs. Pour les données de sortie, il faut se demander : quelles sont les réponses possibles du problème ? Prenons, par exemple, la reconnaissance de chiffre. L'IA doit reconnaître le bon chiffre que l'on dessine. Si je dessine un 5, la réponse doit être 5. Quelles seraient les données d'entrée et de sortie dans ce problème ? Pour les données d'entrée, ce qui est connu et qui est présent à chaque fois que l'on dessine sont les pixels de l'image. Donc chaque pixel du dessin serait une donnée d'entrée. Ce qui varie à chaque fois sont les tons de gris des pixels. En effet, certains pixels seront blancs, noirs et d'autres gris dépendamment du dessin. Si je dessine un 3, certains pixels noirs du 3 seront blancs si je dessine un 5. Donc on attribuera une valeur aux tons de gris. Par exemple, le blanc sera représenté par 0, le noir par 1 et les valeurs entre 0 et 1 seront représentatives des intensités des tons de gris. Donc maintenant, chaque donnée d'entrée est représentée par une valeur. Pour les données de sortie, les réponses attendues sont des chiffres. Il existe 10 chiffres, allant de 0 à 9, donc il y aura 10 données de sortie qui seront chacune associées à un chiffre allant de 0 à 9. Les critères définissant une réponse fournie par l'IA peuvent varier, mais souvent, la donnée de sortie qui aura la plus grande valeur sera la réponse fournie. Donc si je dessine un 5, une intelligence artificielle bien entraînée aura la plus grande valeur sur la donnée de sortie représentant le chiffre 5.

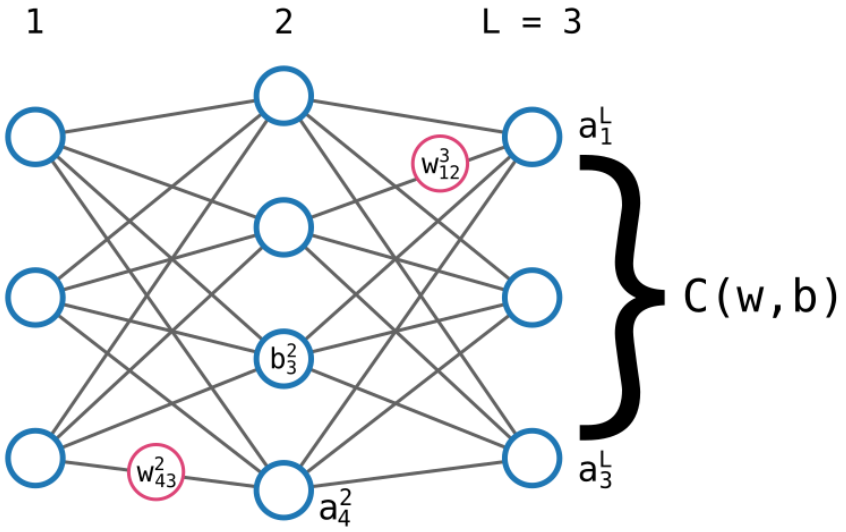
Apprentissage

L'apprentissage est évidemment primordial pour une intelligence artificielle. Comme expliqué légèrement auparavant, il y a plusieurs techniques d'apprentissage. Cependant, seul les réseaux de neurones seront étudiés. Cette technique fait partie d'une grande famille que l'on appelle « apprentissage automatique » ou « machine learning » en anglais. Cette famille est divisée en trois catégories : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. Pour l'apprentissage supervisé, on connait les bonnes réponses du problème en fonction des données initiales. Or, ce que l'on cherche à faire, c'est trouver les meilleurs liens qui permettent de fournir les bonnes réponses pour les bonnes données initiales. Les méthodes utilisées dans cet apprentissage sont, entre autres, les réseaux de neurones, le « deep learning », la

régression statistique et l'apprentissage relationnel. Pour l'apprentissage non supervisé, l'IA a des données d'entrée, mais elle n'a pas de sorties attendues. Elle doit donc prédire les liens entre les données d'entrée et de sortie par elle-même. Les techniques utilisées sont les « K moyennes » et le « clustering ». Pour le renforcement, l'IA a les données d'entrée et on connait les sorties attendues, mais on ne les fournit pas à l'IA. Ainsi, l'intelligence fournit par elle-même les sorties et on la récompense lorsque les sorties sont bonnes. Cette méthode est faible, puisqu'elle demande beaucoup de temps d'entraînement, mais elle est utilisée pour les jeux. Un bon exemple de cette technique est le programme AlphaGo.

Réseau de neurones

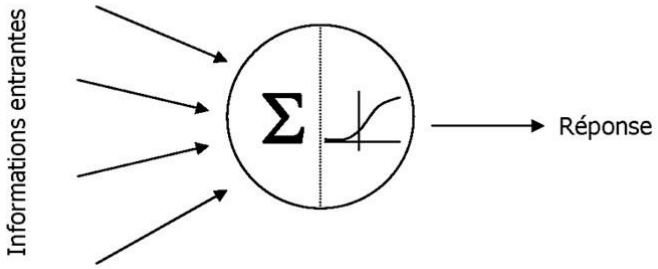
Pour bien comprendre le fonctionnement d'un réseau de neurones, il pourrait être visualisé comme ceci :



[6]

Voici quelques concepts clés avant d'expliquer son fonctionnement. Chaque cercle bleu est un neurone et chaque ligne grise est un poids (w). À l'intérieur même de chaque neurone se produit un phénomène important. Il y a toujours, au départ, une valeur initiale (z) qui, par une fonction d'activation, devient une valeur d'activation (a).

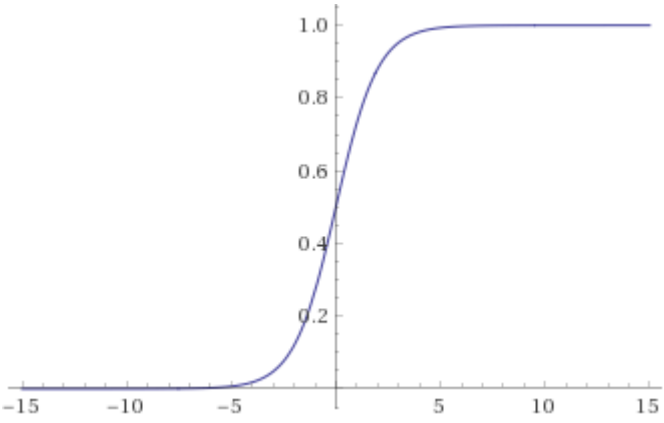
Figure 1 – Neurone artificiel (McCullogh et Pitts, 1943)



[7]

Cette fonction permet d'avoir un certain standard sur les valeurs de chaque neurone. Cette fonction est au choix, mais pour l'explication, j'utiliserai la fonction sigmoïde ($\sigma(x)$) que voici :

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



Voici la simple relation entre la valeur initiale et la valeur d'activation :

$$a = f(z)$$

Donc pour l'exemple :

$$a = \sigma(z)$$

Les neurones alignés verticalement représentent une couche de neurone (l). Les valeurs initiales des neurones de la première couche sont les données d'entrée et les valeurs d'activation des neurones de la dernière couche sont les données de sortie. Donc le nombre de neurone de la première et de la dernière couche change dépendamment du problème. Cependant, le nombre de couches intermédiaires et le nombre de neurones pour ces couches sont au choix. Le dernier élément important d'un réseau est le biais (b). Chaque neurone a un biais, soit un nombre qui modifie la valeur de son neurone. Ces éléments, soit les neurones, les poids et les biais, ne sont en fait que des nombres qui ont chacun un rôle précis. Voici la relation entre ces éléments :

$$z_i^l = \sum_{j=1}^{n} w_{ij}^l a_j^{l-1} + b_i^l$$

Où

- l représente la position de la couche
- i représente la position d'un neurone dans une couche à partir du haut
- j représente la position d'un neurone dans la couche $l-1$ à partir du haut
- w_{ij}^l représente le poids qui modifie le j^{ième} neurone au i^{ième} neurone

Donc si je veux trouver la valeur initiale du premier neurone de la deuxième couche (z_1^2), je dois multiplier toutes les valeurs d'activation de la première couche (a_j^1) à leur poids qui les relie à z_1^2 (w_{1j}^2). Je dois ensuite additionner toutes ces multiplications ($w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + w_{13}^2 a_3^1 + \cdots$). Après ça, j'additionne à cette somme le biais du premier neurone de la deuxième couche (b_1^2) et ce résultat est la valeur de z_1^2. Ensuite, z_1^2 devient a_1^2 et on recommence les mêmes calculs pour trouver la valeur initiale des neurones de la prochaine couche. Donc pour trouver la valeur d'activation des neurones de sortie, on doit effectuer énormément de calculs. Pour faciliter le processus, on effectue les calculs sous forme matricielle, puisqu'il est plus facile de travailler avec des matrices dans un programme informatique avec autant de données. L'équation devient :

$$Z^l = W^l A^{l-1} + B^l$$

Où

- Z^l représente la matrice colonne de toutes les valeurs initiales d'une couche de neurone l
- W^l représente la matrice de dimension j par i de tous les poids d'une couche de neurones l
- A^{l-1} représente la matrice colonne de toutes les valeurs d'activation de la couche de neurones $l-1$
- B^l représente la matrice colonne de tous les biais d'une couche de neurones l

Donc un réseau sans entraînement fourni des données de sortie aléatoires, puisque les poids et biais seront aléatoires. Par conséquent, ce que l'on cherche à faire avec l'entraînement, c'est de trouver la meilleure configuration de poids et de biais qui fournira les bonnes réponses au problème. Pour y arriver, il existe plusieurs méthodes et algorithmes que l'on colle au réseau. Dans ce rapport, la rétropropagation du gradient et l'algorithme génétique sont étudiés.

Rétropropagation

La rétropropagation est une technique d'entraînement qui se base sur les réponses connues du problème pour guider le réseau vers le bon choix de biais et de poids grâce à l'algorithme de rétropropagation.

Les données d'entraînement

Avec cette technique, en plus d'avoir des données d'entrée et des données de sortie, il faut également des données d'entraînement et des données de validation. Les données d'entraînement servent à évaluer la performance du réseau en comparant les données de sortie du réseau et les données attendues, c'est-à-dire les bonnes réponses. Les données de validation servent, tant qu'à

elle, à s'assurer que le réseau est capable de généralisé les liens trouvés à des données qu'il n'a jamais vues.

La fonction de coût

Afin d'avoir un réseau qui puisse apprendre, il est nécessaire de savoir si le réseau actuel performe bien. Pour ce faire, il faut comparer la réponse obtenue à l'activation de la dernière couche de neurones avec la valeur souhaitée.

Par exemple, prenons un réseau de neurones qui identifie si une certaine image est un cercle ou un triangle. Disons que ces images font 20x20 pixels, pour un total de 400 pixels. La première couche de neurones sera donc composée de 400 neurones, chacun représentant la valeur du pixel lui étant associé. Ces valeurs sont les données à analyser, x. Le réseau dira ensuite, dans la dernière couche, si l'image est celle d'un cercle ou d'un triangle. Cette couche aura donc deux neurones. Pour ce qui est de la couche mystère, il n'est pas important de savoir sa composition pour cet exemple.

Le réseau ressemblera donc à ceci :

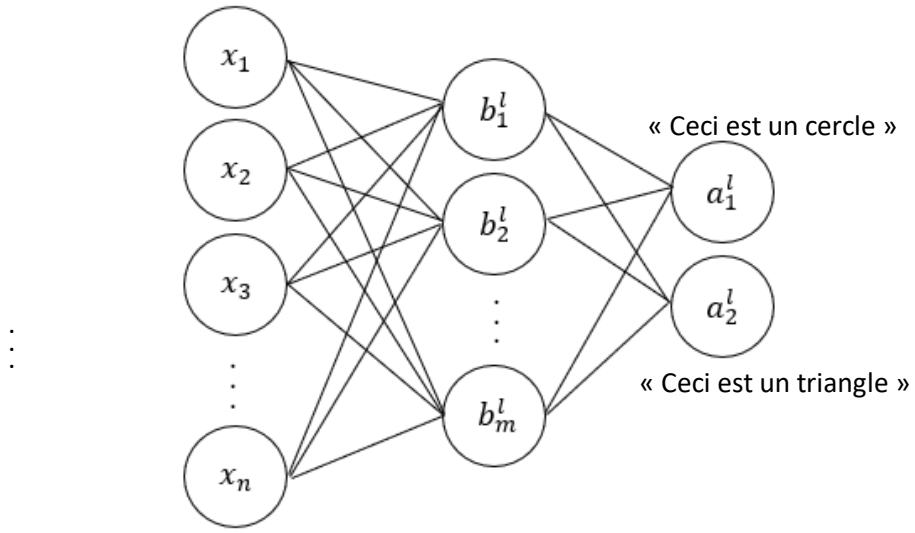
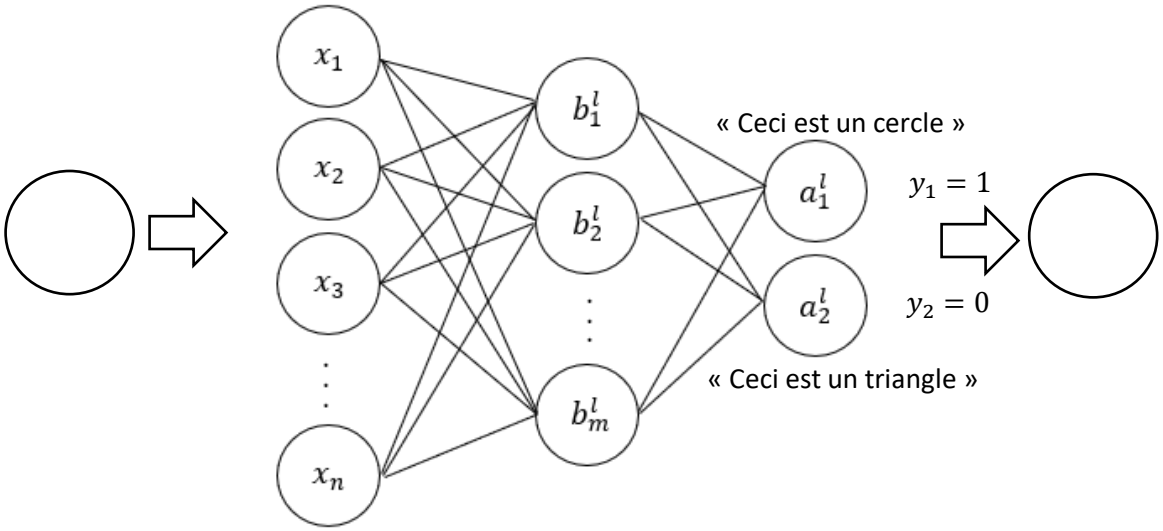


Figure A

Dans un cas idéal, si l'image fournie au réseau est celle d'un cercle, le neurone associé à la réponse « Ceci est un cercle. » s'activerait avec une valeur de 1. De plus, le neurone indiquant que c'est un triangle ne s'activerait pas, c'est-à-dire que son activation serait de 0. Ce cas idéal est la valeur souhaitée pour l'activation de la dernière couche de neurones :



où y_n est la valeur souhaitée pour a_n

Figure B

Cependant, avant que le réseau de neurones soit entraîné, la réponse sera aléatoire et elle pourrait ressembler à ceci :

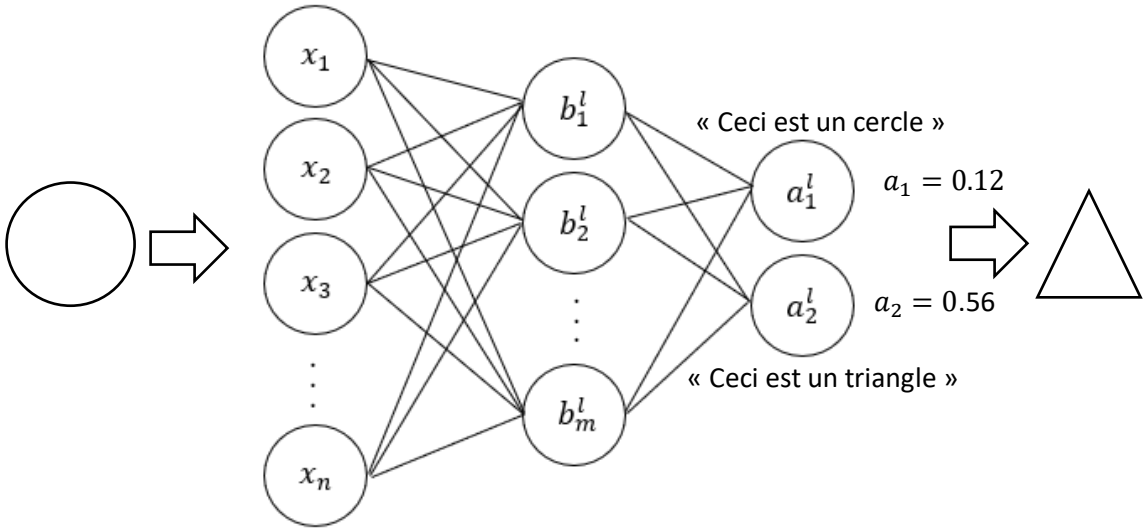


Figure C

Grâce à une certaine fonction, appelé fonction de coût, il sera donc souhaité de quantifier l'écart entre la réponse recherchée (*figure b*) et celle obtenue (*figure c*). La technique de rétropropagation du gradient cherche donc à minimiser cette fonction de coût. En minimisant la fonction, l'écart entre la valeur souhaitée et celle obtenue diminue jusqu'à ce que les deux soient essentiellement identiques. À ce moment, le réseau sera en mesure de correctement identifier l'image présentée :

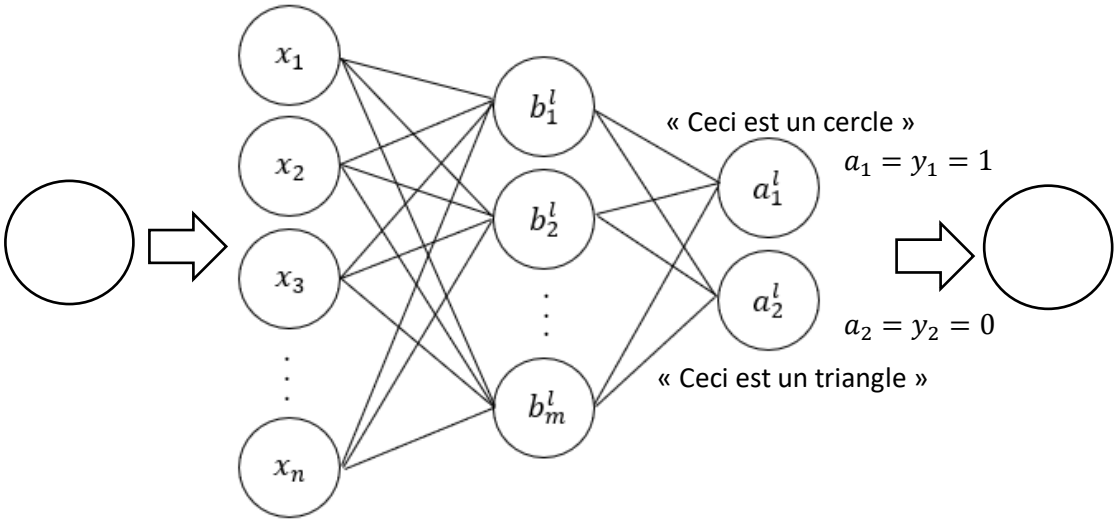


Figure D

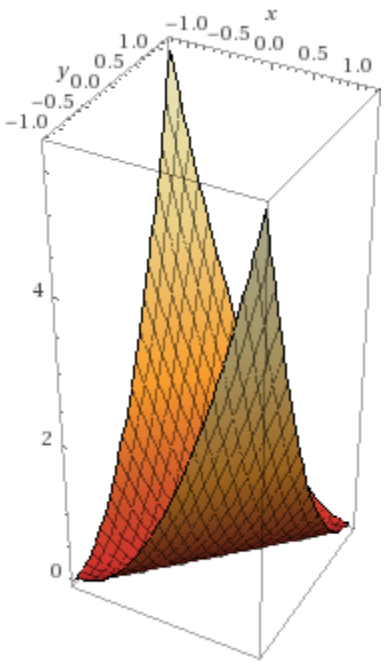
Pour évaluer cet écart entre la valeur souhaitée ($y(x)$) et celle obtenue (a), il faut définir une fonction C. Cette fonction peut prendre différentes formes, comme il sera présenté plus loin dans le rapport. Pour qu'une fonction soit considérée comme une fonction de coût, elle doit être positive en tout point et elle doit se rapprocher de 0 lorsque $y = a$ et s'éloigner de 0 lorsque $y \neq a$. Un exemple de fonction de coût répondant à ces deux critères est la fonction quadratique :

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2 .$$

Où $y(x)$ est un vecteur des réponses souhaitées pour une certaine donnée x dans chacune des neurones de la dernière couche, a est la matrice comprenant les valeurs d'activations de chacun des neurones de la dernière couche et n est le nombre total de données d'entraînement.

Tous les termes de l'équation sont positifs et il est facile de voir que lorsque $y = a$, la fonction sera 0 et lorsque y s'éloigne de a, la valeur de la fonction C augmente. Plus simplement,

on peut visualiser l'allure de cette fonction de coût selon $F(x, y) = (x - y)^2$ et elle ressemble à ceci :



Le choix de la fonction de coût est arbitraire. Cependant, selon la fonction choisie pour les neurones, certaines fonctions sont préférables afin d'éviter de ralentir le réseau. Ce concept sera expliqué en de plus amples détails dans la section traitant des améliorations possibles à apporter au réseau.

La descente de gradient

Il est important de se rappeler que la fonction de coût indique à quel point la réponse donnée par le réseau à l'étude est près de la bonne réponse. Ainsi, pour améliorer le réseau, il faut minimiser la fonction de coût sur l'ensemble des données disponibles en ajustant les poids et les biais. En faisant cela, le réseau sortira une réponse optimale pour le plus grand nombre de données possible.

De plus, lorsque le réseau minimise la fonction de coût, il n'est pas nécessaire de trouver le minimum global de la fonction. En effet, si le réseau est trop entraîné, celui-ci aura de la difficulté à généralisé son raisonnement à des données qui ne font pas partie des données d'entraînement. C'est ce qu'on appelle le sur-apprentissage, un concept qui sera mieux définit avec les améliorations possibles pour un réseau utilisant la descente de gradient. Ainsi, il suffit de trouver un minimum

local de la fonction. Grâce au gradient de cette fonction, il est possible de trouver un de ces minimums. En effet, le gradient étant l'ensemble des dérivées d'une fonction à plusieurs variables, il indique quelle direction augmente le plus la fonction et laquelle la diminue le plus.

En changeant les poids w de Δw, on change la fonction de coût C selon la règle

$$\Delta C \approx \nabla C \cdot \Delta w.$$

C'est la dérivée directionnelle de C dans la direction Δw. Comme présenté dans la théorie, pour que la dérivée directionnelle soit maximum, la direction prise soit celle du gradient de la fonction C. Au contraire, pour aller dans la direction où la dérivée directionnelle est minimum, soit la direction d'un minimum local, il faut avancer d'une certaine mesure η dans la direction opposée au gradient, soit $-\nabla C$.

Ainsi, ce changement Δw, définit par

$$\Delta w = -\eta \nabla C,$$

où η est le taux d'apprentissage du réseau,

permet de dire que la fonction de coût change selon la règle

$$\Delta C \approx -\eta \|\nabla C\|^2.$$

Les poids et les bias du réseau seront donc modifiés selon $\Delta w = -\eta \nabla C$ et $\Delta b = -\eta \nabla C$, donnant les règles

$$w_k \to w_k' = w_k - \eta \nabla C$$

$$b_l \to b_l' = b_l - \eta \nabla C ,$$

ce qui est la descente de gradient.

Tout ce qu'il reste à faire, c'est de déterminer le gradient de la fonction. Il serait possible de le trouver en déterminant les dérivées partielles de chacune des variables du réseau, mais ceci serait très long et demanderait trop de ressources à l'ordinateur. C'est pourquoi la technique de rétropropagation du gradient a été créée, elle permet d'approximer le gradient de façon efficace.

L'algorithme de rétropropagation du gradient

L'idée derrière l'algorithme de rétropropagation du gradient est de propager l'erreur présente dans le réseau à l'envers. Cette erreur représente l'écart présent entre les données sorties par le réseau et celles qui devrait sortir. Grâce à cette erreur, il est possible de calculer le gradient de la fonction de coût plus efficacement qu'en calculant directement les dérivées partielles de chacune des variables. Grâce à ce gradient maintenant trouvé, il sera possible de modifier les variables, soit les poids et les biais, de façon à faire diminuer la fonction de coût.

Cependant, avant de se lancer dans l'algorithme de rétropropagation du gradient, une notion importante sera présentée. L'erreur d'une couche de neurone, δ^l, est la matrice des valeurs auxquelles les activations a doivent diminuer. L'erreur est donc la dérivée partielle de la fonction de coût selon z, $\frac{\partial C}{\partial z_j^L}$ Ainsi, δ, $\frac{\partial C}{\partial w_{jk}^l}$ et $\frac{\partial C}{\partial b_j^l}$ représentent tous les valeurs auxquelles une matrice de variables, soit la matrice des activations, des poids et des biais respectivement, doit changer pour diminuer la fonction de coût à la couche l.

La notation

δ^l est l'erreur à la couche l,

$\nabla_a C$ est le gradient de la fonction de coût selon l'activation a,

$\odot$ est le produit d'Hadamard, définit dans la théorie,

$\sigma'(z^l)$ est la dérivée de la fonction $\sigma(z^l)$ à la couche l,

et $(w^{l+1})^T$ est la matrice transposée des poids de la couche $l+1$, soit la couche suivant celle étudiés.

l, $l+1$ et $l+1$ font références à la couche étudiée, celle suivante et celle précédente, respectivement.

j, k et jk font référence à la position de l'erreur, du biais, de l'activation et du poids étudié.

L'algorithme

L'algorithme se base sur 4 équations, qui seront démontrées sous peu :

BP1 $\delta^L = \nabla_a C \odot \sigma'(z^l)$

BP2 $\delta^l = ((w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$

BP3 $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$

BP4 $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

et va comme suit :

1- Entrer les données x dans les activations des premières neurones a^1_k.

Ici, les données relatives à un entraînement entrent dans le réseau.

2- Pour chaque neurone, calculer $z^l = w^l a^{l-1} + b^l$ et $a^l = \sigma(z^l)$

Les données précédentes avancent dans le réseau, de couches en couches, comme présenté précédemment.

3- Calculer le vecteur d'erreur $\delta^L = \nabla_a C \odot \sigma'(z^l)$

Au lieu d'entrer les données x, il faut calculer l'erreur de la dernière couche.

4- Calculer l'erreur, pour chaque couche $l = L-1, L-2, \dots, 2$, $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1} \odot \sigma'(z^l)\right.$

À partir de cette erreur, l'erreur de chacune des couches précédentes est estimée, en reculant. C'est la rétropropagation du gradient.

5- Trouver le gradient de la fonction grâce aux équations $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ et $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

Preuve des équations de la rétropropagation du gradient

Preuve de BP1

Par définition,

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

Par dérivée en chaine,

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_j^L}\left(\frac{\partial a_j^L}{\partial z_j^L}\right)$$

Puisque $a_j^L = \sigma\left(z_j^L\right)$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_l^L)$$

En forme matricielle :

$$\delta^L = \nabla_a C \odot \sigma'(z^l)$$

Preuve de BP2

Par définition,

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

Par dérivée en chaine,

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\left(\frac{\partial z_k^{l+1}}{\partial z_j^l}\right)$$

En inversant les termes,

$$\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \left(\frac{\partial C}{\partial z_k^{l+1}} \right)$$

Puisque, par définition, $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$

$$\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \left(\delta_k^{l+1} \right)$$

Comme $\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_l^j)$ (*),

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_l^j\right)$$

En forme matricielle

$$\delta^l = \left(w^{l+1}\right)^T \delta^{l+1} \odot \sigma'(z^l)$$

$$(*) \frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

Ces deux dernières preuves ont été prises du livre Neural Networks and Deep Learning [8]. Nous avons effectué les deux preuves suivantes.

Preuve de BP3

Par dérivée en chaine,

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \left(\frac{\partial z_l^l}{\partial w_{jk}^l} \right)$$

Puisque $\frac{\partial z_l^l}{\partial w_{jk}^l} = a_k^{l-1}$ (**),

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l \, a_k^{l-1}$$

$$(**) \frac{\partial z_l^l}{\partial w_{jk}^l} = \frac{\partial \left(w_j^l a_k^{l-1} + b_l^j \right)}{\partial w_{jk}^l} = a_k^{l-1}$$

Preuve de BP4

Par définition,

$$\delta_j^l = \frac{\partial C}{\partial z_l^j}$$

Par dérivée en chaine,

$$\delta_j^l = \frac{\partial C}{\partial b_l^j}\left(\frac{\partial b_l^j}{\partial z_l^j}\right)$$

Puisque $\frac{\partial b_l^j}{\partial z_l^j} = 1$ (***),

$$\delta_j^l = \frac{\partial C}{\partial b_l^j}$$

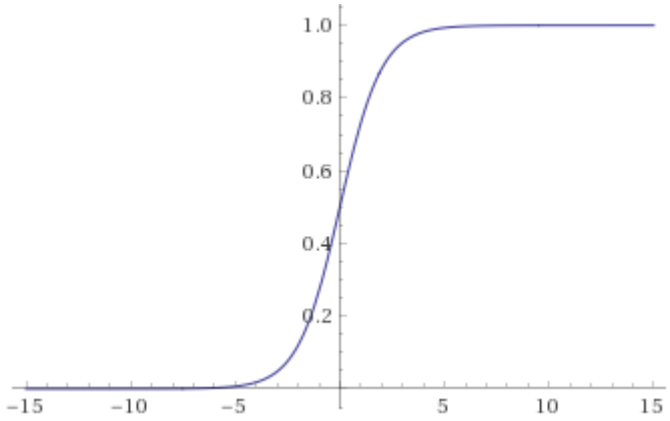
$$(***)\ \frac{\partial b_l^j}{\partial z_l^j} = \frac{\partial\left(z_j^l - w_j^l a_k^{l+1}\right)}{\partial z_l^j} = 1$$

Améliorations

Il existe deux buts pour lesquels une personne voudrait améliorer son réseau de neurone. Évidemment, le but principal est de rendre le réseau le plus efficace possible sur des données qu'il n'a jamais analyser en s'entraînant. Cependant, quelqu'un qui conçoit un réseau de neurone veut également s'assurer que ce-dernier soit le plus efficace possible, c'est-à-dire que l'entraînement se fasse rapidement. Il existe plusieurs méthodes visant à améliorer un réseau de neurone selon l'un ou l'autre de ces deux buts. Cette section en présentera quelques-unes.

Choix de la fonction de coût selon la fonction appliquée aux neurones

Un réseau qui apprend plus vite est préférable à celui qui apprend plus lentement. Il faut donc bien choisir la fonction de coût qui optimisera cet apprentissage. Par exemple, lorsque le réseau est composé de neurones sigmoïdes, l'apprentissage peut s'en voir ralentit pour des activation très petites ou très grandes si la fonction de coût n'est pas bien choisie. Ce ralentissement s'explique par la forme de la fonction sigmoïde :



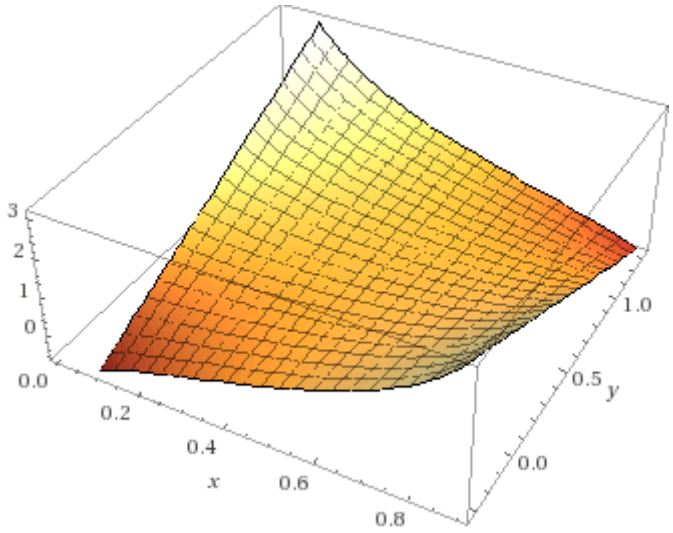
Comme il est possible de voir sur ce graphique, la fonction semble quasiment plate loin du zéro. À ces endroits, la dérivée de la fonction sera également très petite ce qui ne changera que très peu les valeurs des poids et des biais lors de la descente de gradient. Pour éviter ce problème, il faut judicieusement choisir une fonction de coût qui contrera ce problème. Cette fonction est la fonction de coût cross-entropie [9] :

$$C = -\frac{1}{n}\sum_{x}[ylna + (1-y)\ln(1-a)]\,,$$

où n est le nombre de données d'entraînement, la somme se fait pour l'ensemble des données x et y est la valeur numérique de la bonne réponse associé à x (habituellement 0 ou 1, 0 signifiant que la neurone ne devrait pas s'activer et 1 étant la neurone qui devrait s'activer) et a est la valeur obtenue par le neurone.

Pour visualiser cette fonction en 3 dimensions, donc en fonction d'un seul neurone et pour un seul x, il est possible de définir le y de la nouvelle fonction comme le y de la fonction de coût initial, le x comme le a de la fonction de coût initial et de laisser tomber le $-\frac{1}{n}\sum_x$.

Cette nouvelle fonction simplifié, $C_{simp} = -[y \ln x + (1-y)\ln(1-x)]$, ressemble à ceci:



Ainsi, il n'y a plus de ralentissement puisque le $\sigma'(z)$, qui cause problème, disparaît de la dérivée :

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_{jk}^l}$$

$$= \frac{\partial\left(-\frac{1}{n}\Sigma_x \Sigma_y \ \left[y_j ln a_j^l + (1-y_j)\ln(1-a_j^l)\right]\right)}{\partial \sigma}\frac{\partial \sigma}{\partial w_{jk}^l}$$

$$= \frac{\partial\left(-\frac{1}{n}\Sigma_x \Sigma_y \ \left[y_j ln \sigma(z_j^l) + (1-y_j)\ln(1-\sigma(z_j^l))\right]\right)}{\partial \sigma}\frac{\partial \sigma}{\partial w_{jk}^l}$$

$$= -\frac{1}{n}\sum_x \sum_y \left(\frac{y_j}{\sigma(z_j^l)} - \frac{1-y_j}{1-\sigma(z_j^l)}\right)\sigma'(z_j^l)(w_{jk}^l x_{(j-1)k}^l - b_l)$$

$$= -\frac{1}{n}\sum_x \left(\frac{y_j}{\sigma(z_j^l)} - \frac{1-y_j}{1-\sigma(z_j^l)}\right)\sigma'(z_j^l)\, x_{jk}^l$$

$$= -\frac{1}{n}\sum_x \frac{\left(y_j - y_j\left(\sigma(z_j^l)\right) - \sigma(z_j^l) + y_j\left(\sigma(z_j^l)\right)\right)}{\sigma(z_j^l)\left(1-\sigma(z_j^l)\right)}\sigma'(z_j^l)x_{jk}^l$$

$$= -\frac{1}{n}\sum_x \frac{\sigma'(z_j^l)}{\sigma(z_j^l)\left(1-\sigma(z_j^l)\right)}\,(y_j - \sigma(z_j^l)x_{jk}^l$$

$$= -\frac{1}{n}\sum_{x}\left(y_j - \sigma\left(z_j^l\right)\right)x_{jk}^l$$

On dirait qu'un saut inexpliqué s'est produit à la dernière ligne de l'équation. Cependant, il est facilement possible de démontrer que $\sigma'(z) = \sigma(z)\big(1-\sigma(z)\big)$:

$$\sigma'(z) = \left(\frac{1}{1+e^{-z}}\right)'$$

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

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

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

$$= \sigma(z)\left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right)$$

$$= \sigma(z)\big(1-\sigma(z)\big)$$

Le même principe s'applique au biais de la fonction :

$$\frac{\partial C}{\partial b_l} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial b_l}$$

$$= -\frac{1}{n}\sum_{x}\sum_{j}\left(\frac{y_j}{\sigma\left(z_j^l\right)} - \frac{1-y_j}{1-\sigma\left(z_j^l\right)}\right)\frac{\partial\left(\sigma\left(w_{jk}^l x_j^l - b_j^l\right)\right)}{\partial b_l}$$

$$= -\frac{1}{n}\sum_{x}\sum_{j}\left(\frac{y_j}{\sigma\left(z_j^l\right)} - \frac{1-y_j}{1-\sigma\left(z_j^l\right)}\right) * -\sigma\left(z_j^l\right)$$

$$= -\frac{1}{n}\sum_{x}\sum_{y}\frac{\sigma'\left(z_j^l\right)}{\sigma\left(z_j^l\right)\left(1-\sigma\left(z_j^l\right)\right)} * -\left(y_j - \sigma\left(z_j^l\right)\right)$$

$$= -\frac{1}{n}\sum_{x}\sum_{j}\left(\sigma\left(z_j^l\right) - y_j\right)$$

Ici, certaines étapes déjà élaborées lors du développement de la dérivée partielle de la fonction de coût selon les poids ont été omises afin d'alléger le tout.

Agrandir artificiellement les données d'entraînement

Avec plus de données d'entraînement, le réseau est exposé à plus de variations présentes dans le sujet de l'étude et devient ainsi meilleur à le généraliser. Cependant, amasser des données d'entraînement demande beaucoup de temps. C'est pourquoi agrandir artificiellement les données déjà disponibles par l'entremise de différentes modifications appliquées à celle-ci peut être très avantageux, autant pour la performance du réseau que pour le temps que cette technique sauve. Par exemple, pour un programme servant à la reconnaissance d'un certain type d'images, il serait possible d'agrandir le nombre de données d'entraînement en faisant des translations ou des rotations à ces images. Pour l'œil humain, la différence peut sembler insignifiante. Cependant, pour un ordinateur qui se concentre sur la position de chaque pixel de l'image, la différence est immense.

Sur-apprentissage

Le sur-apprentissage, ou « overfitting » en anglais, est un phénomène qui surgit lorsque le réseau s'entraîne trop et commence ainsi à reconnaître les spécificités des données d'apprentissage au lieu d'apprendre la forme générale du sujet à l'étude. Le réseau qui sur-apprend sait moins bien comment généraliser le sujet à l'étude. Dans de telles situations, la performance du réseau sur des données autres que celles qui servent à son apprentissage se voit réduite. Pour contrer ce phénomène, plusieurs techniques différentes ont été développées.

Arrêt prématuré (early stopping)

Comme le nom l'indique, cette technique arrête prématurément les époques d'apprentissage du réseau. Pour savoir quand le réseau doit arrêter de s'entraîner, il faut tester le degré de précision du réseau sur un ensemble de données autre que celui servant à l'entraînement. Lorsque la précision du réseau arrêtera d'augmenter, le réseau arrêtera également de s'entraîner. Les données de validation, mentionnées précédemment, servent à calculer la précision du réseau. En arrêtant prématurément le réseau, celui-ci a moins de temps pour apprendre les spécificités des données d'entraînement [10].

Un problème avec cette technique, est qu'il arrive que la précision de certains réseaux atteigne un plateau pour un certain moment, avant de continuer à augmenter. Dans de telles situations, il suffit d'attendre un certain nombre d'époque d'entraînement pour lequel la précision reste environ la même avant de décider d'arrêter l'entraînement. Il faut cependant avoir une idée générale du comportement du réseau avant de savoir combien d'époque il faut attendre, si même

l'attente en vaut la peine. C'est pourquoi il est préférable de connaître le comportement général du réseau avant d'utiliser cette technique.

Régulation L1

Le but de la régulation est de forcer le réseau à préférer de petits poids w à de grands poids en ajoutant un terme, appelé terme de régulation, à la fonction de coût. Dans le cas de la régulation L1, ce terme est la somme des valeurs absolues des poids du réseau, ce qui donne comme nouvelle fonction de coût :

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

Où C_0 est la fonction de coût non-régularisé, $\lambda > 0$ est le facteur de régularisation et n est la taille du set d'entraînement.

En ajoutant $\frac{\lambda}{n}$, il est possible de contrôler l'importance que devrait attribuer le réseau à la recherche de petits poids. La valeur optimale pour λ se trouve expérimentalement et varie selon le réseau.

Un réseau composé de poids plus petits est préférable puisque les liens qui seront ainsi fortifiés sont ceux qui ont le plus grand impact sur la fonction de coût, soit les motifs les plus récurrents dans le set d'entraînement. En cherchant plutôt les grandes ressemblances, le réseau aura plus de difficulté à faire du sur-apprentissage et fera une meilleure généralisation du sujet à l'étude.

Comme la fonction de coût est modifiée, sa dérivée change également :

$$\frac{\partial C}{\partial w} = \frac{\partial \left(C_0 + \frac{\lambda}{n} \sum_w |w| \right)}{\partial w}$$

$$= \frac{\partial C_0}{\partial w} + \frac{\partial \frac{\lambda}{n} \sum_w |w|}{\partial w}$$

$$= \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}\ sign(w)$$

La règle servant à modifier les poids w lors de la descente de gradient devient donc :

$$w \to w' = w - \frac{\eta\lambda}{n}\ sign(w) - \eta \frac{\partial C_0}{\partial w}$$

Régulation L2

La régulation L2, tout comme la régulation L1, encourage le réseau de neurone à choisir des petits poids. Cependant, l'équation qui permet cela est différente :

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

Tout comme pour la régulation L1, la dérivée partielle de la fonction de coût change :

$$\frac{\partial C}{\partial w} = \frac{\partial \left(C_0 + \frac{\lambda}{2n} \sum_w w^2 \right)}{\partial w}$$

$$= \frac{\partial C_0}{\partial w} + \frac{\left(\partial \frac{\lambda}{2n} \sum_w w^2 \right)}{\partial w}$$

$$= \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}\ w$$

La règle servant à modifier les poids lors de la descente de gradient est donc également modifiée :

$$w \to w' = \left(1 - \frac{\eta\lambda}{n} \right) w - \ \eta \frac{\partial C_o}{\partial w}$$

Contrairement à la régulation L1 qui diminue la valeur des poids par un terme constant, la réduction causée par la régulation L2 est proportionnelle à la taille du poids. Selon l'effet désirée sur les poids du réseau il est donc possible de prendre l'une ou l'autre de ces régulations.

« Dropout »

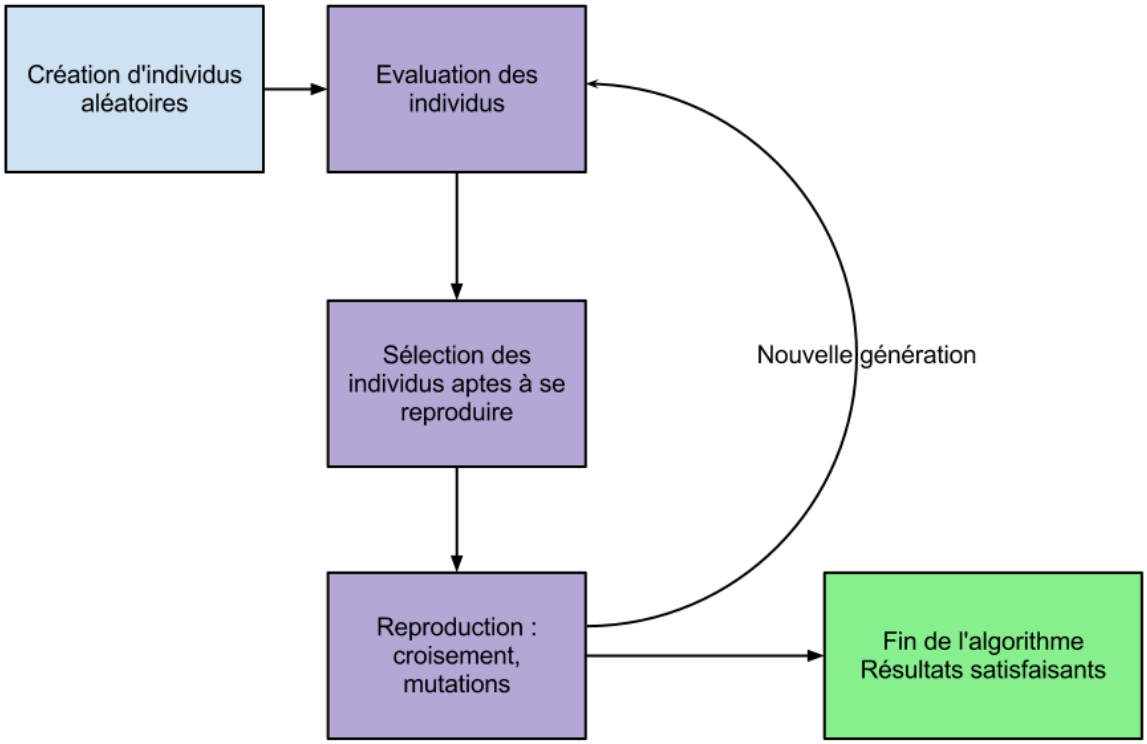
Comme les poids initiaux sont choisis au hasard, différents réseaux de neurones avec exactement le même code ne trouveront pas nécessaire le même minimum relatif pour la fonction de coût. Certains programmes seront donc plus performant que d'autres. Faire une moyenne de plusieurs réseaux de neurones différents serait donc une bonne façon d'optimiser un programme, mais ce serait également couteux en ressources.

Une méthode permettant d'atteindre un résultat similaire, sans pour autant devoir entraîner plusieurs réseaux, est la méthode du « dropout ». Avec cette méthode, à chaque époque d'entraînement, la moitié des neurones d'entrées sont temporairement effacé. Les neurones ainsi enlevés sont choisis au hasard et changent à chaque époque d'entraînement. Ceci revient à entraîner un réseau de neurone différent à chaque fois. Le résultat final, lorsque le programme roule réellement avec toutes les neurones activés, est une moyenne de tous les différents réseaux avec deux fois de neurones qui ont été entrainés au préalable. Cependant, comme il y a deux fois plus de neurones et donc deux fois plus de connections, il ne faut pas oublier de réduire chaque poids d'un facteur de 2.

Algorithme génétique

L'algorithme génétique est une méthode de résolution de problème utilisant une approche génétique, voire probabiliste, basée sur la sélection naturelle Darwinienne. Contrairement à la rétropropagation, l'algorithme génétique ne nécessite presque pas ou pas du tout de données d'entrainement, dépendamment du problème. Dans le cas de la reconnaissance d'image, nous avons utilisé des données pour entrainer le réseau, mais l'algorithme génétique n'en a pas besoin a priori puisque, comme il sera expliqué ci-dessous, il se base sur une fonction de performance qui varie selon le problème et pas à coup sûr sur la fonction de coût, quoique ça soit le cas pour la reconnaissance.

L'algorithme génétique consiste en six principales étapes, soit l'initialisation du système, l'évaluation de la performance, la sélection des éléments plus performants, la recombinaison ou, en termes génétiques, l'enjambement, la mutation et le remplacement de l'ancienne par la nouvelle population. Les mêmes étapes sont utilisées à répétition jusqu'à l'obtention d'un « candidat gagnant », c'est à dire d'un individu vers lequel l'algorithme converge [11].



[12]

Initialisation du système

La population initiale est déterminée aléatoirement, c'est à dire que les gènes des individus sont définis complètement au hasard. Le format des gènes doit être le même pour tous les individus comme si ceux-ci étaient de la même espèce et pour que les croisements soient possibles entre deux individus distincts. Si les gènes sont représentés par des nombres en binaire, par exemple [00101] et [11001], ils ont le même format et ainsi la valeur de 1 dans le premier correspond à la valeur de 0 dans le second. Or, une population peut également être définie comme un ensemble de systèmes, dans le cas des réseaux de neurones par exemple. Les gènes correspondraient donc aux paramètres dont on veut connaitre la combinaison optimale, c'est-à-dire les poids et les biais. Pour

que les croisements soient possibles, il faut que tous les réseaux aient la même structure et que chaque poids et chaque biais dans un réseau aient leurs homologues dans les autres réseaux de la population [11].

Évaluation de la performance

Pour qu'il y ait une évolution possible dans la population de réseaux de neurones, il faut un mécanisme de sélection des individus les plus performants. Pour déterminer le niveau de performance des individus, on utilise une fonction de performance qu'on applique sur chacun des individus afin de les classifier selon la qualité de la réponse qu'ils fournissent au problème. Cette qualité peut être mesurée en fonction de n'importe quel critère choisi, dépendamment de la nature du problème. Par exemple, la fonction de performance, communément appelée fonction « fitness », pour le problème du commis voyageur peut ressembler à $F(x_i) = \frac{1}{d}$, où d est la distance du circuit conçu par le réseau de neurones. Plus la valeur de la fonction de performance serait grande, plus le circuit serait court et plus le réseau serait considéré comme performant.

Si notre population est composée de réseaux de neurones visant à reconnaitre une image, la fonction de performance peut consister en

$$F(x_i) = \frac{1}{(1 + C(x_i))}$$

Où $C(x_i)$ correspond à la fonction de coût du réseau et est nécessairement plus grand ou égal à zéro. Un réseau x_i dont l'efficacité serait maximale aurait donc une fonction de performance de 1, alors qu'un réseau x_i dont l'efficacité serait minimale aurait une fonction de performance de 0. Le niveau de performance des individus serait donc compris entre 1 et 0 en utilisant cette fonction, mais il n'y a pas a priori une fonction de performance qu'il faut utiliser. Par exemple, on peut utiliser une autre fonction de coût que la fonction classique de coût $C(x_i) = \frac{1}{N}\sum_{j=1}^{N}\left(T_j - O_j\right)^2$, soit par exemple $G(x_i) = \frac{\sum_{j=1}^{N}\left(T_j - O_j\right)^2}{\sum_{j=1}^{N}\left(T_j - T_{j+1}\right)^2}$ de sorte qu'on obtiendrait comme fonction de performance ressemblant à

$$F(x_i) = \frac{1}{(1+G(x_i))}$$ [13]

Il est aussi possible de tenir compte du nombre de classifications réussies comme mesure de performance, en utilisant une formule du genre

$$F'(x_i) = \frac{n_r}{n_t}(x_i) - C(x_i)$$

Où n_r est le nombre de classifications réussies et n_t est le nombre de classifications faites au total pour un réseau x_i en particulier. Dans ce cas-là, la fonction de performance varie beaucoup plus et pourrait même être négative si le coût du réseau est trop élevé. Le bon choix de la fonction de performance est capital, car celle-ci renferme à elle seule toutes les informations à propos du problème dont pourra disposer l'algorithme génétique pour arriver à une solution.

Sélection des individus

Lorsque l'efficacité des individus à résoudre le problème a été évaluée par la fonction de performance, il faut choisir les individus dont les gènes pourront avancer à la prochaine génération. Pour de très grandes populations, il est possible de classifier les individus selon leur fonction de performance d'abord et d'automatiquement sélectionner ceux qui présente la valeur la plus élevée. Sinon, une probabilité proportionnelle au niveau de performance calculé précédemment est accordée à chaque individu de la population et la sélection est faite en tenant compte de ces probabilités. Ainsi, les individus ayant le mieux performés auront plus de chances d'être choisis. Si la fonction de performance est $F(x_i)$, alors la probabilité d'être sélectionné rattachée à x_i peut être exprimée comme

$$P(x_i) = \frac{F(x_i)}{\sum_{k=1}^{N} F(x_k)}$$

Où N représente la taille de la population [11]. Utiliser les probabilités pour choisir les individus crée une population beaucoup plus diverse génétiquement et permet de lutter contre le phénomène du « selfish gene », où un seul individu se perpétue au détriment du reste de la population. Cette diversité génétique, même si elle permet d'arriver potentiellement à un meilleur réseau, implique que l'amélioration des réseaux de neurones composant la population sera plus lente, car des réseaux moins performants seront tout de même conservés.

Enjambement ou « Crossover »

Une fois que les individus qui seront les « parents » de la prochaine génération sont choisis, il y a un enjambement des gènes par paire d'individus qui s'effectue. Comme lors de la reproduction des gènes dans la nature, l'algorithme génétique utilise le principe de « crossover » entre les gènes pour augmenter la diversité génétique de la population et courir le risque de produire de meilleurs individus. Si on prend l'exemple d'un couple de valeurs binaires [00001] et [10101], l'enjambement consisterait à changer la troisième valeur du premier individu par celle du deuxième et vice-versa de manière à obtenir la nouvelle paire [00101], [10001]. Dans le cas d'un réseau de neurones, ce principe s'applique difficilement avec les poids et les biais, car ceux-ci sont beaucoup trop liés entre eux et c'est davantage la configuration totale qui a de la valeur que les poids et les biais considérés individuellement. Une alternative est de simplement effectuer une moyenne des poids et des biais entre les deux individus formant la paire de « parents » pour déterminer les poids et les biais ou bien de combiner la moitié des poids et des biais d'un réseau avec l'autre moitié de ceux-ci d'un autre réseau [14]. Même si dans ce cas-là les individus seront identiques du fait qu'ils proviennent de la même moyenne, les mutations se chargeront de diversifier leurs configurations respectives.

Mutation

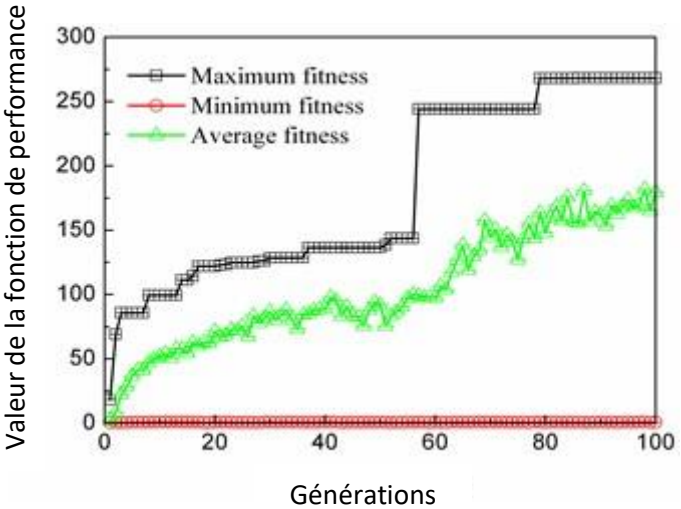
La mutation est un mécanisme essentiel pour créer la diversité génétique nécessaire à l'évolution. Les mutations sont des altérations aléatoires des gènes des individus qui sont produits par les enjambements. Dans le cas de valeurs binaires, par exemple, une mutation pourrait être de changer un 0 pour un 1 ou l'inverse, et ce, au hasard et sans garantie de réalisation. La probabilité des mutations peut-être aussi grande que l'on veut, dépendamment du type de résultats que l'on veut obtenir, mais elle se situe généralement sous 10 %, car on veut éviter de perdre le progrès déjà effectué par la sélection. Pour les réseaux de neurones, les mutations peuvent consister en de légères altérations aléatoires de certains poids et biais ou bien en la combinaison d'un certain pourcentage des biais et des poids d'un réseaux avec un pourcentage plus petit de ceux d'un réseau complètement aléatoire. Par exemple, on combine 90 % des poids et des biais d'un réseau sorti du croisement avec 10 % des poids et des biais d'un réseau aléatoire créé exprès pour la mutation [15].

Nouvelle population

La nouvelle population est constituée par la reproduction entre des paires d'individus lorsque celles-ci forment de nouveaux individus. Cette reproduction est faite par le croisement et la

mutation des individus sélectionnés selon leur performance. Cette nouvelle population, du même nombre que la population initiale, passera par les mêmes étapes et ainsi de suite.

Aperçu de la performance en fonction du temps



[16]

Ce graphique représente l'évolution de la fonction fitness en fonction du nombre de générations. La courbe en noir représente la performance du meilleur individu pour une fonction de performance donnée et la courbe en vert représente la même chose, mais en moyenne pour la population au complet. Plus le nombre de générations est grand, plus la fonction de performance commence à stagner et l'algorithme génétique cesse d'être efficace. Au long terme, l'allure de la courbe est plutôt logarithmique.

Application

Résultats et programmation

La structure du programme

Notre équipe a produit un réseau de neurone et l'a entraîné pour reconnaitre des chiffres manuscrits. Pour les codes, se référé au document 'Neural Network' envoyé. Pour créer notre réseau de neurone, il a fallu utiliser le langage de programmation Python. Ce dernier est le langage par défaut de l'intelligence artificielle et de l'apprentissage profond du fait de ses nombreux

modules créés par la communauté. Le réseau sera codé grâce à une boucle de fonctions qui vont prendre des données, les transformées et les passées à la fonction suivante. La manière dont le réseau est représenté informatiquement est une matrice de matrices. La première matrice englobe les matrices composantes du réseau : neurones, activations, poids et biais. Ensuite, chacune de ces nouvelles matrices est composées de sous-matrices pour chaque couche. Dépendamment du nombre de couches du réseau, leur grosseur change. Tout changement apporté au réseau sera un changement apporté à ces matrices.

$$\begin{bmatrix} [activations] \\ [neurones] \\ [poids] \\ [biais] \end{bmatrix}$$

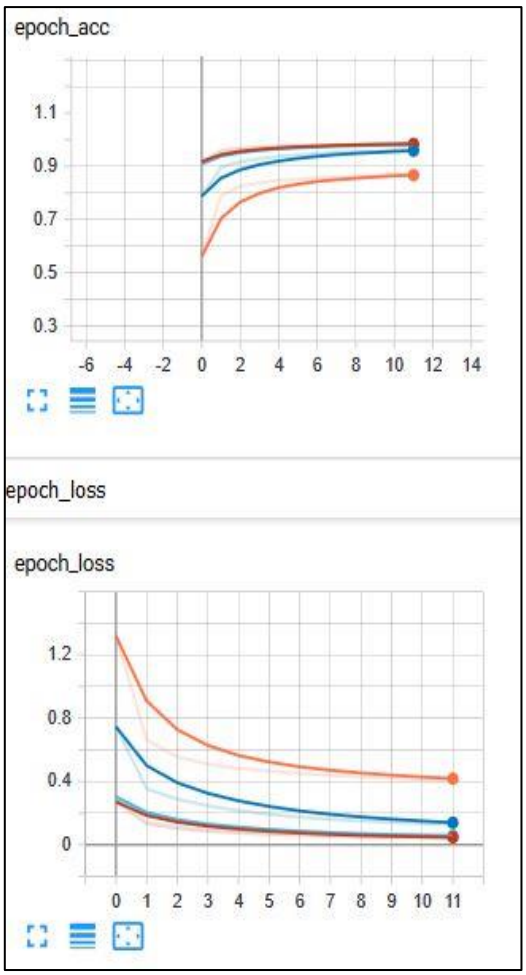
Le réseau peut donc, sous sa forme informatique, être considéré comme un vecteur de matrices de matrices. Une fois le réseau formé, en générant des matrices avec des valeurs aléatoires entre 10 et -10, il reste à définir l'algorithme d'apprentissage. Pour faire des opérations avancées sur des matrices, il est nécessaire d'avoir la librairie 'Numpy'. Elle permet des opérations telles que la multiplication de matrices, la vectorisation d'une matrice, la modification de dimensions. Cela va permettre d'utiliser les fonctionnalités de base des matrices tel que ($[neurone] = [poids][neurone - 1] + [biais]$). L'équation précédente à été utilisée dans la fonction 'passer en avant' de notre programme, qui permet de faire passer les données de la couche de neurones initiales, jusqu'à la couche de neurones finale.

Rétropropagation du gradient

Pour la rétropropagation, il ne suffit que de suivre et d'utiliser les 4 équations de la rétropropagation. En somme, pour le réseau simple utilisant la rétropropagation, il n'y a que des fonctions qui génèrent le vecteur réseau et ses matrices, et des fonctions qui altèrent ce réseau et ses matrices. Tout se passe à l'intérieur du vecteur réseau créé. La seule fonction qui prend des données extérieures est la fonction 'passer en avant' ou « feed forward » en anglais, qui prend les

données d'entrées, soit les données qui doivent être analysées par le réseau. Dans le cas précis de ce réseau, les données d'entrées sont les pixels d'une image représentant un chiffre manuscrit.

Dans cette version du réseau codée de A à Z par notre équipe, il y a eu un petit problème. La fonction de rétropropagation créait une erreur, la variable servant à calculer la variation des paramètres nécessaires, qui diminuait exponentiellement à travers les couches du réseau. Arrivé à la couche initiale, la fonction donnée un vecteur d'erreur ayant des éléments de l'ordre de puissance de -98. Il y avait donc un problème avec les valeurs d'erreur des couches initiales. Le problème n'a pas été résolu. Cependant, le réseau arrivait à s'améliorer jusqu'à un taux de réussite de 76%.



Par la suite, il a été décidé d'utiliser le module « Tensorflow » de Google pour la création du réseau. Ce module avait une librairie de différents types de couches de réseau. Notre réseau initial était un simple réseau à deux couches intermédiaires, à 32 neurones et d'activation sigmoïde. Avec « Tensorflow », un taux de réussite de 97% a été atteint. D'autres activations ont été testées, par exemple la fonction Tanh et la fonction « ReLu ». La fonction « ReLu » a été celle qui a donné les meilleurs résultats. Grâce à « Tensorboard », un sous-module de « Tensorflow », un graphique des résultats des différents réseaux créés a été enregistré. Le tutoriel de google sur Tensorflow a été grandement utile [17].

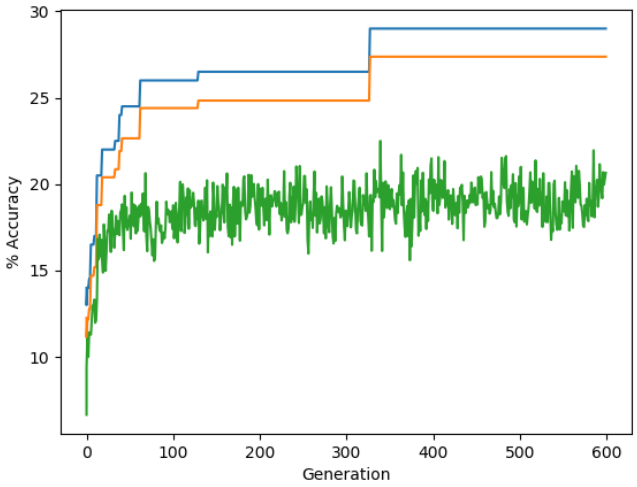
Algorithmes génétiques

Le même réseau a aussi été entrainé par algorithme génétique. Cette fois, au lieu d'avoir une fonction rétropropagation. Il y a une fonction sélection, croisement et mutation. Le programme

qu'on a conçu génère une population de réseaux et sélectionne les meilleurs pour se reproduire. Ensuite des mutations sont appliquées aux réseaux créant un mélange de chaque paramètre avec une valeur aléatoire selon un ratio prédéterminé. Par exemple, avec un ratio de 0.1, la matrice couche 1 (M_1) à l'intérieur de la matrice poids est additionnée à une matrice aléatoire de même dimension (M_2) à l'aide du ratio (r). La fonction ressemble à ceci $M_3 = M_1(1-r) + M_2 r$. La nouvelle matrice remplace l'ancienne. Ainsi tous les paramètres du réseau peuvent être remplacés, donnant une équation générale $R_3 = R_1(1-r) + R_2 r$. La sélection est faite à l'aide d'une fonction fitness.

Commis voyageur

Le programme de notre équipe est polyvalent, donc pour l'appliquer à un nouveau problème, il ne suffit que de changer la forme de la couche d'entrée et de sortie, ainsi que la fonction fitness. Avec ce programme, la population de réseau atteignait un maximum de 60% comme taux de réussite après 3000 générations. La fonction fitness consistait en le taux de réussite moins le coût du réseau. La croissance du taux de réussite semblait logarithmique. Voici un exemple de la performance d'une population de réseaux entrainés sur 600 générations. La courbe bleue est le taux de réussite du meilleur réseau et la courbe jaune est sa valeur fitness. La courbe restante est la valeur fitness moyenne de la population.



Retour sur la programmation

Les problèmes rencontrés lors de la programmation ont été principalement des problèmes d'incompatibilité de dimensions des vecteurs. Chaque couche peut prendre des vecteurs ou des matrices de dimensions bien précises. Avoir les bonnes dimensions partout est donc primordial. Le deuxième problème a été les données. Pour l'apprentissage par rétropropagation, il faut énormément de données. Notre réseau a été entrainé sur la base de données MNIST qui est la base de données par défaut de l'apprentissage profond. Elle consiste en 60 000 images de chiffres manuscrits. Le temps d'entrainement, sans l'optimisation du module « Tensorflow » de Google a

été d'environ 3h. Ensuite, lorsque notre équipe a voulu résoudre d'autres problèmes que la reconnaissance de chiffre, le manque de données nous en a empêchés. Cela nous a fait découvrir que les données sont devenues une ressource stratégique dans la société d'aujourd'hui où ce que l'IA est en pleine effervescence.

Comparaison des deux méthodes

Les deux méthodes qui ont été expliquées et que nous avons implémentées en pratique sont très distinctes parce que, même si les deux servent à l'entrainement d'un réseau de neurones, c'est-à-dire à la modification répétitive du réseau afin d'y instaurer la meilleure configuration possible de poids et de biais, elles sont basées sur des principes différents.

Dans le premier cas, celui de la rétropropagation du gradient, la modification des poids et des biais est très directe, car il est question d'essentiellement propager l'erreur du réseau rétroactivement en modifiant les poids et les biais pour atteindre le minimum de la fonction de coût. Les poids et les biais sont donc déterminés directement en fonction de l'erreur du réseau dans le but de le minimiser, d'où la très grande efficacité de cette méthode. Bien que la rétropropagation soit rapide et efficace en raison de sa précision basée sur ce calcul, elle requiert nécessairement un grand nombre de données d'entrainement, étant donné que l'on doit déterminer à répétition l'erreur du réseau sur laquelle se baseront les modifications apportées aux poids et aux biais. De plus, il est important de préciser que la rétropropagation ne peut être appliquée que quand l'on connait la réponse au problème en question. Par exemple, pour le réseau de neurones reconnaissant les images, il faut non seulement fournir les données d'entrainement au réseau, mais lui spécifier également la réponse correcte qu'on attend de lui associée à chaque donnée afin de lui permettre d'établir son erreur à corriger. Le réseau de neurones entrainé par rétropropagation est donc très performant pour effectuer une tâche précise pour laquelle il a été spécifiquement entrainé avec les bonnes données, comme c'est le cas pour la reconnaissance d'image, mais il est d'aucune utilité face à un nouveau problème dont on chercherait justement la réponse et pour lequel il n'existerait pas une ample base de donnée déjà existante. Un autre problème qui surgit avec la rétropropagation est son incapacité à traiter des fonctions non différentiables en raison de l'omniprésence du calcul différentiel, la rendant impuissante face à des problèmes complexes et appliqués à des situations plus pratiques où on travaille des fonctions imparfaites [18].

Dans le deuxième cas, celui de l'algorithme génétique, la modification des poids et des biais pour optimiser le réseau de neurones est beaucoup moins directe, car celle-ci est faite moyennant la fonction de performance ou « fitness » et à travers un processus évolutif lent. Contrairement à la rétropropagation, l'algorithme génétique ne fait qu'en quelque sorte filtrer les réseaux qui ont de meilleures configurations de poids et de biais plutôt que d'aller directement changer ces paramètres dans une direction déterminée visant à l'améliorer. De plus, dépendamment si l'on veut conserver davantage de diversité dans la population de réseaux, certains réseaux moins performants peuvent même être conservés, selon les probabilités assignées et de la fonction de performance utilisée. Comme l'algorithme génétique est très probabiliste et mise grandement sur le hasard pour arriver à des résultats, notamment à l'aide des mutations, il est beaucoup plus lent et beaucoup moins efficace que la rétropropagation pour un problème de reconnaissance. Il est en ce sens un calque parfait de l'évolution, c'est-à-dire d'un processus très lent, voire logarithmique, et rempli de détours. Or, l'algorithme génétique présente l'avantage de ne pas avoir besoin de données tant qu'il soit possible de définir une fonction de performance qui puisse donner une mesure d'efficacité. Pour la reconnaissance, l'entrainement des réseaux de neurones doit être fait avec des bases de données, mais d'autres problèmes permettent de fixer à l'algorithme génétique une fonction de performance uniquement basée sur un critère général, comme par exemple la distance du circuit dans le cas du problème commis voyageur. Finalement, l'algorithme génétique se prête quant à lui très bien aux problèmes dont on cherche justement les réponses et pour lesquels il n'y a pas de données disponibles, en cela il est l'inverse de la rétropropagation [19].

Conclusion

En conclusion, le réseau de neurones est une technique d'apprentissage énormément utilisée et polyvalente, puisqu'elle peut facilement s'adapter à différents problèmes. En effet, les données d'entrée et de sortie du réseau varient en fonction du problème, puis plusieurs algorithmes peuvent être utilisés afin de trouver la bonne configuration de poids et de biais. Dans ces algorithmes, il y a, entre autres, la rétro propagation du gradient et l'algorithme génétique tels qu'étudiés. Nos recherches et nos résultats montrent que pour la reconnaissance d'image, les réseaux de neurones entrainés par rétro propagation sont nettement supérieurs à ceux entrainés avec l'algorithme génétique. Nos réseaux de neurones de reconnaissance d'image ont réussi à atteindre un taux de succès de 97 % avec la rétro propagation tandis que de seulement 60 % avec l'algorithme génétique. Cependant, notre recherche montre également que la rétro propagation n'est pas adapté à trouver des solutions pour lesquelles il n'existe pas une ample base de données, alors que l'algorithme génétique a beaucoup plus de « créativité », comme le montre nos résultats quand on utilise l'algorithme génétique pour résoudre le problème du commis voyageur.

Cependant, bien qu'elle soit d'une efficacité hors pair, l'intelligence artificielle pose plusieurs problèmes éthiques, autant pour le respect de la vie privée de l'individu que pour les problèmes entourant la création d'une conscience, soit une intelligence artificielle complète, ou encore pour la controverse des armes automatisées. En effet, selon Jean-François Gagné, spécialiste en intelligence artificielle, autant notre vie personnelle, professionnelle que publique sont maintenant documentées [19]. Le spécialiste ajoute :

> « Se promener du point A au point B n'avait pas de valeur avant. Mais quand les données sont structurées, coordonnées, colligées avec celles de mes amis et reliées aux gens que je connais et à mes intérêts, des services et des produits peuvent m'être offerts. On peut influencer mon parcours, ce à quoi je pense, ce que je vois, avec qui j'interagis. » [19]

Notre vie privée a maintenant un prix. De plus, Stephen Hawkins rappelle les dangers entourant la création d'une intelligence artificielle complète qui s'apparenterait à la conscience humaine. Selon lui, « le développement d'une intelligence artificielle complète pourrait mettre fin à l'humanité » [20]. Le scientifique renommé défend son point de vue en expliquant qu'une fois cette intelligence

complétée, elle « décollerait seule, et se redéfinirait de plus en plus vite. Les humains, limités par une lente évolution biologique, ne pourraient pas rivaliser et seraient dépassés. » [20][1]

[1] Entrevue traduit par Le Monde, https://www.lemonde.fr/pixels/article/2014/12/03/hawking-l-intelligence-artificielle-pourrait-mettre-fin-a-l-humanite_4533135_4408996.html

Bibliography

[1] L. Rob, "L'intelligence artificielle gagne du terrain dans les salles de nouvelles," 11 03 2019. [Online]. Available: https://www.ledevoir.com/culture/medias/549562/l-intelligence-artificielle-gagne-du-terrain-dans-les-salles-de-redaction. [Accessed 05 04 2019].

[2] L. Pascal, "Pourquoi 2018 n'a pas été l'année du «deepfake»," 31 12 2018. [Online]. Available: https://www.ledevoir.com/societe/544558/technologie-pourquoi-2018-n-a-pas-ete-l-annee-du-deepfake. [Accessed 05 04 2019].

[3] S. E. Haupt and R. L. Haupt, "Genetic algorithms and their applications in environnemental sciences," Utah State University, Utah, N.D..

[4] E. Plamondon Emond, H. Roulot-Ganzmann and C. Girouard, "L'IA dans la lutte contre les changements climatiques," 23 03 2019. [Online]. Available: https://www.ledevoir.com/societe/science/550291/l-ia-dans-la-lutte-contre-les-changements-climatiques. [Accessed 05 04 2019].

[5] INPRINCIPIO, "Comprendre l'intelligence artificielle," 2018. [Online]. Available: https://www.inprincipio.xyz/comprendre/. [Accessed 12 04 2019].

[6] "MIximum," [Online]. Available: https://i.miximum.fr/images/62O7A2IFI1/. [Accessed 13 mars 2019].

[7] OD-DATAMINING, "Les réseaux de neuronnes expliqués à ma fille," 2018. [Online]. Available: https://od-datamining.com/knwbase/les-reseaux-de-neurones-expliques-a-ma-fille/. [Accessed 3 mai 2019].

[8] M. A. Nielson, "Proof of the four fundamental equations," in *Neural Networks and Deep Learning*, Determination Press, 2018, p. Chapter 2.

[9] M. A. Nielsen, "The cross-entropy cost function," in *Neural Networks and Deep Learning*, Determination Press, 2018.

[10] Wikipédia, "Early Stopping," 18 02 2019. [Online]. Available: https://en.wikipedia.org/wiki/Early_stopping. [Accessed 30 03 2019].

[11] J. Carr, "An Introduction to Genetic Algorithms," Whitman College, Washington, 2014.

[12] R. D. Key, "Schéma général des AG," DoubleZoom, 2019.

[13] A. R. d. M. N. P. S. G. Lima, "Tests with Different Fitness Functions for Tuning for Tuning of Artificial Neural Networks with Genetic Algorithm," Sociedade Brasileira de Inteligência Computacional, Vancouver, 2011.

[14] R. Kemp, "An Introduction to Genetic Algorithms for Neural Networks," University of Cambridge, Cambridge, 2006.

[15] O. Abdoun, J. Abouchabaka and C. Tajani, "Analyzing the Performance of Mutation Operators," International Journal of Emerging Sciences, Kenitra, 2012.

[16] M. Qu, G. Chen and K. Zhang, "Intelligent Identification Method for Whole Aero-Engine Connection Stiffness," Journal of Vibroengineering, Nanjing, 2018.

[17] Google, "Tensorflow," Google, Silicon Valley, 2015.

[18] D. J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms," BBN Systems and Technologies Corp. , Cambridge, 1989.

[19] P. Koehn, "Combining Genetic Algorithms and Neural Networks : The Encoding Problem," Knowville, 1994.

[20] J.-F. Gagné, Interviewee, *Nos données et notre vie privée : trois questions à un spécialiste en intelligence artificielle.* [Interview]. 05 04 2018.

[21] S. Hawking, Interviewee, *Stephen Hawking warns artificial intelligence could end mankind.* [Interview]. 02 12 2014.

[22] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550-1556, 1990.

[23] R. L., "Le gradient d'un fonction scalaire," octobre 2004. [Online]. Available: http://www.edu.upmc.fr/physique/licence/psvp/Documents/gradient.pdf. [Accessed 24 mars 2019].

[24] Wikipédia, "Produit maticiel de Hadamard," 09 12 2018. [Online]. Available: https://fr.wikipedia.org/wiki/Produit_matriciel_de_Hadamard. [Accessed 24 03 2019].

[25] Wikipédia, "Backpropagation," 23 03 2018. [Online]. Available: https://en.wikipedia.org/wiki/Backpropagation. [Accessed 30 03 2019].