CentraleSupélec
Projet long du cursus Supélec
Encadré par : J. Tomasik et A. Rimmer

# Dessines-moi un mouton
# Genrative Adversial Network

François Bouvier d'Yvoire
Mathieu Delmas
Romain Poirot
Paul Witz

Etude des reseaux de neurones en perceptrons
avec application au concept des Generative Adverserial Network

Années 2017-2018

# Dessines-moi un mouton
# Genrative Adversial Network


## Résume

Resume



## Mots clefs

Mots clefs

# Table des matières

# Chapitre 1

# Introduction aux réseaux de neurones et premières applications

## 1.1 Outils utilisés pour le projet

Description des outils mis en place pour le projet et de certains choix techniques.

## 1.2 Réseaux de neurones et fonctionnement

### 1.2.1 Le neurone

L'unité de base du réseau est le neurone, on peut l'imaginer comme une fonction mathématique. Lui sont attribuées n entrées, chacune affectée d'un poids $w_i$ et une fonction mathématique de $\mathbb{R}$ dans $\mathbb{R}$. Le rôle du neurone sera de renvoyer le résultat de la fonction, appliquée à la somme pondérée par leur poids des entrées. On pourrait ajouter un biais comme paramètre de notre neurone afin d'ajuster notre résultat (choisir quand une fonction seuil renvoie 1 par exemple).

Un exemple simple du réseau de neurones est la séparation d'un plan en 2.
Imaginons un neurone à deux entrées $e_1$ et $e_2$, chacune attribuée d'un poids $w_1$ et $w_2$. On affecte au neurone un biais $b$ et une fonction d'activation seuil (Heavyside).
Notre neurone renverra : 1 si $(e_1 * w_1 + e_2 * w_2)–b > 0$ et 0 sinon.

Si les $e_1$ et $e_2$ représentent les abscisses et ordonnées d'un point du plan. On reconnaît dans l'argument de la fonction d'activation l'équation d'une droite affine. Notre neurone pourra donc

distinguer les points du plan selon quel côté de la droite ils se trouvent.

On peut déjà voir qu'une modification des poids entraînera une différente délimitation du plan. On peut donc imaginer faire « apprendre » au réseau quels points délimiter en modifiant ses poids, nous reviendrons sur ce concept par la suite
Cependant, les applications d'un neurone seul sont vite limitées c'est pourquoi on va s'intéresser à en connecter plusieurs entre eux.

### 1.2.2 Réseau de neurones et perceptron

On a déjà vu que le neurone se prêtait bien à la classification de données. On va voir que l'organisation de neurones en réseau permet des classifications bien plus fines.
L'organisation du réseau se fera au moyen de couches : des neurones d'une couche n auront comme entrée la sortie de neurones situés sur la couche n-1. La couche donnant le résultat final sera appelée couche de sortie, toutes les couches qui la précèdent seront appelées couches cachées. Une telle appellation se comprend du fait qu'a priori, nous n'avons aucun moyen de voir ou de corriger les comportements des neurones cachés, puisqu'avec la seule donnée de la sortie, on ne peut pas connaître l'influence des poids cachés sur celle ci

Le Perceptron est un modèle de réseau de neurones sur lequel on va s'intéresser particulièrement. Il consiste en un réseau de propagation vers l'avant (aucune boucle ne peut être trouvée dans le graphe du réseau). L'utilité d'avoir plusieurs couches se comprend facilement. Si on reprend notre exemple du problème de classification des points, on peut imaginer par exemple, quatre neurones qui enverront leur sortie sur un neurone à quatre entrée. Chacun des neurones réalisera la séparation du plan en deux selon le principe déjà évoqué précédemment. Le neurone de la couche de sortie pourra réaliser facilement le rôle d'un ET logique. On vient de sélectionner un carré dans le plan. En étendant le raisonnement, on voit qu'un réseau à deux couches permet de sélectionner n'importe quelle zone convexe de l'espace des entrées (ici du plan). De même, un réseau à trois couches pourra sélectionner n'importe quelle zone concave de l'espace des entrées.
La plupart du temps, on travaillera avec un réseau complètement connecté, c'est-à-dire que toutes les sorties d'une couche sont toutes les entrées de la couche suivante.

### 1.2.3 Apprentissage par rétro-propagation

La rétro-propagation des erreurs est un type d'apprentissage supervisé pour les perceptrons multicouches. Il consiste à calculer l'influence de chaque paramètre sur la sortie et à les mettre à jour en fonction de cette influence.

Les paramètres que l'on fait évoluer sont les poids et les biais. La formule de mise à jour est la suivante :

$$W(t+1) = W(t) + \eta \frac{\partial E}{\partial W}$$

avec $\eta$ le pas de convergence, $\frac{\partial E}{\partial W}$ la matrice de terme général $\frac{\partial E}{\partial W_{i,j}}$
Pour pouvoir mettre à jour les poids, il faut donc calculer les $\frac{\partial E}{\partial W_{i,j}}$.
A la couche $k$ l'influence des poids est donnée par :

$$\frac{\partial E^p}{\partial W_k} = \frac{\partial F}{\partial W}(W_k, X_{k-1}) \frac{\partial E^p}{\partial X_k}$$

Avec $\frac{\partial F}{\partial X}(W_k, X_{k-1})$ la matrice jacobienne de $F$ par rapport à la variable $X_k$. De plus, dans un perceptron on peut noter la sortie de la couche $k$ :

$$Y_k = W_k X_k X_k = F(Y_k)$$

On obtient donc ses 3 équations :

$$\frac{\partial E^p}{\partial y_k^i} = f'(x_k^i) \frac{\partial E^p}{\partial x_k^i}$$
$$\frac{\partial E^p}{\partial w_k^{i,j}} = x_{k-1}^j \frac{\partial E^p}{\partial y_k^u}$$

## 1.3 Application au problème du XOR

Lorsque l'on souhaite travailler sur des algorithmes d'apprentissages par ordinateur il est recommandé de les essayer sur des problèmes connus afin d'en vérifier les performances.
Le problème du XOR est l'un des plus classiques car il apporte de nombreuses difficultés.

L'objectif du XOR est de séparer le plan complexe en quatre cadrants, $(x > 0, y > 0)$, $(x > 0, y < 0)$, $(x < 0, y > 0)$ et $(x > 0, y < 0)$. On restreint le plan à $[-1; 1]^2$. Les sorties attendues par le réseau de neurones sont 1 pour les points tel que $x * y > 0$ et -1 pour les points tels que $x * y < 0$.

Le premier intérêt de ce problème est qu'il est non linéaire, c'est à dire que l'on ne peut pas tracer une droite séparant le plan en 2 qui répond à celui-ci.

C'est en se basant sur la résolution du XOR que nous avons construits notre structure de réseau et vérifié la cohérence de notre code. La littérature propose comme réseau le plus simple pour ce problème une couche cachée de 2 neurones, avec 2 entrées ($x$ et $y$) et 1 sortie dans $[-1, 1]$. Nous avons étudié également quelques autres formes de réseaux pour comparer les résultats.

**Notion de résultats** La notion de résultats nécessite d'être correctement défini afin de pouvoir être interprété correctement, et surtout comparé à d'autres résultats obtenus par nous même ou par d'autres personnes.
Dans le cas des ses apprentissages, le réseau classe les objets que l'on donne en entrée. Généralement l'on défini le résultat par rapport à un pourcentage de succès dans cette classification. Pour l'obtenir on commence par définir une erreur relative, c'est à dire une distance entre la sortie cible et la sortie obtenue. Puis l'on applique un seuil afin de définir si oui ou non le réseau à correctement classifié l'entrée.
Dans le cas du XOR on met en place un seuil de 0.5.
On cherche également à évaluer la vitesse d'apprentissage, ainsi on calcule le pourcentage de succès du réseau à intervalle régulier au cours de l'apprentissage. Les réseaux étant soumis à une forte composante aléatoire (l'ordre d'apprentissage, ainsi que l'initialisation des poids), on effectue des apprentissages dans les mêmes conditions plusieurs fois afin d'obtenir des courbes moyennes, et des intervalles de confiances justifiant nos résultats.

**Réseau en** $2 \to 2 \to 1$ Les résultats obtenus au début sur ce réseau le plus simple semblait tout à fait aléatoire et nous on permit de détecter des erreurs de traductions des équations de rétropropagations en code Python. Nous avons finalement pu obtenir des résultats satisfaisants comme le montre la figure **??**. Cependant ce résultat n'était pas obtenu dans l'intégralité des apprentissages, nous fournissant des résultats très différents, comme sur la figure **??**. La littératures et en particulier les rapports des années précédentes [**?**] et [**?**] nous on montré que dans le XOR n'était effectivement pas juste dans 100% des cas.
Nous avons donc établis soumis le réseau à de nombreux apprentissages, en faisant varier les paramètres ainsi que la forme du réseaux. Voici les résultats les plus intéressants :

**Conclusion sur le XOR** Instabilité du réseau $2 \to 2 \to 1$ et comparaison avec le $2 \to 4 \to 1$ et le $2 \to 2 \to 2 \to 1$

Pas d'apprentissage très petit par rapport à la littérature

Influence des fonctions d'activation

# Bibliographie

[1] K. Steenbergen, F. Janssen, J. Wellen, R. Smets, T. Koonen, "Fast wavelength-and-time slot routing in hybrid fiber-access networks for IP-based services", in *IEEE LEOS Symposium*, Delft, The Netherlands, October 2000.

[2] K. Nichols, V. Jacobson, L. Zhang, "A two-bit differentiated services architecture for the Internet", *IETF RFC 2638*, July 1999.

[3] http ://www.omniorb.org