Rapport JE

raphael.reme

May 2019

Résumé

Le but de l'étude est d'apprendre à un réseau de neurones à classifier des données selon une fonction de frontière donnée, possédant de nombreuses symétries. Il est question de savoir si le réseau arrive à apprendre et utiliser les symétries dans son apprentissage.

Tout est effectué à l'aide de python. Les réseaux de neurones sont faits à la main pour plus de flexibilités sur le modèle. L'usage de la bibliothèque Mxnet est restreint à l'utilisation de ses tableaux multidimensionnels (similaires à ceux de numpy) et à son calcul automatique de gradient.

1 Introduction

L'idée est donc de commencer avec un problème simplifié et avec plein de symétries connues à l'avance afin de tester et essayer de comprendre les difficultés du problème. La fonction de frontière utilisée pour faire les essais est la suivante :

$$f(x) = \begin{cases} 1 - (|x| - \lfloor |x| \rfloor), & if \ \ \lfloor |x| \rfloor \% 2 == 1 \\ |x| - \lfloor |x| \rfloor, & if \ \ \lfloor |x| \rfloor \% 2 == 0 \end{cases}$$

Ce qui donne la fonction suivante :

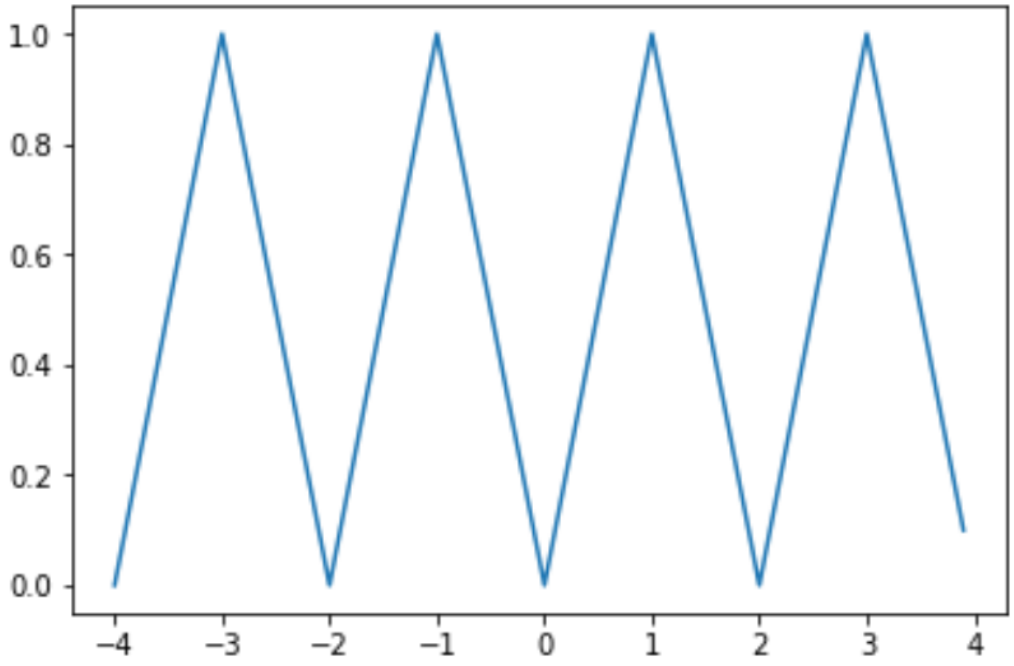

FIGURE 1 – Frontière de décision utilisée

C'est donc un motif régulier qui se répète à l'infini. Le but est d'apprendre au réseau la frontière et d'ensuite ajouter une couche de comparaison entre y et f(x) afin de prendre une décision. Cette fonction de frontière est une fonction linéaire par morceau et continue. Mais telle que défini ici elle est composée d'une infinité de morceaux. Des réseaux de neurones composés de fonctions ReLU peuvent justement apprendre parfaitement des fonctions de

ce type, mais seulement avec un nombre fini de morceaux. C'est pourquoi il sera question de la limite d'apprentissage de la fonction. Cela correspond aux extrémités jusqu'où on veut que le réseau de neurones ait appris correctement la fonction et à partir des quelles il peut diverger de la fonction de séparation. Par exemple si la limite vaut 8 le réseau de neurones doit suivre la frontière entre -8 et 8. En dehors de cette intervalle, il peut diverger de la frontière et en avoir une tout à fait différente.

Cette fonction étudiée avec une limite possède une suite de symétries la simplifiant : On peut replier successivement la fonction sur elle même par rapport à des plans de $\mathbb{R}^2$. Si on suppose que la limite vaut 2^k (il suffit de prendre la première puissance de 2 supérieure à la vraie limite, on apprend à forciori correctement jusqu'à la vraie limite). Alors une suite de symétries est ((1,0), 0), ((-1,0), 2^{k-1}), ((-1,0), 2^{k-2}), ... , ((-1,0), 1).

C'est à dire qu'on plie d'abord par rapport au plan ($x = 0$) du coté des x positifs, puis on plie successivement par rapport aux plans ($x = 2^{k-i}$ du coté des x négatifs et pour i de 1 à k.

Avec un réseau de neurones, on peut plier selon des symétries, en choisissant judicieusement les poids, biais et fonctions d'activations et peut donc réaliser les opérations décrites ci-dessus. Cela donne les résultats suivants :

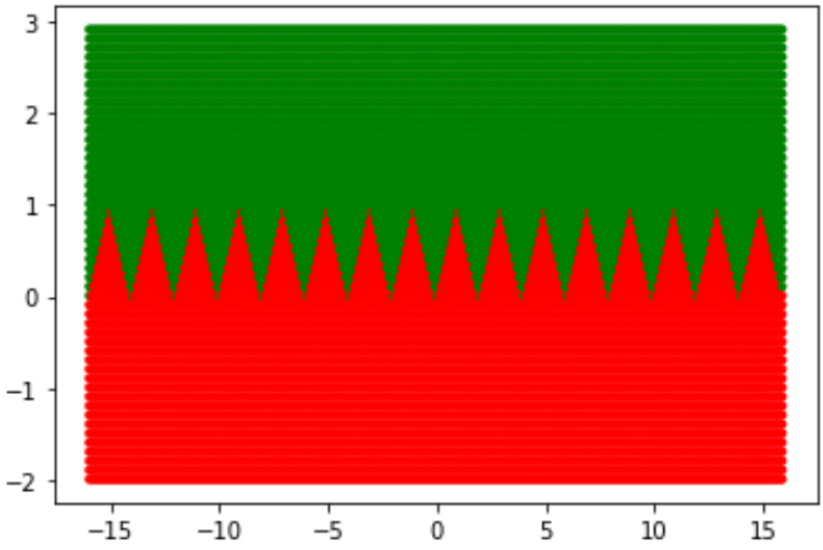

FIGURE 2 – Fonction visée par le réseau

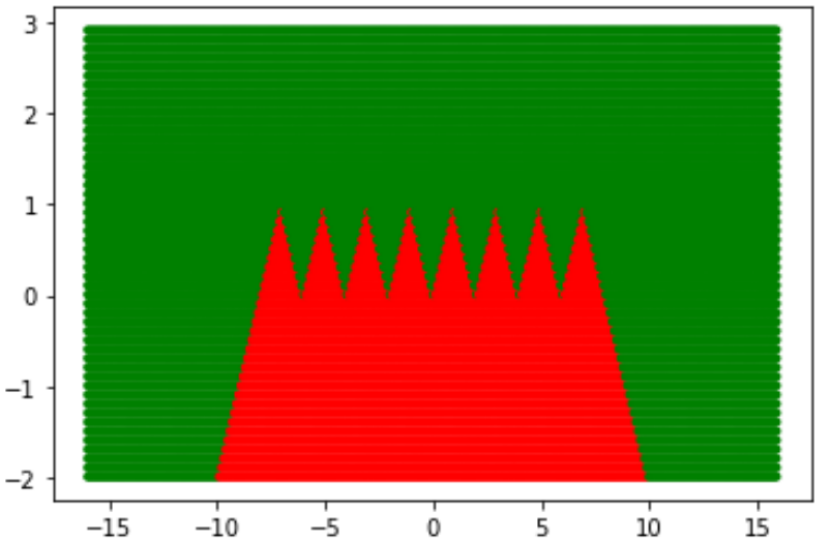

FIGURE 3 – Fonction calculée par un réseau construit à la main à partir de la suite de symétries avec une limite de 8

2 Apprentissage

Dans cette partie il est question d'essayer d'apprendre les symétries et les décisions avec un modèle qui se différenciera selon les parties.

Pour cela nous disposons d'un ensemble d'entraînement généré aléatoirement puis étiqueté, d'un ensemble de test généré aléatoirement puis étiqueté et d'un ensemble pour l'affichage généré régulièrement afin d'obtenir des figures lisibles puis étiqueté.

2.1 Réseaux libres

Le principe est simple, on crée un réseau de neurones avec des fonctions d'activations ReLU et une dernière couche de taille 1 avec une activation sigmoid et on initialise au hasard les poids et biais (loi normal centrée réduite), puis on lui fait apprendre sur l'ensemble d'entraînement. Le but est qu'il arrive à apprendre quasiment parfaitement cet ensemble (on tolère des erreurs très proches de la frontières).

Ici ce qui compte vraiment c'est le choix du nombre de couches et des nombres de neurones sur chaque couche. Pour cela on peut se baser sur le

réseau en introduction, qui lui est capable de faire ce que l'on veut. (Notons qu'il n'a pas exactement la même structure puisque ce dernier utilise des fonctions d'activations plus complexe : Il mélange ReLU et identités sur des mêmes couches).

2.1.1 Résultats

Hélas, il s'avère que même pour une limite de 2, un réseau initialisé au hasard et de même taille que le réseau équivalent de l'introduction, n'arrive pas à apprendre correctement les décisions, et encore moins les symétries.

Pour une limite de 2, un réseau de taille légèrement supérieure (1 neurones de plus sur chaque couche intermédiaire) suffit à apprendre les décisions. (On atteint moins de 5% d'erreur sur l'ensemble de test/train). De plus il se généralise assez bien pour des points en dehors de sa zone d'apprentissage.

En effet l'ensemble d'entraînement est généré dans $[-limite, limite] \times [-1, 2]$ mais pour l'affichage on regarde ce qu'il se passe sur $[-2 \times limite, 2 \times limite] \times [-2, 3]$

Ainsi pour une limite de 2 on obtient ces résultats : (En bleu s'affichent toutes les erreurs faites sur l'ensemble de test)

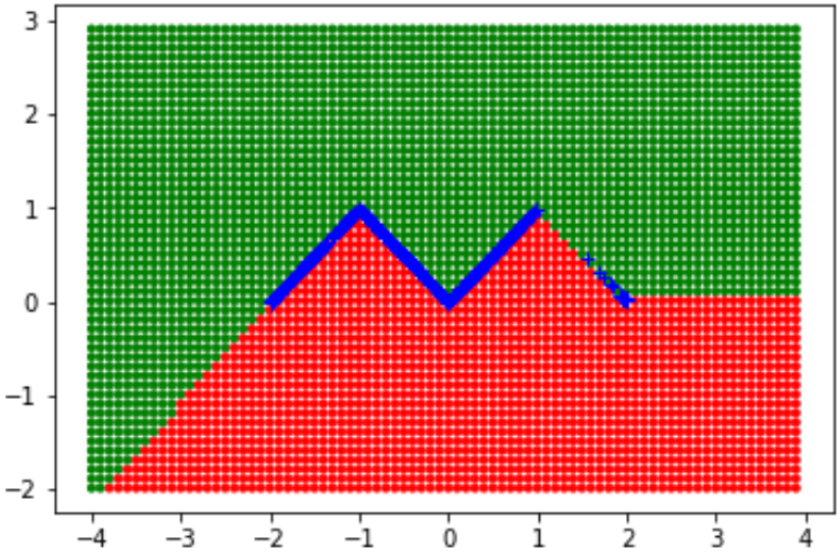

FIGURE 4 – Fonction calculée par un réseau entraîné avec un ensemble généré pour une limite de 2

On peut déjà observer qu'il semble avoir perdu l'utilisation des symétries.

Pour des limites plus élevées cela devient plus compliqué. Déjà pour une limite de 8, il a fallu rajouter 7 neurones sur chaque couche intermédiaire par rapport au modèle de l'introduction. Et on obtient le graphique suivant :

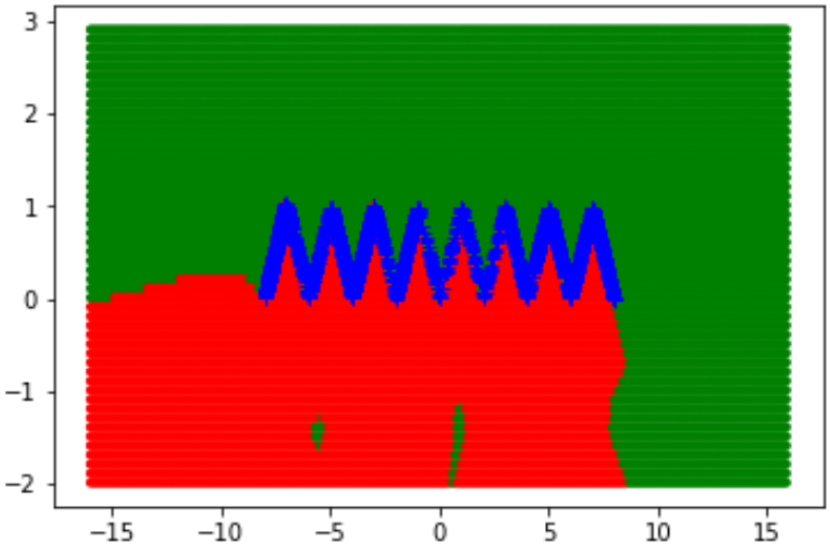

FIGURE 5 – Fonction calculée par un réseau entraîné avec un ensemble généré pour une limite de 8

Ici c'est encore plus flagrant, il n'apparait aucune symétrie. Et pire, le modèle semble trop complexe pour la tâche demandée, en dehors de la zone d'apprentissage, le comportement est plus chaotique. Notamment on peut trouver des zones de vert dans l'intervalle $[-limite, limite]$ et en dessous de -1 (La généralisation ne se passe donc pas si bien) !

Néanmoins si on connait un intervalle de confiance pour l'ordonné des points, on peut alors surement toujours trouver un réseau permettant de faire le travail attendu. Néanmoins il n'apprend pas vraiment les symétries. Et cette solution ne passe pas à l'échelle (C'était notamment l'intéret des symétries, qui réduisait de beaucoup la complexité d'un problème donné)

On peut noter que dans l'apprentissage apparait une dimension aléatoire importante dans le choix initial des poids et biais. A partir de deux configurations initiales différentes, un même modèle ne finit pas dans le même état après l'apprentissage. Notamment un bon moyen d'améliorer l'apprentissage, n'est pas d'essayer d'apprendre longtemps, mais plutot d'aborder le

problème par plusieurs angles, c'est à dire d'apprendre plusieurs fois sur le même modèle mais avec des conditions initiales différentes !

2.2 Réseau particulier : modèle libre de symétrie

Dans cette partie, le modèle considéré est celui d'un réseau de neurones classique (comme depuis le début) mais avec la taille et les activations de celui donné en introduction. (Donc un peu plus restrictif que dans la partie précédente). Le but sera de réapprendre les poids et biais. C'est un modèle qui enchaine donc une rotation (pas au sens mathématique puisque ici il n'y a aucune contrainte sur la matrice de rotation), puis un pliage selon des biais à apprendre, etc. Et enfin une couche de comparaison entre f(x) et y, permettant la décision. Le but est d'apprendre les biais et les poids pour les rotations.

Il faut noter qu'il n'y a aucune contrainte sur les biais ni sur les rotations, ainsi les rotations n'ont pas de raisons à priori d'en être vraiment (ce ne sont que de simples matrices) et le biais au sein d'une même symétries apparait plusieurs fois dans les couches/neurones, mais il n'est pas unifié, il n'y a pas de contrainte sur son évolution, ce qui peut donc mener à différentes valeurs pour un même biais ! Le modèle peut donc permettre à des symétries d'apparaître mais n'est pas restreint à ça. Et il est intéressant de voir si effectivement, dans l'apprentissage les biais s'unifient sur la bonne valeurs, et les matrices deviennt des matrices de rotations.

2.2.1 Liberté sur les bias et les poids

Dans ce cas là on laisse le réseau libre et on regarde ce qu'il apprend. Mais même avec une limite de 2 il n'y arrive pas. Il réduit généralement la fonction de décision à une simple droite et apprend grossièrement à dire si $y > 0.5$. Ce problème apparaissait déjà dans la première partie, mais dans une moindre mesure.

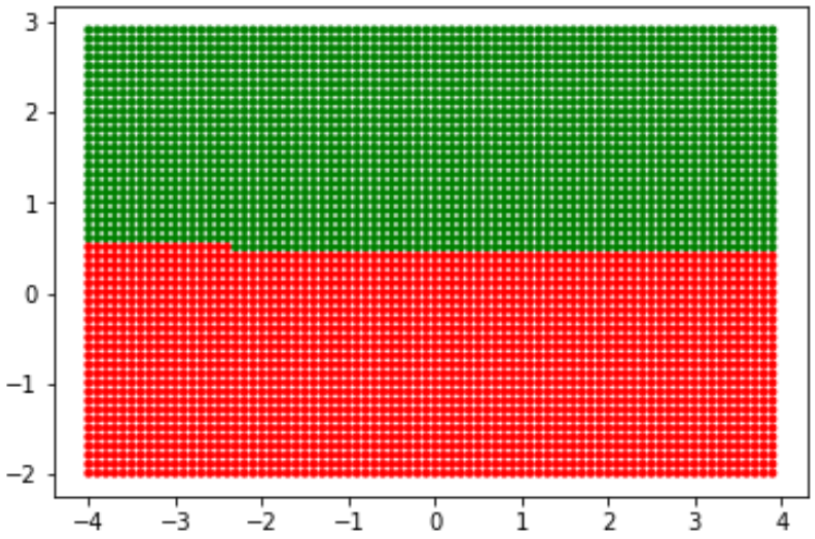

FIGURE 6 – Echec de l'apprentissage pour une limite de 2

Le pire étant qu'il n'apprend aucune symétrie même avec cette architecture contraignante. Les biais ne sont pas unifié et les matrices ne sont pas des matrices de rotation.

2.2.2 Liberté sur les poids ou les bias

Ici on fixe les poids (resp. les biais) correctement, et on essaye d'apprendre les biais (resp. les poids)

Mais ici aussi c'est plutôt un echec. Le réseau arrive tout de même à apprendre des symétries pour une limite de 2, mais pas pour des limites plus grandes. Comme avant il a tendance à trouver une minimum local de la fonction de perte en réduisant à une ou deux droites la fonction de frontières.

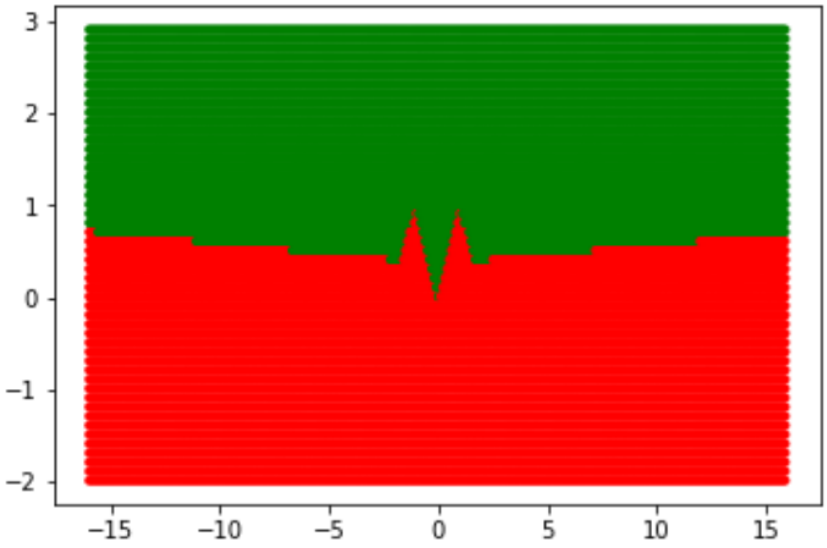

FIGURE 7 – Exemple classique d'un réseau échouant à apprendre la fonction de frontière. Ici la limite vaut 8.

Ces exemples montrent la difficulté du problème, en effet il existe plein de solutions minimisants le problème localement, elle consiste à simplifier le problème à une simple droite. Et cela s'observe d'autant plus que la limite est élevé.

2.3 Réseau de symétrie

Ici le modèle est encore plus restrictif. Il ne se base plus sur la classe initiale de cette étude pour les réseaux de neurones mais sur une nouvelle classe spécialisée pour les symétries et permettant de mutualiser les poids et biais. Dans ce modèle les paramètres sont les angles des rotations et les biais des symétries.

2.3.1 Liberté sur les angles et biais

Dans ce cas encore une fois, c'est un echec. Le réseau apprend des symétries sans grand intéret pour réduire la fonction de frontière à une droite même avec une limite de 2.

2.3.2 Liberté des biais

Connaître les rotations pour l'apprentissage des biais semble avoir un effet très bénéfique sur ce dernier. Le réseau a été capable d'apprendre les biais attendus pour des limites de 2, 4 , 8 et 16. Et n'a pas été testé encore sur des limites plus importantes.

```
//thetas :
0.0 3.1415927 0.0 0.0 3.1415927

//Biases :
-0.0009226046 4.000514 1.9976587 0.9985116
```

FIGURE 8 – Apprentissage des biais : On retrouve 0 suivi des 2^{k-i}. Pour une limite de 8

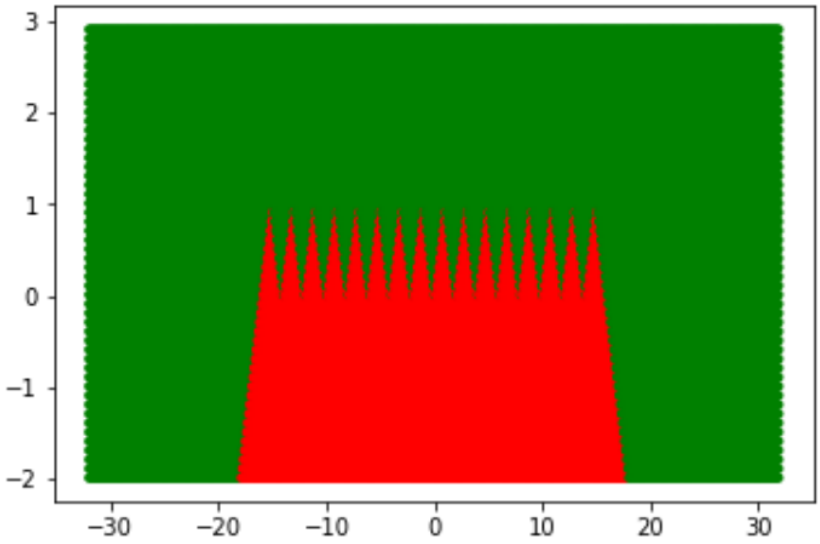

FIGURE 9 – Apprentissage des biais : Testé jusqu'à une limite de 16

2.3.3 Explications

Afin de comprendre pourquoi l'apprentissage ne fonctionne pas bien, il est intéressant de comprendre comment fonctionne l'apprentissage et quelles sont ces pré-requis.

Dans notre cas, on utilise une variante de descente de gradient appelée Adam appliquée à une fonction de coût donnée L. La fonction de coût L choisie dans tous nos modèles ici est l'écart quadratique moyen (MSE).

A chaque étape, chaque paramètre (poids et biais) est mis à jour en fonction de la dérivé de L par rapport à ce paramètre. Les descentes de gradients permettent de trouver des minimums locaux et non globaux. Ici cette variante permet d'essayer de faire un peu mieux. Mais néanmoins on est assuré de trouver le minimum global d'une fonction que si celle si est convexe.

Or si on calcule la fonction de coût en fonction d'un des paramètres de notre modèle, les autres étant fixés aléatoirement on obtient :

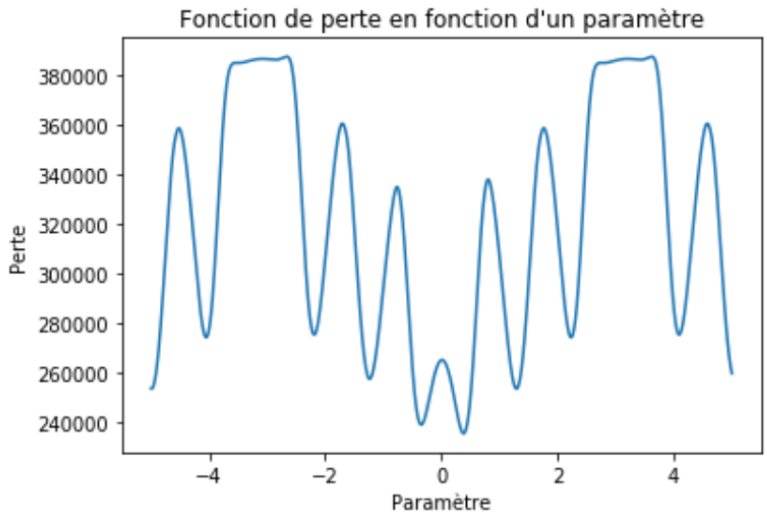

FIGURE 10 – Ici le paramètre est l'angle de la première rotation, avec une limite de 2.

Ce qui est loin d'une fonction convexe. Et il faut imaginer que en réalité cette fonction dépend de tous les paramètres. Cela explique les problèmes rencontrés lors de l'apprentissage. On finit souvent bloqué dans un de ces minimums locaux. Typiquement réduire la frontière à une droite permet de minimiser localement la perte.

3 Pistes à suivre

Il reste encore pas mal de choses à étudier. Certains points méritent plus d'attention que je n'ai pu en porter. Comme l'apprentissage des biais avec les rotations fixées. Savoir si cela peut passer à l'échelle par exemple.

Ensuite, dans les réseaux libres, il n'a été question que de rajouter des neurones au couches. On pourrait envisager aussi de rajouter des couches. Et faire des essais aléatoire pour avoir une idée plus précise de la difficulté avec le passage à l'échelle.

Enfin au vu de la partie 2.3.3, on pourrait envisager de chercher une meilleure fonction de coût. Voici certaines fonctions à considerer : Binary Cross entropy, Hinge Loss, Mean Absolute Error Loss, Mean Squared Logarithmic Error Loss.