CY Tech

Victor VITCHEFF

Résolution d'équations différentielles à l'aide de réseaux de neurones

Rapport de stage de 1^{ere} année effectué du 17 mai 2021 au 8 août 2021

Sous la direction de Mr Aleksey Snytnikov

Novosibirsk State University
Novosibirsk Oblast
Russie, 630090

Table des matières

Introduction

J'ai effectué mon stage de première année d'ingénieur dans une université en Russie. A cause de la crise sanitaire, je n'ai pas pu partir et mon stage est passé en télétravail de la France. 2 professeurs de l'université se sont chargés d'élaborer les sujets du stage et d'accompagner les élèves de CY Tech à travers le stage. L'un d'eux se prénommait Aleksey Snytnikov et c'était mon tuteur université. Parmi les 12 sujets proposés, j'ai pris celui qui me convenait le plus. Je voulais à la fois programmer avec Python mais aussi utiliser mes compétences mathématiques. J'ai aussi été en contact avec ma tutrice au sein de CY Tech, une professeur de lettres madame Cécile D'Agaro.

J'ai décidé de réaliser ce stage car je voulais partir dans un pays étranger. Après avoir regardé sur Arel le choix des différentes universités, j'ai décidé de réaliser mon stage à Novosibirsk en Russie. Je voulais découvrir la Russie et notamment la campagne russe ainsi que les montagnes accessibles depuis Novosibirsk. L'université, quant à elle, me plaisait beaucoup car c'est l'une des meilleurs en Russie et elle possède un grand campus qui paraît très agréable. De plus, mon ancien colocataire à Cergy m'a vivement conseillé de partir là bas car les sujets sont intéressants et la vie agréable. Mon choix de stage s'est donc orienté vers la Russie et j'ai ensuite été séléctionné par CY Tech pour partir là bas. Malheureusement, je n'ai pas eu la chance connaître la vie russe ainsi que l'université à cause de la pandémie Covid.

Dans ce rapport, je vous présenterai la mission qui m'a été attribué ainsi que le déroulement de mon stage en télé travail. Je vais ensuite exécuter un bilan par rapport à ce que j'ai appris et ce qu'il m'a manqué durant ce stage. Je vais adopter un plan chronologique car durant mon stage, les instructions m'étaient données chaque semaine ou lorsque j'avais fini chaque étape de mon projet. Je pense que c'est donc le plan le plus adapté.

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, je tiens à remercier Monsieur Aleksey Snytnikov. Ce fut un super tuteur de stage université et il m'a beaucoup appris pour mon futur ainsi que mon métier plus tard.

Je tiens aussi à remercier ma tutrice d'école Madame Cécile d'Agaro, professeur de lettres, qui a toujours su répondre à mes questions et qui m'a encouragé et féliciter durant mon stage.

Ensuite, je tiens à remercier Madame Audrey Jama, du service relations internationales, qui m'a permis de pouvoir réaliser ce stage en télétravail.

Enfin, je veux remercier Cy Tech pour nous proposer des stages en université grâce à leurs nombreux partenariats.

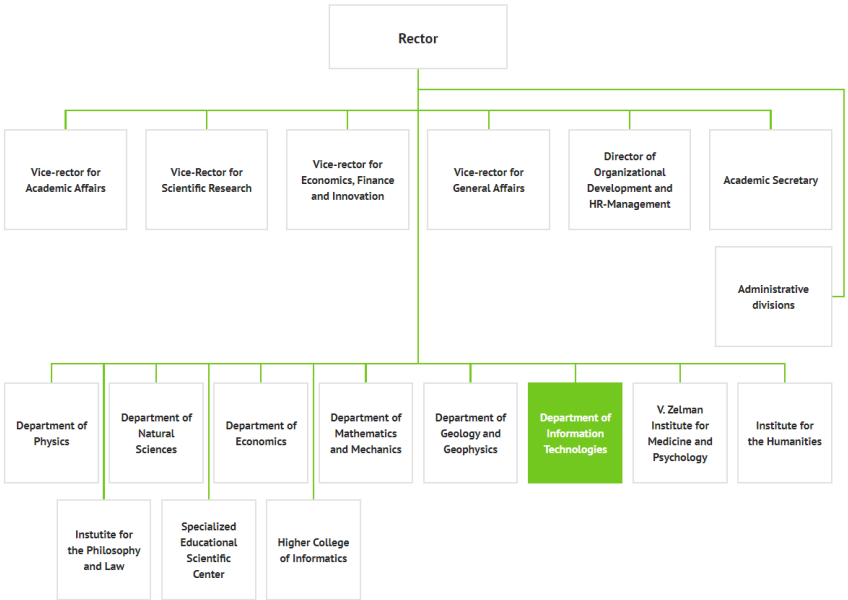
Chapitre 1

Novosibirsk State University

L'université de Novosibirsk possède le statut d'université internationale de recherche de Russie. Elle se situe dans la ville de Novosibirsk à environ 2810km de Moscou vers l'EST. Cette ville est en Sibérie orientale au bord du fleuve l'Ob. Le climat est froid l'hiver et humide l'été. C'est le climat type de la Sibérie en Russie. Dans cette région, l'industrie a donc été beaucoup développé car la Terre est riche en ressources. Mais pour améliorer cette industrie, nous avons besoin d'un institut de recherche. C'est sur ce principe que l'université de Novosibirsk fut construite.

Le début des travaux de l'université ont commencé en 1958. C'est l'académie des sciences de l'URSS qui a ordonné la création de cette université. Dès cette année, des classes préparatoires sont crées pour intégrer l'université. En 1959, l'université ne possédait qu'un seul département : les sciences naturelles. Cependant, il existait déjà 10 matières différentes comme l'analyse mathématique, les équations différentielles ou même l'étude des gaz. C'est dans ces domaines que j'ai effectué mon projet cette année. Le but premier de cette université était d'entraîner le personnel pour ensuite travailler dans des instituts de recherche. Rapidement, de nouveaux départements furent crées notamment ceux de physiques ou géologie. L'université a entraîner de nombreux scientifiques qui ont récolté de nombreux prix ou récompenses. En 2001, l'université récolte la médaille d'or de l'association qui contribue à l'industrie française. En 2002, le campus est agrandi pour que les étudiants puissent pratiquer du sport au sein de l'université. Maintenant, le campus est composé de 7200 étudiants, 2500 professeurs et employés, 100 matières et 45000 alumni. La surface de l'université est très grande, il est facile de s'y loger en tant qu'étudiant et d'y vivre. Selon le Vladimir Potanin's Foundation Ranking (un classement d'université en Russie célèbre), Novosibirsk State University est la meilleure université de Russie.

L'université est composé d'un directeur qui dirige les 6 départements, les 3 instituts (Philosophie et droit, medecine et psychologie et sciences humaines) ainsi qu'un centre scientifique spécialisé. Il est aussi à la tête de 4 sous directeurs ainsi que du pôle administratif de l'université. Il est aussi à la tête du directeur du management de l'université. Le schéma de la structure est ci dessous :



Le département en vert correspond au département où mon tuteur est affilié. Son nom est Aleksey Snytnikov et il fait donc partie du département des technologies de l'information. C'est un professeur de ce département. Il a écrit une thèse sur les performances de système de calcul haute performance. Cette thèse a été supervisé par un autre professeur du même département. Monsieur Snytnikov fut donc un élève de l'université avant.

Akademgorodok, qui signifie literrallement cité académique en russe est le nom du campus de l'université. On parle de Akademgorodok lorsque l'on veut désigner l'université et Novosibirsk lorqu'il s'agit de la ville. Les deux endroits ne se situent pas exactement au même endroit. Il y a 20km d'écart entre les 2 places. Dans ce campus, il est possible de faire du sport, se baigner, se loger. Un quart de l'université est constitué d'une grande forêt. Le campus est fait pour vivre et surtout est construit pour les étudiants russes ou étrangers. Aujourd'hui, Akademgorodok possède 7200 élèves et 45000 sont déjà diplômés depuis sa création.

Chapitre 2

Description, Résultats et Analyse du projet

2.1 Description du projet

2.1.1 Mise en jambes

Mon stage a commencé le lundi 17 mai. C'est au cours d'une première réunion le mardi 18 mai que le sujet nous a été expliqué. Nous avons aussi fait connaissance avec nos tuteurs respectifs. Nous étions 10 élèves durant le meeting et nous avons tous pris connaissance de nos sujets, ceci était expliqué par nos tuteurs. Mon sujet est le titre du rapport : "Résolution d'équations différentielles à l'aide de réseaux de neurones". Le principale but du stage était de trouver une solution à l'équation différentielle de poisson. Mais le tuteur m'a d'abord expliqué pourquoi cette équation en particulière et aussi pourquoi utiliser les réseaux de neurones pour résoudre une équation différentielle.

L'équation de poisson est une équation différentielle qui sert à décrire le plasma dans l'univers. Elle peut s'avérer très complexe en fonction des différents paramètres que l'on choisit au début. C'est pour cette raison qu'elle décrit bien le déplacement du plasma dans notre univers. L'équation s'écrit ainsi :

$$\nabla^2 u = -f$$

u est la fonction que l'on doit résoudre, celle que l'on trace sur le graphique. f est la fonction initiale, la fonction basique, c'est la fonction dont on connaît l'expression. $\nabla^2 u$ est la dérivée seconde de la fonction u. u peut être dérivée selon un ou plusieurs paramètres. Si l'on parle d'un paramètre on parle d'une dimension (1D), s'il y a deux paramètres on parle de deux dimensions (2D) etc. Les paramètres peuvent être la distance noté x et le temps noté t par exemple. Dans le cas d'une dimension, celui où l'on s'intéresse d'abord, l'equation s'écrit de cette manière :

$$\frac{\Delta^2 u}{\Delta x^2} = -f$$

Cette équation peut s'avérer difficile à résoudre si l'on choisit une fonction f dont on ne connaît pas l'intégrale. Par exemple, si on prend $f = e^{-x^2}$ on obtient donc cette équation :

$$\frac{\Delta^2 u}{\Delta x^2} = -e^{-x^2}$$

Résoudre cette équation de manière théorique est très long. C'est pour cela que l'on fait appel à d'autres méthodes pour obtenir un résultat se rapprochant de la réalité. Nous nous sommes intéressé à deux méthodes durant ce projet. La première se nomme la méthode des éléments finis, nous y avons passé une semaine à l'étudier. La deuxième est la méthode des réseaux de neurones. C'est cette méthode que j'ai programmé durant la plupart de mon stage.

Je vais maintenant vous proposer un plan thématique et vous montrer pas à pas les missions qui m'ont été données au cours du stage.

2.1.2 1^{ere} semaine : la méthode des éléments finis

La première tâche qui m'a été donné fut de lire un document donné par mon tuteur afin de comprendre la méthode des éléments finis. Le document fait plus de 300 pages mais nous nous sommes intéressé seulement aux 50 premières pages. Le document nous a expliqué comment marche la méthode mais propose aussi du code pour programmer la méthode. Cela se faisait sur le logiciel de programmation matlab.

Ce principe permet de trouver des solutions sur des points en particulier de l'équation différentielle. Par exemple, on va chercher à trouver une solution au point x=0, x=0.1, x=0.2 etc au lieu de trouver une fonction comme solution. Reprenons l'équation différentielle de départ :

$$\begin{cases} \frac{\Delta^2 u}{\Delta x^2} = -f \\ u(0) = 0 \qquad u(1) = 0 \end{cases}$$

La première étape est de choisir une fonction test que l'on va appeler $\tilde{u}$. Cette fonction doit vérifier les conditions initiales. On peut donc prendre :

$$\tilde{u} = a \times x \times (1 - x)$$
où a est une constante à déterminer

On trouve ensuite a à l'aide de méthodes de dérivation puis on peut estimer le résultat en certains points de l'équation. Le tuteur m'a dit de ne pas m'attarder sur cette méthode donc nous sommes passés ensuite à la méthode des réseaux de neurones.

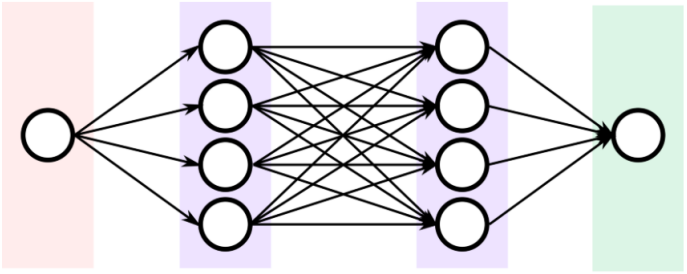
2.1.3 2^{eme} et 3^{eme} semaine : comprendre les réseaux de neurones

Durant cette période, j'ai appris à programmer un réseau de neurones. Le tuteur m'a donné un code en python utilisant le module tensorflow. Ce code est disponible en cliquant ici. A partir de ce code, il a fallu que je comprenne en autonomie comment celui ci fonctionnait et comment le changer pour qu'il soit plus performant. La première équation différentielle sur laquelle j'ai travaillé était la suivante :

$$\begin{cases} \frac{\Delta u}{\Delta x} = 2x \\ u(0) = 1 \end{cases}$$

Cette équation est simple à résoudre de manière analytique. En effet, il est facile de trouver la solution qui est $u = x^2 + 1$. Je vais maintenant vous expliquer comment fonctionne l'approximation par un réseau de neurones que l'on note NN de la fonction u.

Le schéma simple du réseau de neurones que l'on utilise est le suivant :



Sur ce schéma, chaque rond est appelé neurone et chaque flèche est une connexion. Le premier neurone en rose est le neurone d'entrée. Dans notre exemple, cest x. Le dernier neurone en vert est le neurone de sortie. Dans notre cas, c'est l'approximation de $u(x)$. Les neurones en bleus sont les couches cachées du réseau de neurones. Dans cet exemple, il y en a 2 chacun constitués de 4 neurones. Chaque neurone a un poids qui influe sur le neurone de sortie. Nous allons maintenant voir comment faire pour entraîner le réseau afin qu'il possède les meilleurs poids possibles pour approximer u.

Supposons que le réseau de neurones soit une approximation de u. Alors on peut écrire :

$$u(x) \simeq NN(x)$$

En dérivant les deux termes par rapport à x, on obtient :

$$\frac{\Delta u}{\Delta x} \simeq \frac{\Delta NN}{\Delta x}$$

Or, on sait que :

$$\frac{\Delta u}{\Delta x} = 2x$$

Ce qui signifie que :

$$2x \simeq \frac{\Delta NN}{\Delta x}$$

Ou encore :

$$(\frac{\Delta NN}{\Delta x} - 2x)^2 = 0$$

Il faut aussi que le réseau de neurones satisfasse les conditions initiales de l'équation. Sur le même principe, on obtient :

$$(NN(0) - 1)^2 = 0$$

Le but finalement est de minimiser la fonction de perte (noté L) qui est écrite ainsi :

$$L = (\frac{\Delta NN}{\Delta x} - 2x)^2 + (NN(0) - 1)^2$$

Grâce à son module tensorflow, le logiciel python permet de minimiser la fonction de perte. Pour cela, on va choisir 10 points entre 0 et 1. on calcule le point dérivé en chaque point, la fonction de perte et on la minimise. On répète cette opération un grand nombre de fois. Dans l'exemple, on fait ça 1000 fois. On arrive ensuite à trouver L très petit ($L < 0.5$). Une fois que le programme à tourner (environ 3-4 minutes), on trace le graphique sur $[0; 1]$ du réseau de neurones NN et de la fonction originale u. On remarque que l'approximation a bien marché mais que cela peut être amélioré. C'est ce que j'ai réalisé durant ces 2 semaines. J'ai joué avec les paramètres comme le nombre de neurones à utiliser ou encore les méthodes de dérivation en un point d'une fonction. Je vous présenterai les résultats dans la partie "Résultats" du rapport.

2.1.4 4^{eme} et 5^{eme} semaine : Equation de Poisson et réseaux de neurones

Reprenons l'équation de poisson en 1D décrite dans la section mise en jambes :

$$\begin{cases} \frac{\Delta^2 u}{\Delta x^2} = -f \\ u(0) = 0 \qquad u(1) = 0 \end{cases}$$

A titre d'exemple, nous avons choisi $f = x$. Nous avons donc l'équation :

$$\begin{cases} \frac{\Delta^2 u}{\Delta x^2} = -x \\ u(0) = 0 \qquad u(1) = 0 \end{cases}$$

La fonction de perte sera cette fois ci :

$$L = (\frac{\Delta^2 NN}{\Delta x^2} - 2x)^2 + (NN(0))^2 + (NN(1))^2$$

La solution exacte de cette équation de poisson est :

$$u = \frac{1}{6} \times (x - x^3)$$

Après avoir minimsé L, on trace sur un graphique u et NN en fonction de x sur $[0; 1]$. Cela m'a pris beaucoup de temps car il m'a fallu changer les paramètres pour obtenir une solution cohérente sur le graphique. J'ai aussi dû faire face à d'autres problèmes auquelles je ne m'attendais pas.

2.1.5 6^{eme} semaine : Présentation mi-stage

Durant cette semaine, nous avons présenté ce que nous avons fait depuis le début de notre projet. Le tuteur était fier de mon travail. Il m'a dit que l'équation de poisson était très ressemblante à la réalité et que le travail que je fournissais était très bien.

2.1.6 7^{eme} et 8^{eme} semaine : Présentation aux équations différentielles à deux dimensions

La 7^{eme} semaine fut réservée à la présentation d'une nouvelle équation différentielle à 2 dimensions. C'est à dire que l'on choisit une fonction u qui a deux paramètres d'entrée. Elle a le temps noté t et la distance noté x. On peut donc la dériver par rapport à t et par rapport à x. L'équation différentielle associé est noté ainsi et sera appelé équation de convection :

$$\frac{\Delta u}{\Delta t} + \frac{\Delta u}{\Delta x} = 0$$

On décide de fixer le temps t et de le déplacer au fur et à mesure que l'on découvre l'équation. C'est à dire que les graphiques que l'on trace représente u en fonction de x. Certains seront à $t = 0$, d'autres à $t = 0.1$ etc.

A l'équation de convection, nous devons rajouter des conditions initiales pour x ainsi que pour t. Nous avons d'abord mis des bornes aux 2 paramètres. On fixe $x \in [0; 2]$ et $t \in [0; 1]$. La condition initiale sur x sera la suivante :

$$u(x, 0) = \begin{cases} 2 & si \\ 1 & sinon \end{cases} \quad x \in [0.5; 1]$$

Les conditions initiales pour le temps t seront :

$$u(0, t) = 1 \text{ et } u(2, t) = 1$$

Finalement, l'équation de convection s'écrit ainsi :

$$\begin{cases} \frac{\Delta u}{\Delta t} + \frac{\Delta u}{\Delta x} = 0 \\ u(x, 0) = \begin{cases} 2 & si \\ 1 & sinon \end{cases} \quad x \in [0.5; 1] \\ u(0, t) = 1 \qquad\qquad\qquad\qquad u(2, t) = 1 \end{cases}$$

Nous avons utilisé la méthode des éléments finis pour résoudre cette équation. Je me suis aidé d'un Jupyter Notebook crée par la professeure Lorena Barba. Il est disponible à cette adresse ici. On discrétise la distance x en Nx points espacés chacun de δx. Pour une grande précision, δx doit être le plus petit possible. A partir de la définition de la dérivée, on a :

$$\frac{\Delta u}{\Delta x} = \frac{u(x+\delta x, t) - u(x, t)}{\delta x}$$

De la même manière, on discrétise t avec entre chaque temps δt. Et on a :

$$\frac{\Delta u}{\Delta t} = \frac{u(x, t+\delta t) - u(x, t)}{\delta t}$$

Ecrivons cela d'une autre façon. Supposons que l'on se place à un temps $t1$ et à une distance $x1$. Notons $t2$ le temps $t1 + \delta t$ et $x2$ la distance $x1 + \delta x$. Alors, on peut écrire :

$$\frac{\Delta u}{\Delta x1} = \frac{u(x2, t1) - u(x1, t1)}{\delta x} \text{ et } \frac{\Delta u}{\Delta t1} = \frac{u(x1, t2) - u(x1, t1)}{\delta t}$$

En replaçant ces 2 membres dans notre équation différentielle de départ, on obtient :

$$\frac{u(x2, t1) - u(x1, t1)}{\delta x} + \frac{u(x1, t2) - u(x1, t1)}{\delta t} = 0$$

Le temps t étant fixé, on s'intéresse à son avancée dans le temps. On va donc chercher à isoler $u(x1, t2)$. En effet, $t2$ est le temps après $t1$, c'est le temps que l'on ne connaît pas. Ce qui nous donne finalement :

$$u(x1, t2) = u(x1, t1) - \frac{\delta t}{\delta x} \times (u(x2, t1) - u(x1, t1))$$

Grâce à cette formule, je peux facilement coder tout cela en python et obtenir des résultats grâce à cette méthode de la dérivée finie en un point.

2.1.7 9^{eme} et 10^{eme} semaine : Réseaux de neurones sur l'équation de convection

On implémente un réseau de neurones avec 2 entrées et une sortie. Les 2 entrées sont le temps t et la distance x. Ce réseau de neurones est noté NN.

Durant ces 2 semaines, j'ai appliqué le réseau de neurones directement sur l'équation. C'est à dire que j'ai écrit la fonction de perte L comme cela :

$$L = (\frac{\Delta NN}{\Delta x} + \frac{\Delta NN}{\Delta t})^2 + (NN(0,t) - 1)^2 + (NN(2,t) - 1)^2 + (NN(x,0) - u(x,0))^2$$

où x et t sont des valeurs prises dans l'intervalle de départ

Cette méthode d'apprentissage pour le réseau de neurones n'a pas donné de résultat satisfaisant. J'ai donc opté pour une autre méthode, une méthode où le réseau de neurones commence avec un set de données avant d'apprendre directement avec la fonction de perte.

2.1.8 11^{eme} et 12^{eme} semaine : Phase d'apprentissage d'un réseau de neurones

Pendant ces dernières semaines, j'ai appris à entraîner mon réseau de neurones d'abord avec la condition initiale avant d'utiliser tout de suite la fonction de perte. Connaissant les conditions de départ, on peut obtenir un tableau de points à $t = 0$. On initialise ensuite notre réseau de neurones puis on minimalise ensuite la différence entre la sortie du réseau de neurones et la sortie de la condition initiale.

Notons par exemple :

$$u_0(x,0) = \begin{cases} 2 & si \quad x \in [0.5;1] \\ 1 & sinon \end{cases}$$

Cette fonction u_0 est la condition initiale de notre équation de convection. Pour le réseau de neurones, notre première étape sera donc de mettre dans la fonction de perte cette quantité :

$$L = (NN(x,0) - u_0(x))^2$$

De cette manière, à $t = 0$, le réseau de neurones est proche de la solution exacte. On peut ensuite avancer dans le temps ($t = 0.1$ par exemple) mais comme on ne connaît pas la solution exacte on reprend la fonction de perte que l'on a trouvé lors de la semaine 10 :

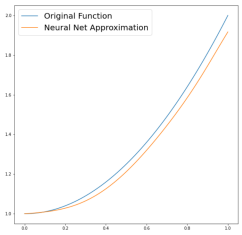
$$L = (\frac{\Delta NN}{\Delta x} + \frac{\Delta NN}{\Delta t})^2 + (NN(0,t) - 1)^2 + (NN(2,t) - 1)^2 + (NN(x,0) - u(x,0))^2$$

Ce moyen me permet d'obtenir des solutions cohérentes par rapport à la solution exacte. La solution avec le réseau de neurones est proche de la solution du professeur Lorena Barba, ces 2 solutions reflétant bien la réalité exacte de la solution. Je vais maintenant vous présenter les résultats que j'ai obtenus au cours de ces différentes semaines.

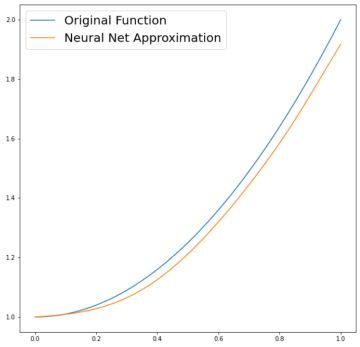
2.2 Résultats du projet

Durant la première semaine, je n'ai pas codé. Je n'ai donc pas eu de résultats, j'ai juste posé des questions à mon tuteur pour mieux comprendre le pdf que celui ci m'a envoyé. J'ai aussi appris à utiliser matlab. J'ai eu des résultats mais les mêmes que sur le pdf. En effet, le code du pdf était difficile à comprendre je n'ai fais que le recopier.

C'est lors de la deuxième semaine que j'ai commencé à coder ou modifier du code tout d'abord. Le tout premier code m'a permis d'obtenir le graphique ci joint :

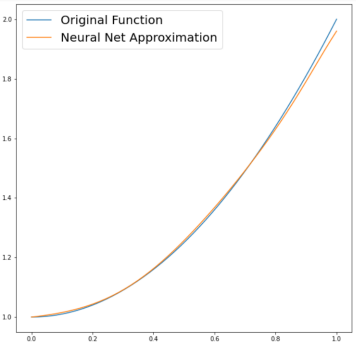


On voit tout de même que le graphique se rapproche de la réalité. Cependant, on peut le rendre encore plus précis en changeant les paramètres de départ. J'ai d'abord joué avec le nombre de couches de réseaux cachées. Il y en avait 4 au départ, j'ai décidé d'en mettre 6 pour obtenir une plus grande précision. Le changement de ce nombre de couches n'a pas influé sur le temps d'execution du programme. Voici le nouveau résultat :

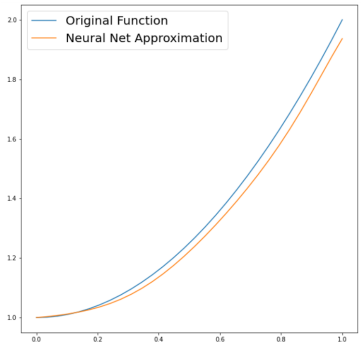


La différence est très minime mais elle se voit lorsque l'on affiche la perte en faisant tourner le code.

J'ai ensuite changé le nombre de points que le réseau de neurones avait en paramètre. Le premier code possédait 10 points à analyser. J'ai d'abord essayer avec 50 points entre 0 et 1 et voici ce que j'ai obtenu :



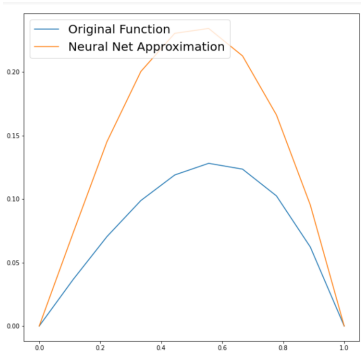
La solution est très proche de ce que l'on recherche mais le temps d'execution fut très long. C'est pour cette raison qu'il m'a fallu redescendre ce nombre de points. J'ai décidé de le mettre à 30 pour obtenir le graphique ci dessous qui est vraiment proche de ce que l'on recherche.



J'ai aussi essayé de changer le lr qui est le taux d'apprentissage du réseau de neurones. Mais après mettre renseigné avec ce document ou celui-là, j'ai observé que le laisser à $lr = 0.01$ était la meilleure chose. Je n'ai donc pas changé ce paramètre.

Le nombre d'occurences pour minimiser la fonction de perte peut être changé aussi mais le bloquer à 5000 est suffisant. En effet, l'augmenter n'augmente que le temps d'execution et le diminuer ne donne pas une perte assez petite. On peut aussi l'adapter en fonction de l'équation qui nous est proposé. Je vais maintenant vous présenter les résultats que j'ai obtenus avec l'équation de Poisson.

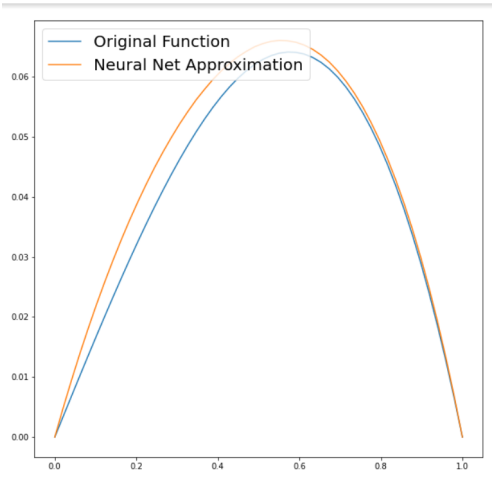
Après avoir changé la fonction de perte ainsi que les conditions initiales pour satisfaire l'équation de poisson, j'ai obtenu ce nouveau graphique :



Ce schéma ne reflète pas ce que je recherche. Certes, l'approximation me montre bien une cloche mais la cloche n'est pas proche de la solution exacte. J'ai donc changé l'approximation du réseau de neurones. C'est à dire qu'au lieu d'approximer la fonction u solution par le réseau de neurones NN, j'approxime la fonction u par $g(x) = x \times (1-x) \times NN(x)$. De cette manière, g satisfait les conditions initiales de l'équation de poisson. La fonction de perte est de cette façon plus simple à obtenir. Elle s'écrit de cette façon :

$$L = (\frac{\Delta^2 NN}{\Delta x^2} + x)^2$$

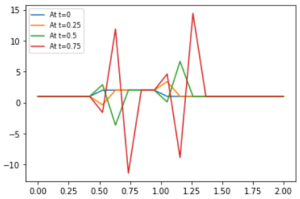
De cette façon, j'ai obtenu un résultat plus proche de la réalité. La façon d'écrire la seconde dérivée permet aussi d'obtenir des résultats plus cohérents. Prenons un exemple. Soit $x1$ un point entre 0 et 1. Je derivais d'abord ce point pour obtenir le point dérivé $x1'$ puis je le dérive une seconde fois pour obtenir $x1''$. Cette méthode est efficace mais une méthode plus efficace consiste à utiliser la définition de la dérivée seconde en un point. De cette façon, j'obtiens une solution de l'équation de poisson vraiment proche de la réalité. Voici cette solution :



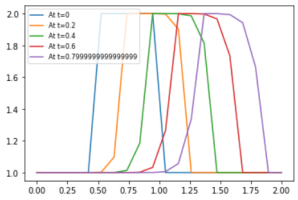
La perte associé est présenté dans la figure suivante. Elle est très faible au cours du temps mais elle ne diminue pas.

```
loss: 0.770928
loss: 0.770886
loss: 0.770848
loss: 0.770814
loss: 0.770782
loss: 0.770752
loss: 0.770725
loss: 0.770700
loss: 0.770676
loss: 0.770654
loss: 0.770634
loss: 0.770614
loss: 0.770596
loss: 0.770579
loss: 0.770563
loss: 0.770548
loss: 0.770534
loss: 0.770520
loss: 0.770507
loss: 0.770495
```

Je suis ensuite passé à l'équation de convection. J'ai d'abord reprogrammé la méthode de Barba Lorena pour obtenir les solutions d'un seul coup. Dans cette méthode on fixe en général δt très petit ($\delta t = 0.001$). J'ai essayé avec $\delta t = 0.1$ mais le graphique n'était pas du tout ressemblant à la solution exacte. En effet, voilà le résultat que je trouve avec $\delta t = 0.1$.

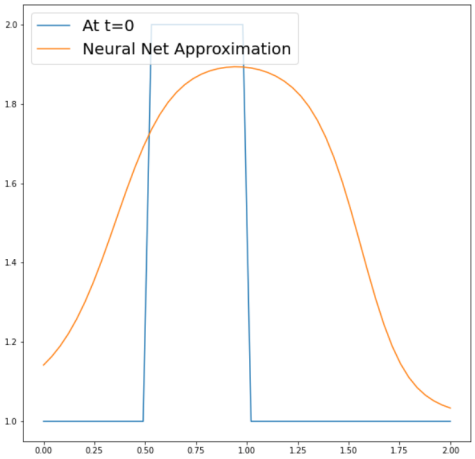


En gardant $\delta t = 0.001$, le code de Lorena Barba m'a permis d'obtenir :

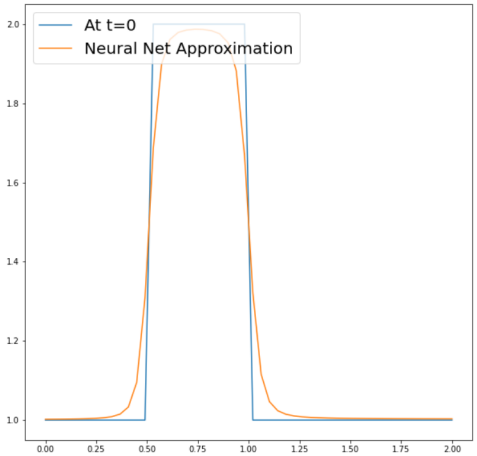


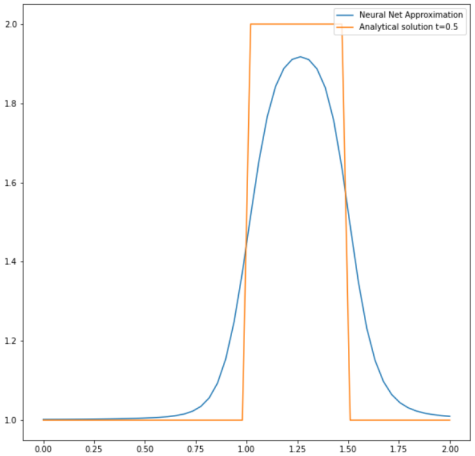
Sur ce graphique, on voit bien l'avancée dans le temps de x.

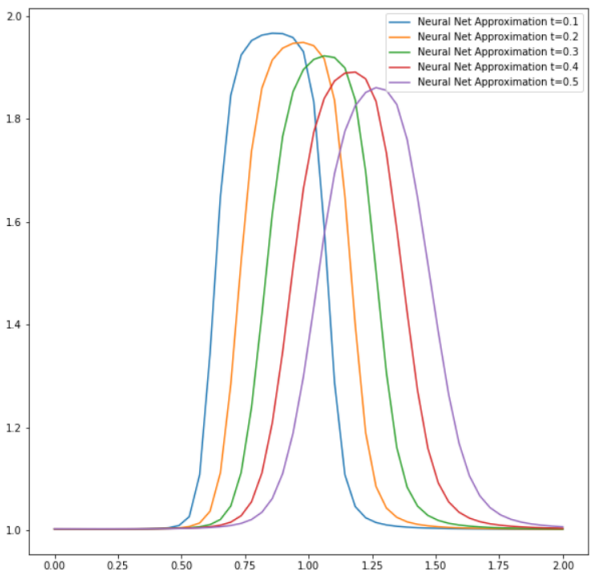
J'ai ensuite fait de même pour le réseau de neurones mais ne sachant pas encore l'entraîner avec un data set, j'obtiens uniquement cette solution :

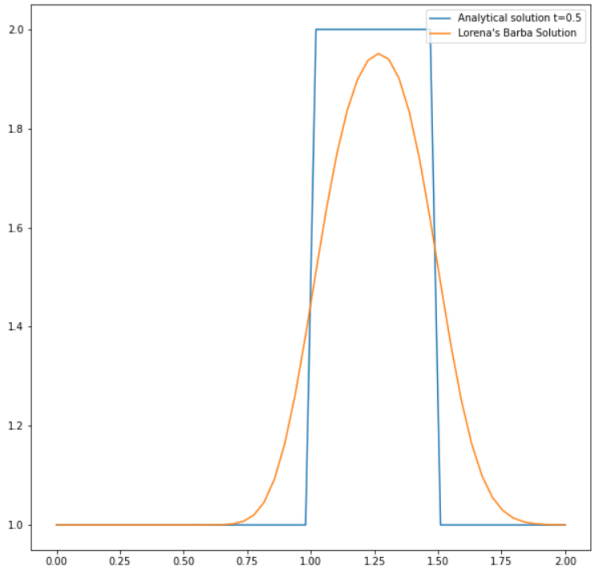


On voit bien que la solution ne correspond pas à ce que je désire. D'où l'importance d'entraîner un réseau de neurones. Après un long travail, j'arrive à obtenir une solution à $t = 0$ mais aussi à $t = 0.1$ etc. Voici les graphiques que j'obtiens qui conclue la partie résultat :









2.3 Analyse du projet

Je faisais souvent face au même problème. C'est à dire que mon tuteur me donnait une tâche à effectuer. La tâche est très facile à comprendre, c'est souvent changer des lignes de code pour arriver à ce que je désire. Mais souvent le problème ne se résolvait pas. Il m'est arrivé de demander à mon tuteur une solution au problème mais celui ci ne me répondait pas ou pas ce que je désirais. Il avait lui aussi beaucoup de travail. J'étais donc obligé de me renseigner sur Internet. Et comme le réseau de neurones est quelque chose de très vaste mais surtout très complexe, on peut tomber sur des articles incompréhensibles qui ne vont pas nous aider ou des articles très utiles dont on a besoin. Et la plupart du temps, j'arrive à résoudre le problème grâce à de la chance ou bien la compréhension du problème. En posant une question précise à mon tuteur, j'arrivais à résoudre le problème auquel je faisais face.

Un des plus gros problèmes auquel je faisais aussi face était de ne pas me perdre sur Internet avec des articles qui n'intéressait pas mon projet. J'ai fait preuve d'une grande curiosité d'esprit mais qui ne s'est pas avéré utile pour mon projet. Je me suis par exemple interessé au taux d'apprentissage d'un réseau de neurones même si cela n'était pas demandé dans mon projet.

Chapitre 3

Bilan du stage

Je pense que la première chose que j'aurai changé aurait été la mobilité. En effet, faire le stage sur place était beaucoup plus sympathique qu'en télé travail. J'ai ressenti un réel manque de motivation dès le début du stage. Je pensais que les tuteurs m'auraient donné cette motivation mais malheureusement cela n'est pas arrivé. Je pense qu'ils étaient aussi peu motivés que nous. C'est la première chose que j'aurais changé pour le stage.

L'un des deuxièmes points que j'aurais aimé changé est la proximité avec le tuteur. En effet, nous avions une réunion par mois et un mail par semaine environ. Il arrivait même que je ne reçoive pas de nouvelles pendant 2 semaines. Peut être qu'avec une réunion par semaine ou même par jour, cela aurait été plus facile de communiquer et d'avancer durant le projet. En effet, à la fin du stage, je n'étais plus en mesure d'avancer. Ayant donné le travail demandé, le tuteur m'a répondu que c'était très bien et je n'ai ensuite rien eu d'autre en réponse. Je pense que j'ai besoin de plus de suivi. Si je devais refaire ce stage, j'aurais été plus bavard avec mes tuteurs.

J'ai pu developper du code dans ce stage. Je pense que cette tâche m'a beaucoup plu. De plus, j'ai codé en python, un langage qui m'est familier et j'ai compris le code qui m'a été donné. Cela me pousse à penser qu'il faut que je sois programmeur plus tard. J'ai besoin de coder pour une entreprise. Le métier de manager m'intéresse moins, je suis à l'aise derrière un ordinateur. Ce stage en télé travail n'a fait que renforcer mon envie de travailler en tant que developpeur informatique. J'espère tout de même trouver un stage passionant dans le domaine de la recherche informatique l'année prochaine en ING2.

Conclusion

Pour conclure, ce stage me fut d'une grande aide pour coder ainsi que pour apprendre l'utilité d'un réseau de neurones. Je sais maintenant coder un réseau de neurones pour résoudre une équation de poisson et une équation de convection qui avance dans le temps. Je ne sais malheuresement pas le faire dans un repère à deux dimensions. Je sais aussi entraîner un réseau de neurones pour le rendre le plus précis possible. Je connais aussi le concept de fonction de perte et de minimisation de celle ci. Je sais le programmer avec le langage de programmation python.

Je regrette de ne pas avoir pu aller en russie mais je suis tout de même heureux d'avoir pu parler en anglais avec un tuteur russe. Ce stage m'offre des perspectives d'avenir. En effet, en ayant travaillé l'utilité des réseaux de neurones, je possède certaines connaissances en deep learning ainsi qu'en intelligence artificielle. Cela fait que je possède un CV encore plus intéressant maintenant.

Annexes

Mon CV

Recherche de Stage en Programmation

Victor Vitcheff

4 les chateaux saint sylvère
95 000 Cergy
Permis B
06 51 65 80 96
vitcheffvi@eisti.eu

Formation

2019-2020 $1^{ère}$ **année en école d'ingénieur**, *CY Tech*, ex-EISTI à Cergy Pontoise, Génie Mathématique.

2017–2019 **Classe préparatoire en école d'ingénieur**, *Ecole Internationale des Sciences du Traitement et de l'Information*, CY Tech, Cergy-Pontoise.

2017 **Baccalauréat Série S**, *mention Assez Bien*, Option SVT spécialité Maths.

Experience professionnelle et projets

2021 **Stage en Télé Travail en échange avec une université**, *Resolution d'équations différentielles à l'aide de réseaux de neurones*, Langage Python, Novosibirsk State University.

2020 **Tuteur à Parkours**, *Suivi d'un groupe de 5 élèves à l'année*, Aide aux devoirs et préparation des DST, Niveau Lycée.

2020 **Réalisation d'une bibliothèque**, *Possibilité d'emprunter un livre et les rendre pour un utilisateur*, Création de classe en POO, Langage Java.

2020 **Création d'un site internet pour la piscine de Cergy-Pontoise**, *Gestion d'une base de données pour les statistiques des différents nageurs*, Langage PHP et MySQL.

2019 **Programmation d'un jeu d'échecs**, *Projet de TIPE à rendre*, Création d'une intelligence artificielle, Langage Python.

Compétences informatiques et mathématiques

Langage OCAML, Python, Pascal, C, HTML, Javascript, PHP, Java, VBA, MySQL

Maths Analyse numérique, optimisation linéaire, théorie des graphes, statistiques

Langues vivantes

Français	Langue Maternelle	*Scolarisation à Londres en 2008-2009*
Anglais	Niveau B2	*Internat à Dublin en novembre 2013*
Allemand	Niveau A1-A2	*Echange à Ludwisbourg en 2011*

Centres d'intérêt

Sports Football, Ski, Surf, Vélo, Course à pied

Code sur l'équation différentielle de convection

Introduction

We want to solve this convection equation :

$$\frac{du}{dx} + c\frac{du}{dt} = 0$$

c has to be understand as a principe of a wave. Here, we consider x as a vector beetween 0 and 2 and t is a value of time. We set up bondary conditions and an initial condition.

$$u(x,0) = \begin{cases} 2 & \text{si } x \in [0.5;1] \\ 1 & \text{sinon.} \end{cases}$$
$$u(0,t) = 1$$
$$u(1,t) = 1$$

We will solve this equation with two methods. The first one is from a jupyler notebook from Lorena Barba. We use the final difference method. I will first of all dicretise the vector x in nx points. And we discretize t in dt points. The discretization of dt is also a problem but we will not see this now. We initialize the array of f who takes the value 2 if x in [0.5;1] or 1 if not. Then, foe each time step and for each points in the vector x we change f with this formula :

$$f_i^{n+1} = f_i^n - c\frac{dt}{dx}(f_i^n - f_{i-1}^n)$$

where i is the moment for x and n is the moment for t.

The second method is with neural networks. So we create a multilayer perceptron neural network with 5 hidden layers of 20 neurons. The neural network's input layer has two inputs : the position x and the time t. we initialize the neural network with random weights and biaises beetween 0 and 1. The output layer will return the sigmoid function plus 1. Because the output value are beetween 1 and 2 due to the initial condition. Let's start the code with this.

```
[ ]  import tensorflow as tf
     import matplotlib.pyplot as plt
     import numpy as np
     import math as ma
```

Here we define the networks parameters and the biaises and weights of the neural network.

```
# Parameters
learning_rate = 0.01

# Network Parameters
n_input = 2  # input layer number of neurons
n_hidden_1 = 32 # 1st layer number of neurons
n_hidden_2 = 32 # 2nd layer number of neurons
n_hidden_3 = 32 # 3th layer number of neurons
n_hidden_4 = 32 # 4th layer number of neurons
n_hidden_5 = 32 # 5th layer number of neurons
n_output = 1    # output layer number of neurons

weights = {
    'h1': tf.Variable(tf.random.normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random.normal([n_hidden_1, n_hidden_2])),
    'h3': tf.Variable(tf.random.normal([n_hidden_2, n_hidden_3])),
    'h4': tf.Variable(tf.random.normal([n_hidden_3, n_hidden_4])),
    'h5': tf.Variable(tf.random.normal([n_hidden_4, n_hidden_5])),
    'out': tf.Variable(tf.random.normal([n_hidden_5, n_output]))
}
```

```
[ ]  biases = {
         'b1': tf.Variable(tf.random.normal([n_hidden_1])),
         'b2': tf.Variable(tf.random.normal([n_hidden_2])),
         'b3': tf.Variable(tf.random.normal([n_hidden_3])),
         'b4': tf.Variable(tf.random.normal([n_hidden_4])),
         'b5': tf.Variable(tf.random.normal([n_hidden_5])),
         'out': tf.Variable(tf.random.normal([n_output]))
     }

     # Stochastic gradient descent optimizer.
     optimizer = tf.optimizers.SGD(learning_rate)
```

In this section, we connected the layers beetween them to create the multilayer_perceptron. The universal approximator (function g) is created. This function approximates the true solution of the previous convection equation.

```
# Create model
def multilayer_perceptron(x,t):
    x = np.array([[x,t]],  dtype='float32')
    # Hidden fully connected layer with 32 neurons
    layer_1 = tf.add(tf.matmul(x,weights['h1']),biases['b1'])
    layer_1 = tf.nn.sigmoid(layer_1)
    # Hidden fully connected layer with 32 neurons
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.sigmoid(layer_2)
    # Hidden fully connected layer with 32 neurons
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    layer_3 = tf.nn.sigmoid(layer_3)
    # Hidden fully connected layer with 32 neurons
    layer_4 = tf.add(tf.matmul(layer_3, weights['h4']), biases['b4'])
    layer_4 = tf.nn.sigmoid(layer_4)
    # Hidden fully connected layer with 32 neurons
    layer_5 = tf.add(tf.matmul(layer_4, weights['h5']), biases['b5'])
    layer_5 = tf.nn.sigmoid(layer_5)
    # Output fully connected layer
    output = tf.matmul(layer_5, weights['out']) + biases['out']
    return (tf.nn.sigmoid(output))

# Universal Approximator
def g(x,t):
  return 1+multilayer_perceptron(x,t);
```

Now we will define the loss function. This function is a good way for the neural network to learn the best weights and biaises for him. First of all, I need to put a value to minimize in the loss function. So I decided to train my data at t=0. We have an array u with the value 1 or 2. It is the initialisation at t=0. So in the loss function I will calculate for k points of X at t=0 the difference beetween

$$f[k]$$

and

$$g(k,0)$$

And I square this difference. For A big number of training steps, the loss become very small and the approximation of the neural network is really close to the reality.

Here are the two functions.

```
[ ]  def loss_apprentissage():
         X=xcoord
         summation=[]
         for i in range(len(X)):
             summation.append((g(X[i],0)-f[i])**2)
         return tf.reduce_sum(tf.abs(summation))


     def apprentissage():
         with tf.GradientTape() as tape:
             loss = loss_apprentissage()
         trainable_variables = list(weights.values()) + list(biases.values())
         gradients = tape.gradient(loss, trainable_variables)
         optimizer.apply_gradients(zip(gradients, trainable_variables))
         return(loss)
```

Here is the definition of the initial condition; the bondary condition and dt for the neural network approximation. dt for the finite difference method has to be smaller so he will be different.

```
u_x0 = 1
u_x2 = 1


def u(x):
    if (x>=0.5) and (x<=1):
        return 2
    else :
        return 1
```

This are the function who train the neural network at t=0.1. This time, in the neural network, the loss function minimise the derivative of g with t and g with x. She also minimize the bondary conditions and the initial condition. With a big number of training steps, the approximation is really good.

```
def training_loss(exa):
    summation = []
    #T=0.09 #Time fixed
    X=xcoord
    Ttab=[0,0.1,0.2,0.3,0.4,0.5,0.6]
    fTab=[0,0.1,0.2,0.3,0.4,0.5]
    for j in range(len(fTab)):
        for i in range(0,X.shape[0]-1):
            #dNN_t = (g(X[i],T+dt)-g(X[i],T))/dt
            #dNN_x = (g(X[i+1],T) - g(X[i],T)) / (X[i+1] - X[i])
            #dNN_t = (g(X[i],T[j+1]) - g(X[i],T[j])) / (T[j+1] - T[j])
            #summation.append((dNN_x + dNN_t)**2 + (g(0,T) - u_x0)**2 + (g(2,T) - u_x2)**2 + (g(X[i],0) - u(X[i]))**2) #For Con
            summation.append((g(X[i],fTab[j]) - exa[j,i])**2)
    for i in Ttab:
        summation.append((g(0,i) - u_x0)**2 + (g(2,i) - u_x2)**2)
    return tf.reduce_sum(tf.abs(summation))

def train_step(exa):
    with tf.GradientTape() as tape:
        loss = training_loss(exa)
    trainable_variables = list(weights.values()) + list(biases.values())
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
    return(loss)
```

This function creates the array f, the initial condition. She also create the array xcoord. This array has nx points beetween 0 and 2. This array is useful to draw the plot.

```
#Initialisation of the function at t=0
def Initialisation(nx):
    xcoord = np.ones(nx)
    tmp=0
    for i in range(nx):
        xcoord[i]=tmp
        tmp=tmp+dx
    f=np.ones(nx)
    for i in range(len(f)):
        f[i]=u(xcoord[i])
    return(xcoord,f)
```

Initialisation of the array xcoord and f. Definition of dt2 which is for the finite difference method.

```
[ ] nx= 50
    dx = 2/(nx-1)
    dt2=0.001

    xcoord,f = Initialisation(nx)
```

We train here the neural network with the data of f.

```
[ ]  apprentissage_steps=2000
     display_step = apprentissage_steps/20
     for i in range(apprentissage_steps):
         apprentissage()
         if i % display_step == 0:
             print(loss_apprentissage())
```

```
tf.Tensor(37.6249, shape=(), dtype=float32)
tf.Tensor(8.240331, shape=(), dtype=float32)
tf.Tensor(6.4542847, shape=(), dtype=float32)
tf.Tensor(6.492918, shape=(), dtype=float32)
tf.Tensor(2.794805, shape=(), dtype=float32)
tf.Tensor(1.416558, shape=(), dtype=float32)
tf.Tensor(0.99472594, shape=(), dtype=float32)
tf.Tensor(0.8136756, shape=(), dtype=float32)
tf.Tensor(0.7073785, shape=(), dtype=float32)
tf.Tensor(0.63360983, shape=(), dtype=float32)
tf.Tensor(0.5778531, shape=(), dtype=float32)
tf.Tensor(0.53346246, shape=(), dtype=float32)
tf.Tensor(0.4969653, shape=(), dtype=float32)
tf.Tensor(0.46639833, shape=(), dtype=float32)
tf.Tensor(0.44061038, shape=(), dtype=float32)
tf.Tensor(0.41885966, shape=(), dtype=float32)
tf.Tensor(0.40059575, shape=(), dtype=float32)
tf.Tensor(0.38529968, shape=(), dtype=float32)
tf.Tensor(0.3724201, shape=(), dtype=float32)
tf.Tensor(0.36137268, shape=(), dtype=float32)
```

Here, we draw the plot at t=0 with the neural network approximation. It is very close to the reality.

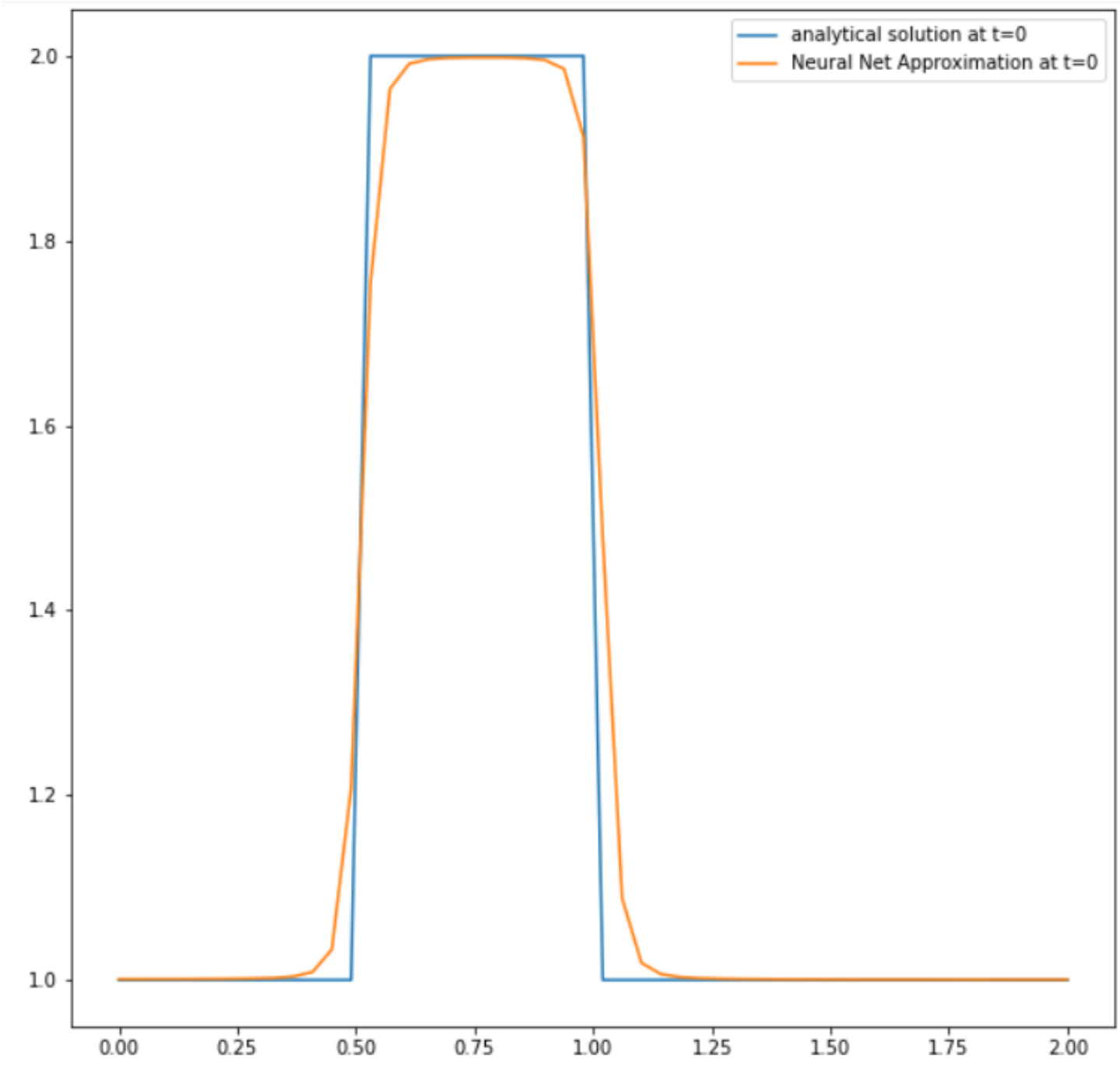
```
from matplotlib.pyplot import figure

figure(figsize=(10,10))



result=[]
T=0
for i in xcoord:
  #result.append(f(i))
  result.append(g(i,T).numpy()[0][0])

plt.plot(xcoord,f,label="analytical solution at t=0")
plt.plot(xcoord, result, label="Neural Net Approximation at t=0")
plt.legend(loc=1, prop={'size': 10})
plt.show()
plt.close()
```


analytical solution at t=0
Neural Net Approximation at t=0
2.0
1.8
1.6
1.4
1.2
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

We train the neural network for t=0.1.

```
def Apply_method(f,nx):
    u1=f.copy()
    for i in range(1,nx):
        u1[i]=f[i] - dt2/dx*(f[i]-f[i-1])
    return(u1)

tabfinal=np.ones((6,nx))
i=1
tabfinal[0]=f
deb=0
obj=0.1
while (i<6):
    tabfinal[i]=tabfinal[i-1]
    while (deb<obj):
        tabfinal[i]=Apply_method(tabfinal[i],nx)
        deb=dt2+deb
    i=i+1
    obj=obj+0.1
```

```
[ ] training_steps = 1000
    display_step = training_steps/5
    #print("for T= "+str(T))
    for i in range(training_steps):
        train_step(tabfinal)
        if i % display_step == 0:
            print("loss: %f " % (training_loss(tabfinal)))
```

```
loss: 30.793159
loss: 7.541766
loss: 2.821058
loss: 1.430565
loss: 1.860978
```

This is the function who applied the finite difference method from Lorena Barba. The function is for only one time step (dt2).

So I decided to apply this function until dt2 >0.1. At each step, I increase a tempory variable with dt2 and when tmp2 >0.1, I stop to apply the function.

Here, we draw two plots.

One with the exact solution and the neural network approximation at t=0.1.

The other plot have the exact solution and the finite difference method approximation.

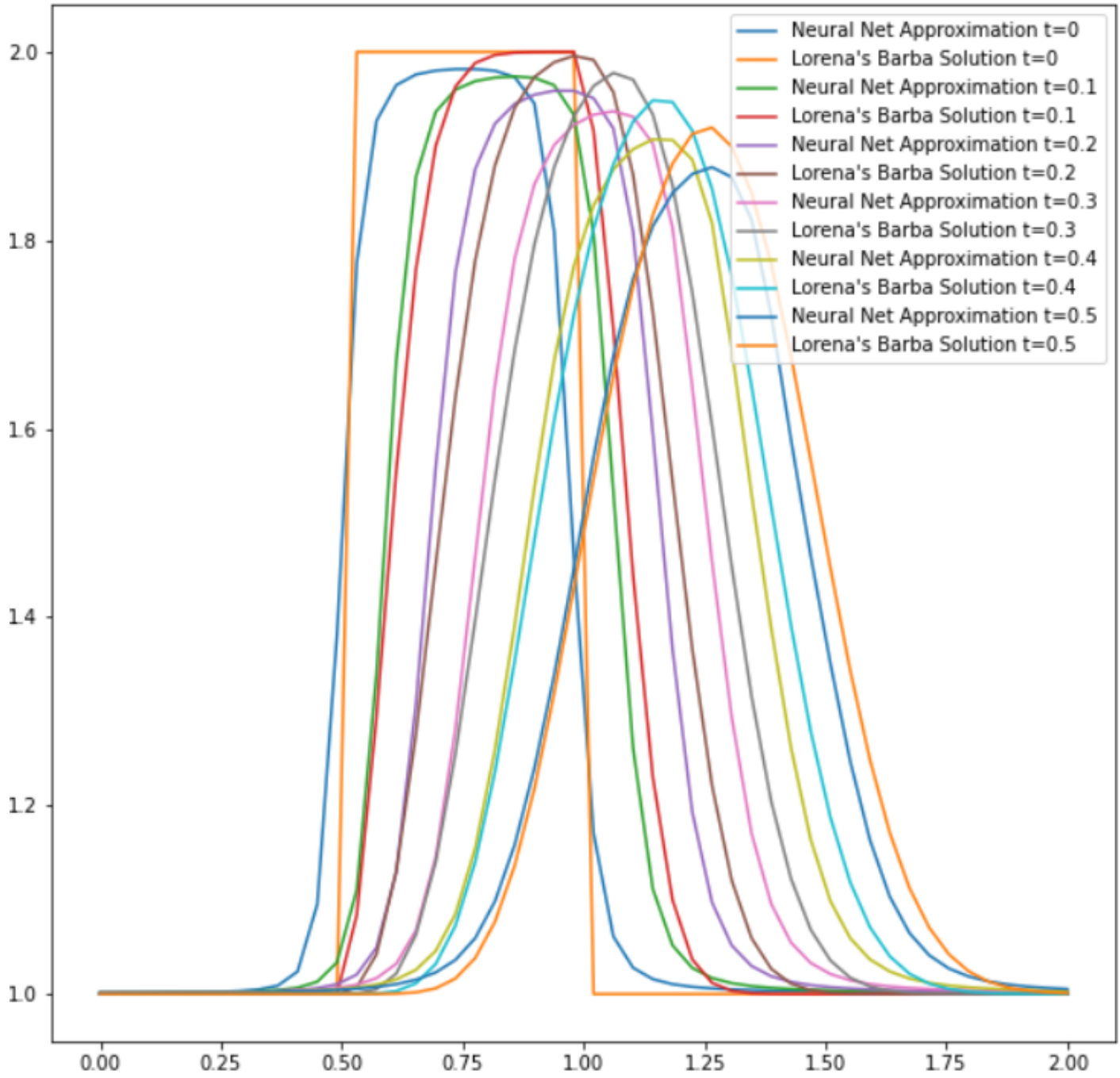
```
X = xcoord
def true_solution(x,t):
    return u(x-t)

result=[]
T=0
T2=0.5
for i in xcoord:
  result.append(g(i,T2).numpy()[0][0])


S=[]
for i in X:
    S.append(true_solution(i,T))

S2=[]
for i in X:
    S2.append(true_solution(i,T2))
tableautemps=[0,0.1,0.2,0.3,0.4,0.5]
figure(figsize=(10,10))
for i in range(len(tableautemps)):
    result=[]
    for j in xcoord:
        result.append(g(j,tableautemps[i]).numpy()[0][0])
    plt.plot(X, result, label="Neural Net Approximation t="+str(tableautemps[i]))
    plt.plot(xcoord,tabfinal[i],label="Lorena's Barba Solution t="+str(tableautemps[i]))

plt.legend(loc=1, prop={'size': 10})
plt.show()
plt.close()
```

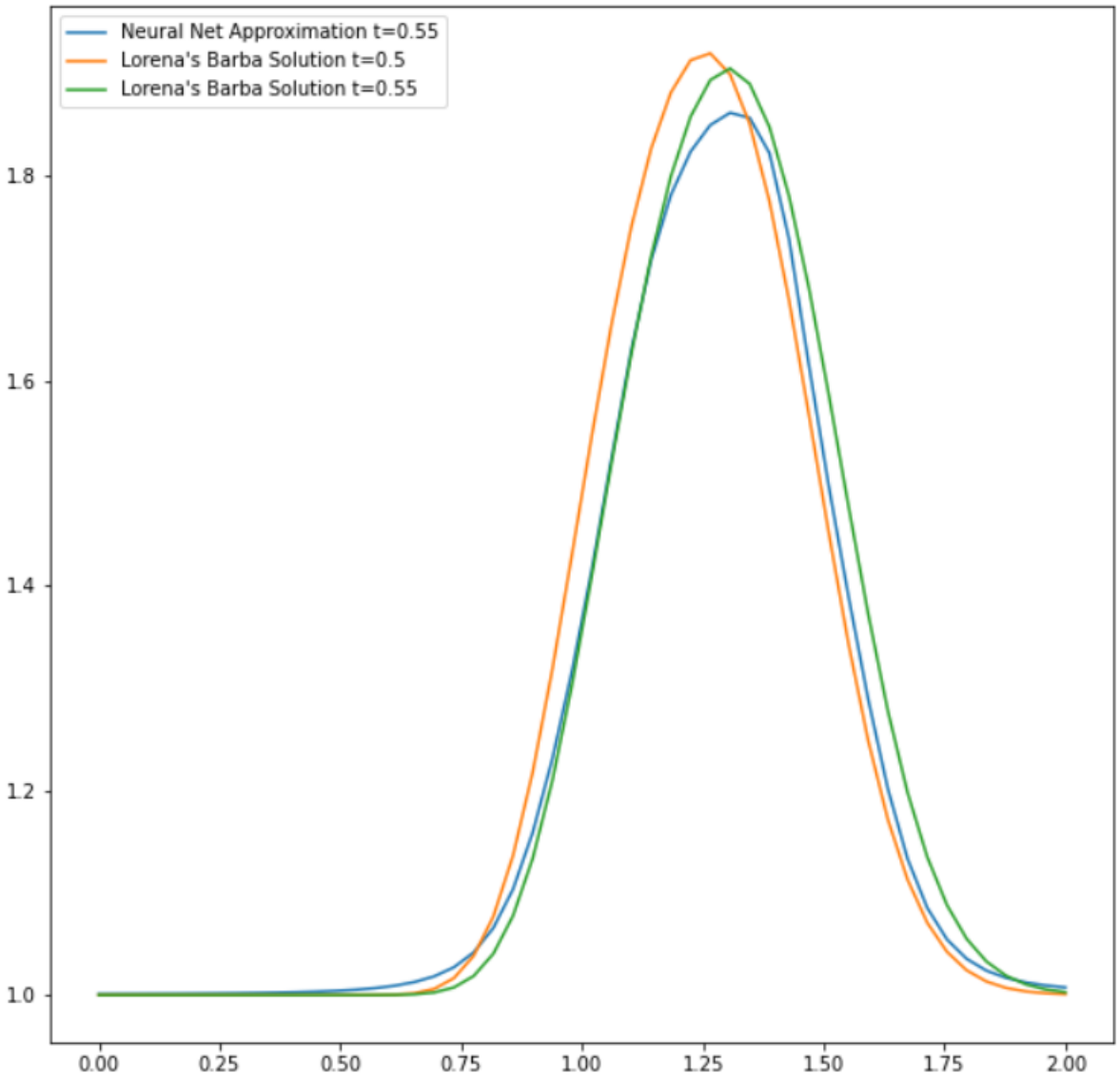

Neural Net Approximation t=0
Lorena's Barba Solution t=0
Neural Net Approximation t=0.1
Lorena's Barba Solution t=0.1
Neural Net Approximation t=0.2
Lorena's Barba Solution t=0.2
Neural Net Approximation t=0.3
Lorena's Barba Solution t=0.3
Neural Net Approximation t=0.4
Lorena's Barba Solution t=0.4
Neural Net Approximation t=0.5
Lorena's Barba Solution t=0.5
2.0
1.8
1.6
1.4
1.2
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

```
f=Apply_method(tabfinal[5],nx)
T=0.501
while (T<0.55):
    f=Apply_method(f,nx)
    T=T+0.001
```

```
result=[]
T=0.5
T2=0.55

for i in xcoord:
  result.append(g(i,T2).numpy()[0][0])

figure(figsize=(10,10))
#plt.plot(X,S,label="true solution t="+str(T))
plt.plot(xcoord, result, label="Neural Net Approximation t="+str(T2))
plt.plot(xcoord,tabfinal[5],label="Lorena's Barba Solution t="+str(T))
plt.plot(xcoord,f,label="Lorena's Barba Solution t="+str(T2))
plt.legend(loc=2, prop={'size': 10})
plt.show()
plt.close()
```


Neural Net Approximation t=0.55
Lorena's Barba Solution t=0.5
Lorena's Barba Solution t=0.55
1.8
1.6
1.4
1.2
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Bibliographie

Code permettant de programmer des réseaux de neurones pour une équation différentielle : https://towardsdatascience.com/using-neural-networks-to-solve-ordinary-differential-equations-a7806de99cdd

Code de Lorena Barba : https://nbviewer.jupyter.org/github/barbagroup/CFDPython/blob/master/lessons/01$_step_1.ipynb$

Article intéressant sur le taux d'apprentissage d'un réseau de neurones : https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0