Rapport de projet CLANU

Simona Miladinova-Seydou Dia

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

3GE1: 2018-2019

SOMMAIRE

1) Introduction.....1
2)Analyse Numérique.....2
3)IF2.....8
3.1 Prise en main.....8
3.2 Premiers développement: voir les données de prédictions.....9
3.3 Seconds développements: Entraînement, Sauvegarde et Ressources.....12
3.4 Troisième développement: optimisation et observation lors de l'entraînement.....16
4)Conclusion Technique.....21
5)Conclusion global du projet et organisation.....22

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Simona MILADINOVA
Seydou DIA

Rapport de projet CLANU

1. Introduction

Dans le cadre de notre formation au département génie électrique de l'INSA Lyon, nous avons été amenés à travailler sur l'étude d'un réseau de neurones permettant de classifier des images à partir d'une base données. Cette dernière est la base de données MNIST(Modified National Institute of Standards and technology). Ce projet à plusieurs objectif. L'étude de ce projet se fera en deux parties. La première consistera à trouver les paramètres optimaux sous matlab de notre réseau de neurones(le nombre d'époques, de couches cachées etc.). La seconde partie se fera en langage et aura pour but de nous familiariser avec l'environnement de développement Qt Creator et d'étudier l'implémentation d'un réseau de neurones en langage C. Cette étude intégrera le développement de fonction et d'étude de l'influence des paramètres et hypers paramètres du réseau.

Vous trouverez dans ce rapport l'ensemble des réponses aux questions des parties d'analyse numérique et d'informatique. Enfin nous apporterons une conclusion à notre projet et vous ferons un récapitulatif de l'organisation du projet.

Nous tenons à insister sur le fait que le rapport est détaillé et présente de nombreuses captures d'écran pour nous permettre à nous, futur ingénieurs, de revenir sur le rapport en ayant un accès facile à l'information dans le cas où nous serons amenés à travailler à l'avenir sur ce type de projet. Ces captures peuvent sembler redondantes pour certains, mais nous sommes convaincus qu'elles pourront s'avérer utiles lors de nos futures expériences professionnelles.

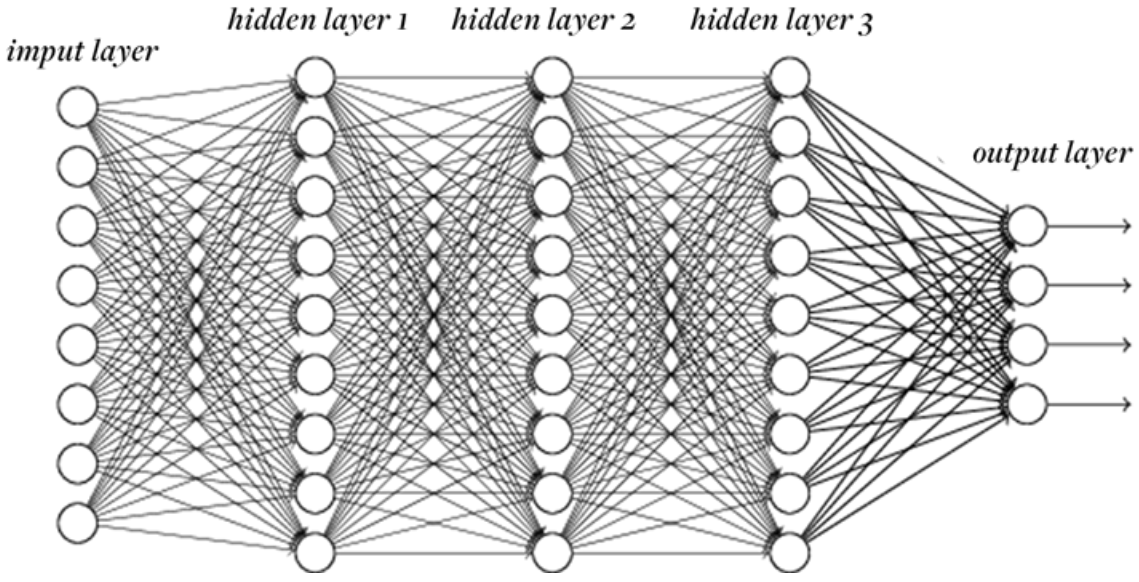

image *extraite de medium.com*

Simona MILADINOVA
Seydou DIA

II. Analyse Numérique

Question 1.

Il a été demandé dans un premier temps de modifier le code mis à notre disposition pour pouvoir réaliser des propagations avant et arrière, mais aussi d'implémenter un code qui permet de déterminer le label d'une image. Voici ci-dessous la fonction implémentée:

```
%-- Forward propagation
[probas,~] = L_model_forward(X, parameters);
%display(probas);
%
y_prediction = zeros(size(probas));

%-- Convert probas to label predictions
for i=1:m

    V=0;    %valeur max
    k=0;
    for j=1:10
        if (probas(j,i)>V)
            V=probas(j,i);
            k=j;
    end
   end
    y_prediction(k,i) = 1; %COMMENTER APRES
end
end
```

Figure 1: Implémentation de la fonction predict

```
%-- Forward propagation: [LINEAR -> RELU]*(L-1) -> LINEAR -> SIGMOID.
%VOTRE CODE ICI
[AL, caches] = L_model_forward(X_mini_batch, parameters);%appel de la fonction Forward propagation
%-- Backward propagation.
[grads]=L_model_backward(AL, Y_mini_batch, caches);%appel de la fonction Backward propagation
%-- Update parameters.
%DECOMMENTER APRES
[parameters] = update_parameters(parameters, grads, learning_rate);
```

Figure 2:Implémentation des fonctions de propagations avant et arrière dans model_sgd

Simona MILADINOVA
Seydou DIA

Nous traçons le coût de validation et le coût d'entraînement pour différentes valeurs d'époques. Nous prenons respectivement num epochs=10 ; 100 et 200. Nous choisissons également de tracer la précision sur l'ensemble d'entraînement et la précision sur l'ensemble de validation.

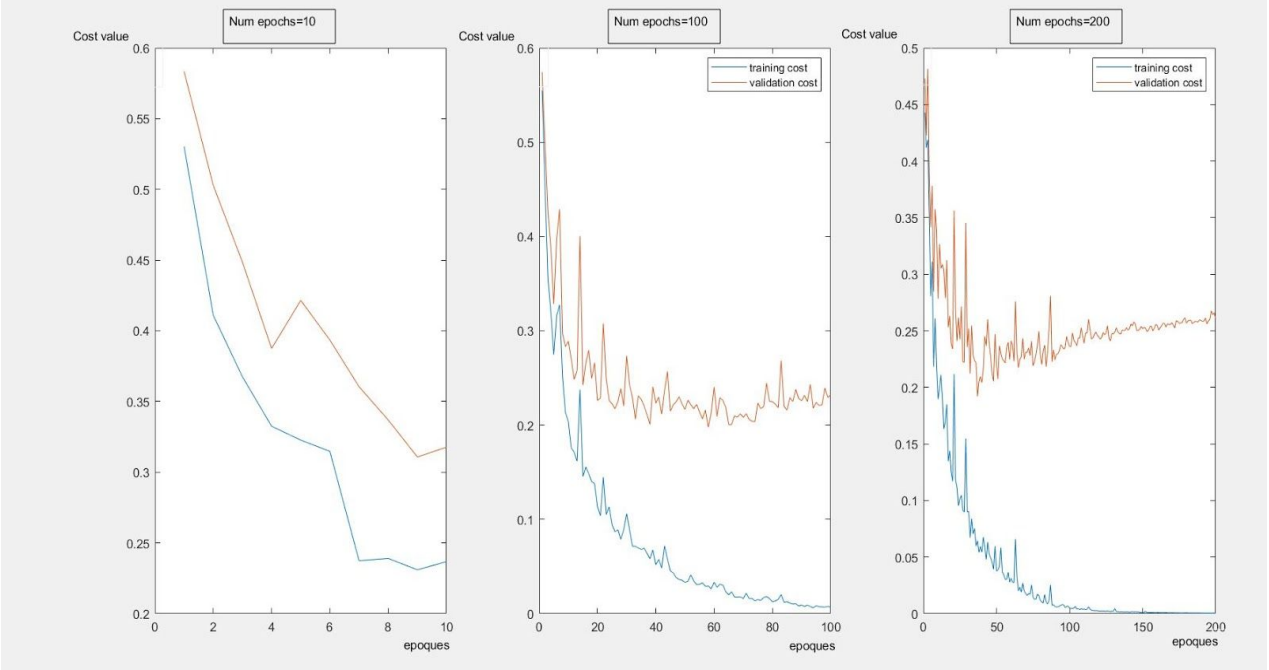

Figure 3 :Tracé du coût de validation en fonction des époques pour un nombre totale d'époques=10;100 et 200;

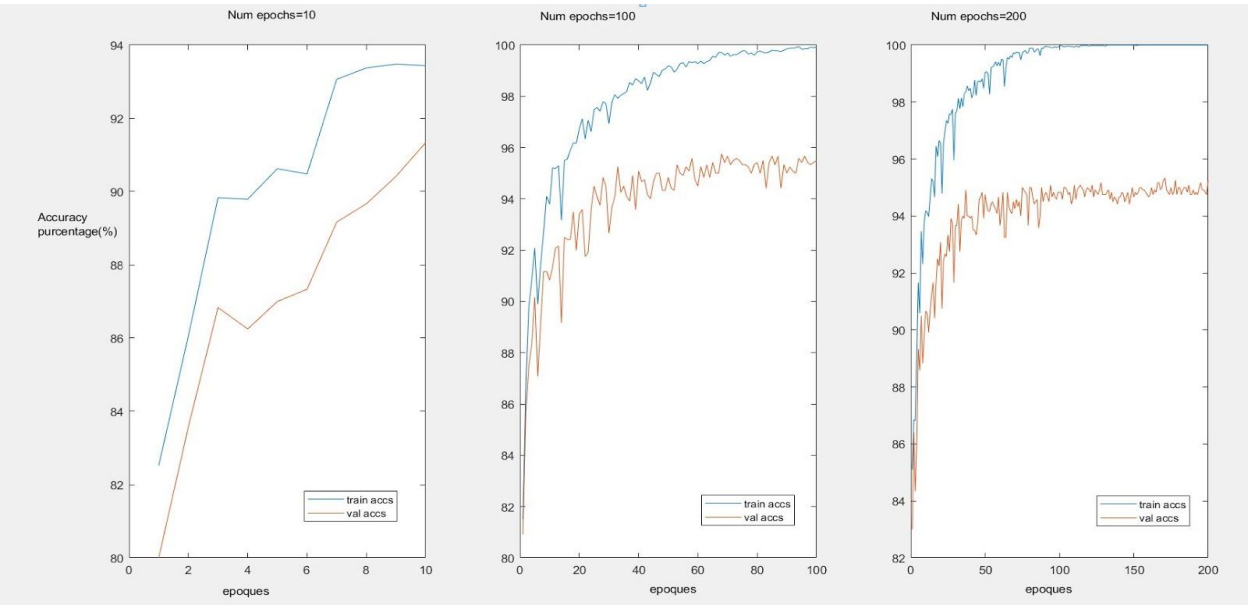

Figure 4 Tracé de la précision en fonction des époques pour un nombre totale d'époques=10;100 et 200;

Nous remarquons qu'en augmentant le nombre d'époques, le coût d'entraînement diminue à la suite de la dernière époque. Cela est normale parce que nous optimisons l'apprentissage sur les données d'entraînement. Cela dit, en s'intéressant au coût de validation, on remarque qu'il diffère grandement du coût d'entraînement lorsque l'on augmente le nombre d'époques.

Question 2.
Dans cette question, nous entraînons le réseau de neurones avec les valeurs données dans l'énoncé pour les paramètres suivants:
n_hidden = 100 neurones
num_examples = 6000 exemples
val split = 0.2
num_epochs = 10, **100, 200**
Le tableau ci-dessous récapitule bien les valeurs obtenues de l'entraînement après la 10e, 100e et 200e époque.

Nombre d'époque	Training cost	Validation cost	Training accuracy (%)	Validation accuracy (%)
10	0.23678	0.31772	93.4375	91.3333
100	0.0066768	0.23199	99.9167	95.5
200	0.00044319	0.26039	100	95.25

Tableau 1: Les valeurs issues de l'entraînement après la dernière époque avec les paramètres de la question 2

Sur les valeurs ci-dessus on voit bien que pour 200 époques j'ai un coût d'entraînement qui est le plus bas. 0.000044319 pour Num_epochs=200 et pour Num_epochs=100 nous avons 0.0066768. Mais on voit bien que le coût de validation est moins bon pour 200 époques. On peut donc conclure que nous sommes en surapprentissage. Nous avons trop entraîné notre réseau sur les données d'entraînement et il n'est pas aussi optimisé que pour d'autres données que le réseau à 200 époques. Nous sommes en apprentissage.

Nous entraînons maintenant le réseau de neurones avec différentes valeurs de couches cachées allant de 10 à 100 par pas de dix couches(10, 20... 90, 100)

Simona MILADINOVA
Seydou DIA

Question 3.

Nous décidons de tracer les coûts d'entraînements et de validations minimaux en fonction du nombre de neurones. Nous fixons 100 époques(toujours pour 6000 exemples).
On remarque que le coût est minimal pour un réseau à 80 neurones et un coût de validation minimal pour un réseau à 90 neurones. Cette étude montre bien qu'avoir plusieurs couches neurones n'est pas synonyme de meilleures convergences et d'optimisation.

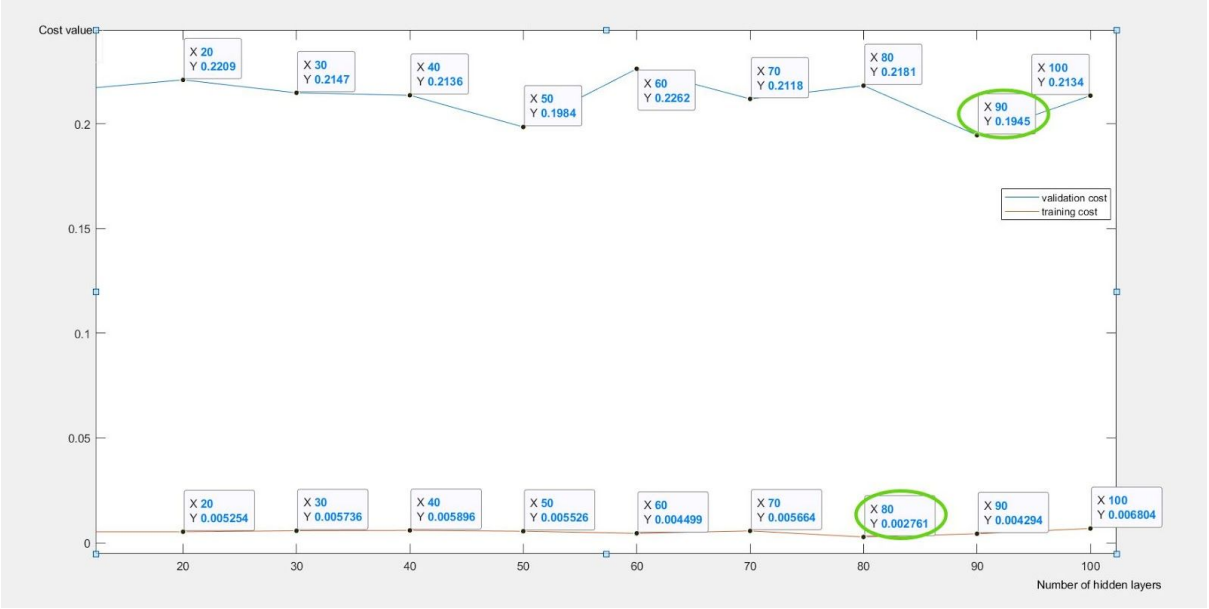

Figure 5 :Tracé du coût de validation et d'entraînement minimal pour une couche cachée allant de 20 à 100 neurones.

Nous cherchons ensuite l'époque correspondant à ce coût minimale et nous récupérons les valeurs de coût de validation, de précision d'entraînement et de validation.

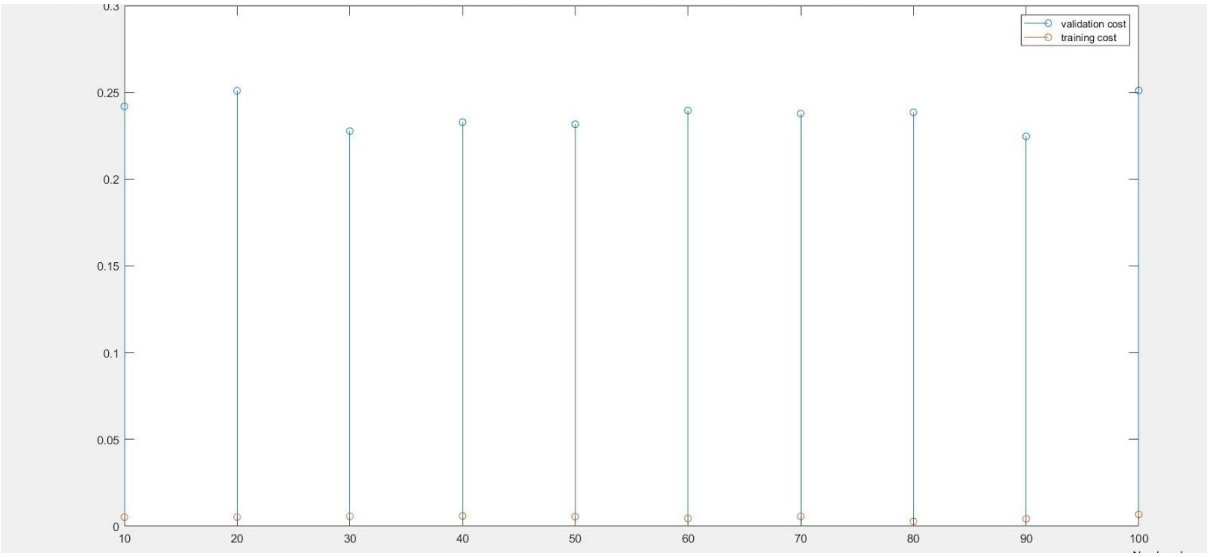

Figure 6 :Tracé du coût de validation et d'entraînement minimal pour différentes époques avec la fonction stem

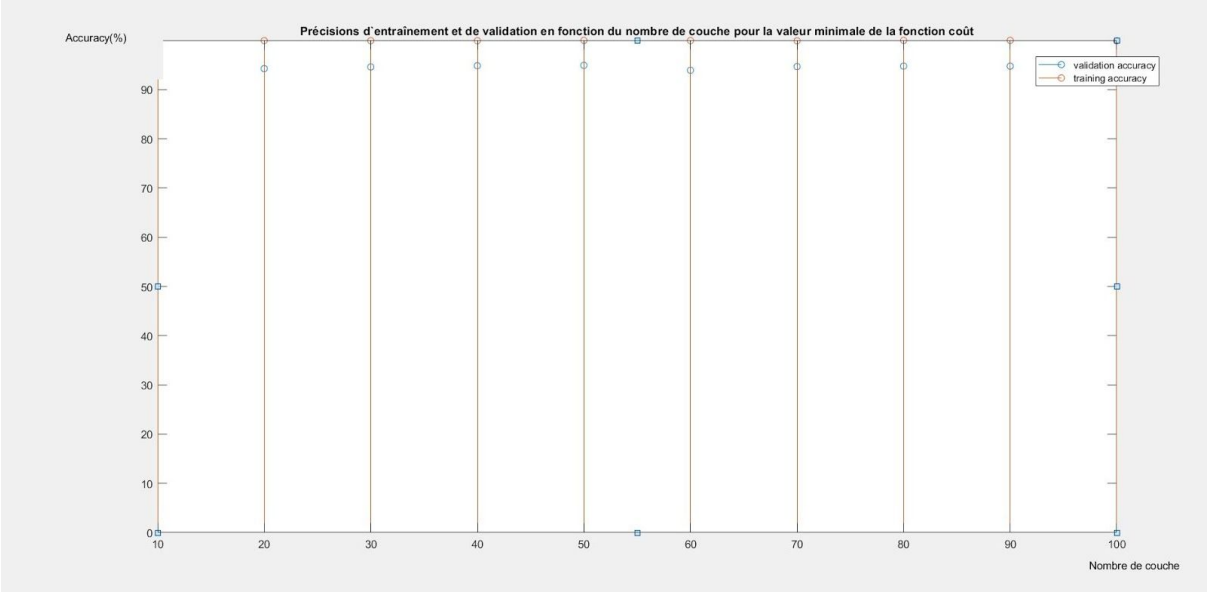

Figure 7 :Tracé de la précision d'entraînement et de validation pour une valeur minimale de de coût d'entraînement en fonction du nombre de neurone de la couche cachée.

Question 4.

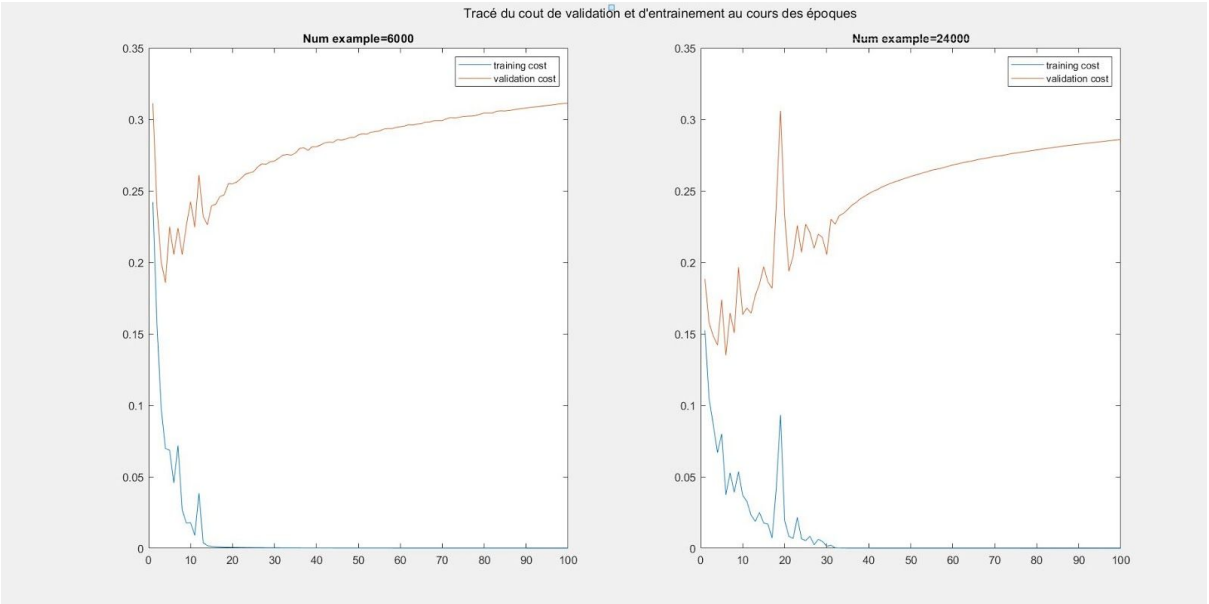

Figure 8 :Tracé de la précision d'entraînement et de validation pour 100 époques pour un nombre d'exemple de 6000 et 24000

Ici on voit bien pour un numéro d'exemple égale à 6000, nous avons un coût d'entraînement nul à partir de l'époque 18 et pour un numéro d'exemple égale à 24000 nous avons un coût d'entraînement nul à partir de la 31ème époque mais le coût de validation est plus faible. Cela est normal parce que nous avons plus de données d'entraînement donc plus de temps de calcul mais une meilleure d'avoir un meilleur coût de validation

Question 5.

Pour tester la précision de notre réseau personnalisé nous implémentons la fonction suivante qui va comparer la matrice de prédiction avec les données de sortie connues.

```
num_examples=10001;
X_test=X_data(:,50000:60000);
Y_test=Y_data(:,50000:60000);
Y_prediction=L_layers_nn.predict_mnist_a_completer(parameters, X_test);

default=0;
for c=1:10001

    for l=1:10

        if(Y_test(l,c)~=Y_prediction(l,c))
            default=default+1;                    %%erreur
        end
    end
end
    accy=((num_examples-default)/num_examples)*100
```

Figure 9 :Implémentation de la fonction accuracy qui permet de tester la précision sur les 10 000 derniers tests de la base de données.

Les données de tests correspondent aux dix mille derniers exemples de la base de données.
Avec un réseau à une couche caché de 105 neurones chacune, 45000 exemples et 17 époques, nous avons une précision de 95.98 %.
Nous ne sommes pas parvenus à augmenter ce taux de précision.

```
Command Window
Validation cost after epoch 17: 0.11025
Training accuracy after epoch 17: 99.975
Validation accuracy after epoch 17: 97.4333

accy =

   95.9804
```

Figure 10 : Précision pour un réseau à 1 couches cachées de **105 neurones**

III. IF2

La suite du projet se déroule sous Qt creator en C++. Nous aurons encore une fois à travailler avec un réseau de neurones, mais cette fois-ci, nous aurons à configurer tout d'abord l'IDE et gérer une chaîne de compilation sous Cmake. Après ce paramétrage nous pouvons commencer notre premier développement.

3.1Prise en main

Question 1.

1- Ajustement de la variable SRC_PATH

```
const string SRC_PATH=" D:\\Cours\\GE\\3GE\\S2\\CLANU\\IF2";
```

Cette valeur est ajustée de manière à indiquer à notre programme où se situent les fichiers de notre projet.

2. Nous passons le projet en mode Debug dans l'IDE Qtcreator

3.Nous exécutons le programme mnist_mlp_test. Pour cela il nous vous préciser en ligne de commande la lecture du fichier de réseau MLP_trained.bin

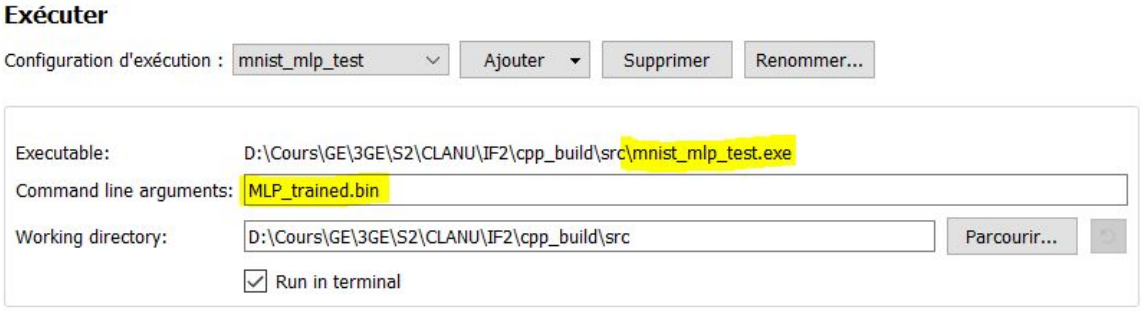

Figure 11:Précision en ligne de commande du MLP_trained.bin

Après avoir exécuté le code, nous retrouvons une précision de 94.27% ce qui est une bonne précision en comparaison de l'étude sous Matlab.

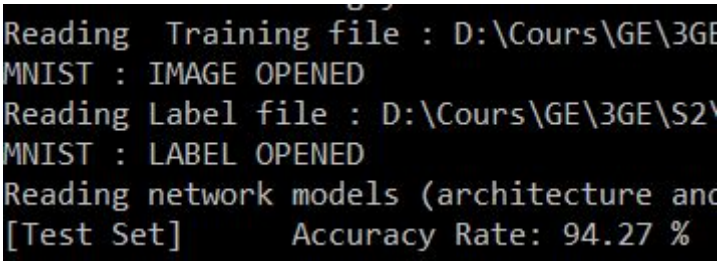

Figure 12: Précision sur le test

Après avoir passé le projet en mode Debug nous pouvons déterminer la structure interne de notre réseau de neurones. C'est-à-dire le nombre de couches et le nombre de neurones dont chacune est constituée.

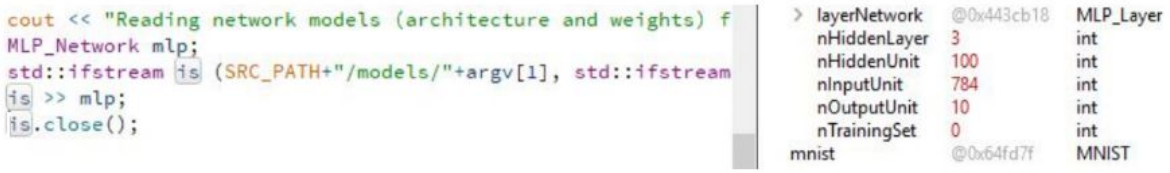

Figure 13: Paramètre du réseau de neurones

On voit bien ici que nous avons un réseau de neurones avec 5 couches (donc 3 couches cachées). La première couche dispose de **784 neurones**, les couches cachées quant à elles disposent de **100 neurones chacune** et la dernière dispose de **10 neurones**. On voit bien ici que l'on retrouve les couches d'entrée et de sortie présentées dans l'énoncé(784 et 10 neurones).

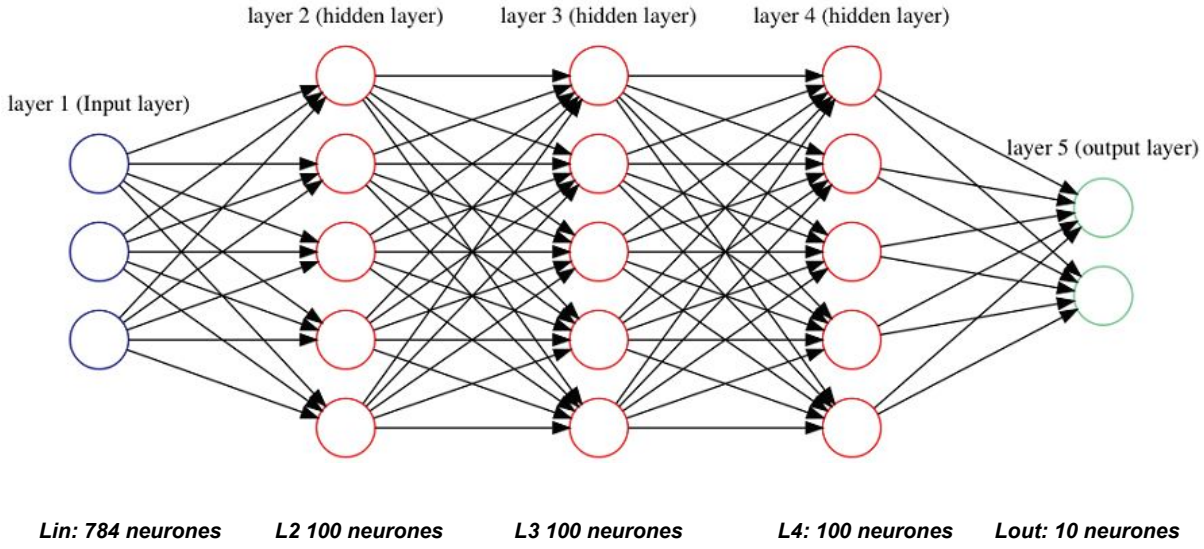

Figure 14: Structure du réseau de neurones

3.2 Premiers développement: voir les données de prédictions

Question 1.
Avant de continuer le développement, nous passons le projet en mode Debug

Question 2.
Nous avons implémenté une fonction qui permet d'afficher les valeurs d'une matrice(notre image) 28x28 en console. Elle prend en argument les dimensions de la matrice et l'indice de l'image.

```
void PrintImage(float *im, int r, int c)
{
// Question 3.2.2 complete this function


int j=0;        //Indice
    for(int i=0;i<r*c;i++)          // tant que je n'ai pas parcouru tous les 28*28
    {
                if(im[i]==1)            //si ==1 alors j'affiche un "X"
                   cout<<"X";
                if(im[i]==0)            //sinon j'affiche une "0"
                   cout<<" ";
            j++;
            if(j==c)                    //quand j'ai parcouru toutes les colonnes; je fais un retour chariot
            {cout<<""<<endl;
             j=0;                       //je réinitialise la variable pour l'affichage de la ligne suivante
            }
    }
}
```

Figure 15: implémentation de la fonction PrintImage

Le code suivant permet d'afficher les indices des images qui ont été mal classées par notre réseau. Elle se situe dans le fichier mnist_mlp_test.cpp.
Le programme va nous afficher une image en fonction de l'index que nous lui aurons passé en argument. Ensuite, il va nous indiquer si elle a été bien classée par notre réseau ou non.

Figure 16: image mal classifiée

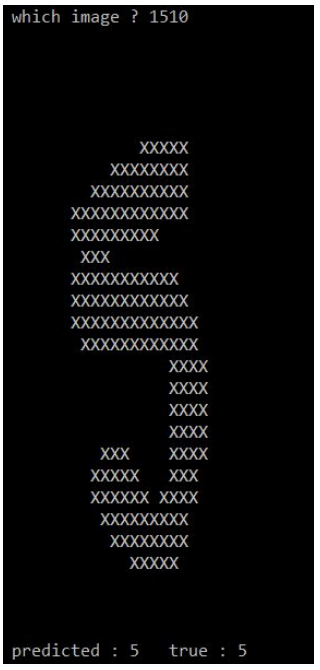

Figure 17: image bien classifiée

Ici, nous avons sélectionné l'image d'index 9944 et le réseau nous a prédit un 8 alors que c'était un 9 (cf. Figure 16).
En revanche, le label de l'image d'index 1510 est bien déterminé. Nous avons bien prédit un 5 pour cette image. (cf. Figure 17).
Nous implémentons aussi la fonction qui permet d'afficher toutes les images qui ont eu un label mal déterminé. Voici ci-dessous la fonction implémentée.

```
//(Question 3.2.3) ADD HERE THE CODE THAT PRINT ALL ERRONEOUS PREDICTED IMAGE INDEX

int m=0;

for(int i=0;i<nTestSet;i++)                                  //pour toutes mes images tests
{
    mlp.ForwardPropagateNetwork(inputTest[i]);                  //j'effectue ma propagation avant
    if(mlp.CalculateResult(inputTest[i],desiredOutputTest[i])==0.F)     //je regarde si les résultats
        {
         cout<<" "<<i;                                                  //si résultat faux alors j'affiche l'indice correspondant
            if(m==30)                                                   //si j'affiche 30 indices sur ligne je fais un retour
                {cout<<endl;                                            //chariot
                m=0;
                }
         m++;
        }
}
cout<<""<<endl;
```

Figure 18: Code permettant d'afficher l'indice d'une image qui est mal classée. Nous affichons au maximum 30 indices par lignes

Voici le résultat que nous obtenons en sortie.

```
[Test Set]      Accuracy Rate: 90.9 %   (compute time : 0.858429)
8 18 33 54 62 66 73 77 92 119 126 149 151 163 185 193 211 233 241 244 245 247 259 264 266 277 300 301 313 318 320
321 325 341 352 358 362 366 381 386 389 391 403 421 435 444 445 448 460 465 479 483 488 495 502 507 511 516 527 528 531
551 571 578 591 606 613 616 627 628 629 638 646 659 683 684 691 707 710 717 740 760 791 801 810 829 839 844 865 866 881
882 924 938 939 950 951 952 956 959 965 982 999 1000 1003 1012 1014 1023 1039 1044 1062 1068 1107 1112 1114 1116 1119 1124 1125 1142 1156
1157 1173 1181 1182 1191 1192 1198 1200 1204 1206 1217 1224 1226 1232 1234 1242 1247 1248 1256 1260 1268 1281 1282 1283 1299 1310 1315 1319 1320 1325
1326 1328 1337 1378 1383 1393 1402 1404 1444 1447 1463 1464 1465 1466 1494 1500 1522 1525 1527 1530 1531 1549 1553 1559 1562 1569 1571 1594 1601 1609
1611 1614 1621 1634 1663 1670 1671 1678 1681 1695 1709 1717 1721 1722 1732 1737 1751 1754 1759 1767 1772 1774 1790 1800 1811 1813 1816 1819 1828 1850
1857 1865 1874 1878 1901 1903 1917 1920 1938 1941 1952 1955 1956 1968 1970 1981 1984 2001 2016 2024 2025 2029 2040 2043 2044 2048 2052 2053 2054 2070
2073 2093 2098 2099 2105 2109 2118 2129 2130 2135 2145 2148 2149 2177 2182 2186 2189 2192 2198 2205 2237 2266 2272 2292 2293 2299 2300 2319 2325 2337
2351 2369 2371 2380 2381 2386 2387 2393 2395 2406 2422 2425 2426 2434 2447 2454 2460 2488 2496 2525 2526 2545 2559 2560 2573 2574 2582 2598 2604 2607
2610 2617 2635 2648 2654 2658 2670 2681 2684 2695 2698 2705 2713 2730 2751 2756 2760 2770 2771 2780 2797 2810 2818 2832 2834 2836 2850 2853 2857 2863
2866 2877 2896 2906 2915 2921 2925 2927 2939 2953 2970 2979 2986 2990 2995 3005 3030 3060 3062 3065 3073 3117 3130 3145 3157 3160 3167 3206 3216 3240
3260 3269 3275 3280 3284 3288 3289 3302 3319 3323 3329 3330 3336 3339 3347 3384 3388 3392 3405 3406 3410 3429 3436 3448 3468 3475 3490 3503 3520 3525
3533 3549 3550 3558 3559 3565 3567 3574 3587 3597 3599 3604 3618 3629 3664 3674 3681 3688 3702 3716 3718 3723 3726 3732 3752 3763 3767 3776 3778 3780
3796 3801 3806 3808 3811 3817 3818 3821 3836 3838 3839 3846 3848 3853 3855 3893 3902 3906 3926 3941 3946 3951 3954 3962 3967 3968 3976 3994 4000 4017
4027 4044 4072 4075 4078 4091 4093 4115 4131 4139 4140 4145 4154 4163 4176 4177 4199 4205 4207 4211 4212 4224 4228 4248 4255 4263 4265 4271 4289 4300
4302 4306 4313 4327 4341 4344 4355 4359 4374 4379 4380 4382 4400 4433 4435 4440 4451 4455 4465 4483 4497 4500 4504 4521 4523 4536 4540 4548 4566 4567
4571 4575 4578 4601 4605 4615 4616 4639 4640 4643 4671 4673 4681 4690 4692 4698 4722 4724 4731 4751 4761 4807 4823 4827 4829 4833 4837 4852 4860 4874
4876 4878 4879 4880 4886 4888 4890 4893 4896 4915 4928 4943 4950 4956 4963 4966 4978 4990 5001 5065 5067 5068 5078 5086 5100 5135 5138 5140 5165 5173
5176 5210 5278 5288 5331 5380 5409 5495 5522 5532 5569 5586 5600 5601 5611 5634 5642 5645 5647 5649 5678 5687 5726 5730 5734 5736 5746 5749 5757 5801
5835 5841 5842 5862 5877 5887 5888 5891 5912 5913 5922 5936 5937 5948 5955 5969 5973 5981 5982 5985 5986 5987 5988 5992 5997 6026 6034 6035 6037 6043
6045 6059 6065 6071 6081 6091 6093 6101 6109 6112 6124 6126 6139 6157 6166 6168 6172 6173 6324 6391 6392 6402 6414 6421 6425 6480 6495 6505 6517 6523
6542 6555 6564 6568 6571 6574 6576 6597 6598 6603 6610 6625 6627 6641 6642 6643 6651 6661 6688 6694 6706 6716 6721 6730 6740 6741 6744 6746 6755 6765
6768 6769 6775 6783 6784 6785 6791 6796 6847 6906 6914 6926 6945 6964 7035 7089 7094 7121 7153 7195 7208 7233 7235 7242 7249 7256 7265 7332 7338 7349
7376 7394 7432 7434 7451 7459 7472 7487 7491 7492 7498 7511 7514 7524 7565 7579 7580 7595 7603 7637 7641 7671 7720 7764 7779 7797 7800 7812 7821 7822
7826 7839 7842 7847 7856 7858 7859 7864 7869 7886 7888 7899 7900 7902 7905 7915 7916 7918 7920 7921 7928 7945 7951 8004 8010 8020 8035 8044 8072 8081
8091 8094 8095 8165 8183 8198 8244 8246 8272 8273 8277 8279 8288 8290 8293 8294 8296 8332 8339 8362 8383 8405 8406 8408 8410 8413 8453 8457 8469 8486
8493 8504 8509 8520 8522 8553 8597 8639 8669 8863 8941 8952 8954 9007 9009 9010 9013 9015 9016 9019 9022 9024 9026 9036 9045 9071 9112 9163 9182 9210
9211 9245 9268 9280 9317 9382 9422 9433 9446 9456 9465 9482 9492 9506 9530 9538 9539 9544 9564 9587 9595 9624 9634 9651 9698 9700 9712 9716 9719 9729
9735 9740 9741 9744 9745 9749 9764 9767 9768 9770 9779 9792 9808 9817 9832 9833 9839 9856 9867 9879 9881 9883 9888 9890 9891 9892 9893 9901 9905 9912
9925 9940 9943 9944 9953 9970 9975 9980 9982
```

Figure 19: Sortie de la console avec tous les indices des images mals classifiés.

3.3 Seconds développements: Entraînement, Sauvegarde et ressources

Question 1.
Dans le fichier MLP_Layer.h on retrouve la fonction d'activation appelée **ActivationFunction:** cf code ci-dessous:

```
//! calcul de la fonction d'activation
inline float ActivationFunction(float value)
    {
    if (activation_function == 'S')  { return 1.F/(1.F + (float)exp(-value));      }   // Sigmoid
    if (activation_function == 'R')  { if (value < 0) return 0; else return value;}   // ReLU
    else return 0;
    }
```

Figure 20:Fonctions d'activation

On remarque que nos fonctions d'activation correspondent à la sigmoïde et la fonction Relu. En fonction du paramètre du constructeur ('S' ou 'R') l'une ou l'autre sera utilisé. Par défaut, c'est la fonction sigmoïde qui est appelée.

Equation:

Relu(rectifier)

$f(x) = x \mathrel{+}= max(0, x)$

Sigmoïde

$\frac{1}{1+e^{-x}}$

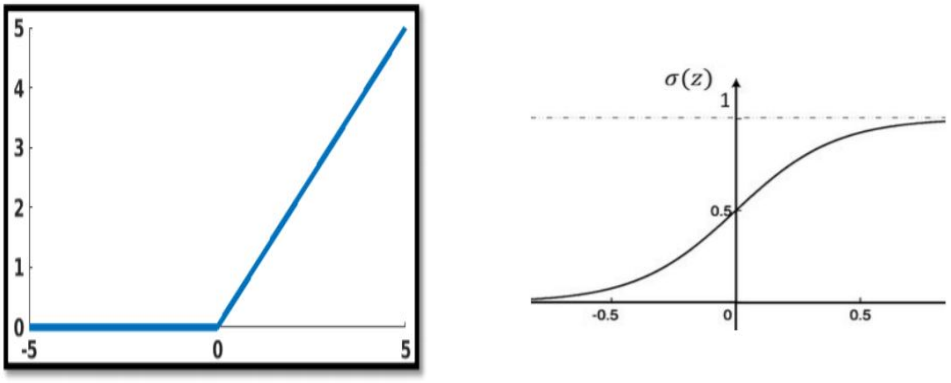

*Figure 21: Fonction de Rectifier et Sigmoïde-extrait de cours **"Réseaux de Neurones"** de Bernard Olivier*

Question 2.

Pour un réseau de neurones profond, la descente de gradient se déroule en trois étapes: La propagation avant, la propagation arrière et l'actualisation des paramètres.

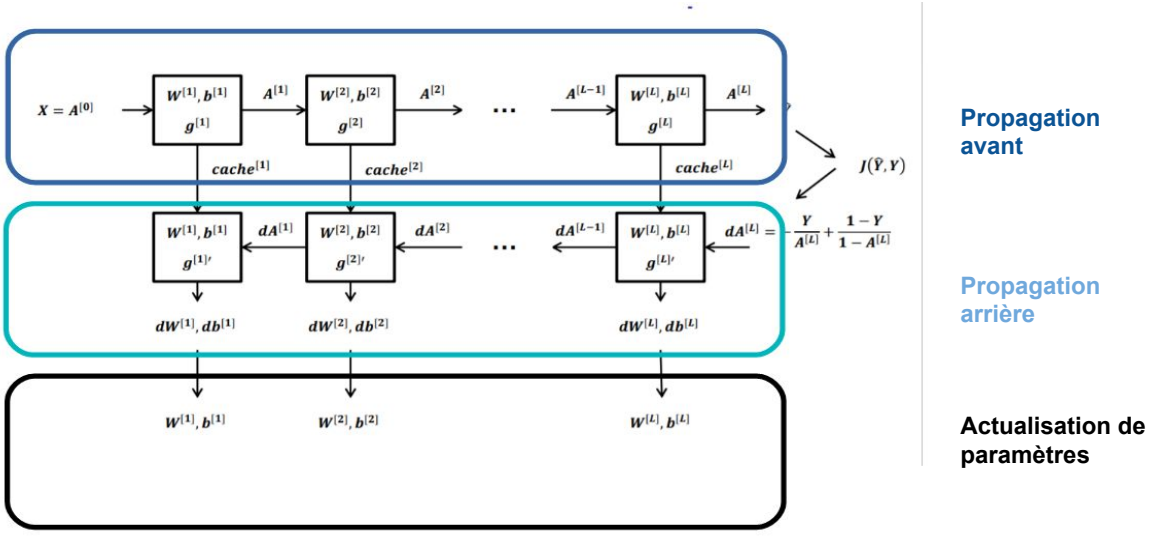

*Figure 22: différentes étapes au cours d'une descente de gradient-extrait de cours **"Réseaux de Neurones"** de Bernard Olivier*

Avec W, la matrice de poids de la couche considérée, b le vecteur des bias et g la fonction d'activation.

La descente de gradient s'effectue selon l'expression mathématique.

$$W = W - \propto \frac{\eth J(W,B}{\eth W}$$

$$B = B - \propto \frac{\eth J(W,B}{\eth B}$$

Il s'agit d'une rétropropagation de gradient. Nous calculons le gradient de l'erreur pour chaque neurone de chacune des couches(de la dernière vers la première) pour chaque neurone.

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$$cache^{[l]}$$

$$W^{[l]}, b^{[l]}, Z^{[l]}, A^{[l-1]}$$

$$dZ^{[l]} = dA^{[l]} \cdot * g^{[l]\prime}(Z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} sum(dZ^{[l]}, 2)$$

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

*-extrait de cours **"Réseaux de Neurones"** de Bernard Olivier*

Question 3.

Le taux d'entraînement correspond au pas de la descente de gradient. Durant les premières époques, il a une très grande valeur et au cours des itérations il diminue. Lorsque nous sommes loin d'avoir minimisé notre fonction(au cours des premières itérations), il a une grande valeur et au fur et à mesure que nous la minimisons il va diminuer.

Voici ci-dessous une illustration

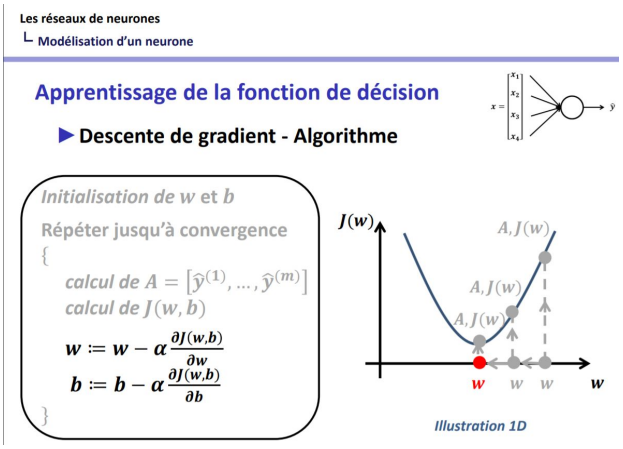

*Figure 23: Principe de minimisation d'une fonction en 1D -extrait de cours **"Réseaux de Neurones"** de Bernard Olivier*

Si nous étions en 1 dimension, le learning rate correspondrait à l'alpha. On voit bien que s'il est trop grand nous minimiserons notre fonction rapidement, mais au risque de ne jamais converger et osciller. Si nous le prenons trop petit, un algorithme qui utilise un seuil mettra beaucoup de temps à exécuter toutes les itérations. Pour notre programme, le principe est le même. Après chaque étape nous diminuons le learning rate selon la suite.

$$x_{n+1} = \frac{x0}{1+n.xn}$$

x0: le learning rate initial
n: l'époque

```
learningRate = initialLR/(1+epoch*learningRate);    // learning rate progressive decay
```

:ligne de commande faisant diminuer progressivement le taux d'apprentissage

Question 4.
Nous effectuons la même manipulation que lors du premier développement, mais cette fois-ci en indiquant que nous travaillerons avec le fichier de training.

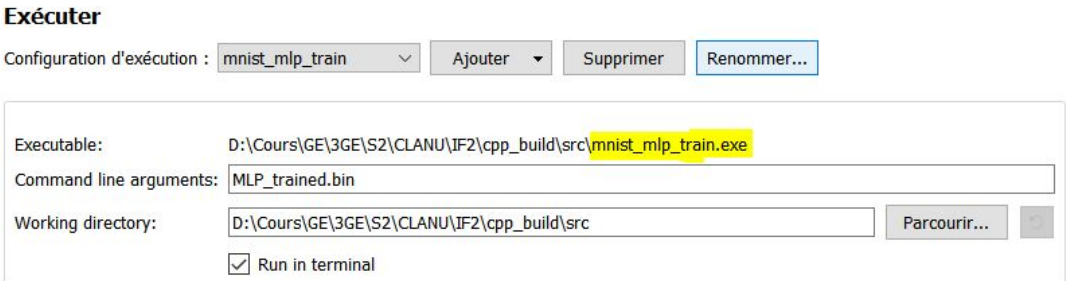

Figure 24: choix de l'exécutable dans l'IDE Qt creator

Question 5.
Nous compilons le projet en mode Release pour une version optimisée et parallèle.

Question 6.
Le nom du fichier de sauvegarde est toujours MLP_train.bin.

Après étude du programme, nous remarquons que le réseau utilise une couche d'entrée de 784 neurones, deux couches cachées de 20 neurones et une couche de sortie de 10 neurones.
Ceci implique;
-que la première couche cachée dispose de 784*20+20 paramètres
-que la Seconde couche cachée dispose de 20*20+20 paramètres aussi-enfin que la couche de sortie dispose de 10*20+10 paramètres
Nous avons donc au total 16330 paramètres.

Ces paramètres sont codés sous forme de float qui à une taille de **4 octets**. L'espace total alloué en mémoire est donc 4*16330 octets ce qui équivaut à **65 320 octet**. Cet ordre de grandeur est normal pour un réseau de neurones de cette taille **(de l'ordre de 65 ko)**

Question 7.

Comparaison de performance

	PC1	PC2	PC3
Processeurs	Core(TM) i7-4770 CPU : 3.40Ghz	Core(TM) i7-4770 CPU : 3.40Ghz	Core(TM) i5-7300 CPU : 2.50Ghz
Mémoire RAM installée	8GB	8.05 GB	8GB
Système Exploitation	Windows 10	Linux	Windows 10
Compilateur	MinGW	MinGW	MinGW
Temps pour l'apprentissage	0.120s	0.069	3.6s

Tableau 2: Comparaison de performances sur deux PC différents

On voit le pour des caractéristiques quasiment équivalent, un ordinateur tournant sous linux exécute le code plus rapidement qu'un ordinateur tournant sous windows 10. Le programme est 1.7 fois plus rapide sous Linux.

Troisième développement: optimisation et observations lors de l'entraînement

Question 1.
Dans cette question, nous cherchons à afficher la précision sur le jeu d'entraînement et sur le jeu de test. Nous tracerons ensuite au cours des époques l'évolution de la précision pour le jeu d'apprentissage et le jeu de test.

Pour cela, nous implémentons nos fonctions d'affichages dans la boucle qui dépend de l'époque dans laquelle nous sommes. Voici ci-dessous le code:

```
for( int i=0; i<nTestSet; i++)
    {
     mlp.ForwardPropagateNetwork(inputTest[i]);
     sums += mlp.CalculateResult(inputTest[i], desiredOutputTest[i]);
    }
accuracyRate = (sums / (float)nTestSet) * 100;
cout << "[TestSet]\t"<<"Accuracy Rate: " << accuracyRate << " %"<<endl;
```

Figure 25: Code permettant d'afficher la précision sur le jeu de test au cours des époques

Il suffit de répéter la même action pour le jeu d'entraînement en modifiant le nom des variables.

On a donc:
accuracyRate = (sums / (float)nTestSet) * 100;
accuracyRate = (sums / (float)nTrainingSet) * 100;

Ces deux lignes de codes sont implémentées dans deux boucles for, elles-même implémentées dans la boucle d'entraînement.
Nous les traçons ensuite en fonction des époques.

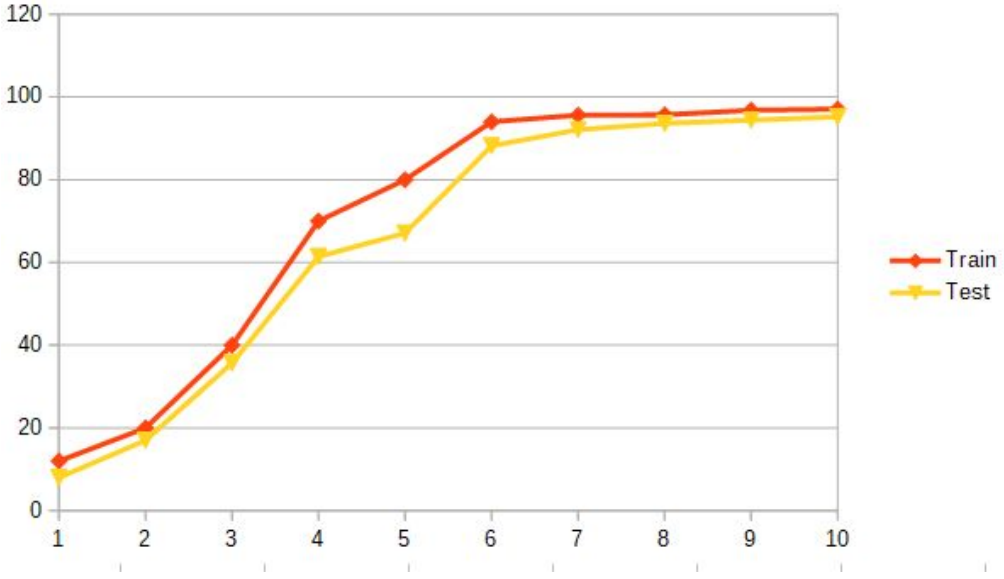

Figure 26: Evolution du coût et du taux d'entrainement

Question 2.

Le but du développement suivant est de sauvegarder notre réseau de neurones au cours des époques de manière à garder celui qui a les meilleures performances. A chaque fois que notre réseau s'améliore, nous le sauvegardons dans un fichier bin présent dans le dossier models (voir page suivante).

```
// Training Result
float TestaccuracyRate=0;
sums=0;
accuracyRate=0.F;
for( int i=0; i<nTestSet; i++)
    {
     mlp.ForwardPropagateNetwork(inputTest[i]);
     sums += mlp.CalculateResult(inputTest[i], desiredOutputTest[i]);
    }
accuracyRate = (sums / (float)nTestSet) * 100;                                    use of old-sty
if(TestaccuracyRate<accuracyRate)
{TestaccuracyRate=accuracyRate;
            string tmp_filenmae = rawname + "_" + to_string(epoch) + "_" + to_string(TestaccuracyRate) + ".bin";
            std::ofstream os(tmp_filenmae, std::ofstream::binary);
            os<< mlp;
            os.close();
 }
```

Figure 27: Sauvegarde du réseau dans un fichier bin

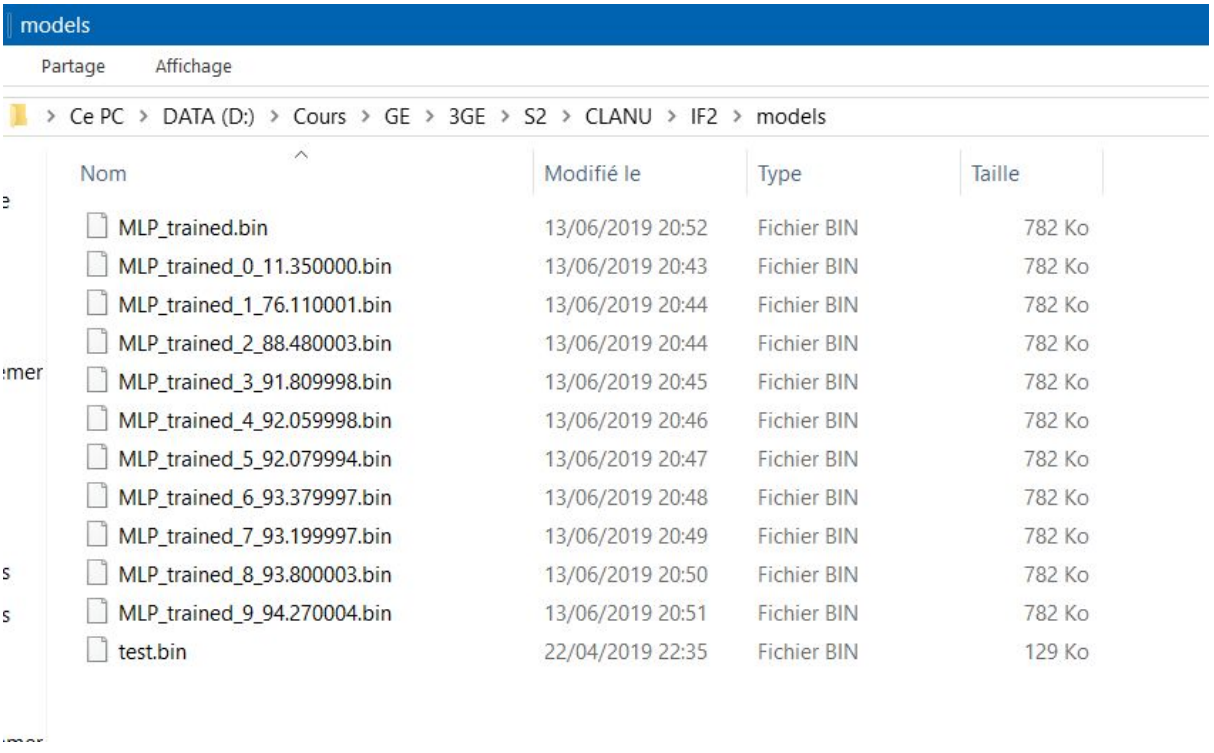

Figure 28: Ensemble des réseau sauvegarder pour 10 époques

Voici l'ensemble des réseaux sauvegarder pour 10 époques.

Question 3.

On voit qu'à un certain moment, plus aucun réseau n'est sauvegardé car nous avons atteint la précision maximale que peut fournir notre réseau.. Après cela il n'est pas nécessaire de continuer à sauvegarder des réseaux de neurones car notre réseau est bien entraîné.

Par exemple pour un réseau avec à **3 couches cachées** chacunes de 3**0 neurones** et **30 époques** le programme va continuer à sauvegarder des réseaux de neurones jusqu'à son terme. Cela veut dire que la valeur maximale de la précision n'est jamais dépassée pour les époques suivantes.

En revanche, si l'on prend un réseau de neurones à 2 couches de 30 neurones chacune avec 50 époques nous arrêtons de sauvegarder les données au bout de 30 époques. Après cela, les taux de précisions ne sont pas plus élevés donc inutile de continuer à sauvegarder les réseaux.

Question 4.

Nous implémentons notre réseau de neurones en C avec les paramètres trouvés à la question 5 de la partie analyse numérique. Nous avions un réseau à **1 couche**, **105 neurones** et **17 époques**. Nous avions un résultat de 95.98%. En implémentant ces même paramètres sous C++, nous obtenons une précision de **96.99%**.

On voit bien que le résultat est meilleur sous C++ que sous Matlab.

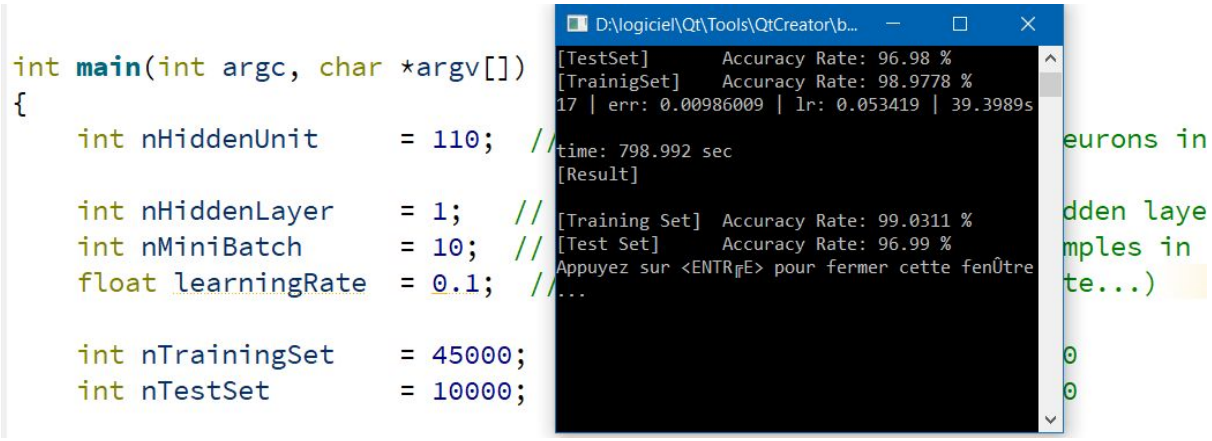

Figure 29: Test en c++ avec les paramètres optimaux trouvés sous Matlab.

	Matlab	C++
Couches cachées	**1**	**1**
Neurones	**105**	**105**
Epoques	**17**	**17**
Précision de test	**95.98%**	**96.99%**

Tableau 3: Comparaison de performance sous C++ et Matlab

Question 5.

Nous allons optimiser notre réseau de neurones pour avoir les meilleurs paramètres qui nous donnerons une meilleure précision de test. Nous allons jouer sur le nombre de couche cachées, le nombre de neurones et le nombre d'époques.

Après de nombreux tests nous retenons les paramètres pour un nombre de **45000 exemples**, de **110 neurones** et de 1**7 époques**. Nous obtenons une précision sur le jeu de test de **97.02 %**

Caractéristique du réseau optimal

	Paramètre
Couches cachées	**1**
Neurones	**110**
Epoques	**17**
Nombre d'hyper paramètres	784*110+110+10*110+10**=87 460**
Mémoire à allouer	87 460*8=**349 ko**
Précision de test	**97.02%**

```
int main(int argc, char *argv[])
{
    int nHiddenUnit     = 110;   /
    int nHiddenLayer    = 1;     //
    int nMiniBatch      = 10;    //
    float learningRate  = 0.1;   /
    int nTrainingSet    = 45000;
    int nTestSet        = 10000;
    float errMinimum = 0.01; // v
```

```
D:\logiciel\Qt\Tools\QtCreator\bin\qtcreator_process_stub.e...
[TrainigSet]    Accuracy Rate: 98.4089 %
11 | err: 0.0140763 | lr: 0.0613195 | 39.3444s
[TestSet]       Accuracy Rate: 96.87 %
[TrainigSet]    Accuracy Rate: 98.56 %
12 | err: 0.0131504 | lr: 0.0597188 | 41.3168s
[TestSet]       Accuracy Rate: 96.88 %
[TrainigSet]    Accuracy Rate: 98.6667 %
13 | err: 0.0123396 | lr: 0.0582538 | 40.6673s
[TestSet]       Accuracy Rate: 96.9 %
[TrainigSet]    Accuracy Rate: 98.7444 %
14 | err: 0.0116175 | lr: 0.0569055 | 39.669s
[TestSet]       Accuracy Rate: 96.96 %
[TrainigSet]    Accuracy Rate: 98.8044 %
15 | err: 0.0109732 | lr: 0.0556583 | 41.0108s
[TestSet]       Accuracy Rate: 96.98 %
[TrainigSet]    Accuracy Rate: 98.88 %
16 | err: 0.0103995 | lr: 0.0544996 | 38.8295s
[TestSet]       Accuracy Rate: 97.02 %
[TrainigSet]    Accuracy Rate: 98.9244 %
```

IV Conclusion Technique

Tout au long de notre étude nous avons pu remarquer que la qualité d'un réseau de neurones ne dépend pas de sa taille ni du nombre de paramètre qu'il contient. Ces derniers varient très fortement en fonctions du nombre et de type de données que nous lui fournissons.

Nous particulièrement apprécié la comparaison des deux PC qui nous a permis de voir que les performances de notre programme peuvent être affectés en fonction du hardware, chose que nous négligeons un peu trop par moment. De plus la comparaison entre matlab et le c a permis de se rendre compte la différence du temps d'exécution entre les langage interprété, comme MATLAB, et les langage compilé, comme C.

V. Conclusion globale du projet et organisation

Nous considérons que ce projet a été très formateur. Il est indispensable pour un ingénieur d'acquérir des notions en réseaux de neurones à l'heure où leur utilisation est de plus en plus courante dans l'industrie. Du secteur de l'énergie à l'aéronautique en passant par la robotique. Ce projet nous a permis de plonger encore plus dans les réseaux de neurones et mettre en applications les notions acquises lors du premier semestre dans le module AN.
De plus, nous ne faisons pas souvent le lien, mais l'organisation a été un point déterminant dans la réalisation du projet.
Nous avons décidé de diviser ce projet en trois parties pour pouvoir livrer à temps ce rapport. La première phase du projet était la compréhension du sujet et la familiarisation avec les outils de développement. La seconde consistait à développer l'application et implémenter les lignes de codes. La troisième et dernière phase quant à elle était la rédaction du rapport. La première phase s'est bien sûr faite à deux pour avoir une compréhension commune du projet. Nous avons pu échanger et lever tout questionnement vis-à-vis de la structure globale du projet.
Pour la seconde et troisième phase, nous avons décidé de les réaliser en même temps. Nous avons alloué des créneaux dans notre emploi du temps pour réaliser le projet. Pour avancer plus rapidement, nous avons décidé de travailler tous les deux sur chacun des thématiques au même moment. Pendant que l'un codait, l'autre rédigeait le rapport et commentait les résultats. Nous alternions souvent les rôles de manière diversifier les tâches de travails de cette manière nous pouvions constamment échanger pour avancer et avoir une vision claire de chacun des éléments du projet.
Nous avons dès le début écarté l'idée de travailler chacun sur une des parties du projet en coupant les tâches en deux. Nous considérons que cette méthode n'est pas efficace et ne permet pas d'avancer plus vite et ne favorise pas l'échange entre nous deux.
Pour conclure ce rapport, nous avons pris du plaisir à réaliser ce projet et nous sommes convaincus que nous serons amenés à réutiliser les réseaux de neurones dans nos futures carrières d'ingénieur