french

Cahier des spÃ©cifications
Sujet 10 : RÃ©seau de Neurones

Thibaut Pepin
Soumia Rezgui
Isaac Szulek
Severine Selaquet
Anthony Montigne
Arezki Slimani

18 avril 2018

Table des matières

1 Introduction

Nous souhaitons analyser des images, un problÃ¨me compliquÃ© qui a besoin de prendre en entrÃ©e une image et dont le but est d'essayer de deviner ce que reprÃ©sente cette image. Pour cela il nous faut une structure qui sera capable de prendre beaucoup de donnÃ©es en entrÃ©e et de capturer des relations complexes entres les entrÃ©es et les sorties,c'est lÃ qu'interviennent les rÃ©seaux de neurones artificiels. Ainsi, afin d'indiquer comment rÃ©aliser le besoin defini par le cahier des charges, il est necessaire de reprendre les besoins du maitre d'ouvrage, en les exprimant cette fois par les maÃ®tres d'oeuvres. En ce sens , nous avons donc rÃ©aliser une description dÃ©taillÃ©e pour notre conception technique. Le document suivant est ainsi la dÃ©finition Ã©crite, en terme de fonctionalitÃ© et de performance, du cahier des charges prÃ©alablement Ã©tablis.

2 Structures de DonnÃ©es

2.1 RÃ©seau de neurones

Pour le choix de notre structure de donnÃ©es nous avons rapidement adoptÃ© la reprÃ©sentation d'un rÃ©seau de neurones comme un ensemble de couches et non comme un ensemble de neurones individuels. Chaque couche doit donc contenir l'ensemble des activations, des biais et des poids des neurones de cette couche. On stocke les activations et les biais dans des vecteurs et les poids dans des matrices. Nous les avons integrÃ© dans une structure couche comme tableau Ã une ou deux dimensions (car les calculs a effectuer necessite d'accÃ©der aux Ã©lÃ©ments sans ordre particulier). Afin de stocker l'ensemble des couches constituant le rÃ©seau de neurone, nous avons optÃ© pour une liste doublement chainÃ©e. En effet, les deux algorithmes necessitant d'accÃ©der aux contenus des couches vont propager des informations de la premiere couche Ã la derniÃ¨re couche pour l'algorithme de propagation. Et de la derniÃ¨re couche Ã la premiÃ¨re couche pour l'algorithme de rÃ©tro-propagation (d'oÃ¹ $le double chainage$). $A finder egrouper les informations g©n©rales d' un r©seau de neurones, nous avons cr©e une petite structu$

struct COUCHE //structure reprÃ©sentant une couche (ensemble de neurones) dans un rÃ©seau de neurone
{
    float* A ; //tableau reprÃ©sentant le vecteur des activation des neurones d'une couche. PossÃ¨de autant d'Ã©lements que de neurones prÃ©sent dans la couche.
    float* B ; //tableau reprÃ©sentant le vecteur des biais des neurones d'une couche de mÃªme taille que le tableau A.
    float** W ; //tableau Ã deux dimensions reprÃ©sentant la matrice des poids entre les neurones de la couche prÃ©cedente et les neurones de cette couche. Ce tableau est donc de taille (taille de la couche actuelle * taille de la couche prÃ©cedente) et est composÃ© d'Ã©lements w_{ij} oÃ¹ $j est le num©ro de la couche pr©cedente et i le num©ro de la couche actuelle$.
    int taille ; //Indique le nombre de neurones prÃ©sent dans cette couche
    float DELTA ; //tableau reprÃ©sentant le vecteur des modifications Ã apporter aux biais de cette couche lors de la rÃ©tro-propagation des neurones d'une couche de mÃªme taille que le tableau A.
    float** DELTA_M ; //tableau Ã deux dimensions reprÃ©sentant la matrice des modifications Ã apporter aux poids de cette couche lors de la rÃ©tro-propagation des neurones d'une couche de mÃªme taille que la matrice W

    struct COUCHE* prec ; //pointeur sur la couche prÃ©cÃ©dente.
    struct COUCHE* suiv ; //pointeur sur la couche suivante.
} ;

typedef struct COUCHE COUCHE ;
typedef COUCHE* Liste_couche ;

struct INFO_RN //structure reprÃ©sentant les diffÃ©rentes informations caractÃ©risant un rÃ©seau de neurones.
{

```
        char** etiquettes ; //tableau de chaine de caractÃ¨re comprenant la signification des neurones de sortie.
        char* nom ; //nom donnÃ© au rÃ©seau de neurones.
        char* date ; //date de crÃ©ation du rÃ©seau de neurones.
        int reussite ; //nombre de fois ou le rÃ©seau de neurones a obtnenu la rÃ©ponse attendue lors de l'apprentissage.
        int echec ; //nombre de fois ou le rÃ©seau de neurones n'a pas obtnenu la rÃ©ponse attendue lors de l'apprentissage.
};

typedef struct INFO_RN INFO_RN ;

struct RN //structure reprÃ©sentant le rÃ©seau de neurones, contenant l'ensemble des couches et les informations du rÃ©seau de neurones.
{
        INFO_RN info ; //les informations du rÃ©seau de neurones.
        Liste_couche couche_deb ; //pointeur sur la 1Ã¨re couche du RN.
        Liste_couche couche_fin ; //pointeur sur la derniÃ¨re couche du RN.
};

typedef struct RN RN ;
```

2.2 Image

Afin de reprÃ©senter une image, nous avons optÃ© pour un tableau Ã une dimension de pixel afin de se rapprocher de la structure des vecteurs d'activations. Chaque pixel Ã©tant une structure contenant la quantitÃ© de rouge, de vert et de bleu prÃ©sent dans un nombre entre 0 et 255 d'oÃ¹ *letypecharquiconvientparfaitementpourdesvaleursdanscetintervalle.*

```
typedef struct Pixel
{
        unsigned char r,g,b ;
} Pixel ;

typedef struct Image
{
        int w,h ;
        Pixel* dat ;
} Image ;
```

2.3 Apprentissage

Le couple d'information sortie attendue et donnÃ©e d'entrÃ©e Ã©tant nÃ©cessaire lors de l'apprentissage. Nous les avons regroupÃ© dans une petite structure.

```
typedef struct Apprentissage
{
        Image* image ;
        char* etiquette ;
} App ;
```

3 Signatures des Fonctions

3.1 Gestionnaires d'apprentissage

3.1.1 void BackProp(RN*, App*, float)

Backprop va effectuer l'algorithme de propagation-inverse puis va modifier les poids et les biais du rÃ©seau de neurones passÃ© en paramÃ¨tre. Pour cela il est nÃ©cessaire d'avoir le couple donnÃ©e d'entrÃ©e et sortie attendue prÃ©sent ici dans la structure App*, on effectuera donc la propagation sur l'image et on comparera l'Ã©tiquette de sortie avec le char* qui est l'Ã©tiquette attendue.

3.1.2 void SigmoidePrimeZ(float* in, float** out, int taille)

SigmoidePrimeZ va rÃ©cupÃ©rer les Ã©lÃ©ments stockÃ©s dans le premier tableau contenant 'taille' Ã©lÃ©ments puis effectuer l'opÃ©ration $x * (1 - x)$ sur chaque Ã©lÃ©ments avant de les Ã©crire dans le deuxiÃ¨me tableau passÃ© en paramÃ¨tre. Cette opÃ©ration n'est pas la dÃ©rivÃ©e de la fonction sigmoide, il s'agit d'une petite optimisation qui nous permet de trouver le mÃªme rÃ©sultat plus rapidement. En effet lors de la propagation inverse il nous est necessaire d'effectuer l'opÃ©ration :
$\sigma'(z) = \sigma(z) * (1 - \sigma(z))$
ou z vient de :
$a^L = \sigma(w^L * a^{L-1} + b^L) = \sigma(z^L)$
or z n'est pas stockÃ© dans notre structure et on peut de toute facon simplifier par :
$\sigma'(z) = a * (1 - a)$
d'ou cette fonction.
Le deuxiÃ¨me tableau est a deux dimension car DELTA_M est disponible au moment ou cette opÃ©ration est effectuÃ©e, on va donc l'utiliser plutÃ´t que de crÃ©er un autre tableau pour stocker le rÃ©sultat.

3.1.3 void MultiplicationMatricielleTransposeeTM(float**, float*, float*, int, int)
void MultiplicationMatricielleTransposeeMT(float*, float*, float**, int, int)

Lors de la retro-propagation certaines matrices doivent Ãªtre transposÃ©es, ce qui est pÃ©nible Ã effectuer. Cependant, ces matrices transposÃ©es sont toujours multipliÃ©es par une autre matrice ce qui nous donne deux cas possibles :
$A^T * B$ ou $B * A^T$
MultiplicationMatricielleTransposeeTM et MultiplicationMatricielleTransposeeMT reprÃ©sentent ces deux cas mais ne transposent pas de matrice, elle effectue un produit matricielle oÃ¹ $lesmatricesnesontpasluesdanslesensconventi$

3.1.4 void Hadamard(float**, float*, float*, int)

Hadamard effectue le produit vectoriel de Hadamard sur les deux premiers tableaux pour stocker le rÃ©sultat dans le troisiÃ¨me tableau. Tous ces tableaux Ã©tant de mÃªme taille, celle ci est passÃ©e en paramÃ¨tre.

3.1.5 void fct_cout(RN, char*)

fct_cout va calculer l'erreur entre le rÃ©sultat obtenu lors de la propagation et le rÃ©sultat attendu, soit le char*, pour chacun des neurones de la derniÃ¨re couche du rÃ©seau de neurones puis va stocker le rÃ©sultat dans le tableau DELTA de la derniÃ¨re couche du rÃ©seau.

3.1.6 void ModifPoids(float**, float**, int, int, int eta)
void ModifBiais(float*, float*, int, int eta)

ModifPoids et ModifBiais vont modifier le premier tableau passÃ© en paramÃ¨tre en lui soustrayant le deuxiÃ¨me tableau multipliÃ© par la vitesse d'apprentissage 'eta', les autres paramÃ¨tres sont les tailles des tableaux.

3.2 Gestionnaire d'entrÃ©es sorties

3.2.1 Image* ChargerBmp(const char* fichier, int, int)
Image* ChargerMnist(const char* fichier, int, int)

ChargerBmp et ChargerMnist vont lire Ã l'emplacement donnÃ© en paramÃ¨tre et renvoyer le contenu du fichier sous forme de la structure Image, si celui ci contient une image au format bmp non compressÃ© ou au mÃªme format que celui utilisÃ© par la base de donnÃ©es MNIST. Le fichier est ensuite supprimÃ© ou partiellement effacÃ©. Les deux entiers correspondent Ã la largeur et la hauteur maximale de l'image acceptÃ©e par le rÃ©seau de neurones dÃ©fini par l'utilisateur.

3.2.2 int Sauver(Image*,const char* fichier)

Sauver va enregistrer l'image passÃ©e en paramÃ¨tre Ã l'emplacement lui aussi passÃ© en paramÃ¨tre au format bmp.

3.2.3 Image* NouvelleImage(int w,int h)

NouvelleImage alloue la mÃ©moire nÃ©cessaire pour stocker une image dont la taille est passÃ©e en paramÃ¨tre dans une structure Image puis renvoie l'adresse de l'image crÃ©Ã©e.

3.2.4 Image* CopieImage(Image*)

CopieImage crÃ©e une copie de l'image passÃ©e en paramÃ¨tre puis renvoie son addresse.

3.2.5 void SetPixel(Image*,int i,int j,Pixel p)

SetPixel modifie le pixel aux coordonnÃ©es i*j de l'image passÃ©e en paramÃ¨tre afin de correspondre au pixel p.

3.2.6 Pixel GetPixel(Image*,int i,int j)

GetPixel renvoie le pixel aux coordonnÃ©es i*j de l'image passÃ©e en paramÃ¨tre.

3.2.7 void DelImage(Image*)

DelImage libÃ¨re la mÃ©moire d'une variable de type Image.

3.2.8 char* ChargerEtiquetteMNIST(const char* fichier)

ChargerEtiquetteMNIST va rÃ©cupÃ©rer la derniÃ¨re Ã©tiquette prÃ©sente dans le fichier passÃ© en paramÃ¨tre puis va supprimer celle-ci du fichier ou supprimer le fichier si celui ci ne contient plus aucune Ã©tiquettes.

3.2.9 App* ChargementCoupleAttIn(char* repertoire, int, int)

ChargementCoupleAttIn va rechercher dans le rÃ©pertoire le premier couple donnÃ©e d'entrÃ©e et Ã©tiquette prÃ©sent tout en supprimant tout fichier au mauvais format ou non reconnu. Les deux entiers correspondent Ã la largeur et la hauteur maximale de l'image acceptÃ©e par le rÃ©seau de neurones dÃ©finie par l'utilisateur.

3.2.10 INFO_RN* ChargerInfo()

ChargerInfo rÃ©cupÃ¨re les structures INFO_RN de tous les rÃ©seaux de neurones enrigistrÃ©s dans le repertoire ../sav/ et le retourne sous forme de tableau.

3.2.11 RN* ChargerRN(INFO_RN info)

ChargerRN initialise et remplit un rÃ©seau de neurones dont les informations sont Ã l'emplacement passÃ© en paramÃ¨tre avant de renvoyer l'addresse de celui-ci.

3.2.12 void SaveRN(RN)

SaveRN va crÃ©er ou modifier tous les fichiers nÃ©cessaires afin de sauvegarder le rÃ©seau de neurones passÃ© en paramÃ¨tre.

3.3 Gestionnaire du rÃ©seau de neurones

3.3.1 RN* Initialisation(INFO_RN)

Initialise un nouveau rÃ©seau de neurones Ã partir des informations renseignÃ©es par l'utilisateur prÃ©sent dans la structure INFO_RN passÃ©e en paramÃ¨tre.

3.3.2 void AjoutCoucheFin(RN, int)

Ajoute une nouvelle couche Ã la liste doublement chaÃ®nÃ©e reprÃ©sentant le rÃ©seau de neurones. L'entier passÃ© en paramÃ¨tre est le nombre de neurones de cette couche.

3.3.3 void AjoutPremiereCouche(RN, int)

Ajoute la premiere couche de la liste doublement chaÃ®nÃ©e reprÃ©sentant le rÃ©seau de neurones. L'entier passÃ© en paramÃ¨tre est le nombre de neurones de cette couche.

3.3.4 void Propagation(Image*, RN)

Propagation va rÃ©cupÃ©rer l'image passÃ©e en paramÃ¨tre et va l'enregistrer dans le tableau des activations de la premiÃ¨re couche du rÃ©seau de neurones. Afin de propager ces informations jusqu'a la derniÃ¨re couche, la formule suivante va Ãªtre appliquÃ©e sur chacune des couches :
$A^j = \sigma(W^j * A^{j-1} + B^j)$
avec j le numÃ©ro de la couche.

3.3.5 void Remplissage(RN)

Attribution de valeurs alÃ©atoires aux biais et aux poids synaptiques du reseau de neurones passÃ© en argument.

3.3.6 char** Reconnaissance(RN)

Retourne les Ã©tiquettes des 3 neurones de la derniÃ¨re couche du rÃ©seau possÃ©dant les activations les plus Ã©levÃ©es sous la forme d'un tableau triÃ© par activation dÃ©croissante.

3.3.7 void MultiplicationMatriceVecteur(float**, float*, float*, int, int)

Effectue la multiplication de la matrice et du vecteur passÃ©s avec en paramÃ¨tres puis enregistre le rÃ©sultat dans le troisiÃ¨me tableau. Les deux entiers sont les tailles de la matrice et du vecteur.

3.3.8 void AdditionVecteurVecteur(float*, float*, float*, int)

Effectue l'addition des deux vecteurs passÃ©s en paramÃ¨tre avec en paramÃ¨tres , puis enregistre le rÃ©sultat dans le troisiÃ¨me tableau. L'entier est la taille des deux vecteurs.

3.3.9 void SigmoideV(float*, float*, int)

SigmoideV applique la fonction sigmoide sur chacun des Ã©lÃ©ments du premier tableau passÃ© en paramÃ¨tre puis enregistre le rÃ©sultat dans le second tableau, l'entier Ã©tant la taille de ces tableaux.

3.3.10 float Sigmoide(float x)

Sigmoide calcule la sigmoide du nombre passÃ© en paramÃ¨tre, soit :
$\sigma(x) = \frac{1}{1+e^{-x}}$

3.3.11 void LibererRN(RN*)

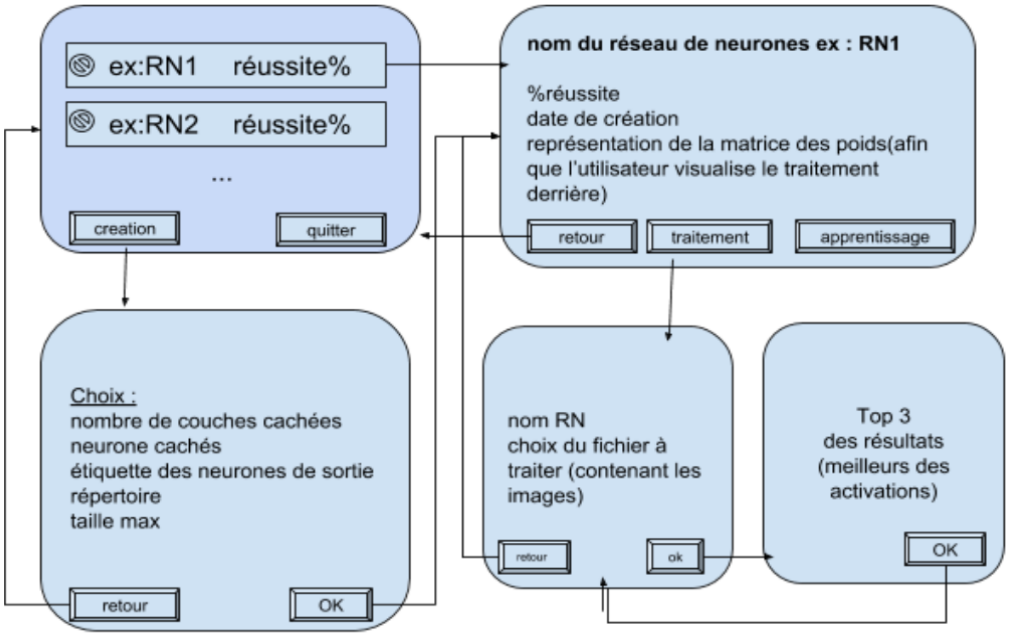
LibÃ¨re la mÃ©moire allouÃ©e par le rÃ©seau de neurones en paramÃ¨tre de la fonction.

4 Interface

4.1 Choix de la bibliothÃ¨que graphique

Afin de rÃ©aliser l'interface de notre application nous avons choisi la bibliothÃ¨que graphique GTK+ qui est une bibliothÃ¨que permettant de crÃ©er des interfaces graphiques GUI (Graphical User Interface) trÃ¨s facilement.Utilisable avec plusieurs langages de programmation. MÃªme si elle a Ã©tÃ© Ã©crite en C, sa structure orientÃ© objet et sa licence ont permis aux developpeurs d'adapter GTK+ Ã leur langage prÃ©ferÃ©. GTK+ s'intÃ¨gre relativement bien sur les systÃ¨mes GNU/Linux.

4.2 SchÃ©ma interface



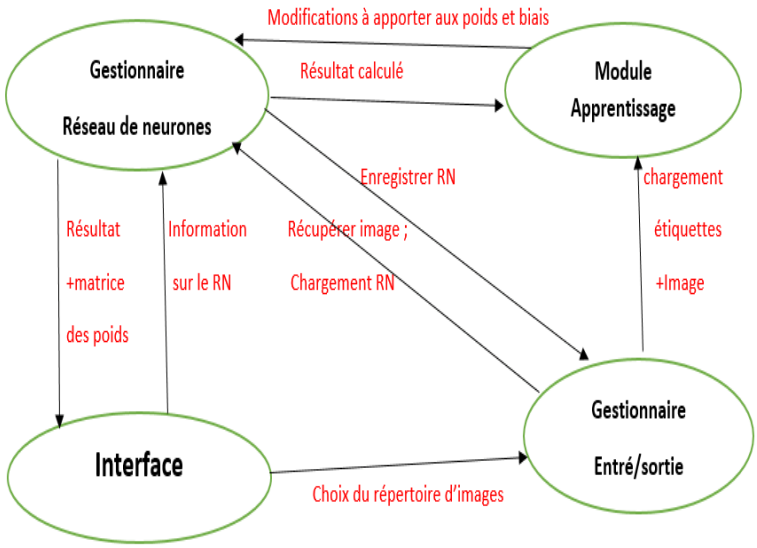
4.3 ScÃ©nario de l'utilisation de l'interface

Lors de l'utilisation de l'application une page s'ouvre Ã l'utilisateur afin qu'il puisse faire ses manipulations. Il a un menu Ã sa portÃ©e oÃ¹ *il peut selectionner des reseaux de neurones qu'il a déjà créé précédemment. Il peut choisir*

– taille Max qui est la taille maximale des images qu'il devra analyser (qui precise la nombre de neurones d'entrée du reseau de neuro
– le nombre de couches cachées ainsi que le nombre de neurones.
– les etiquettes des neurones de sortie.
– choisir le repertoire d'image où elles sont stoquées.
Ainsi il clique sur le bouton valider pour confirmer les informations, ou bien sur le bouton retour qui le redirigera la page d'accueil. A

5 Circulation d'informations entre les differents modules de l'application

5.1 Organigramme et flux d'information



5.2 Explication

Dans l'interface l'utilisateur a la possibilitÃ© de choisir un rÃ©seau de neurones parmis d'autres qu'il a dÃ©jÃ crÃ©e. Pour cela l'information sera transmise au gestionnaire du rÃ©seau de neurones afin qu'il puisse charger un rÃ©seau de neurones via la fonction chargerRN. Il peut Ã©galement crÃ©er un nouveau rÃ©seau de neurones, dans ce cas-lÃ il devra entrer les donnÃ©es nÃ©cessaires pour sa crÃ©ation (la taille maximale, les couches d'entrÃ©es, les couches cachÃ©es ainsi que les Ã©tiquettes et le repertoire des images). Ces informations-lÃ seront rÃ©cupÃ©rÃ©es par le gestionnaire du rÃ©seau de neurones sous forme d'une structure appelÃ©e RN. On utilisera par la suite la structure dans la partie traitement(la propagation). Dans cette partie, le rÃ©seau de neurones devra :
- rÃ©cupÃ©rer une image du gestionnaire entrÃ©e/sortie Ã travers la structure Image.
- calculer le rÃ©sultat et l'envoyer Ã nouveau Ã l'interface.
- envoyer la matrice des poids pour que l'utilisateur visualise au mieux les opÃ©rations qui s'effectuent derriÃ¨re.
Le rÃ©sultat calculÃ© par le gestionnaire des rÃ©seaux de neurones sera rÃ©cupÃ©rÃ© via la fonction propagation par le module apprentissage pour pouvoir l'utiliser dans la retro-propagation. Le module apprentissage va Ã©galement rÃ©cupÃ©rer les Ã©tiquettes ainsi que l'image du gestionnaire entrÃ©e/sortie via la fonction coupleImageEtiquette qui seront utile pour l'apprentissage. Le module apprentissage rÃ©cuperera aussi le top 3 des neurones de sortie via la fonction reconnaissance du gestionnaire rÃ©seau de neurones afin qu'on puisse calculer le nombre de reussite et d'Ã©chec en comparant la sortie obtenue avec celle attendue. Par la suite les modifications Ã apporter aux poids et aux biais seront rÃ©cupÃ©rÃ©es par le gestionnaire du rÃ©seau de neurones afin de les appliquer. Le rÃ©seau de neurones sera par la suite enregistrÃ© dans le gestionnaire entrÃ©e/sortie Ã travers la fonction SaveRN.

6 Conclusion

Pour conclure le choix du langage nous semble appropriÃ© pour la rÃ©alisation de notre application. En effet, nous n'avons pas rencontrÃ© de difficultÃ©s particuliÃ¨res lors de la rÃ©daction du cahier des spÃ©cifications. Cependant, les operations sur les matrices et les vecteurs auraient pu Ãªtre simplifiÃ©es. Par exemple, le langage Python propose des fonctions effectuants des actions pÃ©nibles Ã implementer en langage C (commme la transposition des matrices).