

# PMD

## Perceptron multicouche didactique

Description de l'application et documentation technique

**EIVD classe EI2A**

Frédéric JUNOD *frederic.junod@eivd.ch*  
Mathieu BORNOZ *mathieu.bornoz@eivd.ch*

Yverdon, le 15 juillet 2002.

---

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Avant propos . . . . .	3
1.2	Que fait ce programme . . . . .	3
<b>2</b>	<b>Information générales</b>	<b>4</b>
2.1	Exécution du programme . . . . .	4
2.2	Compilation du programme . . . . .	4
<b>3</b>	<b>Démarche globale</b>	<b>4</b>
<b>4</b>	<b>Description des modules</b>	<b>5</b>
4.1	Base d'entraînement . . . . .	5
4.2	Interface graphique . . . . .	6
4.2.1	Partie événementielle . . . . .	6
4.2.2	Zones de dessin . . . . .	7
4.2.3	Paquetage d'interface . . . . .	7
4.3	Implémentation du perceptron multi-couche . . . . .	7
4.3.1	Package Types . . . . .	7
4.3.2	Package Neuron . . . . .	7
4.3.3	Package Layer . . . . .	8
4.3.4	Package Sample . . . . .	8
4.3.5	Package Perceptron . . . . .	8
4.3.6	La propagation des données . . . . .	8
4.3.7	La correction des erreurs . . . . .	9
4.3.8	La rétropropagation de l'erreur . . . . .	9
<b>5</b>	<b>Difficultés rencontrées</b>	<b>9</b>
<b>6</b>	<b>Améliorations possibles</b>	<b>10</b>
<b>7</b>	<b>Conclusion</b>	<b>10</b>

## 1 Introduction

### 1.1 Avant propos

Cette application est le résultat de plusieurs mois d'intérêt pour les réseaux de neurones et reflète ni plus ni moins les principes du perceptron multi-couche décrit dans le document général "A la découverte des réseaux de neurones".

Il paraît difficile, voir impossible de comprendre la finalité de ce projet sans avoir préalablement pris connaissance du document expliquant ce que sont les réseaux de neurones, quels sont leurs buts et dans quelles circonstances il semble judicieux de les utiliser.

Dans la suite nous ne parlerons pas de l'algorithme général, mais plutôt comment nous avons pu, par la théorie des apprentissages, de l'entraînement et de la correction d'erreurs, arrivé à modéliser les différentes entités dans un langage de programmation, Ada 95 dans notre cas.

### 1.2 Que fait ce programme

Il s'agit d'un programme de reconnaissance de caractères, une reconnaissance qui se fait par le biais de l'apprentissage. Il ne s'agit donc pas d'effectuer un programme qui sait reconnaître des caractères mais qui est capable de les apprendre par l'exemple afin d'acquérir la connaissance puis d'exprimer son avis selon qu'on lui présente tel ou tel caractère.

Dans notre cas le réseau possède une base d'entraînement uniquement basée sur les nombres de 0 à 9. Ce qui à priori ne nécessiterait pas un perceptron multi-couche mais un perceptron simple. Néanmoins pour des soucis d'évolution du programme et pour permettre à l'utilisateur d'effectuer des changements majeurs dans son réseau, nous avons opté pour un perceptron multi-couche totalement modulaire.

Le soucis principal a été de faire une application capable de démontrer de manière visuelle le but d'un tel modèle. Il s'agit donc d'un outils didactique pour effectuer différents tests en fonctions de la base d'entraînements et des différents paramètres.

## 2 Information générales

### 2.1 Exécution du programme

La partie graphique étant totalement écrite en GTK(Ada), il est nécessaire d'installer cette librairie (sous GNU/Linux et Windows) pour exécuter le programme.

Aucune caractéristique matériel n'est exigée mis à part le fait que ce programme est fait uniquement pour les plate-forme Windows et GNU/Linux.

L'intégralité du code a été écrit en Ada 95 est les sources compilées avec le GNAT Version 3.1 sous GNU/Linux et 3.13p sous Windows NT 4.0 (pour plus d'informations consultez le manuel utilisateur du programme).

A noté que l'ensemble du programme à été développé sur la plate-forme GNU/Linux et plus particulièrement la distribution DEBIAN. L'ensemble du programme à été testé sur les deux systèmes d'exploitation sans aucun problème de portabilité. Ce qui renforce nos convictions allant dans le sens que des outils libres comme GTK peuvent être de qualité, portable est totalement adapté au besoin du marché.

### 2.2 Compilation du programme

Pour compiler les sources, des "Makefile" pour GNU/Linux et Windows sont fournis sur le support physique fourni avec l'ensemble du projet.

## 3 Démarche globale

Même si les théories et démonstrations mathématiques liées au perceptron multi-couche existent et sont parfois détaillées et précises, le passage à la modélisation du point de vue de la programmation est encore une étape des plus difficile. Pour ce faire, nous avons préféré regarder différents projets existant (fourni avec le code source) afin de se faire une idée de comment découper les parties de manière à avoir un code compréhensible, extensible, évolutif et reflétant le plus possible la théorie que nous avons vu.

Nous avons plusieurs exemples de code source, la plupart en C d'ailleurs, mais un site à spécialement attiré notre attention par ses exemples précis et concis. Il s'agit du site de l'EPFL (École Polytechnique Fédérale) et plus particulièrement l'espace réservé au "Laboratoire de Calcul Neuromimétique" qui, en plus d'une multitude de publication, offre de nombreux exemples de réseaux de neurones écrit en JAVA avec le code source fourni sous la "Licence Publique Générale de GNU (GPL)". Nous tenons d'ailleurs à saluer cette initiative qui permet à des personnes passionnées par ce domaine d'avoir des points de repère, de bons exemples de ce qu'il faut faire et ne pas faire.

## 4 Description des modules

### 4.1 Base d'entraînement

Les données d'entraînement sont en fait des tableaux de pixel stockées sous formes de tableau de caractères dans des fichiers.

```

.....XX....
.....XXX.....
.....XX.....
....XX.....
...XX.....
..XX.....
.X.....
.X.....
.X.....
.XXXXXXXXXX...
XXXX.....XXX..
XXX.....XX..
XX.....XX.
X.....X.
XX.....X.
.X.....X.
.XX.....X.
..XX.....XX.
...XX.....XX..
...XXXXXXXXX...

```

Il y a exactement 9 fichiers, de 0 à 9 possédant chacun 10 formes d'exemples comme ci-dessus (il s'agit d'un 6) en caractères ASCII afin de pouvoir si on le désire les voir et les modifier directement dans les fichiers.

#### [Paquetage forme]

Ce paquetage s'occupe exclusivement de la gestion de façon à ce que l'on puisse à tout moment savoir quelle est la forme courante, passer à la suivante, passer à la précédente ou encore modifier la forme.

Les principales sous-programmes de ce paquetage sont :

- **procédure Get** qui permet de récupérer une forme par le biais d'un tableau à deux dimension de boolean.
- **fonction Initialiser** qui place une sorte de pointeur (la forme courant) sur la première forme du premier fichier (soit le premier 0 du fichier 0.dat).
- **procédure Modifier** qui permet comme son nom l'indique modifier un pixmap ce qui en réalité remplace un pixmap par un autre.

- **fonction `Forme_Suivante`** qui déplace la forme courante sur la suivante du même fichier (arrivé sur la dernière il retourne sur la première).
- **fonction `Forme_Precedente`** qui déplace la forme courante sur la précédente du même fichier (arrivé sur la première il retourne sur la dernière).
- **fonction `Fichier_Suivant`** qui réalise le même processus que pour les formes mais qui cette fois se déplace de fichier en fichier de manière circulaire.
- **fonction `Fichier_Precedent`** qui réalise le même processus que pour les formes mais qui cette fois se déplace de fichier en fichier de manière circulaire.
- **fonction `Forme_Courante`** Retourne simplement la forme courante pour savoir où l'on se trouve.

Ces différentes sous-programme ont pour seul but de permettre la consultation et l'édition de la base de donnée d'apprentissage. Ce paquetage prend tout son sens dans le programme principale décrit dans les chapitres suivants.

## 4.2 Interface graphique

Une interface, et plus particulièrement pour un programme qui se veut didactique est d'une importance capitale. Elle doit permettre à l'utilisateur de prendre rapidement le programme en main en lui proposant des menus et des boutons suffisamment explicite pour qu'il ne se perde pas dans les méandres de l'application.

Quand au choix de l'outils permettant de créer une telle interface, dans notre cas GTK, il s'est fait tout naturellement puisque étant tous deux de fervents défanneur des logiciels libre nous n'avons pas hésité une seule seconde. La deuxième raison est que nous travaillons depuis plusieurs années sur le système GNU/Linux et plus particulièrement la distribution DEBIAN et que ce n'est pas, ou pas encore le cas de l'école ou nous avons développé ce programme. Nous avons donc pu et cela grâce à GTK créer un programme totalement portable et donc une plus grande souplesse d'utilisation.

### 4.2.1 Partie événementielle

Qui dit interface dit forcément interaction avec l'utilisateur. Une interaction qui est totalement gérée par GTK par le biais de ce que l'on appelle des "Callback". Pour se simplifier la vie nous avons dans un premier tant gérer le gros de l'interface grâce à un programme nommé "GLADE" qui permet de manière visuel, d'une part de placer les différents contrôle sur la fenêtre mais également de gérer les sous-programmes événementiels. Même si ce programme n'est pas encore totalement abouti, il représente néanmoins, ou du moins au début, un gain de temps considérable. Par la suite il est bien plus prudent d'affiner les choses directement dans le code, car "GLADE" gère très mal les modifications et les changements (ajouts, suppression, etc.).

#### 4.2.2 Zones de dessin

Les zones de dessin sont l'un des points cruciaux de notre programme, elles permettent à l'utilisateur de consulter la base de données de pixmap, dessiner la forme à présenter au réseau ou encore dessiner des graphiques.

La particularité de GTK et qu'il s'agit d'une librairie de relativement bas niveau, les problèmes de superposition de fenêtre ne sont pas gérés d'office ce qui dans un premier temps fait disparaître le contenu de vos différentes zones de dessin.

Pour remédier à ce problème nous avons utilisé un paquetage de spécial qui a été utilisé pour la création du logiciel "GIMP" un puissant outil de dessin, et qui nous a permis via un système de double buffer de gérer de manière propre les différents problèmes de superposition mais également de mise à jour des zones de dessin.

Le graphes d'erreur est un bon exemple de ce que l'on peut faire dans les zones de dessin. Malgré le déplacement de la fonction d'erreur et grâce au double buffer, l'affichage est d'une bonne fluidité et permet de mettre en avant et de manière graphique l'évolution de notre réseau de neurones. Il constitue une aide extrêmement précieuse, puisque lui seul permet à l'utilisateur de savoir ou en est l'entraînement du réseau et son niveau d'apprentissage.

#### 4.2.3 Paquetage d'interface

Un paquetage a été tout spécialement conçu pour faire l'interface entre la partie purement GTK(événementielle) et les autres paquetages de notre programme. Ce paquetage permet de gérer toutes les fonctions de dessin :

- Dessin des graphiques
- Récupération de ce que l'utilisateur dessine dans les zones
- Affichage de la base d'entraînement
- L'effacement des différentes zones
- Etc., etc.

Cette manière de faire permet de bien séparer les différentes entités mais également de ne pas "polluer" les parties pure GTK par d'importantes fonctions. Il paraît beaucoup plus logique et c'est d'ailleurs pour cela que nous l'avons fait, de se contenter, dans les sous-programmes événementiels, d'appeler les fonctions désirées.

### 4.3 Implémentation du perceptron multi-couche

#### 4.3.1 Package Types

Ce package rassemble tous les types de données commun entre différentes entités soit :

- les constantes de taille des formes d'apprentissage ;
- les types tableaux de réels pour ces formes.

Ces types et constantes sont implémentées dans un package sans corps que les autres packages incluses dans leurs spécifications.

#### 4.3.2 Package Neuron

Dans ce package sont implémentés les type T\_Neuron et T\_Synapse ainsi que leurs fonctions et procédures associées.

Ces deux types sont privés. Le type `T_Neuron` est un type article rassemblant les variables suivantes :

- `Sum` : représente le potentiel du neurone ;
- `Val_Delta` : le facteur de correction (voir document théorique) ;
- `Output` : la valeur de sortie du neurone, c'est à dire le potentiel passé dans la fonction de seuil ;
- `Inlinks` : tableau de pointeurs sur les synapses provenant de la couche précédente ;
- `Outlinks` : tableau de pointeurs sur les synapses en direction de la couche suivante.

En ce qui concerne le type `T_Synapse` il est composé comme suit :

- `Weight` : le poids de la connections synaptique entre le neurone `From` et `To` ;
- `Data` : la résultat de l'itération précédente qui sera utilisé pour le calcul de la rétro-propagation ;
- `From` : un pointeur sur le neurone émetteur ;
- `To` : un pointeur sur le neurone récepteur.

Les synapse possèdent des liaisons aller-retour (double chaînage) avec les neurones `From` et `To`. Le pointeur aller (de `From` a `To`) est utilisé pour la propagation des données de l'entrée vers la sortie et le retour (de `To` a `From`) pour la rétro-propagation de l'erreur.

#### 4.3.3 Package Layer

Ce package implémente le type `T_Layer` qui est un tableau de `T_Neuron`. Il permet de manipuler ces neurones sous forme de couches et simplifie donc les opérations.

Il est a noter que plusieurs procédures (comme `Set_Output`, `Set_Delta`, ...) surcharges celle du package `T_Neuron` pour un traitement global de la couche.

#### 4.3.4 Package Sample

Ce module fait l'interface entre les données de la base d'entraînement et le réseau de neurone. La procédure `Get_Samples` remplis les tableaux `Input_Samples` et `Output_Samples` qui seront ensuite présentés au réseau par l'intermédiaire de la procédure `Set_Sample` du package `Perceptron`.

#### 4.3.5 Package Perceptron

Ce package gère un perceptron multi-couche en fournissant le type `T_Perceptron` implémenté sous forme d'article a discriminant, ceux-ci fixes la taille de la couche d'entrée et de sortie (`In_Neuron` et `Out_Neuron`).

Détail du type :

- `Input_Samples` : Les exemple d'entrée provenant des fichier d'entraînement ;
- `Output_Samples` : les sortie associées aux entrée de dessus ;
- `Layers` : tableau de pointeurs sur les couches (entrée, sortie, et cachée(s)) ;
- `Error` : l'erreur globale du réseau.

#### 4.3.6 La propagation des données

L'implémentation de cet algorithme n'est que la traduction en Ada de la fonction mathématique vue dans la partie théorique. Il implémenté dans la package `Perceptron` (procédure `Propagate`) et son pseudo-code est le suivant :

```
pour toutes les couches sauf celle d'entree faire
(sens: de la couche d'entrée vers la couche de sortie)
  pour tous les neurones de la couche faire
    effectuer la somme pondérée des synapses en entree
    calculer la sortie en passant la somme par la fonction de seuil.
  fin faire
fin faire
```

#### 4.3.7 La correction des erreurs

Une fois la propagation effectuée, les poids des synapses sont corrigés de la couche de sortie vers l'entrée. Comme pour l'algorithme de propagation des données l'implémentation est la traduction de la formule mathématique.

Pseudo-code de l'algorithme :

```
pour toutes les couches faire sauf la couche d'entree faire
(sens: de la couche d'entrée vers la couche de sortie)
  pour tous le neurones de la couche faire
    effectuer la somme des erreurs en multipliant les erreurs de la couche
    précédente au delta de la couche précédente
    calculer le delta en passant cette somme par la dérivée de
    la fonction de seuil.
  fin faire
fin faire
```

#### 4.3.8 La rétropropagation de l'erreur

Maintenant, grâce à l'algorithme de correction des erreurs, nous savons quel synapse sont responsable des erreur et a quel taux. La rétropropagation de l'erreur vas corriger les poids avec la procédure `Set_Weight`. Pseudo-code :

```
pour toutes les couches faire sauf la couche d'entree faire
(sens: de la couche d'entrée vers la couche de sortie)
  pour tous le neurones de la couche faire
    calculer la somme des sorties de la couche précédente multiplié par
    le delta du neurone
    le nouveau poids de la connection est égal au poids précédent plus
    cette somme
  fin faire
fin faire
```

Cette opération va progressivement faire tendre les réponses du réseau vers une solution correcte.

## 5 Difficultés rencontrées

Nous avons rencontré un certain nombre de problème au niveau de l'implémentation et la gestion des couches cachées. Plus précisément pour les calculs liés à l'algorithme de rétro-propagation. Actuellement et pour des raisons de temps, nous avons finalisé l'ensemble du programme en mode perceptron simple et désactiver les paramètres

permettant de définir le nombre de couches cachées.

Comme dit dans le document "A la découverte des réseaux de neurones" un perceptron simple est tout à fait capable d'effectuer la séparation des classes représentée dans notre cas par des nombres de 0 à 9. Ce petit manque n'affecte donc pas vraiment la qualité de reconnaissance de forme de notre programme, mais, néanmoins il sera certainement possible lorsque nous aurons trouvé l'erreur d'avoir des résultats encore plus précis et cela grâce aux couches cachées et une meilleure modularité du réseau.

## 6 Améliorations possibles

Une extension visant à permettre également la reconnaissance des caractères pourra être apportée dans le future et cela grâce à la modularité de notre perceptron multi-couche. La possibilité d'ajouter des formes d'entraînement supplémentaires pour les nombres ainsi que celles visant à entraîner les caractères serait bien évidemment nécessaire pour parfaire l'ensemble.

La gestion du recadrage des formes (l'uniformisation de la taille des formes) ainsi que le redressement des caractères permettrait de tendre vers une application offrant plus ou moins les mêmes caractéristiques qu'un véritable programme de reconnaissance (ocr) comme nous avons pu le voir dans différents projets présentés dans diverses documentations spécialisées.

Une réécriture du code dans un langage orienté objet (langage Ada ou d'autres) pourrait également contribuer à simplifier le code et mieux définir les différentes entités décrites précédemment.

## 7 Conclusion

Cette application est le fruit de nombreuses heures de travail extra-scolaire car aucun cours n'est dédié aux réseaux de neurones. Même si ces principes sont au départ plutôt déroutants ils montrent bien que dans certains cas leur déploiement peut être d'une grande utilité là où d'autres principes n'offriraient pas vraiment de solutions.

Cela fut pour nous un projet passionnant et ô combien intéressant. Il nous a permis et cela pour la première fois, d'utiliser quelques principes mathématiques vus au cours de nos années d'étude.

En plus d'avoir modélisé ce réseau, nous avons également pu nous familiariser et acquérir une certaine maîtrise de la librairie graphique GTK. Une librairie de relativement bas niveau qui n'est pas simple à prendre en main, mais qui offre une grande souplesse de développement et de très bons résultats sans compter sur le fait qu'elle soit portable, ce qui représente tout de même un argument de taille.