Nouvelles Technologies de l'Information et de la Communication

Projet de Semestre

Page Web Didactique de Visualisation d'Algorithme

Université de Genève - semestre de printemps 2012

Pierre Künzli, Adrien Lescourt

23 mai 2012

Table des matières

1 Présentation

1.1 Introduction

Le but de ce projet est de proposer un outil de visualisation intéractif permettant de présenter un algorithme. Ce type d'outil peut servir dans le cadre de cours d'algorithmique ou lors de présentation au publique afin de fournir une visualisation intuitive du fonctionnement d'un algorithme.

Notre choix s'est porté sur la visualisation d'un réseau de neurones. En effet, ce genre d'algorithme permet d'imaginer un système interactif. En l'occurence, une tâche de classification.

1.2 Réseaux de neurones artificiels

Les réseaux de neurones artificiels constituent un modèle de calcul inspiré du fonctionnement des neurones biologiques. Dans ce modèle, un réseau possède un certain nombre d'entrées et doit déterminer l'état de ses sorties en fonction de ses entrées.

Pour ce faire, on utilise plusieurs couches de neurones, chaque neurone d'une couche étant connecté par des synapses à tous les neurones de la couche suivante. Chaque neurone additionne alors toutes ses entrées (la valeur des synapses en entrée) et injecte cette somme dans une fonction de combinaison non linéaire avant de transmettre la valeur calculée à la couche suivante. Les couches d'entrée et de sortie peuvent elles être linéaires ou non. La structure d'un neurone est montré sur la figure 1 , la structure complète d'un réseau neuronal simple à trois couches est montré sur la figure 2 [1].

La difficulté dans ce genre d'algorithme est de calculer la pondération des synapses de sorte que le réseau fournisse la sortie attendue en fonction de l'entrée. Pour cela, une phase d'apprentissage est nécessaire. Durant cette phase, des données sont fournies en entrée, puis la sortie est comparée avec la sortie attendue et les poids des synapses sont modifiés en fonction des erreurs du réseau. Pour cette phase d'apprentissage, il est soit possible d'utiliser des métaheuristiques pour optimiser le réseau, soit, plus classiquement, un algorithme de rétropropagation du gradient de l'erreur. C'est l'algorithme qui est utilisé dans le cadre de ce projet. L'idée de cet algorithme est de corriger les poids synaptiques en fonction de l'erreur en sortie. Un synapse qui contribue fortement à une erreur se verra fortement modifié. En appliquant itérativement cette technique sur un réseau, on espère atteindre une configuration correcte du réseau. A chaque itération, une fonction de coût représente la qualité du réseau.

La théorie concernant les réseaux de neurones n'est pas le sujet principal de ce rapport, néanmoins, une littérature fournie existe à ce sujet.

1. source : Wikimedia Commons

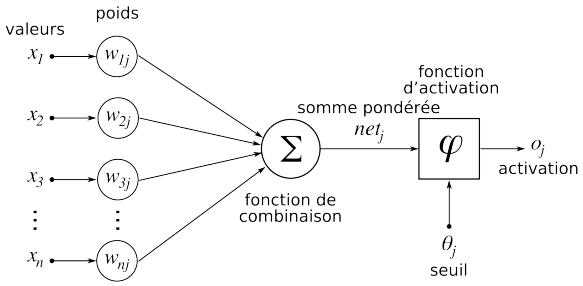

FIGURE 1 – Modèle d'un neurone

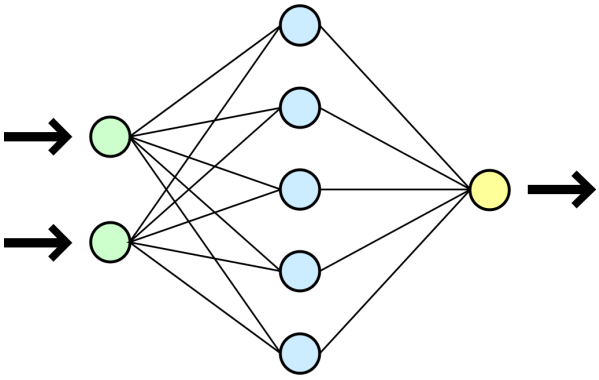

FIGURE 2 – Réseau de neurones simple à trois couches

2 Cas d'utilisation

Le but de ce projet est donc de proposer un outil interactif permettant de visualiser l'optimisation d'un réseau et d'utiliser ensuite ce réseau pour tester son fonctionnement. Afin de ne pas complexifier outre mesure la réalisation du projet, il a été choisi de se focaliser sur un type de réseau et une configuration particulière, à savoir, un réseau capable d'effectuer une tâche de classification. La tâche de classification choisie est la classification de chiffres entre 0 et 9 écrits à la main.

Les cas d'utilisation spécifiés ici ne présentent que les tâches utilisateurs. En effet, le système ne propose que des actions sur des données présentes dans le système. Il n'est pas possible de modifier un réseau modélisé à partir de l'interface web. Pour cela, il faudra manipuler directement le programme. Vu la complexité que cela représente et que de telles modification n'ont pas à être opérées souvent, il n'a pas été prévu d'interface administrateur. Les cas d'utilisation sont présentés ici tels qu'imaginés à l'origine. Il n'a malheureusement pas été possible d'intégrer toutes ces fonctionnalités dans le système. A l'heure actuelle, seuls les cas d'utilisation "classifier" et "calculer, données pré-enregistrées" ont été partiellement implémentés sous une forme différente de celle proposée ici. Les capacités actuelles du système sont détaillées dans la section "Système implémenté". Les cas d'utilisation sont donnés ici avec leurs titres respectifs ainsi qu'un prototype d'écran correspondant pour les deux cas principaux.

Explications

Une série de pages (navigables avec des boutons suivant – précédent) apparaît et explique les différentes phases du déroulement de l'algorithme de façon aussi clair et exhaustive que possible. Les explications sont renforcées par des animations.

Classifier (figure 3)

- Choisir le réseau de classification.
- Dessiner des chiffres dans une aire de dessin.
- Le résultat de la classification est affiché.

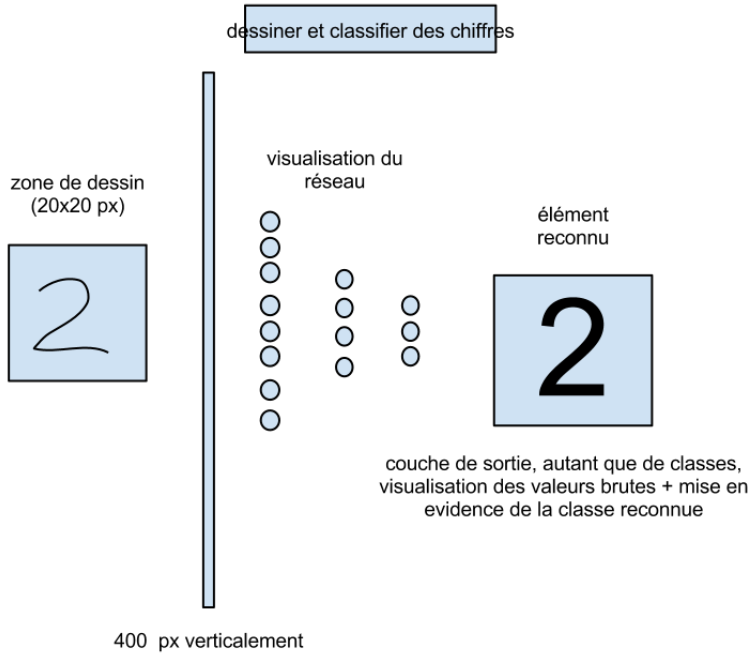

FIGURE 3 – Cas d'utilisation "Classifier"

Calculer, données pré-enregistrées (figure 4)

- Choisir un set de données d'entraînement.
- Démarrer l'optimisation du réseau.
- Afficher la valeur de la fonction objectif pendant l'optimisation.
- Dessiner des chiffres dans une aire de dessin.
- Possibilité de mettre en pause – accélérer – ralentir l'optimisation.
- Le résultat de la classification est affiché.

Calculer, données utilisateur

- Dessiner 10 chiffres.
- Démarrer l'optimisation du réseau .
- Afficher la valeur de la fonction objectif pendant l'optimisation.
- Dessiner des chiffres dans une aire de dessin (en parallèle de l'optimisation du réseau).
- Possibilité de mettre en pause – accélérer – ralentir l'optimisation.
- Le résultat de la classification est affiché.
- L'utilisateur peut valider ou corriger la classe déterminée pour que la donnée rentre dans les données d'entraînement.

Visualiser les données

- Choisir un set de données.
- Les instances s'affichent dans une fenêtre de visualisation.

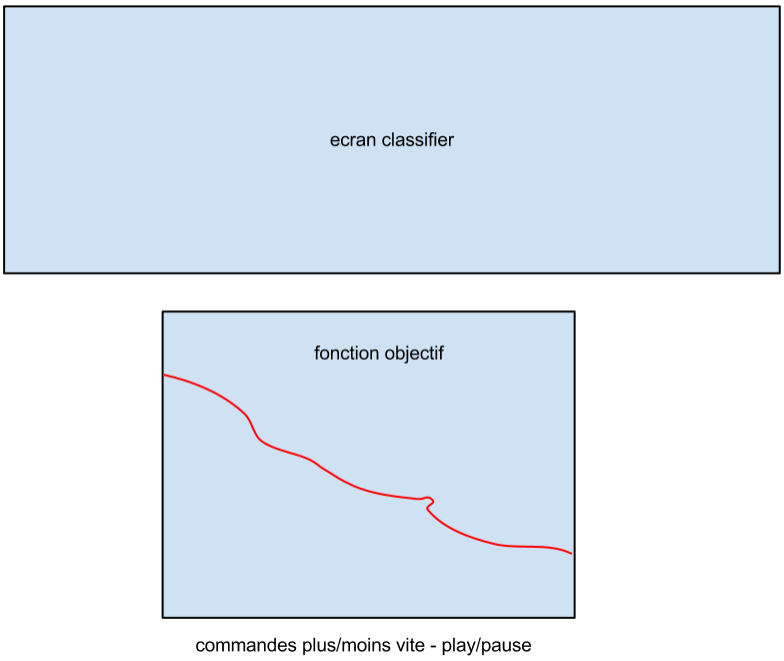

FIGURE 4 – Cas d'utilisation "Calculer, données pré-enregistrées"

3 Conception

Dans cette section, la conception générale du système est décrite. Le détail des étapes à réaliser pour mettre en place le système est donné dans les annexes.

3.1 Protocole de communication

Le système est découpé selon une architecture client/serveur. Globalement, le client ne s'occupe que de récupérer les entrées utilisateur et de présenter les données. Le serveur quant à lui s'occupe de calculer les résultats en fonction des requêtes du client.

Le client peut adresser deux types de requête au serveur : une requête de classification et une requête pour connaître l'état d'un réseau. Les requêtes se font via la méthode `http GET` en `AJAX` et le serveur envoi en réponse un document `xml`.

Etat d'un réseau

Requête :
Le client envoi cette requête afin de connaître l'état complet d'un réseau à une itération donnée. Afin de répondre à cette requête, le serveur doit connaître deux choses : l'identifiant du réseau et le numéro de l'itération voulue. La requête prends alors la forme suivante : `param_`*idReseau*`_`*iteration* où *idReseau* est l'identifiant du réseau et *iteration* est l'itération voulue. Ainsi, si un client veut connaître l'état du réseau *num* à l'itération 500, il enverra la requête `param_num_500`.

Réponse :
Le serveur va répondre à cette requête avec un document `xml` contenant toutes les données caractérisants le réseau. A savoir : le nombre de couches (attribut `nbcouches`), le nombre de neurones par couche (attribut `taillecouches`) et le poids de chaque synapse (attribut `liens`). Le format prévoit également

d'envoyer la valeur locale de chaque neurone, à savoir la somme des entrées pondérée par la fonction d'activation, mais le serveur renvoi toujours la valeur 0 pour cet attribut dans la version actuelle du système. Un exemple partiel d'un tel document `xml` est montré sur le listing 1, la `DTD` correspondante est montrée sur le listing 2.

Listing 1 – Exemple de fichier `xml` en réponse à une requête d'état d'un réseau

```
<reseau>
    <nbcouches>3</nbcouches>
    <taillecouches>2,3,1</taillecouches>
    <couche>
        <niveau>0</niveau>
        <noeud>
            <num>0</num>
            <val>0</val>
            <liens>0.933592407008,2.12853941925,0.390483852058</liens>
        </noeud>
        <noeud>
            <num>1</num>
            <val>0</val>
            <liens>-0.91306200467,0.511798771662,-0.468744634851</liens>
        </noeud>
    </couche>

    ...

</reseau>
```

Listing 2 – DTD des documents `xml` de réponse pour l'état d'un réseau

```
<!DOCTYPE reseau [

<!ELEMENT reseau (nbcouches, taillecouches, couche+)>
<!ELEMENT nbcouches (#PCDATA)>
<!ELEMENT taillecouches (#PCDATA)>
<!ELEMENT couche (niveau, noeud+)>
<!ELEMENT niveau (#PCDATA)>
<!ELEMENT noeud (num, val, liens)>
<!ELEMENT num (#PCDATA)>
<!ELEMENT val (#PCDATA)>
<!ELEMENT liens (#PCDATA)>

]>
```

Classification

Requête :
Le client envoi cette requête afin de connaître le résultat d'une classification. Pour répondre à une telle requête, le serveur doit cette fois ci disposer de trois informations : l'identifiant du réseau, l'itération à considérer et enfin les données brutes à fournir à l'entrée du réseau. Pour ce faire, le client va envoyer une requête de la forme `class_`*data*`_`*idReseau*`_`*iteration*, où *data* est un vecteur de nombres réels, séparés par des virgules, qui sera fournit en entrée du réseau de neurone, *idReseau* est le nom du réseau à utiliser et *iteration* est l'itération à laquelle considérer le réseau. Ainsi, si le client veut fournir l'entrée $\{0.2, 0.5, 1.2\}$ à l'entrée du réseau *num* à l'itération 500, il enverra la requête `class_0.2,0.5,1.2,_num_500`.

Réponse :
Le serveur réponds à cette requête avec un document `xml` contenant le poids de chacune des sorties du réseau (attribut `poids`) ainsi que l'indice du poids le plus fort (*argmax*, attribut `valeur`). Le listing 3 montre un exemple d'un tel fichier et le listing 4 montre la `DTD` correspondante.

Listing 3 – Exemple de fichier `xml` en réponse à une requête de classification

```
<output>
    <poids>0.000387727390373,0.141619070168,0.00292553325452,0.579168814283,...</poids>
    <valeur>3</valeur>
</output>
```

Listing 4 – DTD des documents `xml` de réponse de classifiaction

```
<!DOCTYPE output [

<!ELEMENT output (poids, valeur)>
<!ELEMENT poids (#PCDATA)>
<!ELEMENT valeur (#PCDATA)>

]>
```

3.2 Serveur

Pour implémenter le serveur, le choix s'est porté sur `python` pour plusieurs raisons. Premièrement, il semblait intéressant d'explorer le développement d'une application web avec un langage autre que le traditionnel `php`. D'autre part, le `python` est un langage qui peut facilement être utilisé localement, sans serveur web, ce qui facilite grandement le développement et les tests sur les parties qui présentent une difficulté algorithmique (typiquement l'utilisation de réseau de neurones dans le cas de ce projet). Finalement, le `python` à l'avantage de disposer de nombreuses bibliothèques spécifiques développées pas la communauté, dont notemment des librairies implémentant des réseaux de neurones.

Composants

Le serveur utilisé dans le cadre de ce projet est un serveur `Apache 2` auquel le module `mod-wsgi` à été adjoint. Ce dernier permet à `Apache` d'interpréter du `python`. Il a été choisi d'utiliser un framework `python` pour le web extrêmement simple, à savoir `webpy`. Ce dernier permet en effet d'implémenter simplement le comportement d'une méthode `GET`. Un exemple minimal de programme `webpy` est montré dans le listing 5.

Listing 5 – Exemple de programme minimal `webpy`

```
import web

class main:
    def GET(self):
        # le serveur fait du ping-pong, il repond avec la requete recue
        request = web.input(q = 'web')
        return request

urls = ('/', 'main')
application = web.application(urls, globals()).wsgifunc()
```

Afin d'effectuer les calculs relatifs aux réseaux de neurones, le choix s'est porté sur la librairie `pybrain`, qui semblait fournir tout les outils nécessaires et être assez simple d'utilisation. Il est en effet facile de

construire un réseau selon la configuration voulue, de l'entraîner à partir d'un ensemble de données et de l'utiliser ensuite. Un exemple simple est montré sur le listing 6.

Listing 6 – Exemple d'utilisation de `pybrain`

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.datasets import SupervisedDataSet
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure import SigmoidLayer
# construction d'un reseau - 400 entree, 10 sorties, couche intermediaire de taille 20
net = buildNetwork(400, 20, 10, bias=True, hiddenclass=SigmoidLayer)
# creation d'un ensemble d'entrainement avec 400 entrees et 10 sorties
ds = SupervisedDataSet(400, 10)
# ajouter des donnees d'entrainement
# (input et output sont des vecteurs de nombre reels respectivement de taille 400 et 10)
for ... :
    ...
    ds.addSample(input,output)
# on utilise la backpropagation comme methode d'aprentissage
trainer = BackpropTrainer(net, ds)
# entrainer le reseau pendant 500 iterations
for i in range(500):
    trainer.train()
...
# activer le reseau avec une entree
output = net.activate(input)
```

Génération statique des réseaux de neurones

Au cours du travail, il s'est avéré que l'optimisation d'un réseau de neurones à l'aide de `pybrain` représente une tâche assez longue. Ainsi, afin d'éviter une surcharge trop forte du serveur, la décision à été prise de ne pas générer les réseaux en direct, lorsque l'utilisateur en fait la demande, mais de pré-générer des configurations de réseaux et d'en sauvegarder un certain nombre d'itérations sous forme de fichier afin de pouvoir les charger au moment voulu.

Pour l'instant, chaque réseau est optimisé sur 500 itérations. Les dix premières itérations sont sauvegardées sur le disque, puis, une sauvegarde est effectuée toutes les dix itérations. En effet, pour un réseau de 400 entrées, 10 sorties et une couche intermédiaire de taille 20, la place occupée d'une itération sur le disque est d'environ $300ko$, ce qui peut rapidement devenir trop important si trop de sauvegardes sont faites.

Afin de sauver les réseaux calculés sur le disque dur, le module `python pickle` est utilisé. Ce dernier offre des fonctionnalités de sérialisation-désérialisation d'objets en `python` est permet d'écrire des objets sur le disque facilement. Un exemple d'utilisation de ce module est montré dans le listing 7.
Il est important de noter ici que chaque itération d'un réseau donné est représenté par un réseau à part entière. Ainsi, un réseau nommé `number_simple` à l'itération 120 sera sauvegardé sur le disque dans un fichier nommé `number_simple_120`.

Listing 7 – Utilisation du module `pickle`

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.structure import SigmoidLayer
import pickle
net = buildNetwork(400, 20, 10, bias=True, hiddenclass=SigmoidLayer,inclass=SigmoidLayer)
...
# sauvegarder l'objet net dans le fichier network
f = open("network","w")
pickle.dump(net,f,2)
f.close()
...
# recharger l'objet a partir du fichier
f = open("network","r")
net2 = pickle.load(f)
f.close()
```

Un défaut de `pybrain` est qu'il semble ne pas fournir simplement certaines informations essentielles à propos d'un réseau, notamment le nombre de couches du réseau et la taille de celles-ci. Si cela n'est pas nécessairement gênant dans le cas d'un programme écrit d'un bloc, puisque ces paramètres sont forcéments spécifiés au moment de la création du réseau, cette situation s'avère plus problématique dans notre cas. En effet, les réseaux sont générés par des scripts `python`, puis chargés par la suite par le serveur, qui a alors besoin de connaître ces données pour construire les réponses à envoyer aux clients. Pour palier à ce manque, un fichier nommé `parameters` à été créé. Celui-ci contient, pour chaque réseau à disposition, le nombre de couches du réseau et la taille de celles-ci dans l'ordre. Ainsi, la ligne de paramètre du réseau `number_simple_130` qui possède 400 entrées, 10 sorties et une couche intermédiaire de taille 20 est la suivante : `numbers_simple_130 3 400 20 10`.

Fonctionnement du serveur

Une fois la génération des réseaux effectuée, le rôle du serveur consiste simplement à générer des `xml` en fonction des requêtes du client. Les réseaux ainsi que les paramètres correspondants sont tout d'abord chargés, puis, le serveur utilise ces données pour traiter les requêtes. Le code simplifié de la méthode `GET` est montré sur le listing 8.

Listing 8 – Code simplifié de la méthode `GET`

```
def GET(self):
    # les variables staticNetworks et staticNetworksParams
    # contiennent respectivement les reseaux et leurs parametres correspondant
    global staticNetworks
    global staticNetworksParams
    # recuperation de la requete du client
    request = web.input(q = 'web')
    reqsplit = request['q'].split('_')
    # le client veut obtenir les parametres d'un reseau
    if reqsplit[0]=="param":
        ... determiner quel reseau a partir de la requete ...
        ... stocker le nom du reseau correspondant dans netname...
        # construire le xml demande
        res = makeXMLstatenet(staticNetworks[netname],staticNetworksParams[netname],netname
            )
    # le client envoi une entree pour faire de la classification
    if reqsplit[0]=="class":
        ... construire les donnees d entree dureseau, les stocker dans netinput ...
        ... et determiner le reseau a utiliser, stocker le nom dans netname ...
        # construire le xml demande
        res = makeXMLclassif(staticNetworks[netname],staticNetworksParams[netname],netinput
            ,netname)
    # envoyer le xml au client
    return res
```

On voit donc que les méthodes `makeXMLstatenet` et `makeXMLclassif` constituent les deux fonctions principales du serveur. La première permet de construire un `xml` qui contient la structure complète d'un réseau et la seconde construit un `xml` qui contient le résultat d'une classification. La première fonction se contente de lire des données dans un objet et de les écrire dans un `xml`, il n'est donc pas utile de la montrer ici. Le code simplifié de la fonction `makeXMLclassif` se trouve sur le listing 9.

Listing 9 – Code simplifié de la fonction`makeXMLclassif`

```
def makeXMLclassif(net,netparams,netinput,netname):
    # calculer la sortie du reseau
    output = net.activate(netinput)
    ... ecrire les donnees de output dans res ...
    return res
```

3.3 Client

Pour l'élaboration du côté client, en plus de la structure `html` standard, la visualisation du réseau de neurones nécessite en complément un outil permettant un affichage complexe. Plusieurs technologies étant disponibles, c'est cependant `SVG 1.1` qui est utilisé dans ce projet. En effet, après un début difficile, cette technologie est désormais supportée par le moteur de rendu `Gecko` (utilisé par `Firefox`), par le moteur `Webkit` (Chrome, Safari), et même `Internet Explorer` à partir de sa version 9. Couplé avec `Javascript`, `SVG` permet un rendu vectoriel dynamique et interactif.

Le client est divisé en deux parties rassemblées autour d'un tronc commun `html`. Une partie traite l'affichage de l'état du réseau de neurones à une itération donnée et l'autre partie gère la classification d'un chiffre saisi par l'utilisateur.

Index.html

Le point d'entrée du site se décompose en deux fonctionnalités principales. Il s'occupe de traiter les choix de l'utilisateur en instanciant les objets `SVG` demandés, et il gère aussi toutes les communications avec le serveur.

L'inclusion des objets `SVG` se fait via la balise `object`. La seule difficulté est la transmission de paramètres aux fichiers externes `SVG`. Du côté du `html`, c'est extrêmement simple puisqu'il faut juste ajouter une balise `param` dans la balise `object` de la façon suivante :

Listing 10 – Paramètres HTML vers SVG

```
<object type="image/svg+xml" data="fichierSVG.svg">
    <param name="nomDuParametre" value="mesDonnees" />
</object>
```

Dans le cas de données complexes, comme c'est par exemple le cas avec l'affichage du réseau où c'est tout un fichier `XML` qu'il faut transmettre, on peut créer la balise `object` depuis `javascript` et remplacer 'mesDonnees' par une variable.

Par contre la réception depuis le fichier `SVG` est plus complexe puisqu'il faut établir une fonction qui parcourt la liste des paramètres afin de les charger en mémoire. Or si cette fonction est simple par son fonctionnement, la lecture des paramètres n'est pas des mieux documentée. En fait les valeurs sont accessibles via

Listing 11 – Accès aux paramètres depuis un fichier svg

```
document.defaultView.frameElement.getElementsByTagName("param");
```

qui est un tableau contenant pour chaque balise `param` définie dans l'objet `html` un couple d'attributs `name`, `value`. Si l'on se réfère à l'exemple 10, ce tableau contiendra dans sa première case `nomDuParametre`, `mesDonnees` dans respectivement les attributs `name` et `value`.

Comme annoncé précédemment, c'est aussi `index.html` qui s'occupe de toutes les communications avec le serveur. Toutes les requêtes sont effectuées en `AJAX`, sans utiliser de framework externe.

Affichage du réseau

L'affichage du réseau de neurones s'effectue via la procédure suivante :
- Requête du réseau X à l'itération Y par l'utilisateur.
- Envoi de cette requête au serveur.
- Récupération des données dans un format `XML`.
- Envoi des données `XML` à l'objet `SVG`.
- Parsing des données `XML`.
- Affichage du réseau.

L'itération est choisie via une slide bar `html5` entre 0 et 500. A l'heure actuelle, seul `Chrome` à implémenté cet élément, `Firefox` le remplace par une entrée texte.

Quant à l'affichage du réseau reçu, un élément `<g>` est premièrement défini. Il permet de regrouper un ensemble de formes. Chaque élément ajouté ensuite sera un fils de cet objet. Pour chacune des couches du réseau, on représente les noeuds comme des cercles les un en dessous des autres. Puis chaque noeud d'une couche doit être relié à tous les noeuds de la couche suivante. Selon l'état du réseau, les poids représentant la valeur de ces liaisons sont plus ou moins importants.

Pour le cas du réseau classifiant les chiffres, le réseau comporte beaucoup de noeuds, et énormément de liens. L'image étant composé de 400 pixels et le réseau de trois couches avec une couche intermédiaire de taille 20, cela nous donne 430 noeuds (400+20+10) et 8200 liens (400*20 + 20*10). Il est alors presque impossible de faire varier la largeur des liens directement en fonction des poids. Les poids sont donc regroupés en 6 catégories, en fonction de leur position dans le rapport entre le plus faible et le plus fort. Ces groupes représentent 6 variations possibles des liens. Les plus forts sont dessinés plus larges et plus foncés, et à l'inverse plus le liens est faible plus il est clair et fin. Enfin, pour accélérer l'affichage des couches ayant trop de noeuds (ici seulement la première couche), certains liens trop faibles ne sont pas affichés.

Cette procédure d'affichage est complètement générique. Elle peut donc s'adapter à n'importe quel réseau transmis, pour autant que le fichier `XML` transmit respecte la `DTD`.

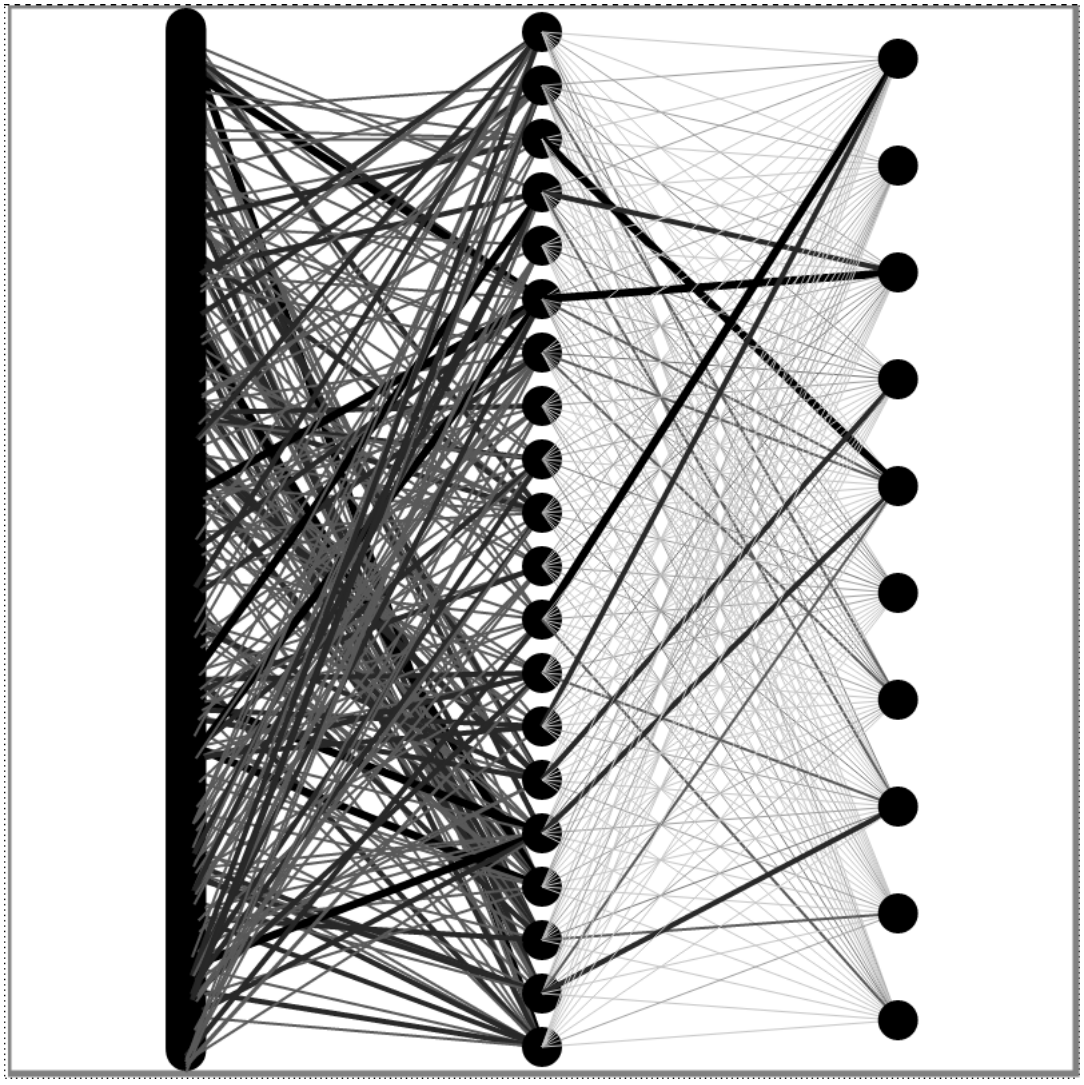

FIGURE 5 – Rendu d'un réseau pour une image de 20*20 pixels, à l'itération 500

Saisie et classification de chiffres

Outre l'affichage du réseau de neurones, le site propose à l'utilisateur de saisir un chiffre afin de le classifier. L'objet `SVG` est composé d'une balise englobante `<g>` comme expliqué au chapitre précédent. Ensuite 20*20 rectangles blancs sont ajoutés de façon à former l'image complète. Chacun de ces rectangles réagit aux événements de la souris de façon à dessiner intuitivement : Le rectangle cliqué change sa couleur en noir. Tant que la souris n'est pas relâchée, tous les rectangles sur lesquels elle passe deviennent noir également jusqu'au relâchement du bouton. Cette méthode de dessin comporte cependant un défaut mineur : si le relâchement du bouton se fait à l'extérieur de la zone `svg` alors l'évènement n'est pas détecté. Dans ce cas, quand l'utilisateur revient dans la zone de dessin, le programme continue à dessiner même si le bouton n'est pas pressé. Il faut donc essayer d'éviter de sortir de la zone quand on dessine un chiffre.

Afin de renforcer les performances de la classification, un contour est ajouté autour du rectangle noir de manière à former un dégradé de gris.

Enfin, une fois le chiffre dessiné, les pixels peuvent être transmis via le bouton `Send number at iteration X`. Quand ce bouton est appuyé, une chaîne de caractères est établie, contenant les niveaux de gris de chacun des rectangles séparés par des virgules. Cette chaîne est alors transmise au serveur qui retourne la classification au format `XML`. Dès réception de cette classification, un objet `SVG` est créé afin d'afficher le résultat. Cet objet est créé de façon similaire à l'affichage du réseau : il récupère puis parse le `XML`. Ensuite un histogramme est créé, contenant le résultat de la classification. Le chiffre détecté sera alors exprimé en rouge.

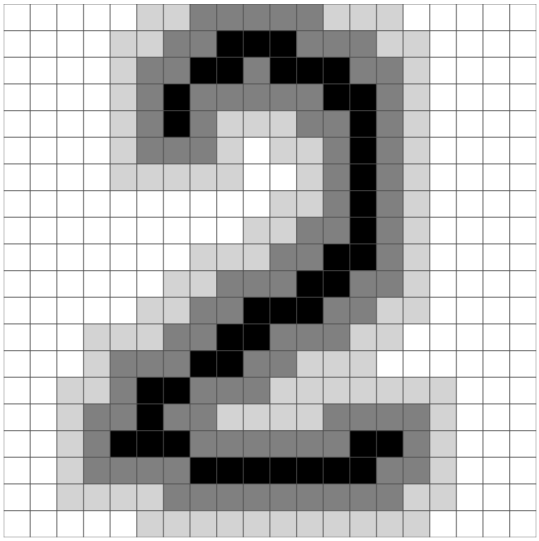

FIGURE 6 – Zone de dessin

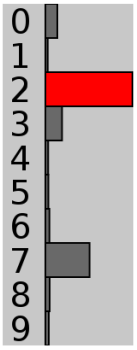

FIGURE 7 – Résultat de la classification de l'image 6

4 Système implémenté

Le système implémenté se décompose en deux parties principales : le code "utilisateur", qui comprends le code du serveur `python` à proprement parler ainsi que le code client tels que décrits dans les sections précédentes, et le code "administrateur", qui se compose d'une série de scripts permettant de générer des réseaux statiques pour le serveur. Plusieurs séries de données d'entraînement accompagnent le code "administrateur".

Dans les sources, le code à destination du serveur se trouve directement dans le répertoire `www` et le code et les données de génération de réseau se trouvent dans le répertoire `generation_NN`.

4.1 Etat actuel et fonctions disponibles

Actuellement, le système propose deux fonctionnalités, dérivées des cas d'utilisateur "Calculer, données pré-enregistrées" et "Classifier". Le système propose donc un premier écran, permettant de visualiser des réseaux à différents stades d'évolution de l'optimisation, tel que montré sur la figure 5. La deuxième fonctionnalité est la classification à proprement parler. Il est possible de choisir un réseau de classification et de l'utiliser afin de classifier des nombres tel que montré sur les figures 6 et 7. Deux modes de saisies sont disponibles, soit avec les niveaux de gris tel qu'expliqué dans la section "Client", soit en noir et blanc uniquement.

Trois réseaux de classification différents sont disponibles. Ceux-ci diffèrent par les données à partir desquelles ils ont été générés.

`numbers_simple` est un réseau généré à partir de chiffres écrits à la main puis numérisés en niveaux de gris. Afin de rendre les données compatibles avec le système de saisie, les niveaux de gris ont étés convertis en noir et blancs selon un seuil.

`numbers_nb` est un réseau généré à partir de données saisies à l'aide du système de saisie noir et blanc.

`numbers_gris` est un réseau généré à partir de données saisies à l'aide du système à niveaux de gris.

4.2 Limitations et évolutions possibles

La première amélioration du système consisterait à mieux configurer le réseau neuronal afin d'en améliorer la qualité de détection. Il serait aussi intéressant de compléter les set de références pour l'apprentissage. Dans l'état actuel, il est nécessaire de dessiner certains chiffres selon certains critères. Par exemple le 1 reconnaissable doit être dessiné comme une simple barre, sans tête ni pied. Ou encore le 4 qui doit être dessiné comme un h inversé.

Il conviendrait ensuite de re-travailler le code sur certaines parties, certains mécanismes ne sont pas encore assez généralisés pour permettre une bonne souplesse du système.

Une fois ces points réglés, il apparaît alors tout à fait possible d'implémenter le reste du système tel qu'imaginé à l'origine à une exception prêt. Effectivement, il semble que la classification en directe représente une tâche trop lourde pour un serveur web, cependant, si il semble qu'une optimisation sur un grand nombre d'itérations soit inenvisageable, effectuer un petit nombre d'itérations en direct serait tout à fait possible.

Globalement, il ressort de ce projet que la combinaison de technologies standards permet aujourd'hui de produire un système interactif performant et compatible avec tout les navigateurs récents. Par ailleurs, l'utilisation de `python` pour la conception de services web apporte un ensemble d'outils très puissants dans le cas où l'on souhaite réaliser des tâches non standards pour le web.

5 Annexes

5.1 Configuration du serveur

Pour installer `Apache 2` , `wsgi` et `webpy` sur une distribution `Ubuntu 10.04` il faut installer les paquets suivants :
– `apache2`
– `python-webpy`
– `libapache2-mod-wsgi`
Il faut ensuite modifier le fichier de configuration `/etc/apache2/sites-available` pour obtenir le fichier suivant :

Listing 12 – Configuration du serveur

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    WSGIScriptAlias /ntic/main /var/www/ntic/main.py/
    Alias /ntic/main/static /var/www/ntic/static/
    AddType text/html .py

    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>

    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    <Directory /var/www/ntic>
        Order deny,allow
        Allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog /var/log/apache2/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog /var/log/apache2/access.log combined

    Alias /doc/ "/usr/share/doc/"
```

```
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>
</VirtualHost>
```

Pour avoir un retour dynamique des erreurs d'`apache`, la commande suivante est très utile (en root) : `tail -f /var/log/apache2/error.log`

Le code suivant permet d'écrire dans la sortie d'erreur en `python` :

Listing 13 – Ecrire dans la sortie d'erreur en `python`

```
import sys
print>>sys.stderr,"message"
```

5.2 Installation de `pybrain`

Afin d'installer `pybrain` sur un système `Ubuntu 10.04`, il faut préalablement installer les paquets suivants :
– `git`
– `python-setuptools`
– `python-scipy`
– `gfortran` (ou tout autre compilateur `fortran`)
Les commandes suivantes permettent ensuite d'installer le module :

Listing 14 – Installation de `pybrain`

```
git clone git://github.com/pybrain/pybrain.git
cd pybrain
sudo python setup.py install
```