

EPSI Bordeaux

Réseaux neuronaux

TP - Perceptron

LAMARCHE Maxime – DA SILVA Arnaud
01/03/2016

Table des matières

Introduction.....	2
Compréhension générale	2
Interface graphique.....	3
Développement des premières fonctions.....	4
Les variables utiles.....	4
La fonction « Learn »	5
La fonction « CalculationOfObtainedValue »	6
La fonction « Find »	6
Première utilisation du perceptron.....	7
Développement avancé.....	8
La fonction « Training »	8
Mise en place des nouvelles variables, et modification des fonctions	8
Développement de fonction	9
Etude des résultats	10
Diminuer le nombre de d'apprentissages	10
Augmenter le taux d'erreur.....	10

Introduction

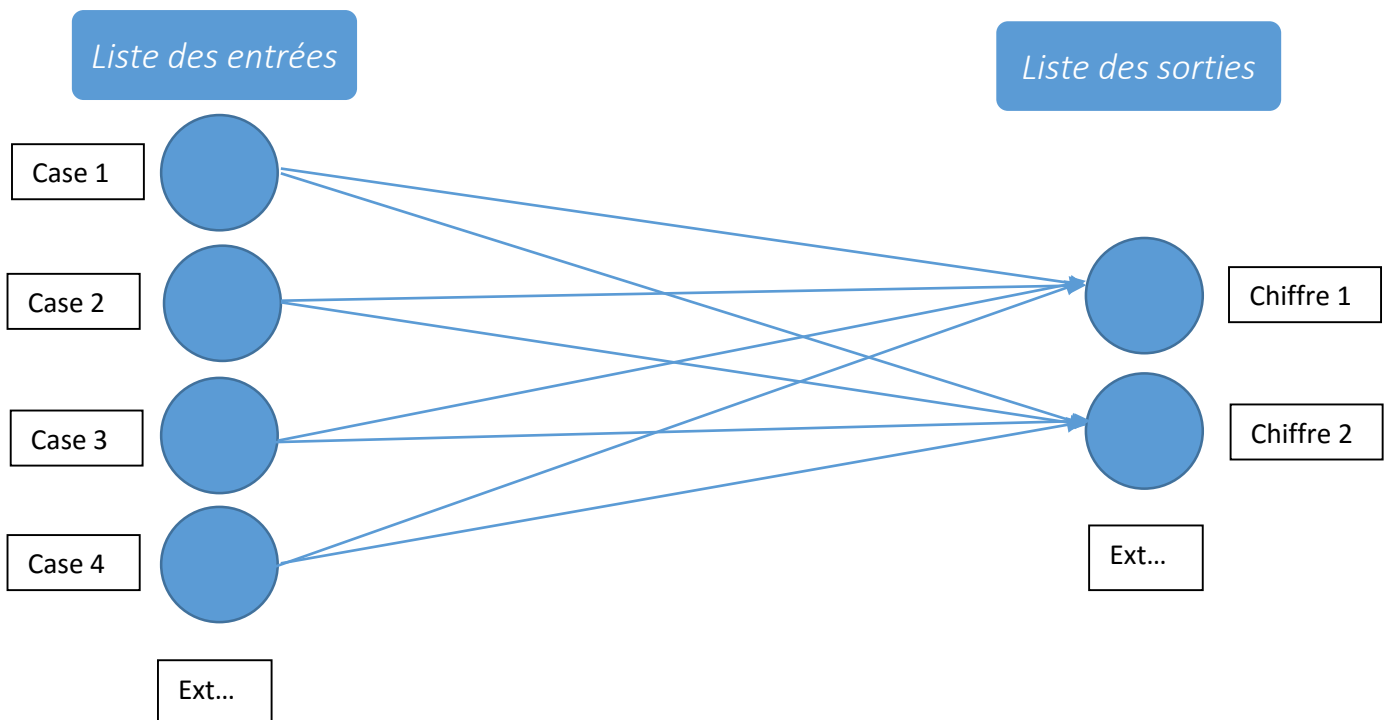
Compréhension générale

Nous avons décidé de développer le projet en JavaScript afin d'obtenir une interface graphique.

L'interface graphique se découpe en deux parties :

- La grille permettant de dessiner le chiffre
- La zone contenant les boutons permettant d'apprendre, et de trouver les chiffres dessinés.

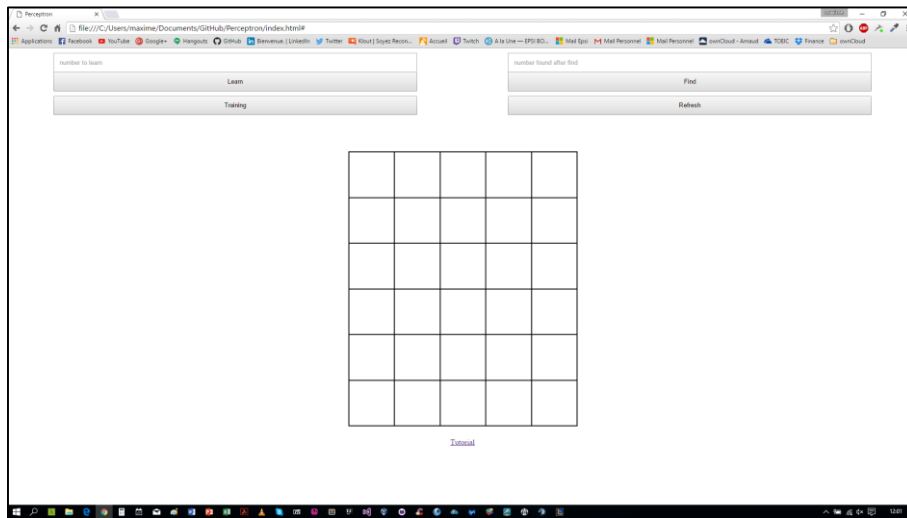
En fait, chaque case de la grille correspond à un neurone, et chaque chiffre pouvant être appris correspond à une sortie.



En faisant varier les poids, le seuil d'activation, et le têt d'apprentissage, il faut obtenir un système capable de reconnaître les chiffres dessinés.

Interface graphique

Le premier développement à faire est l'interface graphique. Je ne vais pas expliquer cette partie, mais juste montrer le rendu final.

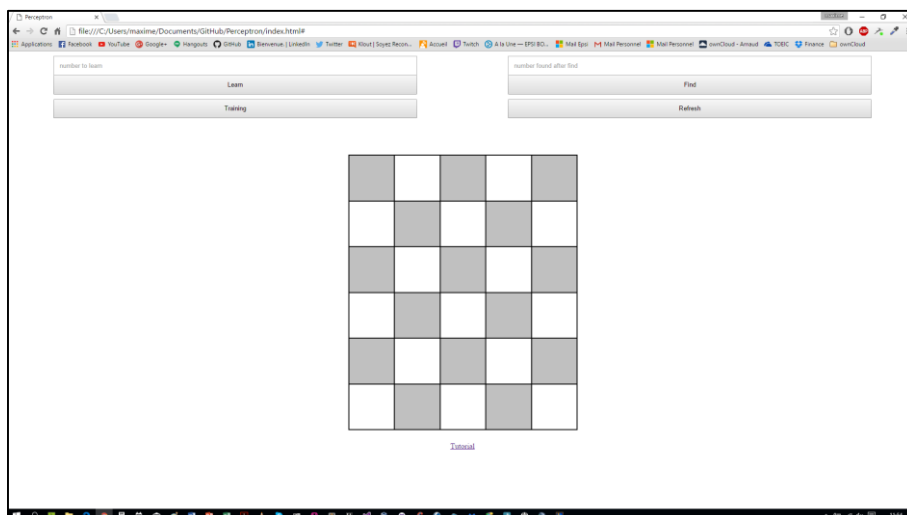


Nous avons donc :

- Une grille qui contient 30 cases
- Un bouton « Learn » permettant d'apprendre les chiffres au perceptron
- Un bouton « Find » permettant d'afficher le chiffre reconnue par le perceptron
- Un bouton « Training » (je reviendrais dessus plus tard)
- Un bouton « Refresh » permettant d'effacer la grille

Je rappelle que chaque case correspond à un neurone, ce qui nous fait un réseau à 30 neurones pour 10 sorties (0, 1, 2, 3, ..., 9).

Nous définissons comme « Actif » un neurone si celui-ci est rempli, et inactif ceux qui sont vide (les neurones actifs sont gris et les inactifs sont blanc).



Nous avons tous les outils graphiques pour dessiner les chiffres que le perceptron doit apprendre.

Développement des premières fonctions

Les variables utiles

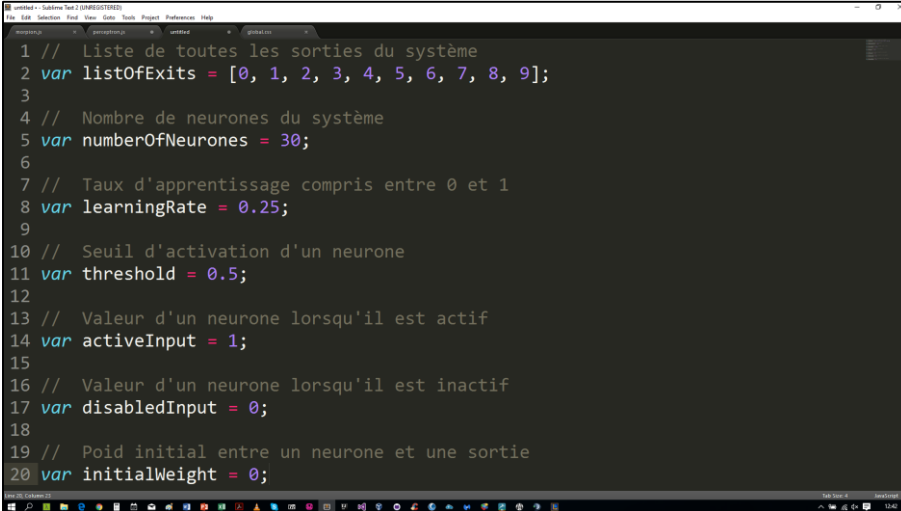
Il est important de définir les variables indispensables à l'utilisation du perceptron, et qu'elles soient dynamiques et non pas en dure dans le code (vous verrez pourquoi lors de l'étalonnage des valeurs).

Ces valeurs sont :

- Le taux d'apprentissage
- Le seuil d'activation
- La valeur d'un neurone actif
- La valeur d'un neurone inactif
- Et le poids initial

Personnellement, j'ai ajouté également :

- La liste des sorties possibles
- Le nombre de neurones



```
1 // Liste de toutes les sorties du système
2 var listOfExits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
3
4 // Nombre de neurones du système
5 var numberOfNeurons = 30;
6
7 // Taux d'apprentissage compris entre 0 et 1
8 var learningRate = 0.25;
9
10 // Seuil d'activation d'un neurone
11 var threshold = 0.5;
12
13 // Valeur d'un neurone lorsqu'il est actif
14 var activeInput = 1;
15
16 // Valeur d'un neurone lorsqu'il est inactif
17 var disabledInput = 0;
18
19 // Poids initial entre un neurone et une sortie
20 var initialWeight = 0;
```

La fonction « Learn »

Cette fonction a pour but de faire apprendre un chiffre au perceptron. Elle prend en paramètre le chiffre à apprendre (en fait c'est l'ID du chiffre à apprendre correspondant à ma liste de sorties), et fonctionne comme ceci :

- On boucle sur toutes les sorties possibles
 - On définit la valeur attendue en fonction de l'ID du chiffre à apprendre en paramètre
 - Si la sortie en cours est celle que l'on souhaite apprendre, la valeur attendue est égale à 2 * le seuil
 - Sinon, la valeur attendue est égale à la valeur d'un neurone inactif
- On boucle sur tous les neurones
 - On calcul la valeur obtenue
 - Pour obtenir la valeur obtenue, on boucle sur tous les neurones actifs, et on somme le produit de la valeur de l'input en cours et le poids entre ce neurone et la sortie (cf. La fonction « CalculationOfObtainedValue »)
 - On met à jour le poids entre le neurone et la sortie grâce à la formule suivante :

$$P = P + (E - O) * I * L$$

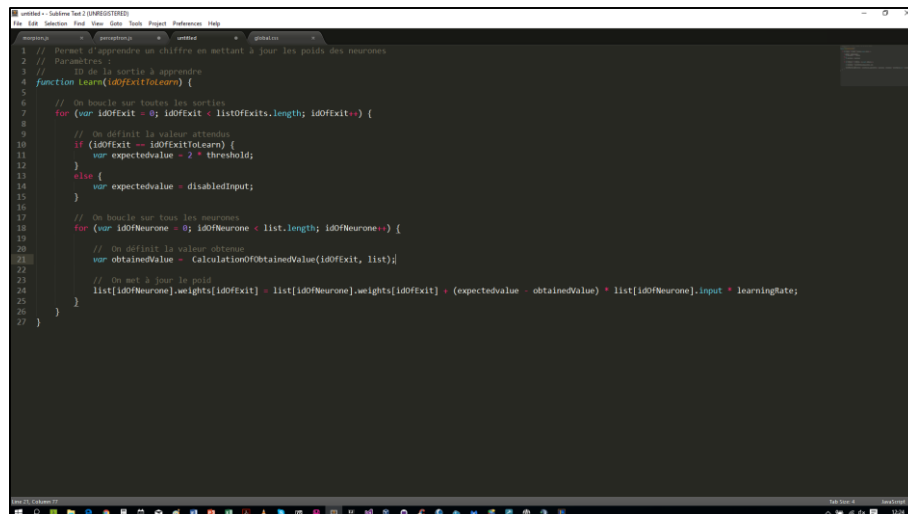
P : Le poids du neurone pour une sortie

E : La valeur attendue

O : La valeur obtenue

I : La valeur du neurone

L : Le taux d'apprentissage

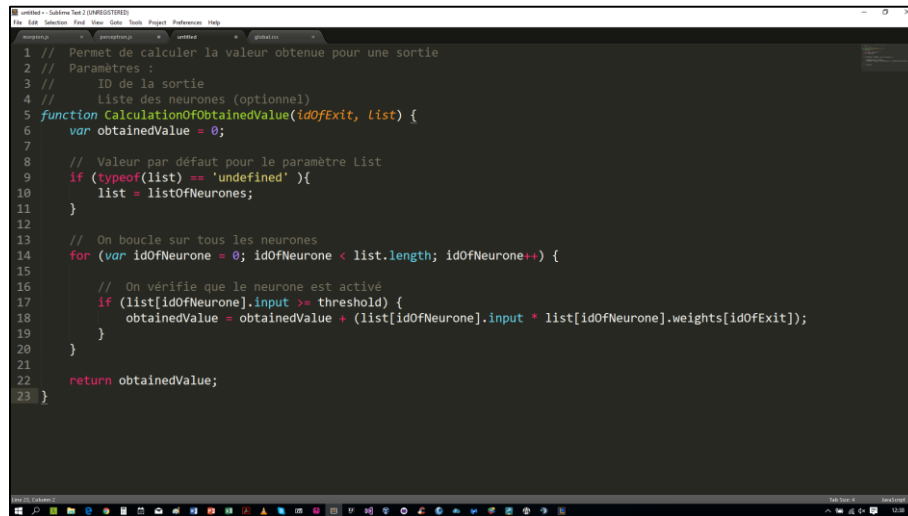


```

1 // Permet d'apprendre un chiffre en mettant à jour les poids des neurones
2 // Paramètres :
3 // ID de la sortie à apprendre
4 function learn(idOfExitToLearn) {
5
6   // On boucle sur toutes les sorties
7   for (var idOfExit = 0; idOfExit < listOfExits.length; idOfExit++) {
8
9     // On définit la valeur attendue
10    if (idOfExit == idOfExitToLearn) {
11      var expectedValue = 2 * threshold;
12    }
13    else {
14      var expectedValue = disabledInput;
15    }
16
17    // On boucle sur tous les neurones
18    for (var idOfNeurone = 0; idOfNeurone < list.length; idOfNeurone++) {
19
20      // On définit la valeur obtenue
21      var obtainedValue = CalculationOfObtainedValue(idOfExit, list);
22
23      // On met à jour le poids
24      list[idOfNeurone].weights[idOfExit] = list[idOfNeurone].weights[idOfExit] + (expectedValue - obtainedValue) * list[idOfNeurone].input * learningRate;
25    }
26  }
27 }
  
```

La fonction « CalculationOfObtainedValue »

Je ne vais pas revenir sur cette fonction, elle est simple et expliqué plus haut.



```

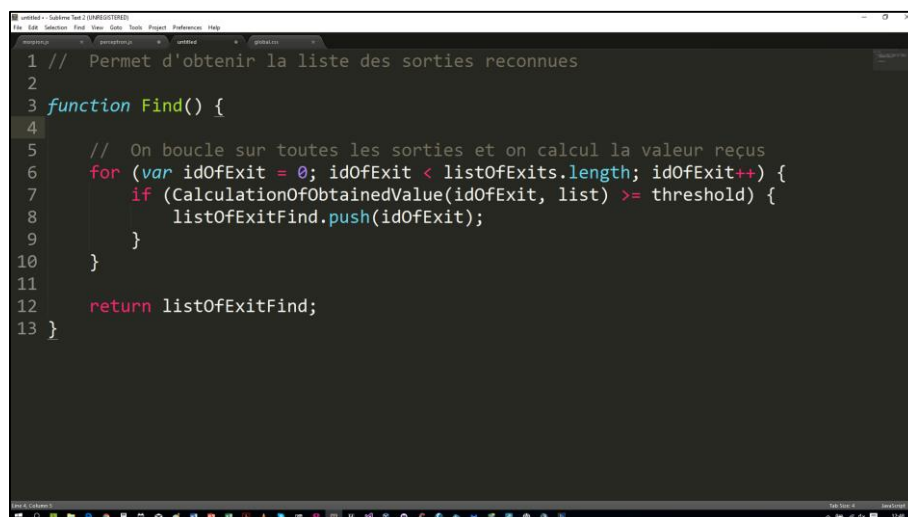
1 // Permet de calculer la valeur obtenue pour une sortie
2 // Paramètres :
3 //   ID de la sortie
4 //   Liste des neurones (optionnel)
5 function CalculationOfObtainedValue(idOfExit, list) {
6   var obtainedValue = 0;
7
8   // Valeur par défaut pour le paramètre List
9   if (typeof(list) == 'undefined') {
10    list = listOfNeurones;
11  }
12
13  // On boucle sur tous les neurones
14  for (var idOfNeurone = 0; idOfNeurone < list.length; idOfNeurone++) {
15
16    // On vérifie que le neurone est activé
17    if (list[idOfNeurone].input >= threshold) {
18      obtainedValue = obtainedValue + (list[idOfNeurone].input * list[idOfNeurone].weights[idOfExit]);
19    }
20  }
21
22  return obtainedValue;
23 }

```

La fonction « Find »

Cette fonction, bien que obligatoire, reste très simple. Son but est de trouver tous les chiffres reconnus par la fonction « Learn ». Elle fonctionne comme ceci :

- On boucle sur toutes les sorties
 - Si la valeur obtenue par tous les neurones actifs pour cette sortie (cf. fonction « CalculationOfObtainedValue ») est supérieur au seuil, alors le chiffre est reconnu.



```

1 // Permet d'obtenir la liste des sorties reconnues
2
3 function Find() {
4
5   // On boucle sur toutes les sorties et on calcul la valeur reçus
6   for (var idOfExit = 0; idOfExit < listOfExits.length; idOfExit++) {
7     if (CalculationOfObtainedValue(idOfExit, list) >= threshold) {
8       listOfExitFind.push(idOfExit);
9     }
10  }
11
12  return listOfExitFind;
13 }

```

Première utilisation du perceptron

Nous avons toutes les fonctions pour que notre perceptron fonctionne. Il faut néanmoins faire varier plusieurs paramètres afin d'obtenir un rendu acceptable.

Le nombre de sorties :

Moins il y en a, moins de temps il faudra pour les apprendre. Pour commencer, essayez d'apprendre uniquement le 1 et le 2.

Le taux d'apprentissage :

Il est compris en 0 et 1. Plus il est petit, plus l'apprentissage sera long mais sera fin. Plus il est grand, plus l'apprentissage est rapide mais les valeurs oscillent énormément (et permet aussi un plus large choix d'erreurs, j'y reviendrais plus tard). Une valeur initiale à 0.25 est un bon choix.

Pour le reste, j'utilise des valeurs arbitraires qui peuvent être modifié :

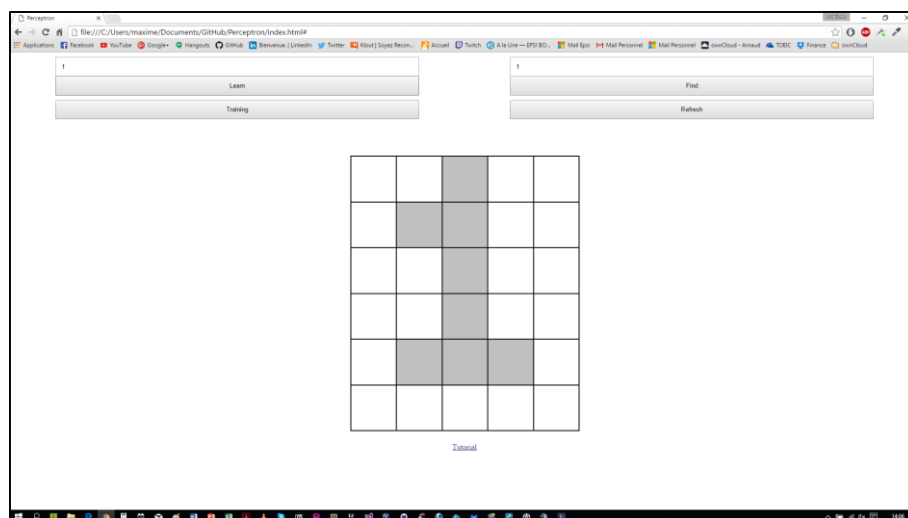
Le seuil : 0.5

La valeur d'un neurone actif : 1

La valeur d'un neurone inactif : 0

Le poids initial : 1

C'est avec tous ces paramètres, et les fonctions définis plus haut, le perceptron peut apprendre chaque chiffre en cliquant plusieurs fois sur « Learn ».



Développement avancé

La fonction « Training »

Avec notre perceptron fonctionnel, nous voulons forcément lui faire apprendre de plus en plus de chiffres. Vous verrez que l'apprentissage peut être rapide pour 2 chiffres (ne pas se décourager si l'apprentissage est long, la suite va vous permettre de savoir rapidement si l'apprentissage fonctionne), mais ça peut décourager beaucoup de personnes de cliquer une centaine de fois sur « Learn » en prenant le temps de vérifier que aucun chaque dessins correspond, ext...

La solution est simple : Automatiser l'apprentissage des chiffres.

Mise en place des nouvelles variables, et modification des fonctions

Pour ce faire, nous allons créer une nouvelle fonction, la fonction « Training ».

Pour commencer, il nous faut les différents modèles de chiffres. Nous allons donc créer un tableau à deux dimensions, qui contiendra la liste des ID de neurone activés pour dessiner tel ou tel chiffre, et la liste des neurones (contenant les neurones actifs ou non en fonction de la liste renseigné).

C'est lors de la création du perceptron (et de la grille), que nous initialiserons chaque nouvelles listes.

```
1 // Liste des chiffres
2 var listOfNumbers = [
3
4 // Chiffre 0
5 [[], [6, 7, 8, 11, 13, 16, 18, 21, 22, 23]],
6
7 // Chiffre 1
8 [[], [2, 6, 7, 12, 17, 21, 22, 23]],
9
10 // Chiffre 2
11 [[], [1, 2, 3, 8, 11, 12, 13, 16, 21, 22, 23]],
12
13 // Chiffre 3
14 [[], [1, 2, 3, 8, 12, 13, 18, 21, 22, 23]],
15
16 // Chiffre 4
17 [[], [1, 6, 8, 11, 12, 13, 14, 18, 23]],
18
19 // Chiffre 5
20 [[], [1, 2, 3, 6, 11, 12, 13, 18, 21, 22, 23]],
21
22 // Chiffre 6
23 [[], [1, 2, 3, 6, 11, 12, 13, 16, 18, 21, 22, 23]],
24
25 // Chiffre 7
26 [[], [1, 2, 3, 8, 12, 13, 14, 18, 23]],
27
28 // Chiffre 8
29 [[], [1, 2, 3, 6, 8, 11, 12, 13, 16, 18, 21, 22, 23]],
30
31 // Chiffre 9
32 [[], [1, 2, 3, 6, 8, 11, 12, 13, 18, 21, 22, 23]]
33 ]
```

```
1
2 for (var idOfNumber = 0; idOfNumber < listOfNumbers.length; idOfNumber++) {
3   if (listOfNumbers[idOfNumber][1].indexOf(idOfNeurone) != -1) {
4     listOfNumbers[idOfNumber][0].push(new Neurone(activeInput, listOfWeights));
5   }
6   else {
7     listOfNumbers[idOfNumber][0].push(new Neurone(disabledInput, listOfWeights));
8   }
9 }
```

Nous allons également modifier toutes nos anciennes fonctions qui utilisent la liste de neurone initial pour qu'elles puissent aussi se baser sur nos nouvelles listes. Ainsi, il suffit de rajouter un paramètre « list ».

Développement de fonction

Le principe de cette fonction est de réaliser toutes les étapes d'un test classique, et de boucler jusqu'à ce que tous les tests soient bons.

Pour simplifier l'apprentissage, nous allons également modifier la fonction « Learn » : Tant que le chiffre à apprendre n'est pas appris, on recommence.

```

1 // Permet d'apprendre un chiffre en utilisant à jour les poids des neurones
2 // Paramètres
3 // ID de la sortie à apprendre
4 // Liste des neurones (optionnel)
5 function Learn(idOfExitToLearn, list) {
6   var numberIsLearn = false;
7
8   // Valeur par défaut pour le paramètre list
9   if (typeof(list) === "undefined") {
10     list = listOfNeurons;
11   }
12
13   // On boucle tant que le chiffre n'est pas appris
14   while(numberIsLearn == false) {
15
16     // On boucle sur toutes les sorties
17     for (var idOfExit = 0; idOfExit < listOfExits.length; idOfExit++) {
18
19       // On définit la valeur attendue
20       if (idOfExit == idOfExitToLearn) {
21         var expectedValue = 2 * threshold;
22       } else {
23         var expectedValue = disabledInput;
24       }
25
26       // On boucle sur tous les neurones
27       for (var idOfNeuron = 0; idOfNeuron < list.length; idOfNeuron++) {
28
29         // On définit la valeur obtenue
30         var obtainedValue = CalculationOfObtainedValue(idOfExit, list);
31
32         // On met à jour le poids
33         list[idOfNeuron].weights[idOfExit] = list[idOfNeuron].weights[idOfExit] + (expectedValue - obtainedValue) * list[idOfNeuron].input * learningRate;
34       }
35
36       // On récupère la liste des chiffres reconnus
37       var listOfExitsFind = Find(list);
38
39       // On vérifie si au moins 1 chiffre est appris
40       if ((listOfExitsFind.length == 1 && listOfExitsFind[0] == idOfExitToLearn)) {
41         numberIsLearn = true;
42       }
43     }
44   }
45 }

```

Le principe de la fonction « Training » est le suivant :

- On boucle tant que tous les chiffres ne sont pas appris
 - On boucle sur toutes les sorties afin de les apprendre
 - On boucle sur toutes les sorties afin de vérifier que tous les chiffres sont appris
 - Si au moins 1 chiffre n'est pas appris, on recommence

En plus d'apprendre tous les chiffres, cette fonction nous permet d'afficher en combien de clique, nous apprenons tous les chiffres.

```

1 // Permet d'apprendre automatiquement les chiffres
2 S("#training").click(function() {
3   var numberAreLearn = false;
4   var numbersCompteur = 0;
5
6   // On boucle tant que tous les nombres ne sont pas appris
7   while(numberAreLearn == false) {
8
9     var listOfTotalExitsFind = [];
10
11     // On boucle sur toutes les sorties
12     for (var idOfExit = 0; idOfExit < listOfExits.length; idOfExit++) {
13
14       var listOfNumbersFind = Find(listOfNumbers[idOfExit][0]);
15
16       // On ne fait apprendre le chiffre que si c'est nécessaire
17       if (listOfNumbersFind.length != 1 || listOfNumbersFind[0] != idOfExit) {
18         Learn(idOfExit, listOfNumbers[idOfExit][0]);
19         numbersCompteur++;
20       }
21     }
22
23     // On boucle sur toutes les sorties
24     for (var idOfExit = 0; idOfExit < listOfExits.length; idOfExit++) {
25
26       var listOfNumbersFind = Find(listOfNumbers[idOfExit][0]);
27
28       // On définit comme "appris" un chiffre si le chiffre sortant après le find est celui attendu
29       if (listOfNumbersFind.length == 1 && listOfNumbersFind[0] == idOfExit) {
30         listOfTotalExitsFind.push(true);
31       }
32     }
33
34     // On vérifie que tous les chiffres sont appris
35     if (listOfTotalExitsFind.length == listOfExits.length) {
36       numberAreLearn = true;
37
38       alert("End of training in " + numbersCompteur + " learn to " + listOfExits.length + " numbers");
39     }
40   }
41 })

```

Etude des résultats

A ce moment, nous pouvons apprendre jusqu'à 10 chiffres (0 à 9), le perceptron est donc totalement fonctionnel. Il est temps de l'optimiser.

Diminuer le nombre de d'apprentissages

Le premier axe de recherche a été de faire diminuer le nombre de clique nécessaire à l'apprentissage des 10 chiffres. J'ai donc fait varier le taux d'apprentissage et le poids initiale jusqu'à obtenir des résultats satisfaisants.

Avec le taux d'apprentissage à 0.07 et le poids initial à 0, le perceptron apprend les 10 chiffres en 169 cliques au lieu de 446 cliques pour la configuration que je vous ai donnée plus haut.

Ainsi nous apprenons plus vite, mais après vérification, le taux d'erreur est très faible. En effet, si pixel est retiré ou ajouté du model initial, Il y a plus de 80% de chance que le perceptron ne reconnaisse plus le chiffre. Ceci nous amène au deuxième axe de recherche.

Augmenter le taux d'erreur

Pour permettre à un modèle d'être reconnue avec plus ou moins d'approximation, la première solution est d'augmenter le taux d'apprentissage (0.25). Ainsi, on peut ajouter ou retirer plusieurs pixels et avoir un perceptron qui reconnaît tout de même les différents chiffres.

La seconde solution serait d'ajouter aléatoirement des pixels actifs, permettant un étalonnage des poids qui prend en compte les pixels en plus. Cette méthode fonctionne aussi avec des pixels en moins.

La troisième solution est de faire comme si le chiffre ainsi dessiné a une ombre autour de lui. En rendant tous les neurones aux alentours actifs, le perceptron prendra en compte les erreurs de quelques pixels d'écarts et continuera à reconnaître les chiffres. Cette méthode demande tout de même une résolution élevée (nombre de pixels grand) et elle est compliquée à mettre en place sur une résolution de $6 * 5$ comme la nôtre.