Date : mercredi 31 décembre 2014

# I4 Perceptron - Documentation technique

---

L'objectif du TP est de créer un réseau de neurones artificiels pour répondre au problème de la reconnaissance de chiffres dessinés dans un canevas de pixels. Chaque pixel a deux états possibles : « activé » ou non. Il fut conseillé d'utiliser un **perceptron**, le modèle le plus simple des réseaux neuronaux. L'**apprentissage supervisé** est quant à lui imposé.

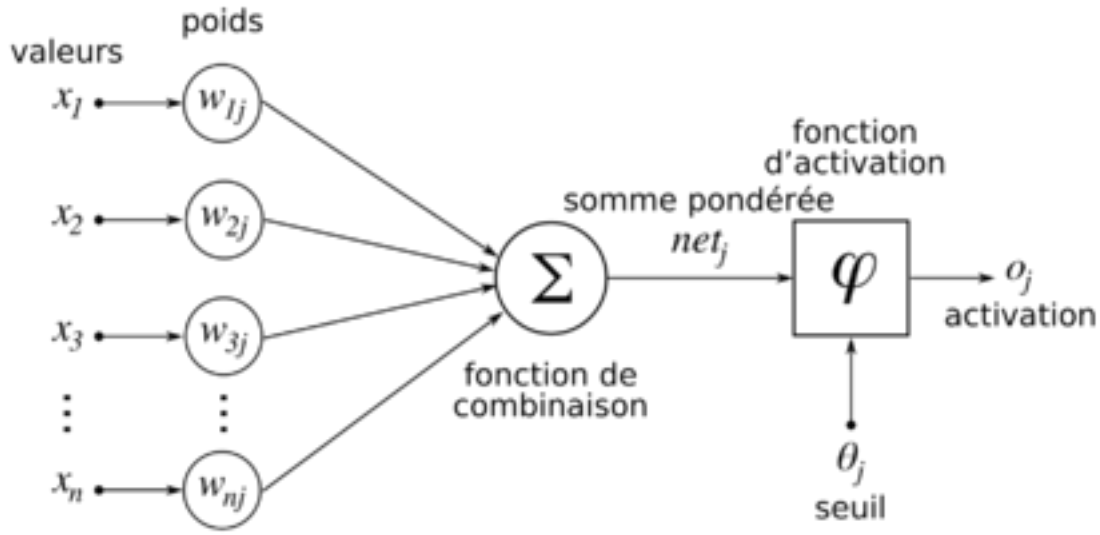
Cette documentation vous aidera à comprendre la logique suivie pour répondre au sujet. D'une part, les raisons pour lesquelles on optera pour un réseau **feed-forward** plutôt qu'un simple perceptron seront exposées plus en détail. D'autre part, l'algorithme de passage d'une matrice NxM à une matrice 5x3 sera détaillé.

A la différence de l'algorithme de passage (non demandé par le sujet), le code du réseau à couche multiple présent dans le dossier src > models > mlp peut être réutilisé pour traiter d'autres projets que celui énoncé. Les classes ont été créées dans l'optique d'être réemployées pour n'importe quel problème ayant recourt à un MLP (Multiple Layer Perceptron).

## I. Le réseau feed-forward

### A. Le neurone artificiel

Un réseau de neurones est en général composé d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente. Le perceptron désigne un réseau de neurones ne contenant aucun cycle. Sa version la plus simplifiée est un réseau mono-couche avec une seule sortie. Le perceptron du OR en est un exemple.

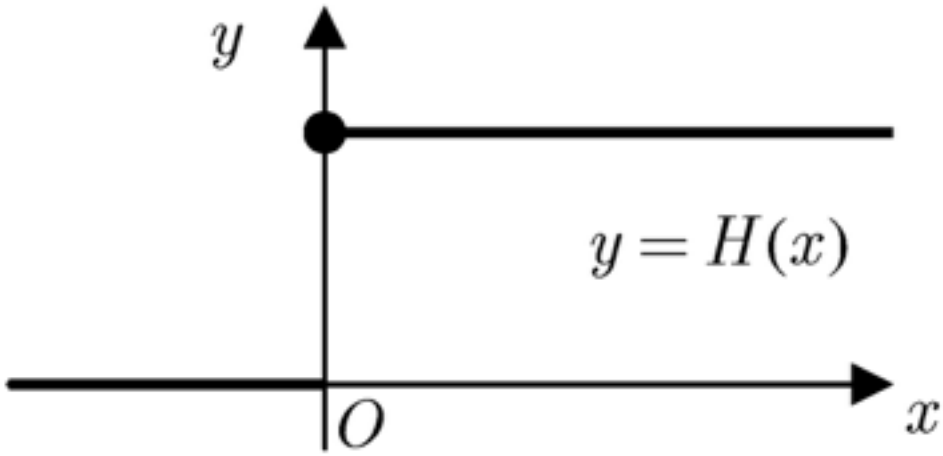


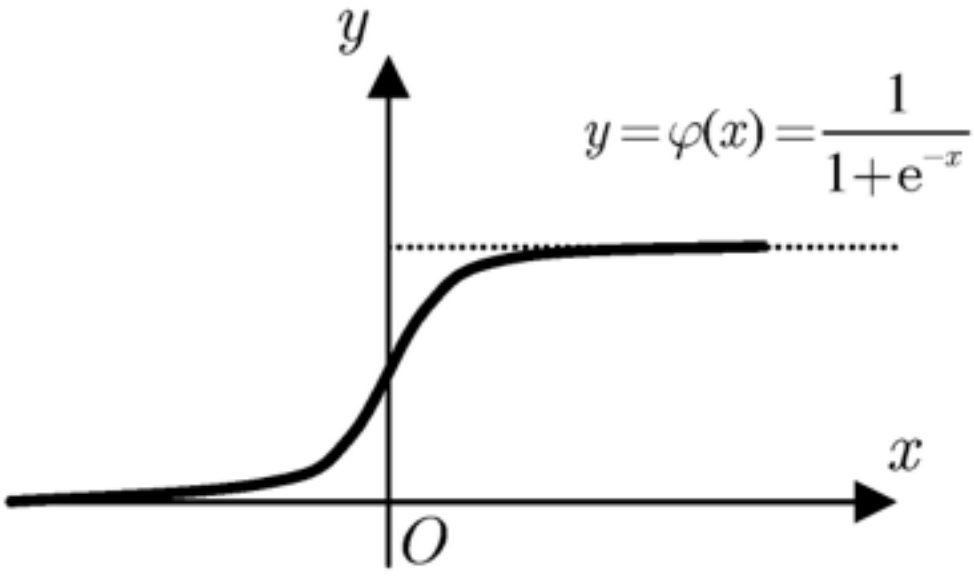

Un **neurone artificiel**, appelé aussi **neurone formel,** reçoit des valeurs d'entrées, traite ces dernières et retrouve une valeur de sortie qui peut différer selon un seuil désigné. De ce fait, l'objet **Neuron** est construit comme ci-dessous :

- **nbInputs** : le nombre d'entrées

- **weights** : le tableau des poids associés. Chaque poids est initialisé avec un décimal compris entre -1 et 1.
- **output** : la valeur de sortie
- le seuil d'activation, ici appelé **biais**, est stocké dans le tableau **weights**, à l'indice **nbInputs**, d'où une initialisation jusqu'à nbInputs + 1 exclu.

La fonction d'activation **Evaluate** utilise une fonction de transfert de type **sigmoïde**. Cette dernière permet d'obtenir une valeur comprise entre 0 et 1.De plus sa valeur en 0 étant à 0,5 permettrait de faciliter les traitements si on souhaite retourner uniquement deux valeurs bien distinctes. On obtiendrait alors un comportement similaire à la fonction **Heavyside**. Or pour la reconnaissance des chiffres, nous aurons besoin de savoir à quel point la valeur de sortie est forte. L'heavyside n'est donc pas la plus appropriée.

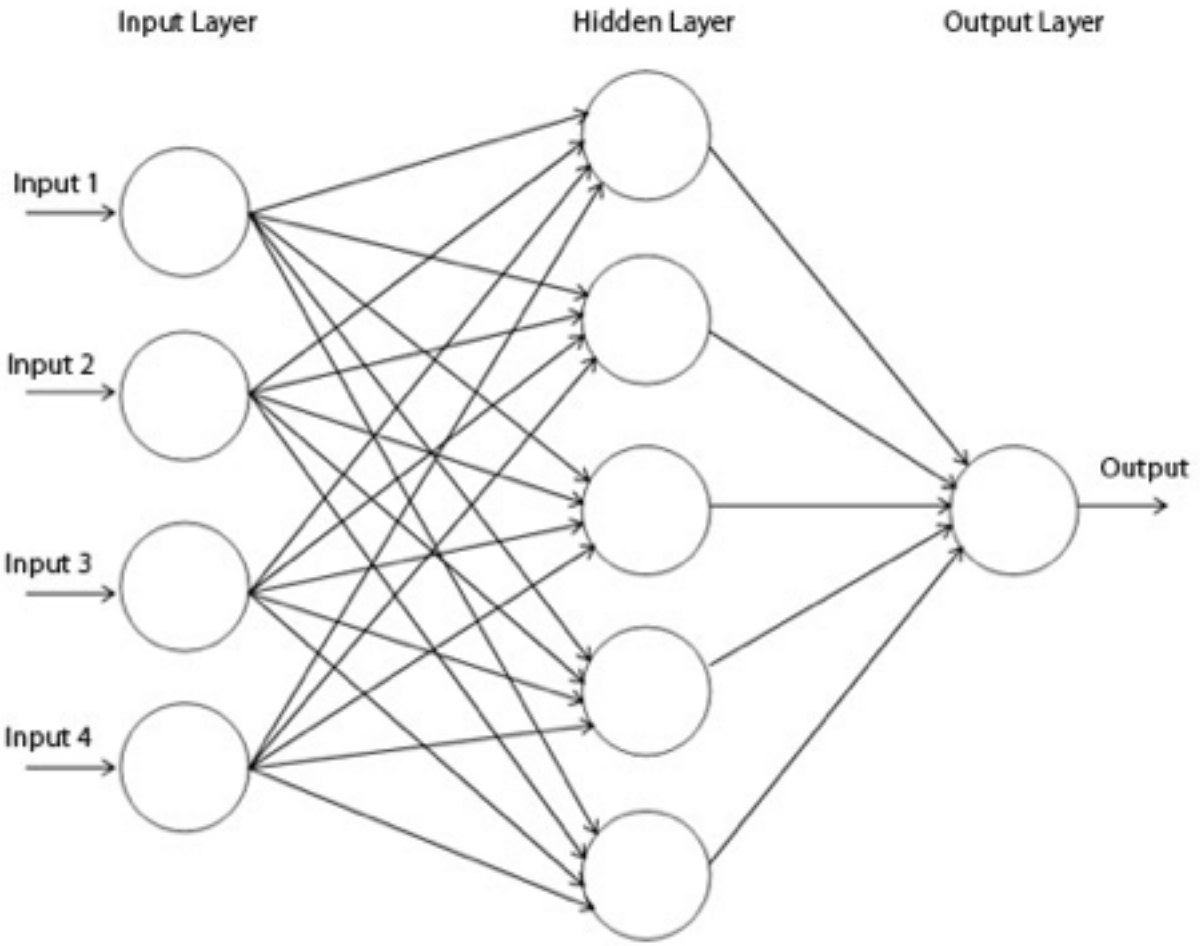


Pour ce TP, nous garderons les valeurs décimales retournées par la sigmoïde, permettant ainsi de créer une reconnaissance de 10 chiffres. Ceci sera abordé plus en détail ultérieurement. Notons principalement qu'une **sigmoïde** permet de **savoir plus facilement vers quelle direction on doit s'orienter pour améliorer les résultats**.

## B. L'apprentissage

Le perceptron est un réseau adapté pour un **problème linéairement séparables**. Or pour la reconnaissance de chiffres, cela sous-entend qu'un pixel activité ne serait associé qu'à un chiffre uniquement. De ce fait, on souhaite dépasser cette limite et donc nous utiliserons un réseau **feed-forward**, nommé également **réseau à couches.** Le nom feed-forward provient du fait que l'information ne va que dans un sens : de l'entrée (inputs) vers la sortie (output). Ce type de réseau est composé d'une ou plusieurs couches de neurones cachés.




On a 10 chiffres à reconnaître grâce à un nombre de pixels modifiable. Or un algorithme de traitement expliqué dans la seconde partie permet de toujours récupérer une matrice 5x3. On a alors **15 entrée possibles**. Le **nombre de neurones cachés est fixé à 10** pour coïncider aux 10 chiffres possibles, ceci dit, on peut obtenir des résultats satisfaisant avec plus ou moins de neurones cachés. Par conséquent, on a **un seul neurone de sortie** pour désigner le chiffre reconnu. Pour résumer, on utilisera une **réseau de neurones 15 - 10 - 1**

L'algorithme d'apprentissage de **Widrow-Hoff** n'est adapté que pour le perceptron. Nous emploierons donc un **apprentissage par rétropropagation du gradient**. En anglais on emploie le mot **« backpropagation ».** Ceci permet de corriger les poids entre les neurones de sorties et les neurones cachés dans un premier temps puis de propager l'erreur en arrière tout en corrigeant les poids entre les neurones cachés et les entrées.

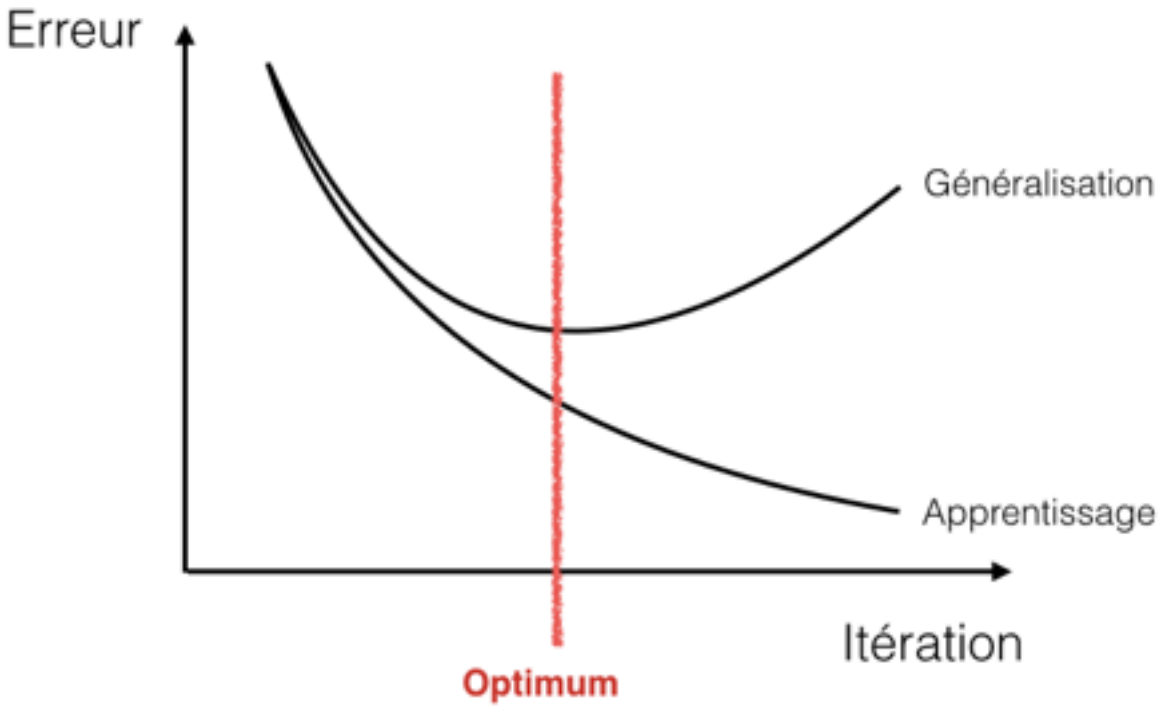
La nouvelle valeur du poids est calculée grâce à la formule vue en classe :

$$P = P + t * (A - O) * E$$

P = Poids de la connexion
t =Taux d'apprentissage
A = Sortie attendue
O = Sortie obtenue
E = Entrée

Vous trouverez dans le code une notion de « delta » qui correspond en fait à « (A-O) ». Comme les traitements sont un peu plus complexes avec la présence de couches cachés, les tableaux de deltas permettent de rendre le code plus lisible.

Pour terminer notre partie apprentissage, nous porterons notre attention sur la problématique du **surapprentissage**. On suppose qu'au bout d'un certain temps, à partie d'une certaine itération, le système de neurones arrive à des performances optimales pour ensuite perdre de son efficacité et régresser.




De ce fait, l'historique de totalGeneralisationError et betterGeneralisation permet de repérer quand le suraprentissage apparait. On stoppe alors le processus d'apprentissage pour rester à l'optimum.

Les performances de réseau de neurones sont visibles et testables dans l'onglet

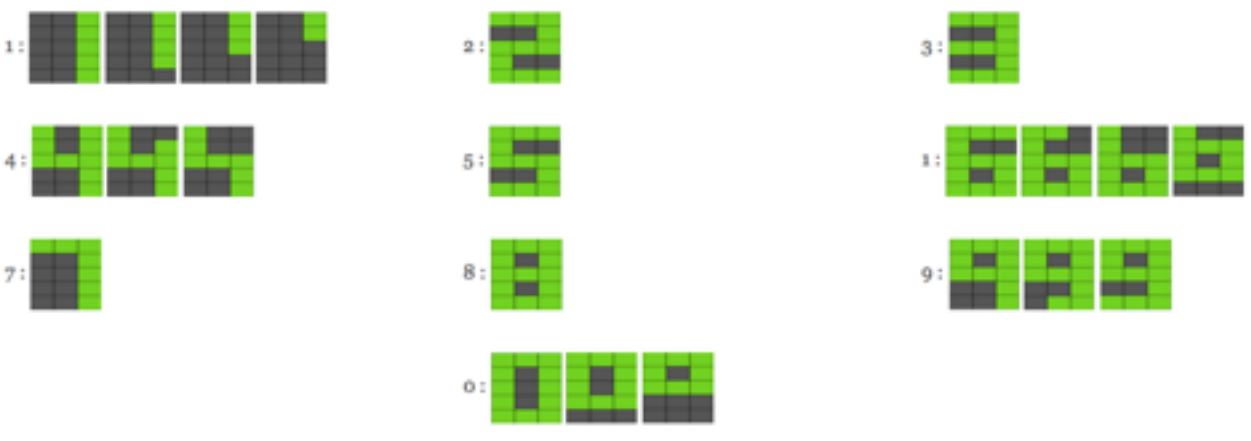

**« Multiple Layer Perceptron tests ».** Vous pouvez visualisés les paramètres estimés comme étant les plus optimisés suite aux nombreux tests.

La valeur de sortie finale est le chiffre reconnu. Elle est obtenue par la multiplication du taux par 10 puis arrondi à l'entier le plus proche. Vous pouvez observer que les taux obtenus pour chaque chiffre sont plus que remarquables.

0 : 0.01044404260374806 8
1 : 0.09729272189943028
2 : 0.20047699550409695
3 : 0.2990762315841404
4 : 0.40127027349643035
5 : 0.500376913660931
6 : 0.5983468647249116
7 : 0.6992660409513305
8 : 0.8003341875484068
9 : 0.8914532126691013

## II. Algorithme de passsage

L'autre problématique de ce TP réside dans la reconnaissance pure du nombre. En effet si on se contente d'apprendre un nombre dans la matrice où l'utilisateur peut dessiner alors le nombre appris ne sera repéré que s'il est dessiné au même endroit. Par conséquent, un algorithme de passage a été développé pour toujours récupérer une matrice de 5 lignes sur 3

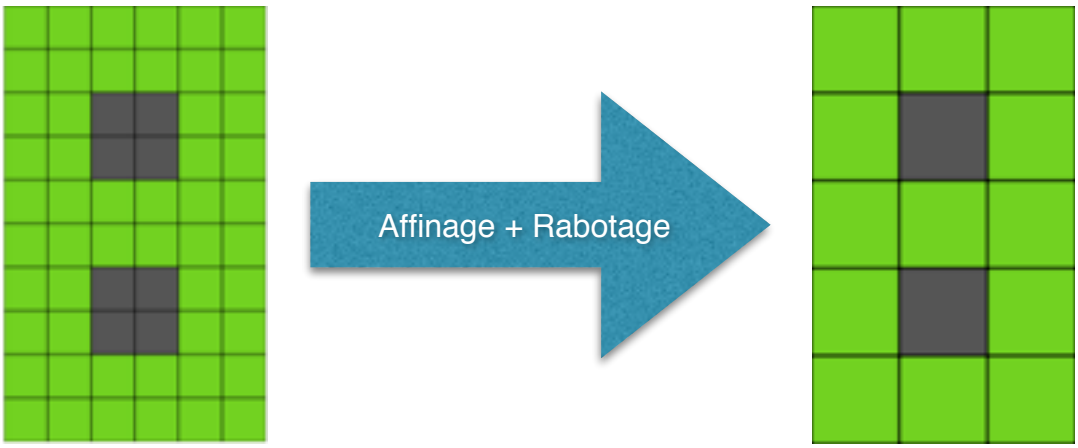


colonnes. Une matrice 5x3 correspond à la disposition minimal pour écrire les 10 chiffres au format digital.

Dans un premier temps on ne garde que la matrice la plus petite possible. Les pixels activités dessinent un nouveau cadre avec lequel on travaillera. On enlève alors les pixels en dehors de ce cadre, les non activés superflus. Cette opération est l'équivalent du bouton « découper la sélection » d'un éditeur d'image.

Si la matrice obtenue est plus petite qu'une 5x3, on rajoute des lignes en bas ou des colonnes à gauche. Ce comportement a été pensé avec le chiffre « 1 » qui est le plus problématique. Il parait plus naturel de dessiner un « 1 » à droite d'une case plutôt qu'à gauche. Quant au fait de coller le chiffre le plus en haut, on réfléchira plus sous un aspect mathématiques et performances machines. En effet, il est plus rapide de rajouter une ligne à la fin d'un tableau plutôt que de décaler toutes les existantes puis remplir la première avec les nouvelles valeurs.

Si la matrice est plus grande, on exécute un pré-traitement d'affinage et de rabotage.




Ces deux actions permettent d'obtenir un chiffre avec les traits les plus fins possible. Voici l'exemple d'un « 8 » pour une illustration plus concrète :

Dans le cas où le la matrice d'origine est proportionnelle, on fait simplement le rapport en hauteur et en largeur. De ces coefficients, on découpe la matrice d'origine en parties de coefficientHauteur x coefficientLargeur pixels. La valeur du nouveau pixel est « activée » si la somme des pixels activés de la partie est égale ou supérieur à la majorité.

Dans le cas contraire, on récupère le quotient et le reste du nombre de lignes et de colonnes modulo le nombre minimum de lignes et de colonnes respectivement. Les traitements horizontaux et verticaux sont différents.

Pour le nombre de colonnes, on ne note que deux cas :

- le reste est égal à 1 : on rajoute une ligne en bas pour respecter la logique énoncée ci-dessus
- le reste est égal à 2 : on rajoute une ligne en bas et en haut pour ne pas générer un écrasement de l'image. Ensuite les opérations sont les mêmes que ceux décrits dans le cas d'une matrice proportionnelle permet de conserver l'information.

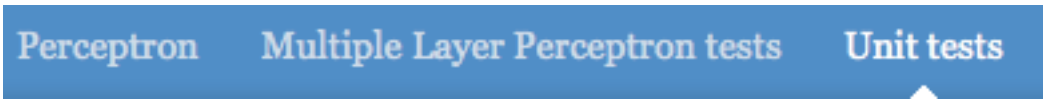
Pour le nombre de ligne, les situations sont plus nombreuses :

- le reste est égale à 1 : on tronque la ligne qui a le moins de pixel activés. Il est plutôt probable que cette ligne soit en fait un allongement d'un segment.
- le reste est égal à 2 : on exécute le même traitement que si le reste était égale à 1 et cela deux fois

- le reste est égale à 3 : par soucis de non création d'effet d'écrasement et de non perte d'informations, une ligne sera créée en haut et en bas.
- le reste est égal à 4 : une ligne est rajoutée en bas.

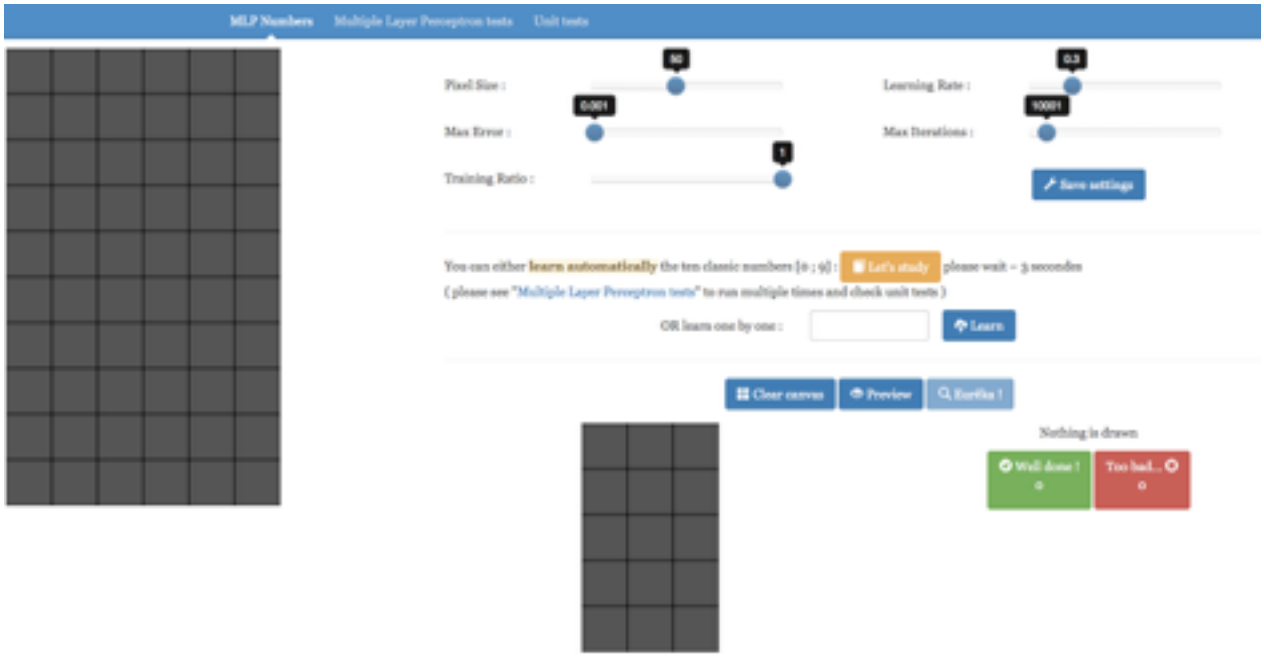
Il est évident que cet algorithme est perfectible, cependant il permet de reconnaître les chiffres déplacés, rétrécis ou agrandis avec un taux de réussite à 100%.

Tous les cas permettant de tester chaque comportement énoncé sont traduits en tests unitaires consultables dans l'onglet « **Unit Test** »




Afin de réaliser la reconnaissance des chiffres, un apprentissage supervisé est réalisable grâce à réseau feed-forward ainsi qu'un algorithme de pré-traitement de matrices. Je vous invite à tester la solution proposée avec l'interface « **Perceptron** ». Cette dernière vous permet de modifier les paramètres du système de neurones, de lancer un **apprentissage automatique massif** ou d'exécuter une apprentissage ponctuel, de visualiser le résultat de l'algorithme de pré-traitement et enfin de vérifier si le chiffre dessiné est bien reconnu. Les informations complémentaires sur les poids de chaque neurones sont disponibles pour les plus curieux.

SI vous souhaitez uniquement évaluer les performances de réseau de neuronnes, sans l'algorithme de pré-traitement, je propose de vous concentrer sur l'onglet « **Multiple Layer Perceptron tests** ».




**Perceptron information :**