Mini-Rapport

Exploration de la notion de méta-apprentissage

Dans quelle mesure un système apprenant peut prendre conscience de ses performances et altérer son comportement ?

Yann Boniface, Alain Dutech, Nicolas Rougier
Matthieu Zimmer

23 avril 2012

1 Introduction

Notre intérêt s'est tourné vers l'article [Cleeremans Alex, 2007] et ses 2 types de réseaux proposés. Dans un premier temps, nous avons cherché à reproduire et expliquer les résultats donnés, et ensuite à des solutions pour tirer profit des paris réalisés.
Nous nous sommes également penchés sur [Pasquali Antoine, 2010] dont nous avons reproduit les expériences, mais leurs enjeux nous semblent encore vagues.

2 Dupliquer le premier réseau

2.1 Les bases

En premier lieu, rappelons la structure des réseaux et les résultats de l'article :

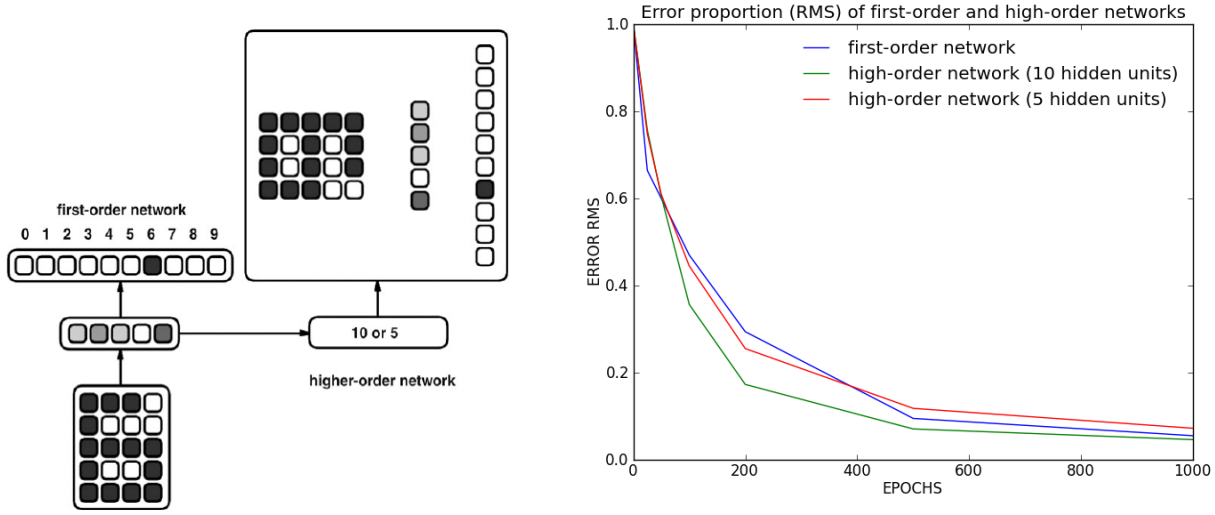

FIGURE 1 – [Cleeremans Alex, 2007] Architecture connexionniste avec méta-représentations

La formule RMS utilisée à une époque e est la suivante :

$$rms\ proportion_e = \frac{rms_e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(o_{i,e} - d_i)^2}}{max(rms_{e'}),\ \forall e' \in epochs}$$

$$with \begin{cases} n : number\ of\ neurons\ on\ the\ output\ layer \\ o_{i,e} : value\ obtained\ for\ the\ i^{th}\ neuron\ at\ the\ e^{th}\ epoch \\ d_i : value\ desired\ for\ the\ i^{th}\ neuron \end{cases}$$

Nous avons décomposé les erreurs pour mieux comprendre le fonctionnement de l'architecture.

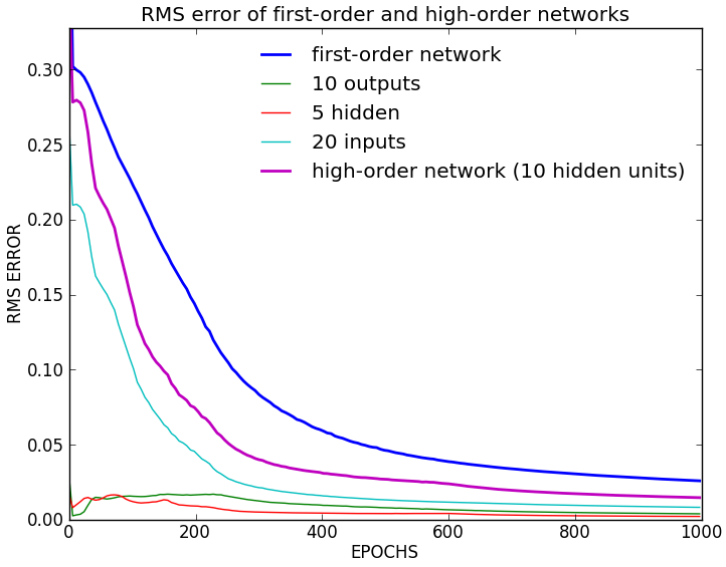

FIGURE 2 – **Erreur RMS sans proportion**. Le réseau supérieur à 5 unités cachées n'est plus considéré, et celui à 10 est découpé en 3 courbes pour représenter les 3 couches à reproduire. La courbe violette est donc la somme des 3 courbes des couches.

On peut en conclure 2 choses :

- la couche caché et la couche de sortie ne posent aucuns problèmes d'apprentissages
- les performances du second réseau dépendent principalement de sa capacité à reproduire les entrées

Il reste une question en suspend :

Pourquoi le second réseau apprendrait-il plus rapidement que le premier ?

Nous n'avons pas la réponse. Par contre, nous pouvons nous intéresser à l'erreur de classification.

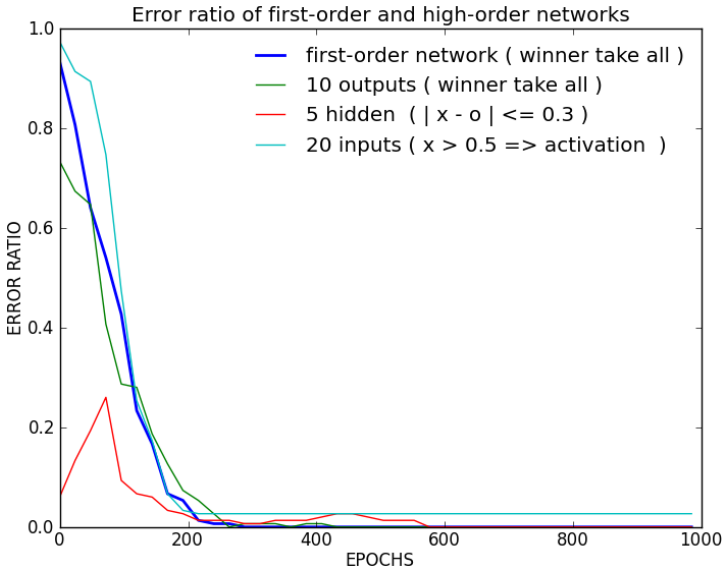

FIGURE 3 – **Erreur de classification**. Le réseau supérieur à 5 unités cachées n'est plus considéré, et celui à 10 est découpé en 3 courbes pour représenter les 3 couches à reproduire.

Il est alors beaucoup moins flagrant que le second réseau apprend mieux que le premier. Évidemment, parler de classification de la premier couche (courbe 20 inputs) et de la couche caché (courbe 5 hidden) est relativement dérisoire.
La question de la rapidité d'apprentissage reste donc ouverte.

2.2 Les rouages

Notre attention s'est également portée sur les mécanismes qui permettaient la réalisation de cette architecture. Nous avons alors pu remarquer que les neurones de la couche cachée du premier réseau se stabilisaient très rapidement (autour de la $50^{\text{ième}}$ époque en moyenne), le tout permettant au second réseau d'avoir des entrées très peu variables, favorisant et permettant donc son apprentissage.

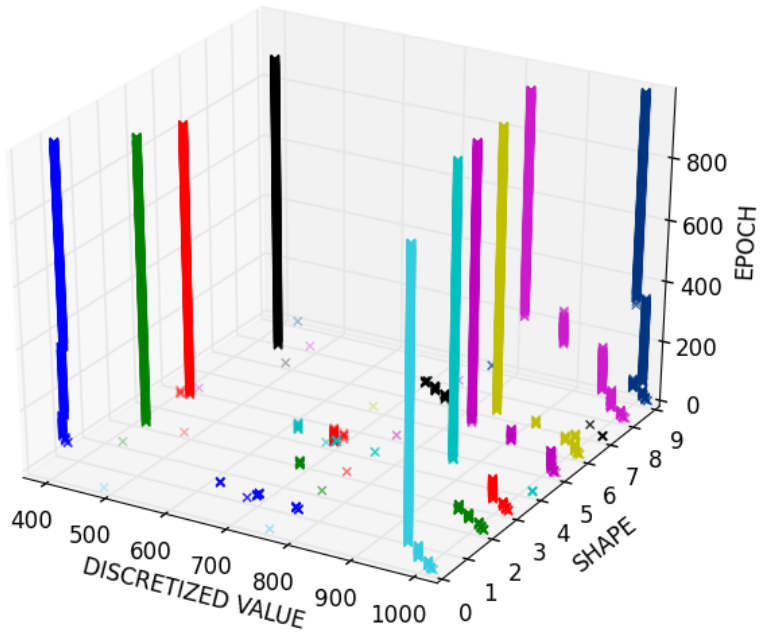

FIGURE 4 – Valeurs discrétisés de la couche caché du premier réseau. Chaque couleurs représentant un des 10 chiffres en entrée. Les courbes deviennent rapidement stables.

2.3 Ré-haussement

Pour revenir à l'importance des entrées, nous avons réalisé que le second réseau n'était capable de dupliquer le premier qu'uniquement parce que ces entrées sont triviales. Ainsi nous avons augmenté le nombre d'entrées en passant sur des chiffres manuscrits [Semeion Research Center, 1994], tout en augmentant proportionnellement le nombre de neurones des couches cachées :

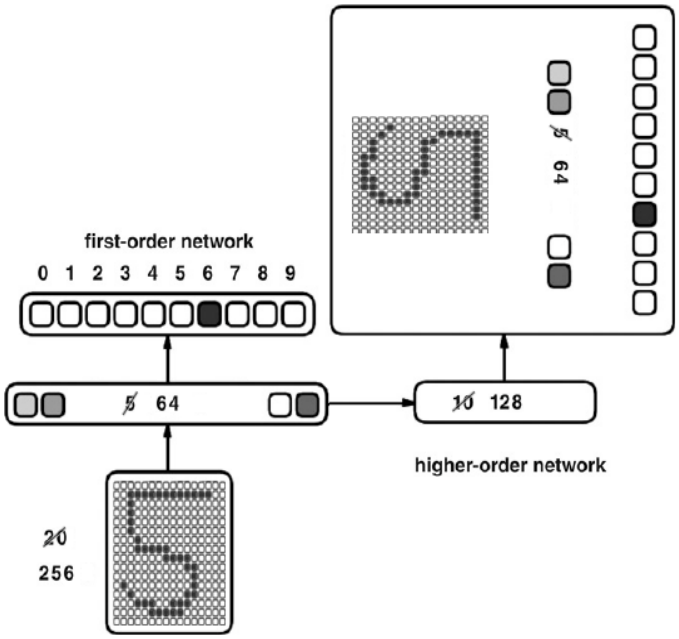

FIGURE 5 – Architecture avec méta-représentation pour chiffres manuscrits

Les performances du second réseau se sont alors écroulées :

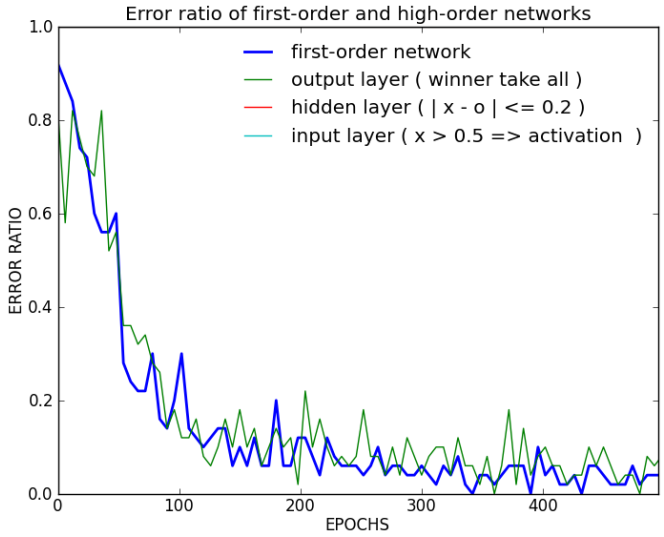

FIGURE 6 – **Erreur de classification** de l'architecture sur des chiffres manuscrits. L'erreur du second réseau est toujours divisé en 3.

FIXME Il n'est alors plus capable que de reproduire la couche de sortie. Petit bémol tout de même, le critère de classification pour la couche caché est peut-être trop rigide, il suffit qu'un neurone diffère de 0.2 (de la valeur voulue) pour considérer que c'est un échec.

Tentons d'expliquer cette chute :
La première chose à comprendre est que l'augmentation du nombre de neurone n'a rien à voir avec cette perte. En effet, nous avons fait les simulations avec les chiffres triviales

(seulement 10 formes différentes) agrandit en 16×16 et tout se passe très bien.
Cette chute provient du fait que les entrées proposées sont très nombreuses. Il s'opère ainsi une première étape de classification dans la couche caché du premier réseau. Donc différentes formes peuvent être représentées par la même couche cachée. Sauf que le second réseau n'ayant en entrée que la version factorisé (la couche caché), il lui est impossible de retrouver les formes initiales.

2.4 Représentations

Enfin, nous avons remarqué qu'en bloquant l'apprentissage entre la couche cachée et les entrées du premier réseau, puis en changeant de tâche, le réseau était capable de réapprendre la nouvelle tâche, ce qui prouve bien la présence d'une représentation des entrées dans la couche cachée.

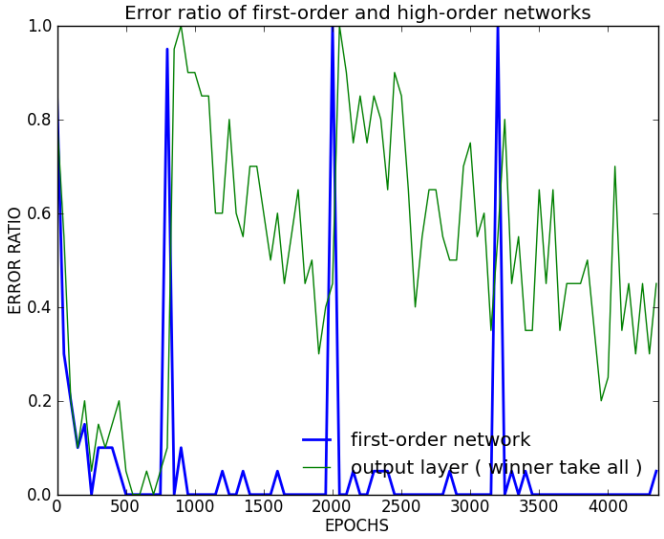

FIGURE 7 – **Erreur de classification** de l'architecture sur des chiffres manuscrits. Apprentissage bloqué à l'époque 800. Changement de tâche aux époques : 800, 2000, 3200

2.5 Remarque

Il faut cependant remarquer qu'un simple perceptron est suffisant pour réaliser la tâche du premier réseau (même sur les chiffres manuscrits), et donc, qu'il est possible que dans le cas d'un problème non linéairement séparable cette architecture soit invalidée.

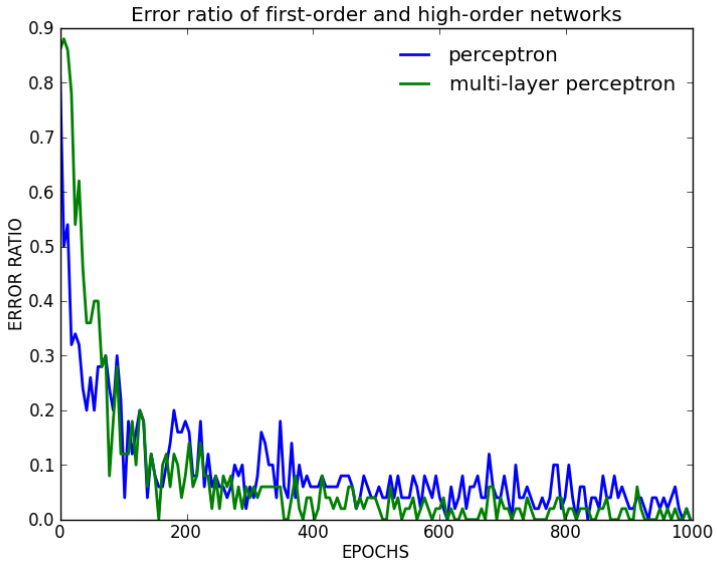

FIGURE 8 – **Erreur de classification** d'un perceptron et d'un perceptron multi-couches sur la base de chiffres manuscrits.

3 Parier sur le premier réseau

3.1 Les bases

Rappelons également la structure des réseaux et les résultats :

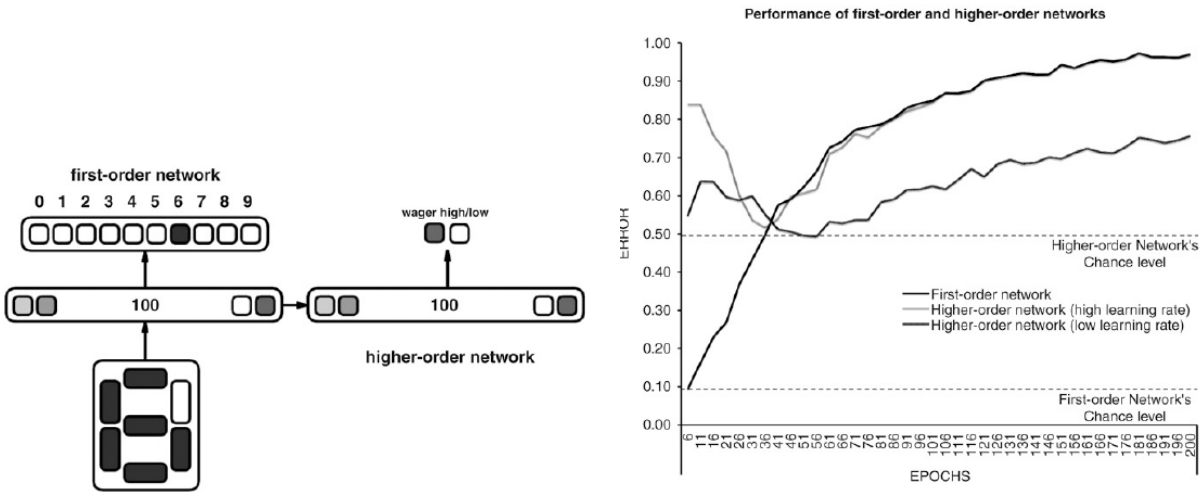

FIGURE 9 – [Cleeremans Alex, 2007] Architecture connexionniste avec paris

La première chose que nous avons faites à été d'améliorer les performances du second réseau en modifiant quelques paramètres (initialisation des poids sur [-1 ; -1], momentum à 0.5).

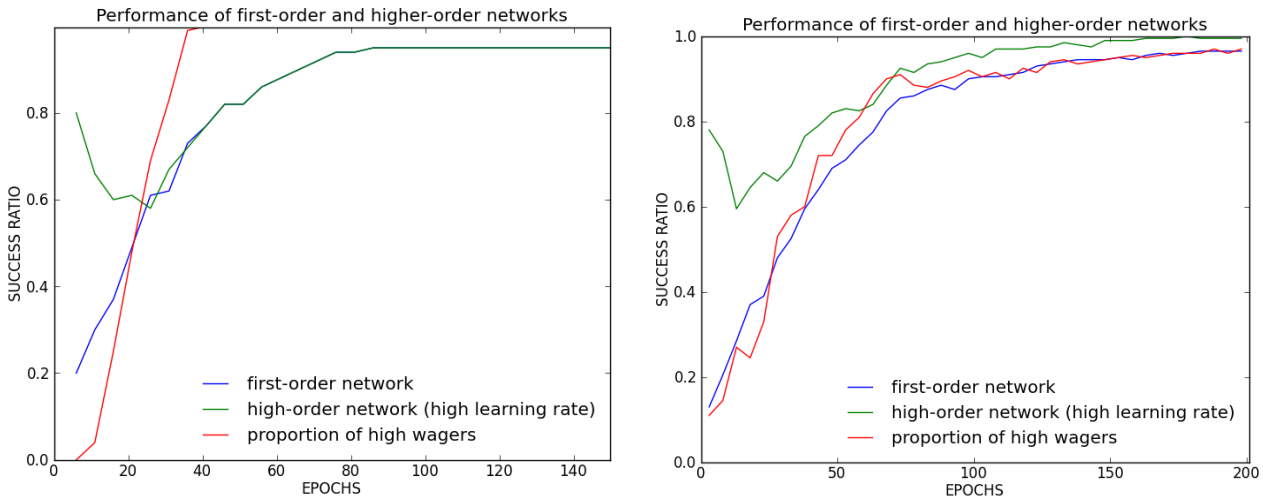

FIGURE 10 – Performance de l'architecture. À gauche la base, à droite la version améliorée. On ne considère plus le réseau avec apprentissage faible, mais on a ajouté le taux de paris hauts.

Contrairement à celui de l'article, il ne se contentera plus simplement de parier haut à chaque coups (après 40 époques). Il aura une longueur d'avance sur le premier réseau sur toute la durée de l'apprentissage. On pourra donc tirer profit de cette avance.

3.2 Feedbacks

À partir de cette différence de performances, nous avons imaginé plusieurs architectures, qui améliorent plus ou moins les performances de reconnaissance du réseau sur des chiffres manuscrits :

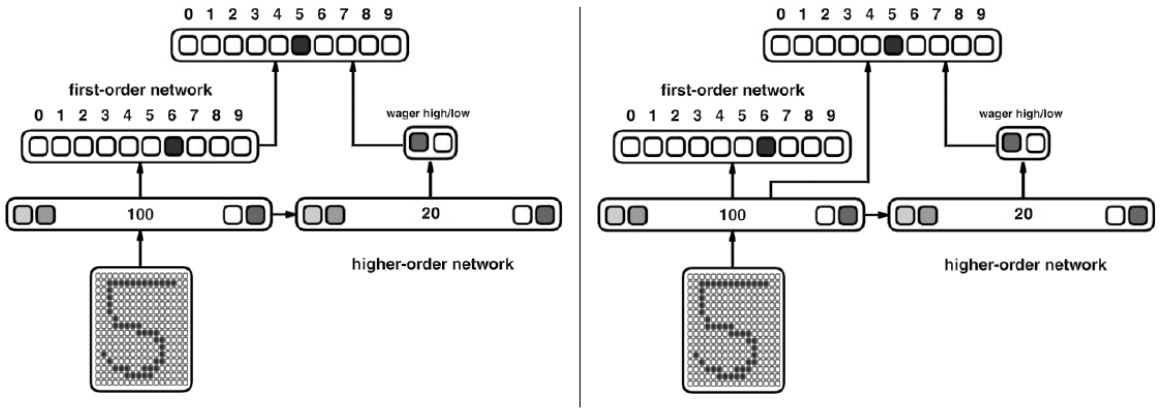

FIGURE 11 – Architecture avec 3$^{\text{ième}}$ réseau

Dans ces 2 architectures, nous nous contentons de connecter un 3$^{\text{ième}}$ réseau qui doit tirer des conclusions à partir d'informations sur les 2 premiers.

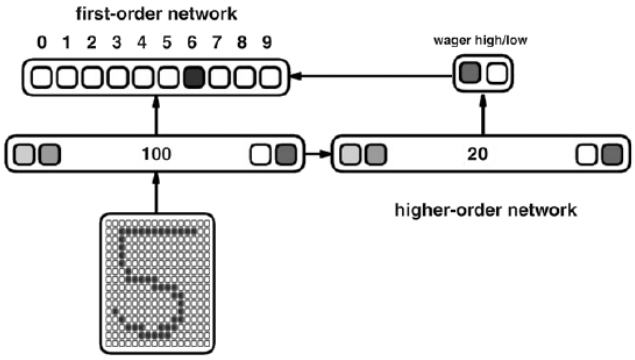

FIGURE 12 – Architecture par fusion

Ici, nous mélangeons un apprentissage par descente de gradient (sur le premier et second réseau) et un apprentissage perceptron (entre les 2 couches de sorties).

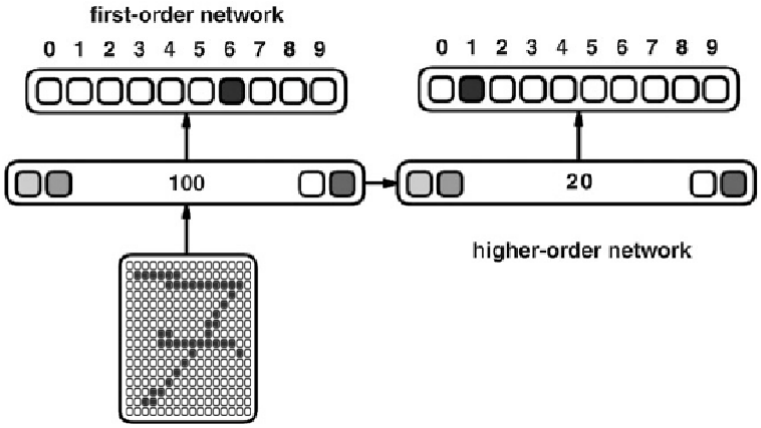

FIGURE 13 – Architecture par intuitions

Cette architecture est légèrement différente dans le sens où elle n'enregistre plus de pari mais l'indice du n$^{\text{ième}}$ neurone le plus actif contenant la bonne réponse. Exemple : le réseau supérieur sort 2 -> la réponse est le 3$^{\text{ième}}$ neurone le plus actif de la couche de sortie du premier réseau.

Nous avons aussi essayé quelques modèles où le réseau supérieur servait de superviseur à l'apprentissage du premier réseau. Par exemple, s'il parie haut, l'apprentissage du premier réseau sera faible, sinon il sera accentué.

4 La suite

Ce que nous continuons d'étudier :

– validation sur des expériences plus complexes (qui ne peuvent être résolue directement par un perceptron)

- relation entre la taille de la couche cachée du premier réseau et le taux de paris avantageux
- nouveau critère de classification pour la couche caché dans la première architecture sur les chiffres manuscrits
- approfondir les intérêts du second article [Pasquali Antoine, 2010]
- de nouvelles architectures axées méta-apprentissage où le réseau d'ordre supérieur contrôle le premier (taux d'apprentissage, momentum, entrées à approfondir, ...) telle une conscience

Références

[Cleeremans Alex, 2007] Cleeremans Alex, Timmermans Bert, P. A. (2007). Consciousness and metarepresentation : A computational sketch. *doi :10.1016/j.neunet.2007.09.011.*

[Pasquali Antoine, 2010] Pasquali Antoine, Timmermans Bert, C. A. (2010). Know thyself : Metacognitive networks and measures of consciousness.

[Semeion Research Center, 1994] Semeion Research Center, o. S. o. C. (1994). Semeion handwritten digit data set. 1593 handwritten digits from around 80 persons.

A Algorithme

Ensemble d'inscrutions executé pour une époque et un tirage d'entrées/sorties.
Cet algorithme se déroule en plusieurs temps :

a On calcule la sortie du premier réseau en ignorant le second (valeur à 0)
b On calcule la sortie du second réseau
– On calcule la sortie du premier réseau avec les sorties de b)

$first\_order.calc\_hidden\_layer(samples.inputs)$
$high\_order.calc\_output\_layer(first\_order.hidden\_layer)$
$first\_order.calc\_output\_layer(first\_order.hidden\_layer,\ [0, ..., 0])$

$h\_output \leftarrow ampli(high\_order.output\_layer)$
$right\_houtput \leftarrow [0,\ 0]$
if $good\_answer(first\_order)$ **then**
  $right\_houtput[1] \leftarrow 1$
else
  $right\_houtput[0] \leftarrow 1$
end if
$first\_order.calc\_output\_layer(first\_order.hidden\_layer,\ h\_output)$

$calc\_stats()$

$high\_order.train(first\_order.hidden\_layer,\ right\_houtput)$
$first\_order.train(samples.inputs,\ samples.outputs,\ h\_output)$

L'interêt se porte sur

function $train(inputs,\ outputs,\ add)$
  for $i = 0 \rightarrow output\_neurons.length$ **do**
    $y_{output}[i] \leftarrow g'(output\_neurons[i].a) \times (outputs[i] - output\_neurons.state)$
  end for

  for $i = 0 \rightarrow hidden\_neurons.length$ **do**
    $w\_sum \leftarrow \sum_{j=0}^{output\_neurons.length} output\_neurons[j].weights[i] \times y_{output[j]}$
    $y_{hidden}[i] \leftarrow g'(hidden\_neurons[i].a) \times w\_sum$
  end for
  $update\_weights\_hidden\_layer(y_{hidden})$

  for $i = 0 \rightarrow output\_neurons.length$ **do**
    $output\_neurons[i].update\_weights\_gradient(y_{output}[i],\ hidden\_neurons,\ add)$
    $output\_neurons[i].update\_weights\_perceptron(outputs[i],\ hidden\_neurons,\ add)$
  end for
end function
function $update\_weights\_gradient(error,\ intputs,\ add)$
  $calc\_output(inputs + add)$

  for $j = 0 \rightarrow inputs.length$ **do**
    $dw \leftarrow weights[j] - last\_weights[j]$
    $p \leftarrow error \times inputs[j]$

$weights[j] \leftarrow weights[j] + learning\_rate \times p + momentum \times dw$
end for
end function
function $update\_weights\_perceptron(goal,\ intputs,\ add)$
$calc\_output(inputs + add)$

for $j = inputs.length \rightarrow inputs.length + add.length$ **do**
$dw \leftarrow weights[j] - last\_weights[j]$
$p \leftarrow (goal - state) \times add[inputs.length - j]$
$weights[j] \leftarrow weights[j] + \frac{learning\_rate \times p + momentum \times dw}{add.length}$
end for
end function