Etude de la stabilité de réseaux de neurones par le calcul des nombres de Lyapunov.

Thomas Voegtlin

Résumé

Un réseau de neurones au cours de l'apprentissage est un système dynamique non linéaire dont l'évolution peut être étudiée à l'aide des outils généraux utilisés pour décrire le comportement de ces systèmes. En particulier, on peut se demander si le réseau va ou non converger vers un état voulu, ou bien si, comme d'autres systèmes dynamiques, il se comportera de manière chaotique.

Les nombres de Lyapunov sont une mesure du degré de chaos d'un système dynamique. Appliqués aux réseaux de neurones, ils permettent de savoir, dans une certaine mesure, si le système est correctement paramétré.

Stage de première année de magistère d'informatique, effectué en juin-juillet 1996 au Laboratoire de Formes et de Recherche en Intelligence Artificielle (LAFORIA) de l'Université de Paris VI à Jussieu.

Directeur de stage: Florence d'Alché-Buc.

Directeur de l'équipe connexionniste du Laforia: Patrick Gallinari.

---

1 Réseaux de neurones.

1.1 Définitions.

Un **neurone formel** est un automate possédant plusieurs entrées et une sortie. L'automate calcule sa sortie en appliquant une fonction de seuil à la somme de ses entrées, pondérée par des poids de connexions associés aux entrées.

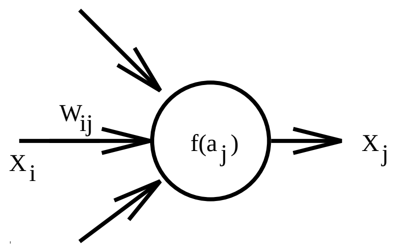

Figure 1: un neurone

$a_j = \sum_i x_i w_{ij}$ est la somme pondérée des sorties des neurones précédents.

$x_j = f(a_j)$ est la sortie du neurone j.

$w_{0j} = -1$ (biais) par convention

Un **perceptron** est un réseau de neurones, multicouche, pour lequel les sorties d'une couche sont reliées aux entrées de la couche suivante. La première couche est la couche d'entrée et la dernière est la couche de sortie. On appelle couches cachées les couches intermédiaires.

Les réseaux de neurones ont la capacité de pouvoir apprendre une tâche à partir d'exemples, à travers des modifications des poids de leurs connexions. En général, on fournit au réseau sur ses entrées, des exemples de la tâche à réaliser, et on applique un algorithme d'apprentissage à ce réseau.

Dans le cas d'un apprentissage supervisé, on donne aussi au réseau la "réponse", c'est-à-dire les sorties voulues. L'interêt des réseaux de neurones réside alors dans leur faculté de généralisation , c'est-à-dire qu'un réseau correctement entraîné pourra, par la suite, traiter correctement des entrées non apprises.

L'algorithme d'apprentissage utilisé pour entraîner des réseaux multicouches est l'algorithme de **rétropropagation**.

La définition initiale de McCulloch et Pitts donne des valeurs booléennes aux entrées et aux sorties. On utilise cependant des entrées et sorties continues, ainsi qu'une fonction de seuil continue afin de disposer d'outils mathématiques comme l'énergie du réseau, le gradient d'erreur, la température, etc., qui permettent d'appliquer des algorithmes d'apprentissage, comme celui de rétropropagation du gradient, lorsque le réseau est organisé en couches. Toutes les variantes de cet algorithme utilisent des valeurs continues.

L'apprentissage est un problème d'optimisation : Quelles sont les valeurs à donner aux poids des connexions pour minimiser le nombre d'erreurs commises par le réseau. Pour attaquer ce problème on définit une énergie qui doit décroître au cours de l'apprentissage. Elle est représentative du nombre d'erreurs effectuées par le réseau.

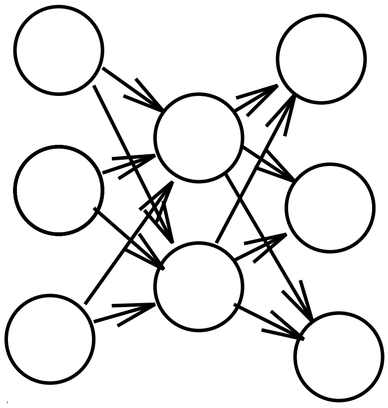

Figure 2: un perceptron à une couche cachée

En général, on choisit une énergie égale à la somme des carrés des différences entre les sorties desirées et les sorties données par le réseau :

$E = \sum_{i \in sorties} (valeur_i - valeurvoulue_i)^2$

1.2 La rétropropagation. Le problème de la stabilité.

La rétropropagation d'un gradient à travers un réseau est une mise en oeuvre de la descente de gradient à travers une composition de fonctions.

La valeur initiale des poids est aléatoire. On présente un exemple au réseau, c'est-à-dire qu'on initialise les neurones de la couche d'entrée . Les neurones calculent leur valeur x_i en fonction des sorties des neurones précédents, de l'entrée vers la sortie.

$x_j(t) = f(a_j(t)) = f(\sum_{j \in A(j)} w_{ij} x_i)$

On a noté $A(j)$ l'ensemble des neurones antérieurs au neurone j, c'est-à-dire les neurones qui ont leur sortie reliée à j.

Ensuite, on modifie les poids des connexions selon une loi du type :

$w_{ij}(t+1) = w_{ij}(t) - \epsilon \frac{\partial E}{\partial w_{ij}}$

de sorte à minimiser l'erreur E par une technique de descente de gradient.

La dérivée précédente s'exprime comme produit de deux termes:

$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ij}}$

on a : $\frac{\partial x_j}{\partial w_{ij}} = x_i f'(a_j)$

$\frac{\partial E}{\partial x_j} = 2(d_j - x_j(t))$ si le neurone j appartient à la couche de sortie.

et $\frac{\partial E}{\partial x_j} = \sum_k w_{kj}(t) y_k(t)$ sinon.

on peut regrouper les termes qui dépendent de i et ceux qui dépendent de j:

$\frac{\partial E}{\partial w_{ij}} = x_i(t) y_j(t)$

avec: $y_j(t) = 2f'(a_j(t))(d_j - x_j(t))$

si le neurone j appartient à la couche de sortie.

et $y_j(t) = f'(a_j(t)) \sum_k w_{kj}(t) y_k(t)$ sinon.

Le terme $y_j(t)$ est le **gradient en le neurone** j .

C'est de cette formule que vient le nom de rétropropagation: pour calculer les y_i d'une couche, il suffit de calculer ceux de la couche précédente, qui interviennent dans la somme. Donc le calcul est relativement simple. De plus, cette formule peut s'interpréter intuitivement: Si le neurone j de la couche de sortie est loin de sa sortie désirée, c'est à cause des connexions des neurones de la couches précédente vers i, d'où modification des poids, mais c'est aussi à cause de l'état d'activation des neurones de la couche précédente, qui possèdent une erreur égale à la somme ponderée des erreurs en sortie de ces neurones.

La preuve formelle de la convergence de l'algorithme de rétropropagation a été donnée [1] moyennant les hypothèses suivantes:

- Le réseau possède une architecture adaptée.
- Les exemples fournis dans l'échantillon sont cohérents.
- Le pas d'apprentissage est suffisament petit, ou bien il tend vers zéro, selon une série de somme infinie.

Pourquoi s'intéresser, dans ce cas, à la stabilité du réseau? Le résultat précédent est théorique et ne rend pas compte de plusieurs points essentiels:

- Le risque de tomber dans un minimum local:On fait décroître E selon un chemin imposé et dans certains cas, l'algorithme peut rester bloqué dans un minimum local de la fonction d'énergie, sans jamais trouver le minimum global.

- Le choix des paramètres du réseau: Ce choix est crucial vis-à-vis des performances, qu'il s'agisse du choix de ϵ où d'une variante sur le modèle de la rétropropagation, ou encore du choix de l'architecture.
- Le temps de calcul: Comment choisir le pas de calcul pour optimiser ce temps. Dans la pratique, l'algorithme de rétropropagation est très lent si on l'utilise dans sa version de base, car on ne sait pas bien choisir ϵ, le pas du gradient.

2 Systèmes dynamiques.

2.1 Définitions.

Un système dynamique est défini par une équation d'état, du type $X_{t+1} = F(X_t)$. Il peut se comporter de manière chaotique, c'est-à-dire d'une manière difficile à distinguer d'un processus aléatoire. Il existe des outils pour quantifier le degré de chaos d'un système, en particulier les nombres de Lyapunov.

2.2 Les nombres de Lyapunov.

Les nombres de Lyapunov permettent de quantifier l'évolution d'un système dynamique, et en particulier de savoir si son comportement est de type chaotique. Il sont une généralisation aux systèmes non linéaires des valeurs propres des systèmes linéaires.

L'équation d'état d'un système linéaire peut se mettre sous la forme: $X_{t+1} = AX_t$ où A est une matrice carrée . Le comportement d'une solution est lié aux amplitudes des valeurs propres de A. Par exemple, si ces valeurs propres sont distinctes (ce qui est le cas général), alors pour une valeur initiale $X_0 = \sum_i a_i v_i$, où les v_i sont les vecteurs propres, correspondant aux valeurs propres $\gamma_1 > ... > \gamma_n$, la solution de l'équation d'état est $X_t = A^t X_0$. Et la norme de X_t croît géométriquement selon la plus grande valeur propre γ_1 de A. Dans ce cas particulier, les exposants de Lyapunov sont les nombres $\lambda_i = \log \gamma_i$.

Alors $\frac{1}{t} \log ||X_t|| = \frac{1}{t} \log ||A^t X_0||$ tend vers la plus grande valeur propre de A pour presque toutes les conditions initiales X_0.

Les exposants de Lyapunov donnent le taux de croissance asymptotique des solutions de l'équation d'état. Le taux de croissance asymptotique d'une solution sera λ_1 pour presque toutes les conditions initiales, excepté pour des vecteurs appartenant à un sous espace strict de l'espace des phases.

Le taux de croissance des solution depend donc du signe de l'exposant principal λ_1. Si $\lambda_1 > 0$, les solutions vont diverger, sinon elles vont converger.

Ces exposants de Lyapunov se généralisent aux systèmes dynamiques non linéaires. L'équation d'état d'un systeme non linéaire est quelconque: $X_t = F(X_{t-1})$. (On supose que le système est autonome, c'est-à-dire que F ne depend pas de t). Alors on note J_t la matrice jacobienne de F au point X_t : $J_t = DF(X_t)$. Les exposants de Lyapunov pour ce système sont définis par:
$\lambda(X) = \lim_{t->\infty} \log |m_i(t)|$ où les $m_i(t)$ sont les valeurs propres du jacobien J_t. Si $X_0 <> X_{eq}$ et X_t tend vers X_{eq} c'est-à-dire que X_0 est dans le bassin d'attraction de X_{eq}, les exposants de Lyapunov de X_0 et de X_{eq} sont les mêmes. En général tout point dans le bassin d'attraction d'un attracteur a les mêmes exposants de Lyapunov que cet attracteur.

L'interêt des nombres de Lyapunov pour des systêmes non linéaires est qu'ils quantifient la dépendance du système aux conditions initiales. Il n'existe pas de définition universellement admise du chaos; nous nous en tiendrons à la définition suivante: On dit qu'un système se comporte de manière chaotique si il est très sensible à de faibles variations des conditions initiales X_0. Par exemple, si X_t^1 et X_t^2 sont deux trajectoires qui ne diffèrent que par leurs valeurs initiales X_0^1 et X_0^2, on peut écrire:

$$X_1^t - X_2^t = Y_t$$

et si Y_t est petit:

$$Y_{t+1} = F(X_t^1) - F(X_t^2) = J_t.Y_t$$

où J_t est la jacobienne de F au point X_t.

Aussi, pour des variations toujours petites, on a :

$$Y_t = J_t J_{t-1} ... J_0 Y_0.$$

L'évolution dans le temps d'une perturbation initiale suffisament petite est donc approximée par un système linéaire. Et les exposants de Lyapunov donnent ici le taux de croissance des perturbations du systême. Si $\lambda_1(X_0, Y_0)$ est positif alors la perturbation Y_0 croît exponentiellement au cours du temps. $\lambda_1 > 0$ indique que le système présente une dépendance importante aux conditions initiales. A l'inverse, $\lambda_1 < 0$ est une condition de stabilité d'un système dynamique: celui-ci converge vers un état stable, ou bien alors les valeurs d'adhérence sont en nombre fini. [3]

2.3 Les réseaux de neurones sont des systèmes dynamiques.

L'apprentissage d'un réseau peut être vu comme l'évolution d'un système dynamique en temps discret. L'espace des phases est l'espace des poids des connexions, la fonction F est la fonction qui modifie les poids selon les règles de la rétropropagation à chaque étape.

Nous noterons les poids des connexions par un vecteur X_t de R^c, ou c est le nombre de connexions du réseau. A l'étape t: $X_t = F(X_{t-1})$ où F est la fonction d'apprentissage.

On distingue deux types d'apprentissage:

- Le gradient global: Avant de modifier les poids, on soumet tous les exemples au réseau et le gradient d'erreur est la superposition des gradients de tous les exemples dont on dispose. Dans ce cas, la fonction de modification des poids est la même à chaque étape.
- Le gradient stochastique: On tire un exemple au hasard, et on s'en sert pour modifier les poids selon la loi de rétropropagation. Cette méthode est plus rapide, et la convergence est indépendante de l'ordre dans lequel sont présentés les exemples.

Enfin on peut bruiter l'exemple choisi à chaque fois, de manière à favoriser les capacités du réseau à généraliser. En effet, le bruitage est l'ajout aux exemples d'une petite composante aléatoire au cours de l'apprentissage. On ne présente donc jamais deux fois la même entrée au réseau. Ceci évite de générer un réseau trop sensible aux entrées, et introduit donc de la robustesse dans les poids des connexions. De plus, on fournit bien plus d'exemples au réseau, et de cette manière on ne lui permet pas d'apprendre "par coeur" les exemples mais on l'oblige à en extraire des règles générales.

Ces deux types d'apprentissage sont modélisés par deux types de systèmes dynamiques: en effet dans le premier cas (gradient global), la fonction F de modification des poids ne change pas au cours du temps. Dans le cas du gradient stochastique ou bien si on effectue un bruitage, la fonction F n'est pas la même à chaque étape $F = F_t$, et $X_t = F_{t-1}(X_{t-1})$. Cependant, on peut considérer que le processus (F_t) possède certaines propriétés statistiques qui permettent de continuer les calculs précédents.

2.4 Exposants de Lyapunov pour des systèmes stochastiques non linéaires.

Les exposants de Lyapunov peuvent se généraliser (encore!) à une fonction d'état qui change de manière aléatoire à chaque étape. Cette théorie sera abordée brièvement ici, pour montrer que les résultats théoriques sont valables pour l'apprentissage par gradient stochastique.

Revenons tout d'abord au cas linéaire:

Un processus aléatoire est dit stationnaire si sa structure en terme de probabilité est invariante au cours du temps [6]. On dit pour un tel processus qu'une de ses variable v est ergodique si , pour une réalisation donnée, la moyenne de v au cours du temps (en tant que variable aléatoire) tend, lorsque ce temps tend vers l'infini, vers la moyenne de v sur l'espace des réalisations du processus (sans tenir compte du temps).

Soit $\{A_{t=0}^{\infty}\}$ une suite aléatoire ergodique de matrices, On considère le système $X_{t+1} = A_t X_t$ Soit $H_t = A_{t-1}...A_1 A_0$ tel que la solution du système linéaire prenne la forme générale $X_t = H_t X_0$. La définition de l'exposant de Lyapunov pour ce système est [2]: $\lambda(X_0) = \lim_{t->\infty} \frac{1}{t} \log ||H_t X_0||$

Cette valeur $\lambda(X_0)$ va dépendre du vecteur initial X_0 et de la réalisation du processus $\{A_t\}$. Mais si la suite (A_t) est fixée il existe au plus $p <= d$ valeurs possibles pour $\lambda(X_0)$, d étant la dimension de l'espace. De plus, le théorème multiplicatif ergodique d'Oseledec établit que: Si le processu A_t est ergodique, alors, en dehors d'un ensemble de mesure nulle, l'ensemble des valeurs possibles pour $\lambda(X_0)$ est le même pour toutes les réalisations de la suite de matrices (cf Eckmann et Ruelle 1985). Ainsi, dans ces conditions, nous pouvons définir les exposants de Lyapunov $\lambda_p < .. < \lambda_1$ et on a, avec une probabilité 1, $\lambda(X_0) = \lambda_i$ pour un i appartenant à 1...p et pour tout X_0 [2]. Le théorème d'Oseledec établit aussi que les ensembles $E_i = X_0 \in R^d / \lambda(X_0) <= \lambda_i$ forment des sous-espaces avec $E_p \in .. \in E_1 = R^d$ et $\lambda(X_0) = \lambda_i \iff X_0 \in E_i - E_{i+1}$. Comme pour les systèmes définis plus-haut, λ_1 donne le taux de croissance d'une solution avec probabilité 1. Plus précisément, $\lambda(X_0) = \lambda_1$ pour tout X_0 pris en dehors d'un sous-espace de R^c de dimension $(c-1)$ ou inférieure.

Pour des systèmes dynamiques non linéaires, la généralisation précédente est toujours valable, car les exposants de Lyapunov du système non linéaire ne sont autres que les exposants de Lyapunov du système Jacobien associé. On note F_t la fonction d'état au temps t, et J_t le jacobien de F_t au point X_t: $J_t = DF_t(X_t)$

On définit alors ainsi les exposants de Lyapunov:

$\lambda(X_0, Y_0) = \lim \frac{1}{t} \log ||J_{t-1}..J_0 Y_0||$

Pour des réseaux de neurones en gradient stochastique, on suppose que le processus J_t satisfait les conditions d'ergodicité du théorème d'Oseledec (c'est une condition qui ne porte que sur la suite d'exemples présentés au réseau, et celle-ci vient d'une fonction pseudo-aléatoire) et $\lambda(X_0, Y_0)$ vaut 1 pour presque tous les X_0 avec probabilité 1. (Ici Y_0 joue le rôle de X_0 dans le cas linéaire.)

On pose: $Y_t = J_{t-1} Y_{t-1}$ (ie) $Y_t = J_{t-1}..J_0 Y_0$ pour la suite.

3 Calcul des nombres de Lyapunov.

En fait, il nous suffit de calculer le plus grand d'entre eux, qui suffit à indiquer si le système se comporte ou non de maniere chaotique.

Il existe deux catégories de méthodes pour estimer les nombres de Lyapunov. En général elles s'attachent uniquement à l'exposant principal λ_1. La plupart de ces méthodes ont été mises au point pour travailler sur des données expérimentales. (Wolf ou Eckmann et Ruelle).

Les méthodes “directes” proposées par Gluckenheimer (1982) et implémentées par Wolf (85), Parker et Chua [3] sont basées sur la supposition que $||Y_t||$ va croître exponentiellement au taux λ_1 au cours du temps. On observe le taux de croissance d'une perturbation infinitésimale en relevant la différence entre deux trajectoires, pour calculer λ_1. On choisit des valeurs de X_t suffisament proches pour faire l'approximation.

Dans notre cas, nous pouvons effectuer une simulation et nous ne sommes pas restreints à des données expérimentales: nous pouvons choisir la valeur et la direction de la perturbation initiale.

Les méthodes indirectes proposent d'estimer le jacobien de F et d'en calculer les valeurs propres. On intègre donc $X_t(X_0)$ pendant un temps T, et on estime le jacobien. L' exposant de Lyapunov est: $\lambda_1 = \frac{1}{T} \log |m_1(T)|$, où m_1 est la plus grande valeur propre de J_t.

Enfin une nouvelle méthode a été proposée par Bertrand Augereau (96) [4], qui est adaptée aux réseaux de neurones. Cette méthode est en fait a l'origine de mon stage.
Augereau propose de calculer un nombre de Lyapunov instantané, avec une complexité qui est la même que celle de la rétropropagation. L'exposant de Lyapunov est obtenu en passant à la limite dans le calcul de ces exposants instantanés. L'interêt est de pouvoir intervenir au cours de la rétropropagation sur les paramètres du réseau de manière à donner une valeur voulue aux nombres de Lyapunov instantanés . Ainsi, il propose un algorithme de rétropropagation qui calcule à chaque itération le pas d'apprentissage de sorte que les nombres de Lyapunov instantanés soient tous égaux à une constante négative, pour assurer la convergence du système.

3.1 La méthode de Parker et Chua.

Cette méthode repose sur le fait que presque chaque perturbation croît avec un taux λ_1. On considère donc l'évolution de deux systèmes proches au temps zéro et on regarde à quelle vitesse ils convergent ou divergent. En pratique, celà revient à effectuer deux rétropropagations sur deux réseaux proches. On crée donc un premier réseau, et le second est obtenu en ajoutant une petite perturbation au hasard.

J'ai procédé de la manière suivante:

- Choix d'un temps T.
- Choix d'un nombre d'itérations it.
- Créer deux réseau, distants de δX_0.
- Pendant T itérations, laisser les deux réseaux évoluer. Calculer le taux de divergence entre les deux réseaux, rapporté au temps T.
- Réajuster la taille de la petite perturbation, et reprendre le calcul précédent it fois.

Pour choisir T et it, il faut trouver un compromis: en effet, lorsque le nombre de Lyapunov est grand en valeur absolue, la perturbation peut devenir assez vite nulle du fait de la précision limitée du calcul, ou bien elle peut devenir trop grande et le calcul n'est plus valable. C'est pourquoi on répète l'opération it fois: D'une part, la précision est améliorée, d'autre part une seule opération peut échouer (si δX devient nulle) ou non, en fonction du choix initial de δX_0. Il vaut mieux augmenter le nombre it de tentatives plutôt que de diminuer le temps T, car si T est trop petit, la perturbation ne croît pas forcément avec la plus grande valeur propre, elle n'a pas le temps de prendre la bonne direction.

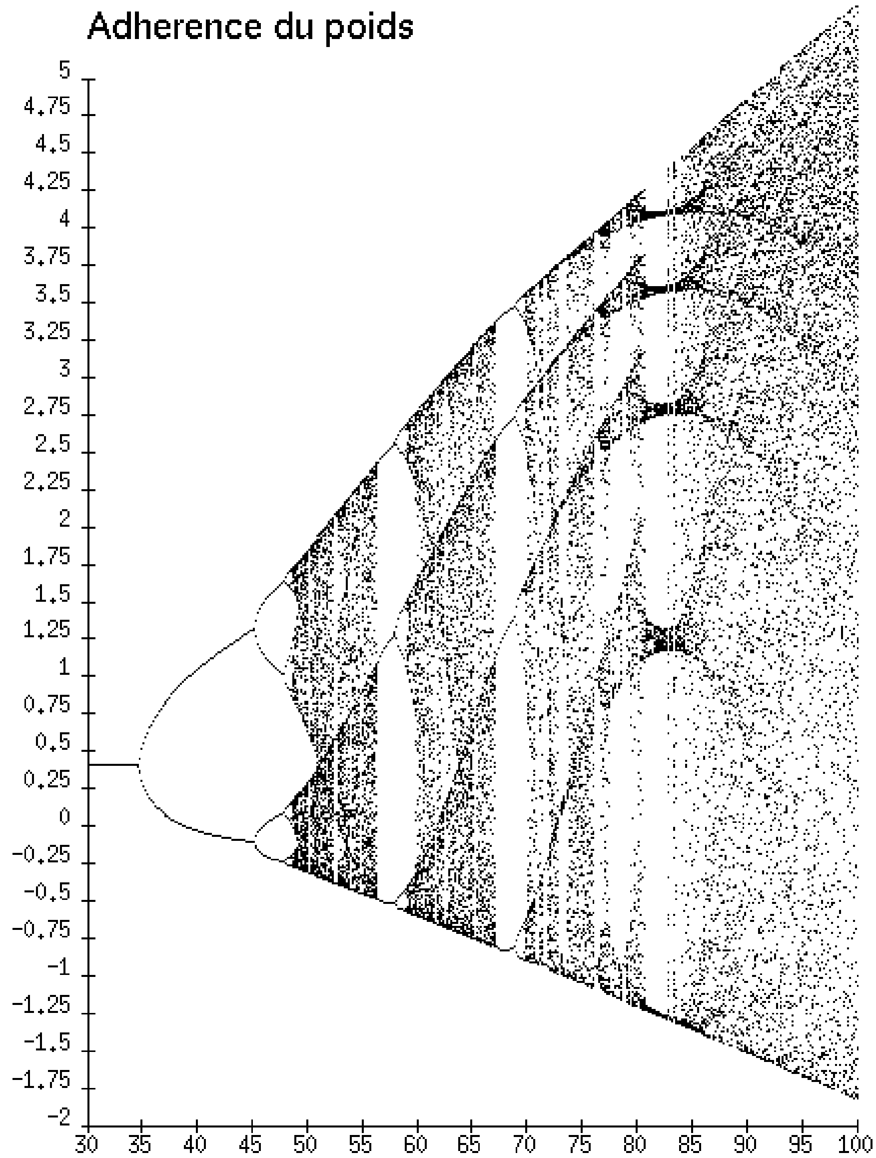

Figure 3: *Diagramme de bifurcation du poids de la connexion entre l'entrée et la sortie*

3.2 La méthode d'Augereau.

Le calcul proposé par Augereau est repose sur des **nombres de Lyapunov instantanés** $\lambda_n = \frac{1}{n}\sum_{p=0}^{n-1} ln(1+\epsilon\sum_{(i.j)\in C_A} L_{ij}^p)$ et le nombre de Lyapunov n'est autre que la limite de λ_n lorsque n tend vers l'infini. Les coefficients L_{ij} sont relatifs aux connexions qui appartiennent au sous-réseau C_A dont dépend la sortie A. Par la suite, tous les réseaux présentés ont une seule sortie. Ce calcul ne me semble pas rigoureux (il est présenté plus en détail en annexe), mais l'expérience montre qu'on obtient les mêmes résultats avec les deux méthodes (les courbes sont les mêmes, à certaines limitations près).

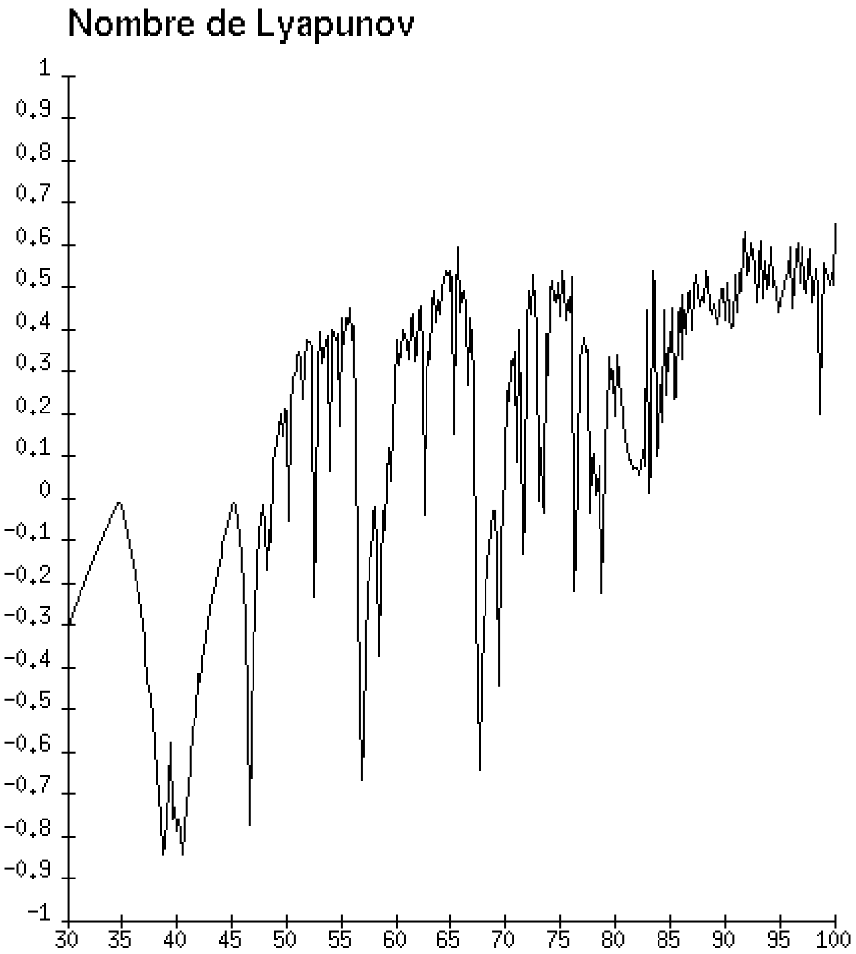

Figure 4: *Nombre de Lyapunov mesuré par la méthode de Parker et Chua*

3.3 Résultats comparés

La méthode directe est lente. Voici tout d'abord un exemple, présenté par Augereau dans sa thèse, et que j'ai repris: l'apprentissage d'un réseau trivial, sans neurone caché, avec seulement une connexion entre un neurone d'entrée et une sortie. On fait varier en abscisse (figures 4 et 5) le pas d'apprentissage de la rétropropagation. Ce système ne possède évidemment qu'un seul bassin d'attraction, donc le nombre de Lyapunov est indépendant des valeurs initiales des poids. Pour cette raison, j'ai fait converger le système avant de calculer le nombre de Lyapunov, afin d'avoir moins d'itérations à effectuer.

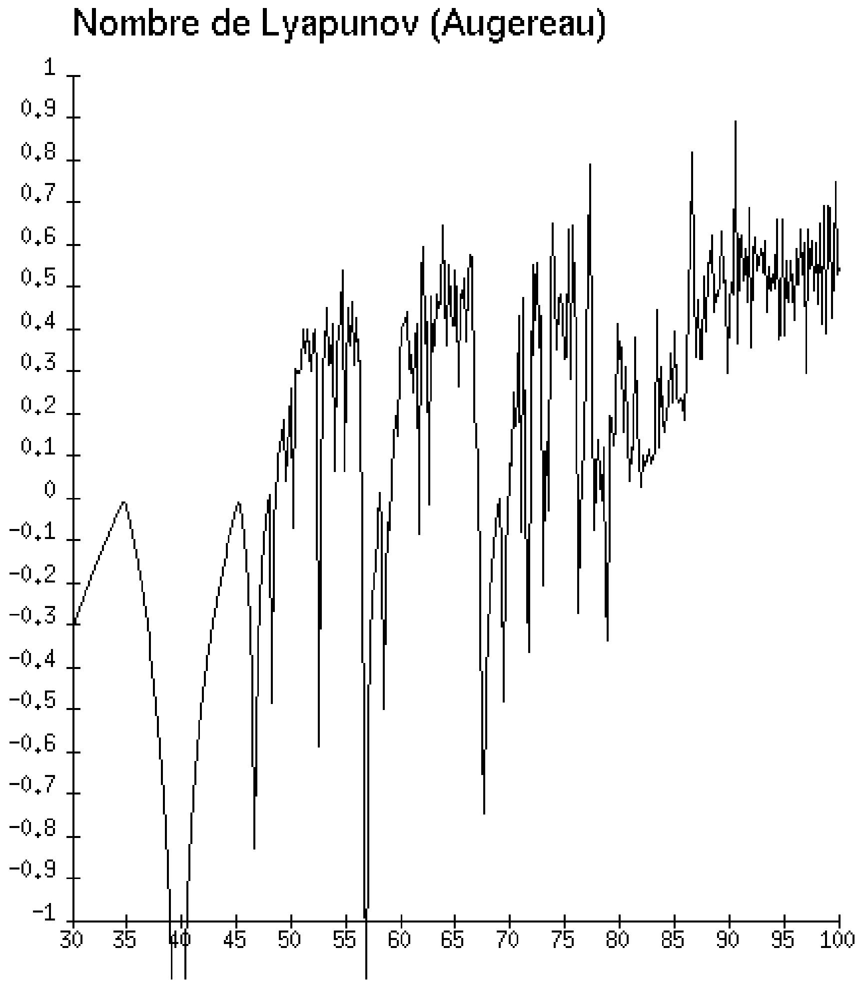

Figure 5: *Nombre de Lyapunov calculé à l'aide de la méthode d'Augereau*

On constate en premier lieu que les bifurcations de l'ensemble des valeurs d'adhérence correspondent à des pics d'annulation du nombre de Lyapunov. Ce dernier devient positif dans les régions où le comportement du système est chaotique. Enfin, il existe des intervalles de valeurs du pas d'apprentissage où le système passe d'un état chaotique à un état momentanément ordonné, avec un nombre fini de valeurs d'adhérences. On peut constater que sur les deux graphiques, le nombre de Lyapunov repasse sous zéro à ce moment...

La différence majeure entre les deux algorithmes est que la méthode de Parker et Chua ne permet évidemment pas de calculer des nombres très grands en valeur absolue. En effet elle repose sur une mesure directe et les nombres mesurables sont proportionnels au nombre de décimales des chiffres utilisés. On peut déjouer cette limitation en faisant un nombre réduit d'itérations, comme indiqué plus haut, et avec les limitations que l'on sait. Dans l'exemple ci-dessus, par exemple, cette astuce permet de faire descendre les pics plus bas, et de les rendre similaires a ceux de la figure 4 . Mais le nombre d'itérations nécessaire devient alors très vite rhédibitoire... A l'inverse, la méthode d'Augereau est précise et rapide. Cela tient au fait que c'est quasiment une méthode de jacobien.

3.4 Un exemple de réseau à une couche cachée

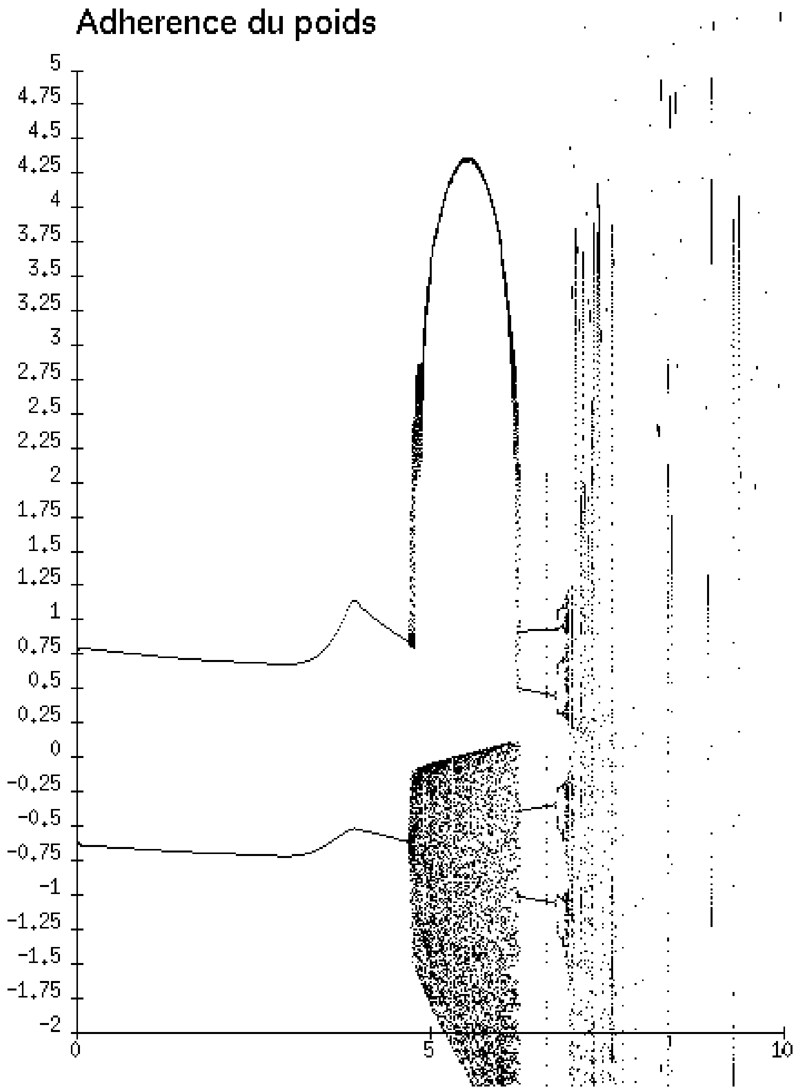

Figure 6: *Diagrammes de bifurcation. En haut, le poids en entrée, et en dessous le poids en sortie.*

L'exemple de la figure 6 est aussi un exemple trivial, destiné à vérifier que l'algorithme fonctionne bien pour des réseaux multicouches. J'ai trois neurones, un en entrée, un caché et un en sortie. On constate cependant que le poids en sortie est bien plus instable.(pour chaque valeur du pas je suis parti des mêmes conditions initiales). Les comportements des neurones sont ici chaotiques au même moment. Cependant pour des réseaux possédant plusieurs sorties, deux sorties diférentes peuvent avoir des diagrammes de bifurcation totalement différents (cf Augereau [4]).

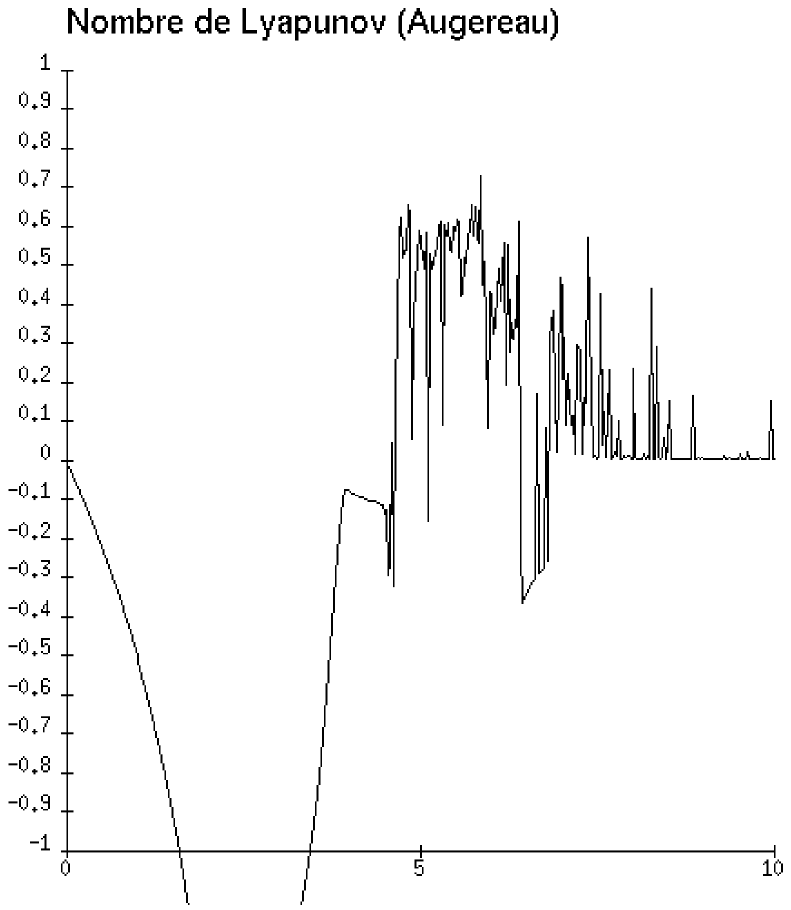

Figure 7: *Nombre de Lyapunov. Les résultats sont corrects.*

4 Le problème de la généralisation. Techniques de régularisation

4.1 Introduction

L'intérêt majeur des réseaux de neurones réside dans leur capacité à généraliser ce qu'ils ont appris, c'est-à-dire qu'ils sont parfois capables, lorsqu'on leur présente une entrée non apprise, de donner une sortie "correcte", cohérente avec l'ensemble sur lequel s'est fait l'apprentissage. Pour être capable de généralisation, un réseau doit extraire des lois générales à partir de l'ensemble d'apprentissage, et réagir en fonction de ces lois générales. A l'inverse, un réseau peut parfois apprendre "par coeur" une base d'apprentissage, en particulier lorsque celle-ci est trop petite, et être par la suite incapable de généraliser ce qu'il a appris à de nouvelles entrées.

Pour qu'un réseau puisse apprendre "par coeur" une base d'exemple, il doit comporter un certain nombre de neurones, fonction du nombre d'exemples. A l'inverse, pour favoriser les capacités de généralisation, on cherche souvent à réduire le nombre de neurones et à élargir la base d'apprentissage.

Une technique utilisée pour augmenter le nombre d'exemples de la base consiste à bruiter les entrées, en leur ajoutant une faible composante aléatoire de maniere à obtenir une base infinie. Le bruitage permet de rendre le réseau régulier, c'est-à-dire qu'une faible variation des entrées ne perturbera pas les sorties, puisque la sortie voulue des exemples bruités est la même que celle des exemples de la base d'origine.

Inversement, il existe des techniques qui mettent l'accent sur l'architecture du réseau, en cherchant à minimiser le nombre de neurones. On peut par exemple supprimer des connexions jugées inutiles, puis appliquer la technique suivante (cf Krogh et Hertz [1]):

- éliminer les neurones qui sont isolés, ou bien qui n'ont pas de sortie.
- détecter les paires de neurones dont la sortie est fortement corrélée, et en supprimer un des deux, en remplacant les connexions qui partaient de ce neurone par des connexions depuis le neurone laissé intact.(dans ce cas il faut refaire un apprentissage du réseau après l'operation).

On peut aussi procéder de maniere constructive, en partant d'une architecture minimale et en l'augmentant petit à petit...

4.2 Application à la technique de weight decay

Nous disposons d'un outil de mesure de la stabilité des réseaux de neurones. Nous pouvons par exemple l'utiliser pour étudier l'influence de ces techniques de régularisation sur la stabilité.

Le **weight decay** est une méthode pour détruire les connexions inutiles. Ceci permet aussi de réduire l'espace de recherche des solutions, et donc d'accélerer la convergence.

On ajoute un terme dans l'énergie, de manière à pénaliser les connexions inutiles:

$E_{decay} = E + \alpha \sum_{i \in \{1 \ldots c\}} w_{ij}^2$

J'ai essayé d'obtenir des résultats sur l'influence de l'apport d'un decay sur la stabilité du réseau. Mais le calcul d'Augereau ne se généralise pas avec cette formule, et le calcul avec la méthode directe est très long et n'est pas appliquable à des réseaux de taille importante. Je ne suis parvenu à rien de significatif...

5 L'Algorithme proposé par Bertrand Augereau

5.1 Principe. Avantages

Bertrand Augereau propose un algorithme de rétropropagation à pas adaptatif, dont le but est de fixer la valeur du nombre de Lyapunov du réseau. En effet, le nombre de Lyapunov instantané valant

$$\lambda(n) = \log\left(1 + \epsilon \sum_{i,j \in C_A} L_{ji}(n)\right),$$

il est possible de de donner une valeur fixe négative au nombre de Lyapunov du système en choisissant un pas variable $\epsilon(n)$ de sorte à avoir $\lambda(n) = \lambda_0 \forall n$. Dans ce cas le nombre de Lyapunov, c'est-à-dire la limite de $\lambda(n)$ est evidemment λ_0. En quelque sorte on cherche a "forcer" la stabilité du réseau.

Cet algorithme possède des proprietés intéressantes. L'expérience montre que le pas ϵ varie considérablement au cours de l'apprentissage, du moins pour des petits reséaux.

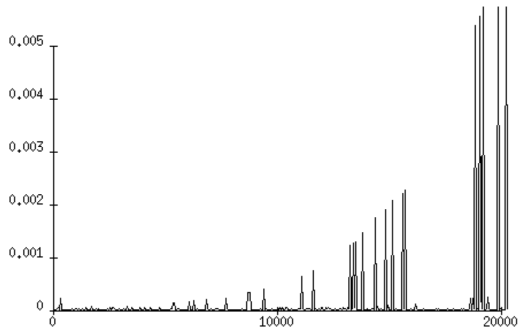

Figure 8: *Variation du pas d'apprentissage*

On peut se représenter l'apprentissage comme le déplacemment d'un point ou d'une bille sur une hypersurface de dimension égale à la dimension de l'espace de recherche (le nombre de connexions), la dernière dimension représentant l'énergie. Les descentes de gradient font en général suivre à ce point la plus grande pente de la surface sur laquelle il se trouve. Avec un pas ϵ constant l'amplitude de chaque déplacement dépend uniquement de la pente locale. Et si le point tombe dans un minimum, la dérivée de l'énergie devenant nulle, la bille tend à s'immobiliser. Des variantes font décroître le pas d'apprentissage de sorte à assurer la convergence de la méthode dans tous les cas, d'autres choisissent de donner à ϵ une valeur telle que l'amplitude de chaque pas soit la même.

En pratique cependant ce n'est pas tout-à-fait comme si le point se deplacait à la surface, car le pas du gradient intervient: Une itération correspond à un saut dans une direction, et si des obstacles sont présents sur le chemin, ils sont ignorés: la bille passe "à travers la paroi".

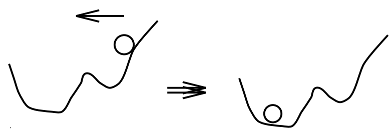

Figure 9: *Une itération correspond à un saut dans la direction de la plus grande pente.*

Après une itération, le point peut parfois même se retrouver à un endroit plus élevé que sa position précédente, selon la valeur de ϵ. En particulier l'algorithme d'Augereau, avec son pas d'apprentissage très variable, possède la propriété de pouvoir échapper aux minimums locaux de cette manière. Si le réseau se trouve pris dans une region ou $\sum L_{ji}$ est petite, (typiquement une cuvette dont le fond est "plat") alors ϵ prend une grande valeur et le réseau echappe ainsi au minimum local. Malheureusement rien ne distingue, sur le plan mathématique, un minimum local du minimum global, et donc il n'est pas certain que cet algorithme soit très stable...

En effet lorsque le réseau est dans une cuvette, le relief est plat et le pas prend une valeur élevée, et les approximations d'Augereau deviennent evidemment fausses. Alors le véritable nombre de Lyapunov du système n'est pas celui que l'on cherche à imposer. Les sauts correspondent donc à des ruptures de la régularité des nombres de Lyapunov instantanés. Il faut en conclure que dans ce cas il n'est *pas possible* de forcer la valeur du nombre de Lyapunov du réseau par cette méthode.

Voici (figure 10) le graphe de la fonction d'erreur globale du réseau correpondant au pas d'apprentissage montré figure 8. Le réseau s'échappe d'une zone ou la convergence est trop faible (discontinuité) , qui peut être un minimum local sans que ce soit nécessairement le cas. La discontinuité correspond bien à un pic de $\epsilon(n)$ sans que ce pic soit le plus important. Par la suite, des pics bien plus importants n'engendrent pas de changement brutal de l'erreur commise par le réseau, sans doute parce que la pente est alors trop proche de zéro.

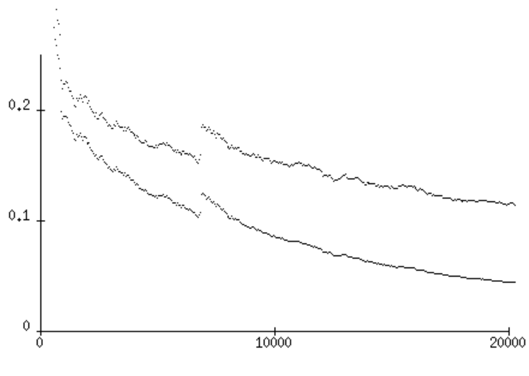

Figure 10: *Erreur en fonction du nombre de pas.*

(Il y a deux courbes car celle du dessus représente l'erreur commise sur un échantillon d'exemples différent de l'échantillon d'apprentissage).

5.2 Le surapprentissage.

On ne cherche cependant pas toujours à atteindre le minimum de l'erreur sur la base d'apprentissage, mais à ce que l'erreur soit minimale lorsque le réseau doit généraliser ce qu'il a appris. Et le minimum de l'erreur sur la base d'apprentissage ne correspond pas toujours à une erreur minimale en généralisation. Lorsque la base d'apprentissage est trop petite, il se produit un phenomène de **surapprentissage**: plus on fait durer l'apprentissage du réseau et moins il est capable de généraliser.

Les données utilisées ici proviennent d'un problème de classification de deux formes différentes d'ondes. Le réseau possede 21 neurones en entrée et un neurone en sortie. Les échantillons d'apprentissage sont issus du problème de reconnaissance proposé par L. Brieman, J.H. Freidman, R.A. Oslhen and C. J. Stone dans le livre "Classifiaction and Regression Trees" édité par Belmont (C.A. Wadsworth) en 1984. La base d'apprentissage possede 192 exemples et la base de généralisation 200 exemples.

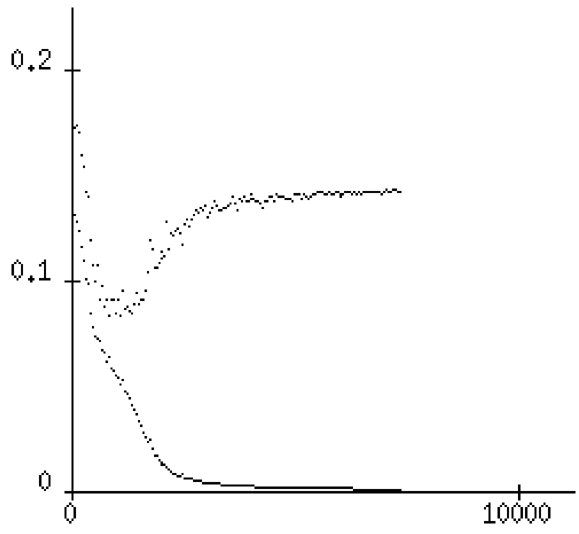

Figure 11: *Trop d'apprentissage dégrade les performances. En abscisse, le nombre d'itérations, et en ordonnée l'erreur globale sur une partie la base d'apprentissage (10 exemples) et, au-dessus, sur une partie restreinte de 30 exemples de la base de généralisation*

La figure 11 est un peu artificielle: j'ai utilise des bases restreintes pour l'apprentissage et pour la généralisation afin d'accentuer le phénomène. En utilisant des bases plus importantes le surapprentissage est moins visible.

5.3 Etude des performances.

J'ai cherché à comparer les performances de la rétropropagation classique (ϵ constant) et de l'algorithme d'Augereau ($\lambda(n)$ constant), sur le plan de la dégradation des performances en généralisation. Les courbes suivantes ont été obtenues à partir de conditions initiales identiques, et avec différentes valeurs du nombre de Lyapunov λ. Le gradient est stochastique, la base d'apprentissage fait 10 exemples et la base de généralisation 200 exemples. Evidemment on ne peut pas comparer la vitesse relative des deux algorithnes car le pas n'est pas le même. Pour cette raison, voici les courbes obtenues avec une seule valeur de ϵ pour la méthode classique et différentes valeurs de λ pour la méthode d'Augereau, qui font varier la vitesse de convergence. On constate cependant une différence au niveau de la forme des courbes...

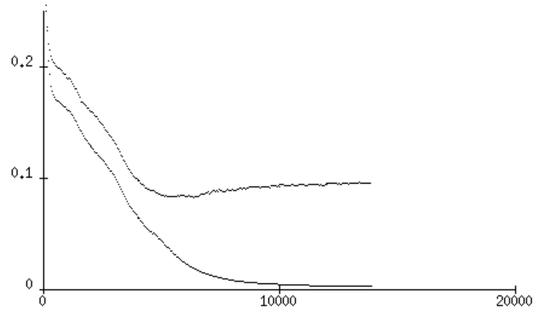

Figure 12: *Rétropropagation classique.*$\epsilon = 0.0003$

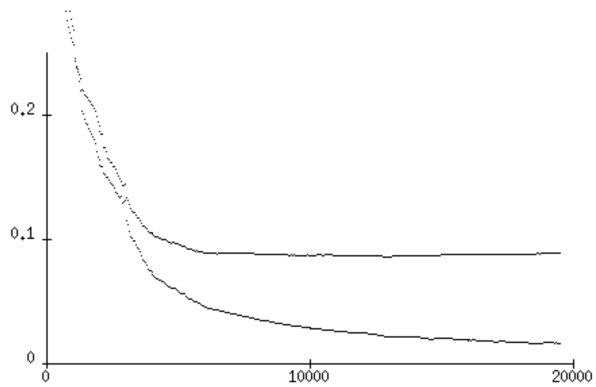

Figure 13: *Méthode d'Augereau.*$\lambda = 0.003$

On constate tout d'abord que le surapprentissage est plus marqué dans le cas de l'algorithme d'Augereau qu'avec la rétropropagation classique. L'algorithme "à nombre de Lyapunov imposé" n'est pas intéressant sur le plan de la généralisation. A priori l'explication n'est pas claire: on impose au réseau d'évoluer, de converger à vitesse constante, alors que l'apprentisage est terminé et que l'on devrait se trouver dans un état stable.

Les quatre courbes avec la rétropropagation d'Augereau sont de la meme forme: Seule la vitesse varie. J'ai vérifié que dans le cas le plus lent (avec $\lambda = 0.003$) l'erreur en généralisation revient bien, après un temps très long, à des valeurs similaires à ce qu'on trouve dans le cas le plus rapide.

La convergence est d'autant plus rapide que λ est négatif, mais on ne peut pas espérer

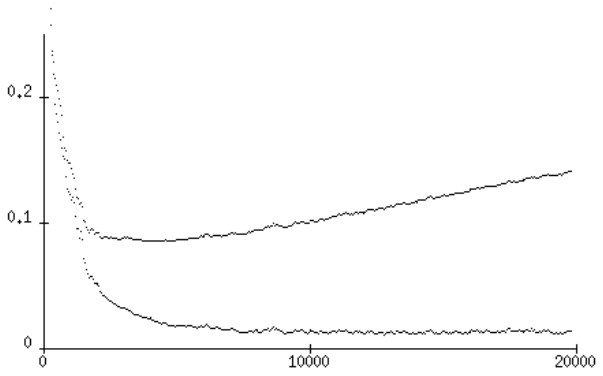

Figure 14: *Méthode d'Augereau.*$\lambda = 0.01$

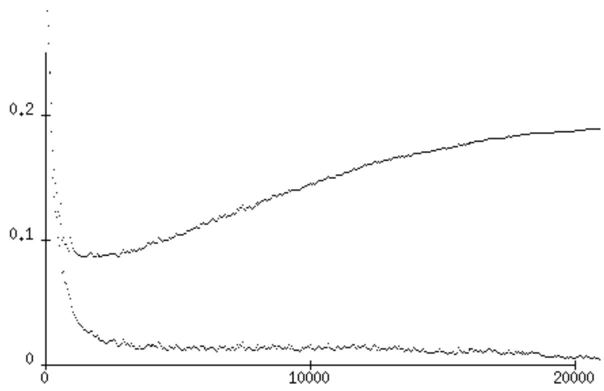

Figure 15: *Méthode d'Augereau.*$\lambda = 0.02$

donner à λ des valeurs trop petites sans quoi l'algorithme fournit des valeurs du pas qui sont trop elevées et on tombe dans des phénomênes de saturation, avec des sauts fréquents de la fonction d'erreur. Ici sur ces graphiques j'ai volontairement produit des cas ou la courbe d'erreur ne "saute" pas pour pouvoir comparer les courbes entre elles.

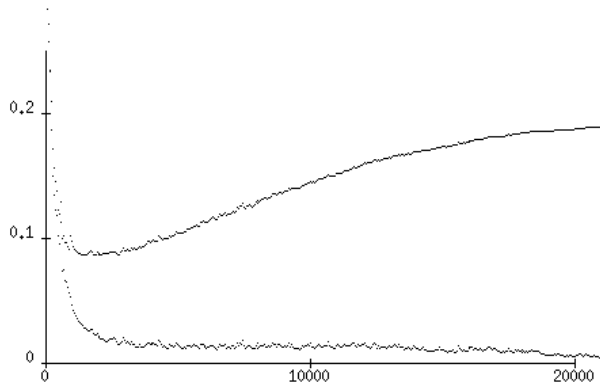

Figure 16: *Méthode d'Augereau.*$\lambda = 0.03$

6 Conclusion

La méthode de calcul des nombres de Lyapunov présentée par Bertrand Augereau est très efficace, elle est certainement la plus adaptée à des réseaux de neurones. Néanmoins elle est très limitée: on ne peut pas l'appliquer à des algorithmes du second ordre, ou à la technique de weight decay...(les formules ne se généralisent pas).

Son algorithme d'apprentissage possède des propriétés intéressantes et inhabituelles. La plus intéressante est peut-être l'aptitude d'échapper aux minimums locaux. Cependant pour en tirer pleinement parti, on a intérêt à appliquer une stratégie de type "pocket": garder en mémoire la meilleure solution que l'on a rencontrée au cours de la recherche des poids, car la courbe d'erreur n'est pas décroissante.

Enfin il faut garder présent à l'esprit le fait que cette méthode ne parvient pas à imposer une valeur au nombre de Lyapunov du réseau: celui-ci choisit, par moments de "sauter" ailleurs dans l'espace des phases, et à ces moments les approximations faites dans ce calcul deviennent fausses...

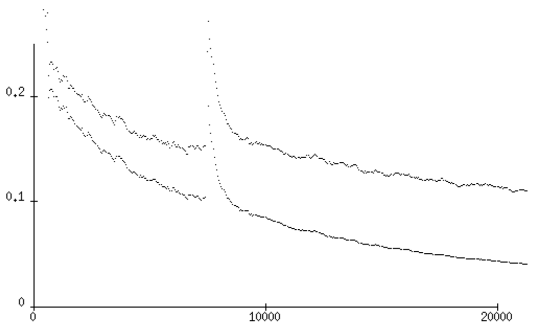

Figure 17: *Les sauts correspondent à des ruptures de régularité des nombres de Lyapunov instantanés.*

A Le calcul proposé par Augereau

On note T la fonction vectorielle de modification des poids: $w(n+1) = T(w(n))$, soit $w_{ji}(n+1) = T_{ij}(w_{ji}(n))$ pour tout i et j.

Alors $T_{ji}(w) = w_{ji} + \epsilon\Delta_{ji}(w)$

Bertrand Augereau se base sur une approximation numérique pour calculer les valeurs propres de la matrice jacobienne:

$$J_c(w) = \left(\frac{\partial T_i(w)}{\partial w_j}\right)_{i,j \in \{1 \ldots c\}}$$

deux cas se présentent pour les dérivées partielles:

$\frac{\partial T_i(w)}{\partial w_j} = 1 + \epsilon\frac{\partial \Delta_i(w)}{\partial w_i}$ et $\frac{\partial T_i(w)}{\partial w_j} = \epsilon\frac{\partial \Delta_i(w)}{\partial w_j}$

D'où l'expression de la matrice en fonction des dérivées des poids et du pas du gradient:

$$J_c(w) = \begin{pmatrix} 1+\epsilon\frac{\partial \Delta_1(w)}{\partial w_1} & \ldots & \epsilon\frac{\partial \Delta_1(w)}{\partial w_c} \\ \ldots & 1+\epsilon\frac{\partial \Delta_i(w)}{\partial w_i} & \ldots \\ \epsilon\frac{\partial \Delta_c(w)}{\partial w_1} & \ldots & 1+\epsilon\frac{\partial \Delta_c(w)}{\partial w_c} \end{pmatrix}$$

pour calculer ce jacobien, Augereau fait l'approximation suivante:

$$\frac{\partial \Delta_{i_1}(w)}{\partial w_{j_1}}\frac{\partial \Delta_{i_2}(w)}{\partial w_{j_2}} = \frac{\partial \Delta_{i_1}(w)}{\partial w_{j_2}}\frac{\partial \Delta_{i_2}(w)}{\partial w_{j_1}}$$

cette approximation lui permet de négliger les termes en $\epsilon^n, n > 1$ et de poser

$$det(J_c(w)) = 1 + \epsilon\sum_{i=1}^{c}\frac{\partial \Delta_i(w)}{\partial w_i}$$

Cette approximation permet aussi le calcul des valeurs propres:

$\beta_1(w_n) = 1$ d'ordre $c-1$ et $\beta_2(w_n) = 1 + \epsilon\sum_{i=1}^{c}\frac{\partial \Delta_i(w_n)}{\partial w_i}$ d'ordre 1.

On cherche à étudier l'évolution d'une petite déviation Δw_n, et pour ce faire, Augereau étudie des grandeurs géométriques. En effet, à chaque étape de la rétropropagation et à chaque élément du spectre de Lyapunov correspond un sous-espace propre, l'espace des vecteurs propres associés à cet élément. Pour $\beta_1(w_n)$ on a un espace $E_{1,n}$ de dimension c-1 et pour $\beta_2(w_n)$ on a un espace $E_{2,n}$ de dimension 1. Δw_{n-1} se décompose dans chacun des sous-espaces orthogonaux selon les vecteurs propres $P_{1,n-1} \in E_{1,n-1}$ et $P_{2,n-1} \in E_{2,n-1}$: $\Delta w_{n-1} = P_{1,n-1} + P_{2,n-1}$. A l'étape suivante, $\Delta w_n = \beta_1(w_{n-1})P_{1,n-1} + \beta_2(w_{n-1})P_{2,n-1}$

Augereau définit à partir de ces deux vecteurs $P_{1,n-1}$ et $P_{2,n-1}$ un élément de surface R_{n-1}. De plus, il définit R_n basé sur la décomposition de Δw_n en :$\Delta w_n = \beta_1(w_{n-1})P_{1,n-1} \in E_{1,n-1}$ et $\beta_2(w_{n-1})P_{2,n-1} \in E_{2,n-1}$.

Au lieu de suivre l'évolution de la norme de Δw_n on mesure l'augmentation de surface de R_n. Ainsi on a $\Theta(R_{n-1}) = ||P_{1,n-1}||.||P_{2,n-1}||$ et $\Theta(R_n) = ||\beta_1(w_{n-1})P_{1,n-1}||.||\beta_2(w_{n-1})P_{2,n-1}||$

Le taux de convergence est alors, selon Augereau, $\theta = lim_{n->\infty}(\prod_{i=0}^{n-1})^{\frac{1}{n}}$ où les θ_i sont les "taux locaux" $\theta_i = \frac{\Theta(R_n)}{\Theta(R_{n-1})} = |\beta_1(w_i)\beta_2(w_i)|$

Par la suite, on définit les nombres de Lyapunov instantanés comme les logarithmes de ces taux locaux.

Une telle démarche ne me semble pas rigoureuse, bien que les résultats obtenus lors des calculs soient satisfaisants. En effet la direction des vecteurs propres change à chaque itération, et la notation R_n est donc confuse: elle désigne à la fois l'élément de surface porté par les vecteurs $P_{1,n-1}$ et $P_{2,n-1}$ et celui porté par $P_{1,n}$ et $P_{2,n}$...

NB: les coefficients L_{ji} dont il est question en 3.2 sont définis par:

$$L_{ij} = \frac{\partial \Delta_{ji}}{\partial x_{ji}}$$

Alors d'après le calcul des valeurs propres précédent, on a bien:

$$\lambda_n = \frac{1}{n}\sum_{p=0}^{n-1} ln(1 + \epsilon \sum_{(i.j)\in C_A} L_{ij}^p)$$

B Choix d'implémentation.

SN est un simulateur de réseaux de neurones regroupant divers algorithmes de rétropropagation, selon des méthodes variables: algorithme du gradient, gradient conjugué, méthode de Newton, etc. Il est essentiellement programmable en LISP. Le noyau du simulateur a été écrit en C et incorporé à l'interpreteur LISP TL3, sous forme de bibliothèques de procédures bas niveau (création du réseau, manipulations directes...). Les procédures de haut niveau du simulateur (apprentissage...) sont regroupées dans des bibliothèques en LISP, et permettent de générer et d'entraîner les réseaux avec des procédures standard. Le but de mon stage était d'implémenter une méthode de calcul des nombres de Lyapunov en LISP, qui soit integrée au simulateur SN, et de l'appliquer à divers algorithmes de rétropropagation.

La méthode d'Augereau passe par le calcul de nombres L_{ji} relatifs aux connexions entre les neurones i et j. Comme le simulateur SN est architecturé autour d'un noyau en C ou sont définis tous les champs des objets neurone, synapse... et ou sont effectuées toutes les opérations de bas niveau sur ces champs, il eut été normal de modifier ce noyau pour incorporer le champ L_{ji} au type synapse.

Malheureusement les sources du simulateur n'étant pas disponibles au laboratoire, il m' était impossible de modifier le noyau. Programmer en LISP ou en C une structure contenant les L_{ji} et disjointe des structures utilisées pas SN aurait conduit à des procédures peu pratiques, compte tenu de l'architecture du simulateur. J'ai donc opté pour une troisième solution, peu propre mais qui a le mérite de tirer parti du noyau de SN: Les champs internes des connexions ne sont pas tous utilisés par l'algorithme classique de rétropropagation. En particulier , le champ s-sigma contient la derivée seconde du poids de la connexion, et n'est utilisé que par des méthodes dites du second ordre. J'ai choisi d'y mettre la valeur L_{ji}, et de renommer ce champ ainsi que les procédures qui y donnent accès. Je ne pense pas qu'il y ait de cas où l'on puisse avoir besoin à la fois de ce champ et de calculer L_{ji}, la méthode décrite par Augereau étant adaptée à la seule rétropropagation classique.

Références

[1] Krogh et Hertz, **Introduction to Neural Networks**

[2] Daniel F. Mc AFFREY, Stephen ELLNER, A.Ronald GALLANT et Douglas W. NYCHKA.,**Estimating the Lyapunov Exponent of a Chaotic System with nonparametric regression**, Journal of the American Statistical association, Sept 1992.

[3] Thomas S. PARKER et Leon O. CHUA, **Chaos: A Tutorial for Engineers**, Proceedings of the IEEE, Aout 1987.

[4] Bertrand Augereau, **Contrainte de dynamique non linéaire pour l'amélioration de l'apprentissage et l'élaboration de l'architecture des réseaux neuronaux multicouches** ,Université de Poitiers,1996.

[5] William A. GARDNER,**Introduction to Random Processes with application to Signals and Systems**, McGraw-Hill.

[6] Murray ROSENBLATT, **Stationnary Sequences and Random Fields**, Birkhauser Boston, 1985

Table des Matières