Identification de loi de comportement par réseaux neuronaux entraînés par algorithmes évolutionnaires

Guillaume GERBER
Rapport de stage du DEA d'Analyse Numérique de Paris VI
effectué au Centre de Mathématiques Appliquées de l'Ecole Polytechnique
sous la direction de M. Marc Schoenauer

Mars-Juin 2000

Remerciements

Je tiens tout d'abord à exprimer toute ma gratitude et ma reconnaissance à Mr Marc Schoenauer, responsable de l'équipe Evolution Artificielle et Apprentissage de l'Ecole Polytechnique, pour m'avoir permis d'effectuer ce stage dans des conditions optimales et de m'avoir encadré avec attention tout au long de ces quatre mois.

Je tiens également à remercier vivement :

- Mr François Jouve, chercheur au CMAP, pour toute l'aide qu'il m'a apportée concernant la partie mécanique,
- Sana Ben-Hamida, Sofiane Oussedik, Jean-Phillipe Ayanides et Kamal Abboud, thésards au CMAP, ainsi que Cyrille Barbot, également stagiaire de DEA, pour leur bonne humeur quotidienne et leurs conseils avisés qui ont permis de résoudre mes problèmes informatiques,
- Mme Geo Boleat, secrétaire du CMAP, pour la gentillesse de son accueil lors de mon premier jour au laboratoire,
- les responsables de la maintenance informatique du CMAP.

Table des matières

Introduction

Que se soit par exemple pour l'identification d'une loi physique ou encore pour le contrôle d'un système dynamique, l'ingénieur se retrouve parfois confronté à un problème de recherche de fonction pour lequel aucun exemple d'entraînement n'est disponible. Il s'agit typiquement d'un problème inverse dont on sait résoudre le problème direct et qui peut s'exprimer sous la forme d'un problème d'optimisation dans un espace fonctionnel, mais qui ne peut être résolu par une méthode déterministe de type descente de gradient.

Avec le développement du calcul évolutionnaire, de multiples recherches ont permis de mettre en évidence les bonnes capacités des algorithmes évolutionnaires (AEs) à répondre à ce type de problème mal posé. Ainsi, le calcul évolutionnaire offre une alternative facile à mettre en oeuvre et qui permet souvent de trouver une approximation satisfaisante de la solution. On retiendra néanmoins que pour obtenir un résultat par évolution artificielle, il faut payer le prix de nombreuses résolutions du problème direct.

L'objet du stage que j'ai effectué au Centre de Mathématiques Appliquées de l'Ecole Polytechnique (CMAP) est l'étude d'une approche évolutionnaire appliquée à des réseaux neuronaux pour la résolution d'un problème inverse qui intervient en mécanique : l'identification de loi de comportement.

Le premier chapitre présente les principaux outils mis en jeu, c'est à dire les réseaux de neurones “feed-forward” et les algorithmes évolutionnaires. Avant de s'intéresser au problème mécanique, nous verrons dans la deuxième partie une application plus simple d'apprentissage neuronal par algorithmes évolutionnaires sur un problème classique de contrôle dynamique : le “Lunar Lander”. Enfin, dans la troisième partie, nous tenterons d'évaluer la validité de l'apprentissage évolutionnaire des réseaux de neurones sur l'identification de loi de comportement.

Chapitre 1

Présentation des outils

1.1 Les réseaux de neurones

Après quelques définitions précisant le principe des réseaux de neurones, nous nous attarderons plus particulièrement sur le perceptron, qui a été le principal modèle utilisé au cours de cette étude. Nous terminerons en détaillant la dérivation d'un réseau feed-forward, qui est nécessaire pour l'identification neuronale de loi de comportement.

1.1.1 Généralités sur les réseaux de neurones artificiels

Les réseaux de neurones au même titre que les algorithmes évolutionnaires font partis des métaphores biologiques utilisées pour la résolution de problèmes numériques. Comme leur nom l'indique ils s'inspirent du fonctionnement cérébral de l'être humain, en se basant principalement sur le concept de neurone.

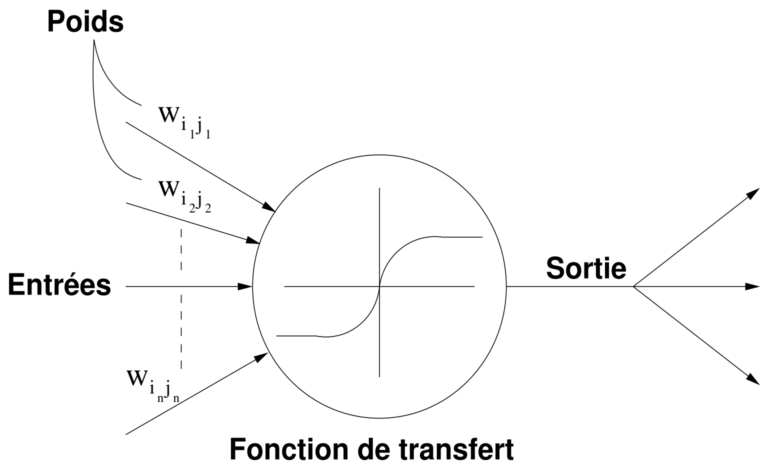


FIG. 1.1 – *Représentation d'un neurone artificiel*

Un neurone artificiel est un processeur élémentaire. Il recoit un nombre variable d'entrée en provenance de neurones en amont. A chacune de ces entrées est associé un poids représentatif de l'importance de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals.

Le traitement réalisé par un neurone se déroule en deux phases : La première consiste à faire la somme pondérée des entrées, qu'on appelle activation. A partir de cette valeur, une fonction de transfert calcule la valeur de l'état du neurone. C'est cette valeur qui sera transmise aux neurones en aval.

Les neurones ainsi connectés forment un réseau qui communique avec l'extérieur par l'intermédiaire de neurones d'entrée et de neurones de sortie, les neurones intermédiaires étant appelés neurones cachés. Dans le cadre de ce stage les réseaux de neurones servent de base pour représenter des fonctions : les entrées du réseau sont les variables et la sortie représente l'image.

L'atout majeur des réseaux de neurones réside dans leur capacité d'apprentissage qui est permise par la possibilité de modifier les poids des connexions.

1.1.2 Réseaux feed-forwards et perceptrons

Les réseaux feed-forward forment une catégorie de réseaux qui se caractérisent par le fait que le signal s'y propage unilatéralement, autrement dit ils ne possèdent pas de boucle retour. Cette spécificité facilite le calcul de la simple propagation tout comme celui des dérivées. C'est la raison pour laquelle nous avons choisi cette catégorie de réseaux pour représenter les fonctions qui sont appelées à de nombreuses reprises pour le calcul des deux problèmes directs étudiés dans le cadre de ce stage.

Le type de réseau que j'ai effectivement utilisé au cours de ce stage est en fait un modèle particulier de réseaux feed-forwards : le perceptron.

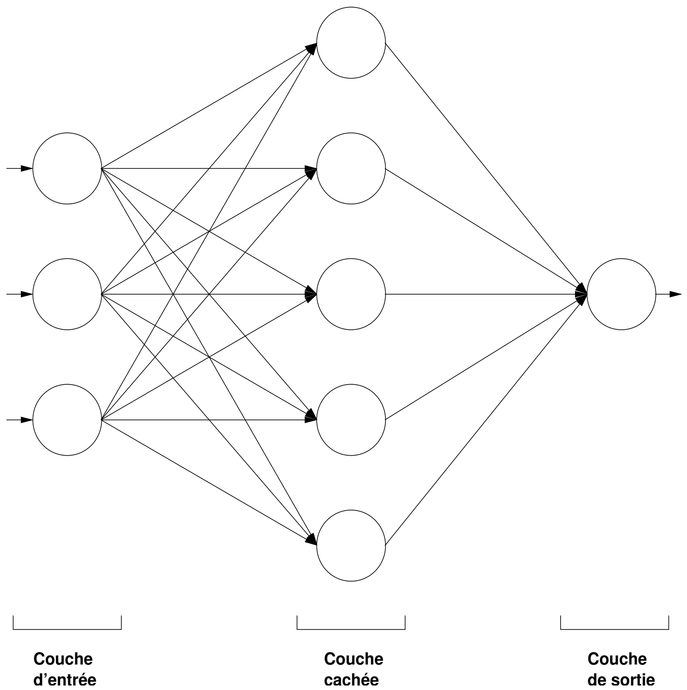


FIG. 1.2 – *Organisation d'un perceptron*

Les neurones d'un perceptron sont organisés en couches. Il n'y a pas de connexions entre neurones d'une même couche et chaque neurone d'une couche est connecté à tous les neurones de la couche suivante. Les perceptrons sont les réseaux traditionnellement utilisés pour l'interpolation de fonctions par back-propagation. Cette dernière est une méthode d'entraînement dont le principe est d'ajuster les poids par une descente de gradient.

1.1.3 Résultats théoriques relatifs aux réseaux neuronaux

Un premier résultat d'approximation

Le théorème de Kolmogorov (1957) fait partie des résultats importants qui ont amené de nombreux chercheurs à s'intéresser aux réseaux de neurones. Ce théorème révèle que pour toute fonction dans un espace de dimension N, il existe un réseau feed-forward à trois couches, avec 2N+1 neurones cachés munis de fonctions de transfert appropriées, capable de l'implémenter exactement. Ce théorème présente néanmoins le défaut de ne donner aucun renseignement concernant la nature des fonctions de transfert. Hecht-Nielsen (1987) a répondu à cette question en démontrant que toute fonction peut être approchée avec une précision arbitraire par un réseau perceptron doté de neurones cachés à fonctions sigmoïdales. Ce résultat signifie donc que les perceptrons à fonctions de transfert sigmoïdales sont des approximateurs universels. En revanche, on remarquera qu'il ne fournit aucune information concernant le nombre de neurones cachés nécessaires.

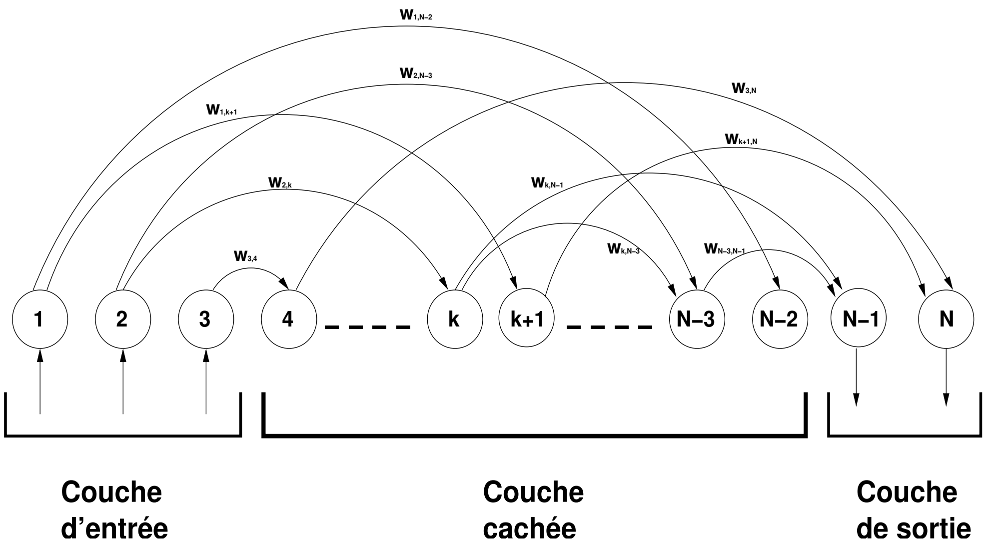
Approximation des dérivées

L'objectif de l'identification de loi de comportement, qui nous intéressera dans la troisième partie, n'est pas seulement la loi elle-même, mais également ses dérivées premières et secondes qui sont les valeurs effectivement utilisées au cours du calcul d'élasticité. La question est alors de savoir si les réseaux de neurones, qui serviront de base de recherche aux lois de comportement, sont capables d'approcher convenablement les dérivées de n'importe quelle fonction. Des travaux de Hornik, Stinchcombe et White (1991) ont montré que les réseaux feed-forward à trois couches avec des neurones cachés munis de sigmoïdes peuvent approximer non seulement une fonction quelconque mais également toutes ses dérivées. Ce résultat théorique même s'il ne renseigne toujours pas sur le nombre de neurones cachés m'ont conduit à préferer pour la suite la sigmoïde aux autres types de fonction de transfert pour lesquels il n'existe pas de tels résultats.

1.1.4 Dérivation d'un réseau feed-forward

La résolution du système de l'élasticité requiert les dérivés premières et secondes de la loi de comportement. Voici donc les formules qui permettent d'obtenir les dérivées premières et secondes d'un réseau feed-forward qui sera sensé représenter une loi de comportement.

Considérons un réseau feed-forward quelconque :



Appelons w_{ij}, $i < j$ le poids de la connexion reliant le neurone i au neurone j. Soit I_k l'ensemble des neurones situés en amont du neurone k et qui lui sont connectés. Soit f la fonction de transfert des neurones cachés. Notons d'autre part x_k les entrées du réseau et o_k la sortie du neurone k.

Simple propagation :

– pour $n \in$ couche d'entrée:

$$o_n = x_n$$

– pour $n \in$ couche cachée :

$$o_n = f\left(\sum_{k \in I_n} w_{kn} o_k\right)$$

– pour $n \in$ couche de sortie :

$$o_n = \sum_{k \in I_n} w_{kn} o_k$$

Dérivée première :

– pour $n \in$ couche d'entrée :

$$\frac{\partial o_n}{\partial x_i} = \delta_i$$

– pour $n \in$ couche cachée :

$$\frac{\partial o_n}{\partial x_i} = \left(\sum_{k \in I_n} w_{kn} \frac{\partial o_k}{\partial x_i}\right) f'\left(\sum_{k \in I_n} w_{kn} o_k\right)$$

– pour $n \in$ couche de sortie :

$$\frac{\partial o_n}{\partial x_i} = \sum_{k \in I_n} w_{kn} \frac{\partial o_k}{\partial x_i}$$

Dérivée seconde :

- pour $n \in$ couche d'entrée :

$$\frac{\partial^2 o_n}{\partial x_i \partial x_j} = 0$$

- pour $n \in$ couche cachée:

$$\frac{\partial^2 o_n}{\partial x_i \partial x_j} = \left(\sum_{k \in I_n} w_{kn} \frac{\partial^2 o_k}{\partial x_i \partial x_j} \right) f' \left(\sum_{k \in I_n} w_{kn} o_k \right) + \left(\sum_{k \in I_n} w_{kn} \frac{\partial o_k}{\partial x_i} \right) \left(\sum_{k \in I_n} w_{kn} \frac{\partial o_k}{\partial x_j} \right) f'' \left(\sum_{k \in I_n} w_{kn} o_k \right)$$

- pour $n \in$ couche de sortie :

$$\frac{\partial^2 o_n}{\partial x_i \partial x_j} = \sum_{k \in I_n} w_{kn} \frac{\partial^2 o_k}{\partial x_i \partial x_j}$$

Indiquons que si f est une sigmoide, on peut limiter le nombre d'exponentielles à calculer en considérant les relations :

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

$$f''(x) = f(x)(1 - f(x))(1 - 2f(x))$$

1.2 Les algorithmes évolutionnaires (AEs)

Après une brève présentation des algorithmes évolutionnaires, nous nous intéresserons plus particulièrement à leur application aux réseaux de neurones en définissant une représentation génotypique des réseaux à structure fixe. Nous présenterons ensuite les stratégies d'évolution qui ont été le principal algorithme utilisé dans le cadre de ce stage pour faire évoluer les populations de réseaux de neurones. Enfin, nous évoquerons la librairie utilisée pour l'implémentation de toute la partie évolutionnaire : la librairie EO.

1.2.1 Généralités sur les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des algorithmes stochastiques généralement utilisés en dernier recours pour résoudre des problèmes d'optimisation pour lesquels il est impossible d'utiliser une méthode déterministe telle qu'une méthode de descente de gradient. Leur principal inconvéniant est que leur application demande toujours un temps calcul très important en comparaison d'un algorithme déterministe.

Ils prennent leur inspiration dans les mécanismes de la sélection darwinienne et de la génétique de l'évolution. Leur but consiste à optimiser une fonction définie par l'utilisateur appelée fonction d'adéquation ou de fitness. Pour atteindre cette objectif, l'algorithme travaille en parallèle sur une population d'individus distribués dans la totalité de l'espace de recherche. Chaque individu ou chromosome est généralement constitué d'un ensemble d'éléments ou gènes. L'objectif est de chercher la combinaison optimale de ces gènes qui donne lieu au maximum d'adéquation. A chaque itération, appelé génération, est créée une nouvelle population avec le même nombre d'indvidus. Cette nouvelle génération consiste généralement en des individus mieux adaptés à l'environnement tel qu'il est représenté par la fonction d'adéquation. Au fur et à mesure des générations les individus vont tendre vers l'optimum de la fonction de fitness. Pour assurer cette convergence les algorithmes évolutionnaires se basent sur une heuristique très simple : les meilleures solutions seront trouvées dans des régions de l'espace de recherche contenant des proportions relativement élevées de bonnes solutions et, en outre, ces régions peuvent être identifiées par échantilonage robuste et judicieux de l'espace des solutions.

Le fonctionnement général d'un algorithme évolutionnaire peut être schématisé de la façon suivante :

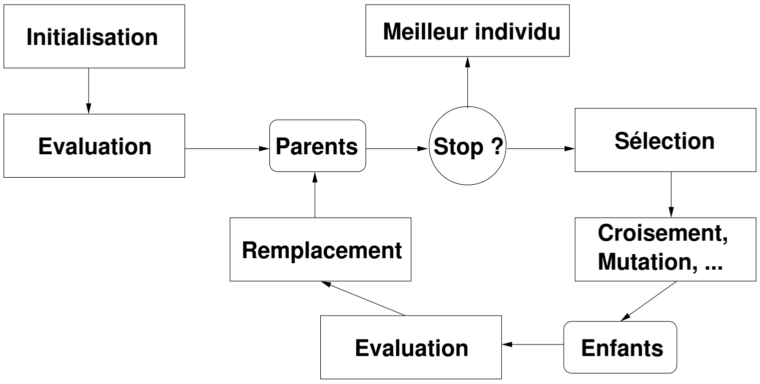


FIG. 1.3 – *Schéma général d'un algorithme évolutionnaire*

Un algorithme évolutionnaire est composé de cinq étapes :

initialisation :

la population initiale s'obtient en générant des individus uniformément dans l'espace de recherche.

évaluation :

l'algorithme évalue la fonction d'adéquation f_i de chaque individu i. C'est en général l'étape qui consomme la quasi-totalité du temps calcul.

sélection :

l'algorithme sélectionne dans la population les individus qui auront des descendants en créant une population intermédiaire de parents. Cette sélection qui simule le mécanisme de la sélection naturelle en biologie est biaisé vers les individus qui ont la meilleure fitness. Elle peut être soit déterministe soit stochastique.

reproduction avec croisement et mutation :

l'algorithme recombine les individus sélectionnés au moyen d'opérateurs génétiques tels que la mutation et le croisement, qui d'un point de vue algorithmique peuvent être vus comme des mécanismes servant à changer localement les solutions représentées par les parents (mutation) ou à les combiner. La mutation agit en modifiant aléatoirement un ou plusieurs gènes d'un chromosome, tandis que le croisement échange ou mixe des gènes d'un parent avec ceux de l'autre.

remplacement :

l'algorithme choisit parmis les enfants et les membres de la population les individus qui constitueront la nouvelle génération. Ce remplacement peut être générationnel c'est à dire que les enfants obtenus remplacent totalement les individus de l'ancienne génération (Simple Genetic Algorithm). Il peut également être elitiste : la nouvelle géneration est constitué des meilleurs individus parmis les enfants et les individus de l'ancienne génération (Evolution Strategies +).

Outre le choix des opérateurs, trois paramètres doivent être fixés : le nombre d'individus dans la population, le taux de croisement et le taux de mutation.

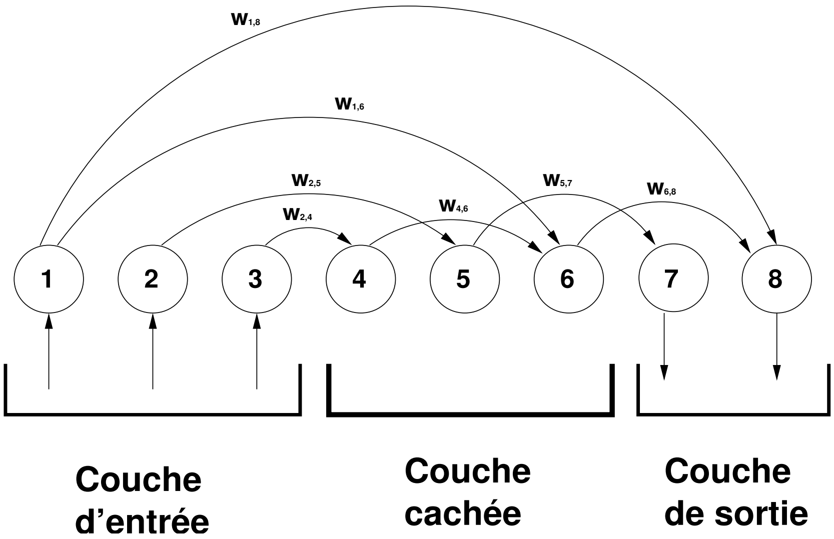
Les algorithmes évolutionnaires les plus connus sont les algorithmes génétiques (GA) qui sont principalement utilisés sur des problèmes d'optimisation sur chaines de bits. Les algorithmes évolutionnaires s'appliquent également à des problèmes réels avec notamment les stratégies d'évolution (ES). Enfin indiquons qu'avec la programmation génétique (GP) ont été définis des opérateurs déstinés à faire évoluer des arbres.

La section suivante présente la façon dont on a représenté les réseaux de neurones en tant que chromosome dans le cadre de ce stage.

1.2.2 Génotype d'un réseau feed-forward

Pour faire évoluer une population de réseaux de neurones il est nécessaire de définir la représentation génotypique de ces réseaux considérés dès lors comme des chromosomes. Je me suis limité dans le cadre de ce stage à fixer à chaque fois préalablement la topologie pour les réseaux de la population. Le fait figer ainsi la structure des réseaux entraîne que les seules variables sont les poids et que l'on se retrouve finalement avec un problème d'optimisation paramétrique. Ceci implique que l'on peut représenter de manière équivalente un réseau de neurones par un vecteur de réels dont les composantes représentent les poids des connexions du réseau.

Si l'on considère par exemple le réseau feed-forward suivant:



où w_{ij}, $i < j$ est le poids de la connexion reliant le neurone i au neurone j. On peut ranger naturellement les poids dans un vecteur donné par :

$$(w_{1,6},w_{1,8},w_{2,5},w_{3,4},w_{4,6},w_{5,7},w_{6,8})$$

Dans la suite on se servira donc de cette représentation sous forme de vecteur de réels pour appliquer aux réseaux neuronaux les opérateurs génétiques de croisement et de mutation. La section suivante présente un algorithme évolutionnaire qui est fréquemment utilisé pour l'optimisation de chaînes de réels et qui se révélera être l'algorithme le plus intéressant pour faire évoluer une population de réseaux de neurones.

1.2.3 Les stratégies d'évolution

Les stratégies d'évolution sont une variante d'algorithme évolutionnaire utilisée dans le cadre de l'optimisation de fonctions réelles. Elles se distingent tout d'abord des algorithmes classiques par le fait qu'elle ne comprend pas d'étape de sélection puisque tous les individus de la population génèrent le même nombre d'enfants. Mais leur principale originalité réside dans l'opérateur de mutation qui s'adapte au fur et à mesure de l'évolution en tâchant d'orienter les membres de la population dans les directions les plus prometteuses. Cette fonction est réalisée en rattachant à chaque gène un paramètre de contrôle σ_i évoluant en parallèle et qui détermine la variance de la mutation gaussienne qui est appliquée à ce gène.

Voici le détail du déroulement d'un algorithme ES :

- on génère une population de μ individus. Chaque individu est pris comme un vecteur de paires de réels (x^i,σ^i), $i = 1,...,\mu$ où les x_i représentent les poids des connexions du réseau et les σ_i correspondent aux paramètres de la mutation adaptative
- la fitness de chaque individu $(x^i,\sigma^i,)$ $i = 1,...,\mu$ de la population est évaluée en se basant sur la fonction objectif à optimiser, $f(x^i)$

- si l'on doit procéder au croisement entre deux individus, on les sélectionne au hasard dans la population :

$$\left(x^{i_1},\sigma^{i_1}\right) = ((x_1^{i_1},...,x_n^{i_1}),(\sigma_1^{i_1},...,\sigma_n^{i_1}))$$

$$\left(x^{i_2},\sigma^{i_2}\right) = ((x_1^{i_2},...,x_n^{i_2}),(\sigma_1^{i_2},...,\sigma_n^{i_2}))$$

  Il y a deux types d'opérateurs de crossover :

le croisement discret :

on échange les gènes aléatoirement

$$\left(x^{i_1},\sigma^{i_1}\right) = ((x_1^{\varepsilon_1(i_1)},...,x_n^{\varepsilon_n(i_1)}),(\sigma_1^{\varepsilon_1(i_1)},...,\sigma_n^{\varepsilon_n(i_1)}))$$

$$\left(x^{i_2},\sigma^{i_2}\right) = ((x_1^{\varepsilon_1(i_2)},...,x_n^{\varepsilon_n(i_2)}),(\sigma_1^{\varepsilon_1(i_2)},...,\sigma_n^{\varepsilon_n(i_2)}))$$

où : $\varepsilon_k = Id$ ou $\varepsilon_k(i_1) = i_2$ et $\varepsilon_k(i_2) = i_1$

le croisement intermédiaire :

$$\left(x^{i_1},\sigma^{i_1}\right) = ((\alpha x_1^{i_1}+(1-\alpha)x_1^{i_2},...,\alpha x_n^{i_1}+(1-\alpha)x_n^{i_2}),(\alpha\sigma_1^{i_1}+(1-\alpha)\sigma_1^{i_2},...,\alpha\sigma_n^{i_1}+(1-\alpha)\sigma_n^{i_2}))$$

$$\left(x^{i_2},\sigma^{i_2}\right) = (((1-\alpha)x_1^{i_1}+\alpha x_1^{i_2},...,(1-\alpha)x_n^{i_1}+\alpha x_n^{i_2}),((1-\alpha)\sigma_1^{i_1}+\alpha\sigma_1^{i_2},...,(1-\alpha)\sigma_n^{i_1}+\alpha\sigma_n^{i_2}))$$

où α est un paramètre variant entre 0 et 1.

- chaque individu (x,σ) de la population engendre le même nombre λ/μ d'enfants par mutation. On commence par calculer un paramètre κ

$$\kappa = \tau N(0,1)$$

Puis chaque enfant (x',σ') est obtenu à partir du parent (x,σ) par :

$$pour\ i = 1,..,n, \quad \sigma_i' = \exp(\kappa + \tau' N(0,1))$$

$$et \quad x_i' = x_i + N(0,\sigma_i')$$

où les paramètres τ et τ' prennent selon les recommandations de Schwefel les valeurs suivantes :

$$\tau \propto \frac{1}{\sqrt{2\sqrt{n}}} \qquad \tau' \propto \frac{1}{\sqrt{2n}}$$

- enfin le remplacement est déterministe. Il peut se faire soit parmi les λ enfants $(\mu,\lambda) - ES$ ou soit parmi les μ parents et les λ enfants $(\mu + \lambda) - ES$.

1.2.4 La librairie EO

Pour l'implémentation des algorithmes évolutionnaires, j'ai utilisé la librairie EO. Il s'agit d'une librairie en "Open Source" écrite en C++ qui possède les élements de base pour programmer des algorithmes évolutionnaires. Son principal intérêt réside dans son caractère abstractif marqué qui permet de définir puis de faire évoluer facilement de nouveaux objets tels que les réseaux de neurones. Elle est téléchargeable sur internet à l'adresse : *http://www.eeaax.polytechnique.fr/EO/*

Chapitre 2

Contrôle neuronal d'un module lunaire

Avant d'appliquer les réseaux de neurones à l'identification de loi de comportement je me suis interessé à un problème plus simple d'apprentissage neuronal par algorithmes évolutionnaire : contrôler l'atterrissage d'un module lunaire.

2.1 Cadre du problème

Dans ce problème on s'intéresse à un module qui s'apprête à alunir. Celui-ci possède un réseau de neurones dont le rôle est de contrôler la poussée verticale des réacteurs en fonction des paramêtres physiques du module que sont l'altitude et la vitesse verticale. L'objectif est de trouver un réseau capable d'atterrir le plus doucement possible.

Le module est laché depuis une altitude a^0 avec une vitesse initiale v^0. La gravité qui s'exerce sur le module est celle de la Lune : $1.5m/s^2$. On suppose que la quantité de fuel est limitée à 100 unités. Chaque unité de fuel consommée produit une unité de poussée. La poussée maximale est limitée à 10 unités et la variation de la masse du module due à la consommation du fuel n'est pas prise en compte. On désire en outre que le réseau soit capable d'alunir à partir de conditions initiales comprises dans un intervalle de [500m,1500m] pour l'altitude et de [-10m/s,10m/s] pour la vitesse. Les informations fournies au réseau en entrée sont l'altitude et la vitesse du module et c'est l'unique sortie du réseau qui commande la poussée.

La figure suivante décrit le déroulement d'un alunissage réalisé par l'un des meilleurs réseaux obtenu au cours des tests effectués :

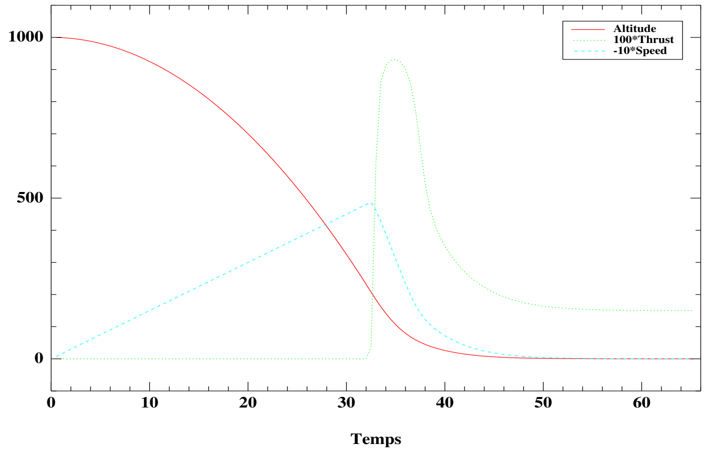


FIG. 2.1 – *Déroulement d'un alunissage controlé par un réseau de neurone obtenu par évolution artificielle*

2.2 Structrure des réseaux utilisés

Les réseaux utilisés sont des perceptrons à une couche cachée. La topologie est fixée tout au long des essais.

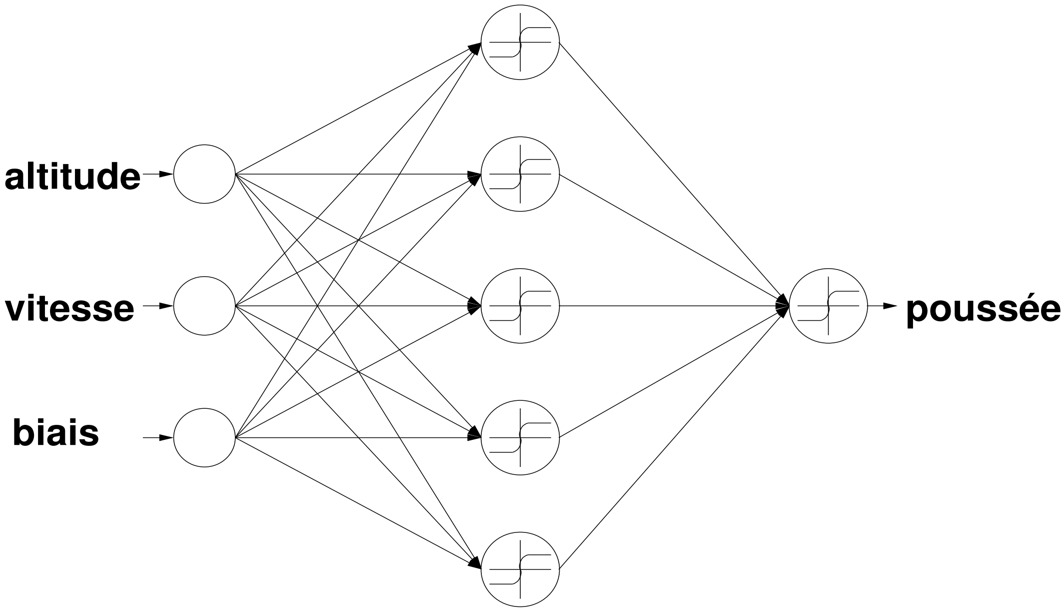


FIG. 2.2 – *Contrôleur neuronal pour le module lunaire*

Le nombre de neurones d'entrée est de trois : un neurone pour l'altitude, le deuxième pour la vitesse et le troisième est le neurone de biais qui a pour valeur de sortie 1. Concernant le nombre d'unités cachées, j'ai choisi d'appliquer le modèle de Kolmogorov qui à un réseau à n entrées et une sortie, attribue $2n + 1$ neurones cachés, c'est à dire 5 dans notre cas. Les fonctions de transfert sont des sigmoides de paramètre $\beta = 1$. L'unique sortie du réseau est un neurone à fonction de transfert également sigmoidale et qui renvoie donc une valeur comprise entre 0 et 1. La commande effectivement envoyée aux réacteurs est alors calculée par :

$$Poussee = Sortie * PousseeMax$$

La topologie des réseaux étant fixée a priori seuls les poids sont ajustés au cours de l'évolution. Le problème ainsi défini équivaut donc à un problème paramétrique à 21 variables.

2.3 La fonction de fitness

Pour que le réseau final soit en mesure d'alunir depuis n'importe quelle condition initiale située dans les intervalles, l'algorithme entraine les réseaux sur toute la région $[500m,1500m] \times [-10m/s,10m/s]$. Ainsi, pour évaluer un individu, on lance N simulations en considérant à chaque fois des conditions initiales choisies uniformément dans l'intervalle $[500m,1500m] \times [-10m/s,10m/s]$. La fitness est alors calculée à partir de la moyenne des vitesses d'alunissage obtenues :

$$Fitness = \frac{1}{1 + \left(\frac{\sum_{i=1..N} VitesseAlunissage(a_i^0, v_i^0)}{N} \right)}$$

De cette façon, on obtient une fonction de fitness pouvant varier entre 0 et 10 et qui doit être maximisée.

Dans tous les essais qui ont été entrepris dans la suite, le nombre de simulations effectuées lors de chaque évaluation de fitness est resté constant N=5.

2.4 Résultats avec un SGA

Reprenant l'approche envisagée dans l'article [Ronald and Schoenauer 1994], j'ai commencé par tester le classique algorithme génétique de John Holland : le Simple Genetic Algorithm (SGA).

Les opérateurs du SGA effectivement utilisés sont les suivants :

- Sélection à la roulette-wheel
- Croisement arithmétique entre les poids vus comme un vecteur de réels
- Mutation gaussienne sur des poids choisis au hasard dans le réseau. La variance de la mutation est initialisée à 10, puis décroit géométriquement d'un facteur 0.999 à chaque génération
- Remplacement générationnel avec elitisme

Les paramètres choisis sont :

- Initialisation des poids des réseaux dans [-10,10]
- Taille de la population = 50
- Taux de croisement = 0.6
- Taux de mutation = 0.2

– Nombre de générations = 2000

La figure suivante présente les résultats obtenus en moyenne sur 30 essais :

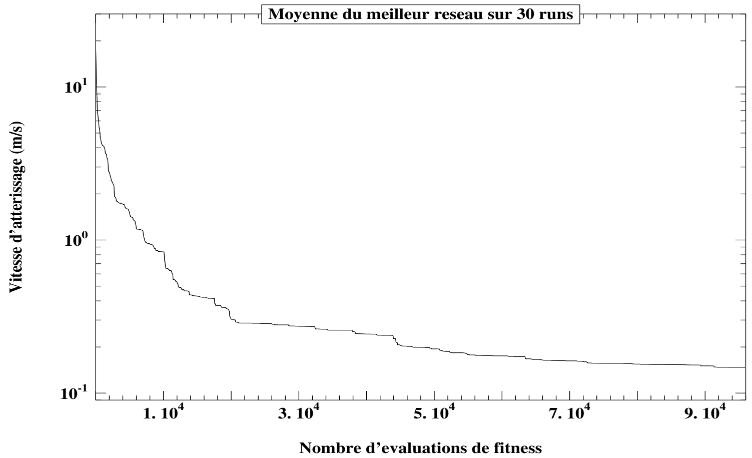


FIG. 2.3 – *Evolution de la vitesse d'alunissage du meilleur réseau avec un algorithme SGA*

L'algortithme SGA permet donc d'obtenir en moyenne au bout de 2000 générations un réseau capable de poser le module lunaire à une vitesse d'environ 15 cm/s ce qui représente un atterrissage largement satisfaisant dans la réalité. Très vite je me suis néanmoins tourné vers les stratégies d'évolution (ES) qui se sont avérées être nettement plus performantes en terme de rapidité.

2.5 Résultats avec la mutation adaptative

L'algorithme testé dans cette section est un ES(20,100), avec remplacement générationnel. La taille de la population est fixée à 20 individus et chaque parent engendre à chaque génération cinq enfants par mutation adaptative. Comme pour l'algorithme SGA les poids sont initialisés dans [-10,10]. Enfin concernant l'opérateur de mutation¸ les variances de départ associées à chaque poids d'un réseau sont choisies uniformément dans l'intervalle [0,10]. La vitesse finale moyenne sur 30 essais est représentée par le graphique suivant :

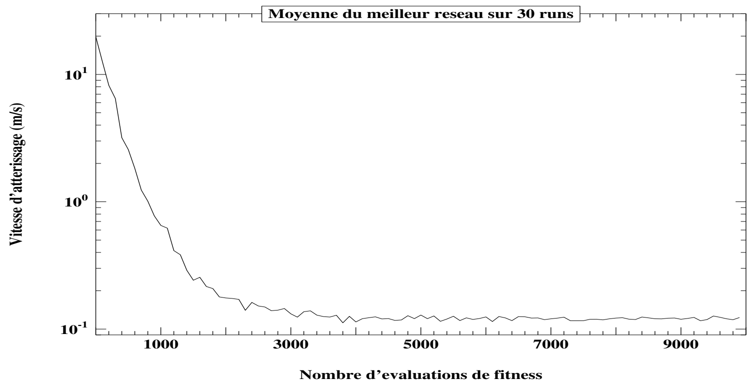


FIG. 2.4 – *Evolution de la vitesse d'alunissage du meilleur réseau avec un algorithme ES*

On constate que l'algorithme ES permet d'obtenir non seulement une vitesse d'alunissage plus faible, mais surtout d'accélérer considérablement la convergence par rapport à SGA. Ainsi pour atteindre un réseau capable d'alunir en moyenne à environ 10 cm/s, l'ES requiert plus de dix fois moins d'évaluations de fitness que le SGA de la section précédente.

2.6 La question de la légitimité du croisement entre des réseaux de neurones

De nombreux chercheurs sont opposés à l'utilisation du croisement dans les algorithmes évolutionnaires chargés de faire évoluer des réseaux de neurones, car ils estiment qu'ils ont un effet destructeur sur les réseaux. On peut aisément observer la mauvaise influence que peut avoir un croisement en considérant le cas du crossover discret entre deux réseaux identiques du point de vue phénotypique mais génotypiquement différents :

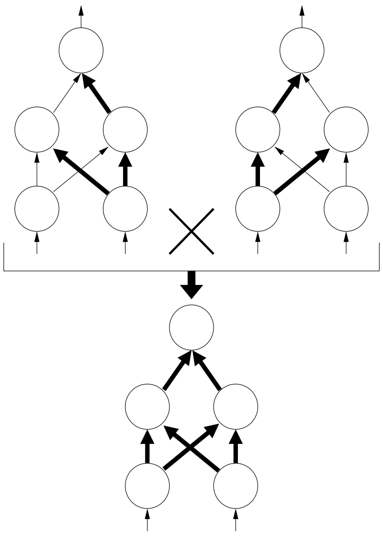


FIG. 2.5 – *Croisement entre deux individus de phénotypes identiques mais de génotypes différents*

Idéalement on souhaite qu'à l'issue d'un croisement entre deux individus identiques sur le plan phénotypique, on récupère un enfant ayant un phénotype équivalent aux parents, ce qui n'est pas du tout le cas dans l'exemple que l'on vient de voir. Le fait qu'il n'y ait pas injection de l'espace génotypique vers l'espace phénotypique implique donc qu'il faut rester prudent quant à l'usage du croisement entre réseaux neuronaux. Néanmoins on peut remarquer que lorsque la population commence à converger, c'est-à-dire lorsque les réseaux commencent à adopter tous la même configuration, le croisement prend plus de sens.

Pour essayer d'observer l'influence du croisement sur le problème du module lunaire j'ai comparé des stratégies d'évolution avec crossover (arithmétique) avec des ES sans crossover :

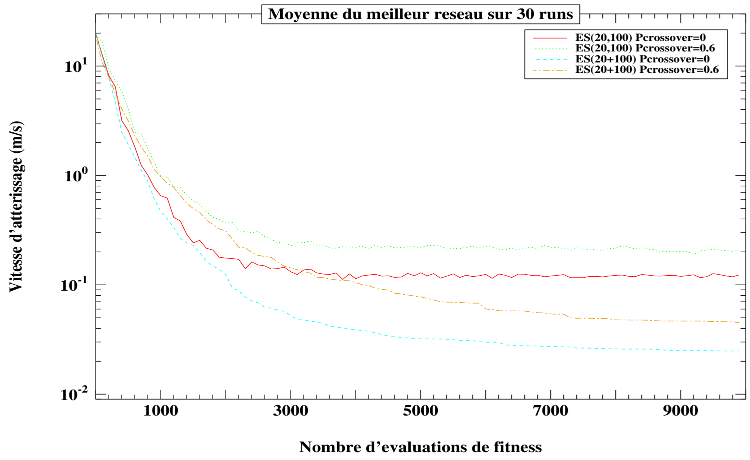


FIG. 2.6 – *Influence du croisement sur l'efficacité de l'algorithme ES*

On constate que pour chacun des types de stratégies d'évolution, le croisement dégrade les résultats, ce qui confirme que le crossover peut avoir un effet néfaste sur les réseaux. Enfin, on note que l'ES elitiste obtient de meilleures vitesses d'alunissage que l'ES non elitiste.

Chapitre 3

Identification neuronale de loi de comportement

3.1 Introduction

L'industrie de la plasturgie connait actuellement un essor considérable avec l'apparition de nouveaux polymères aux propriétés toujours plus pointues. Aussi, pour tirer partie au maximum de ces nouvelles propriétés mécaniques, l'analyse par éléments finis est devenu un outil indispensable pour les ingénieurs designers. En effet, la FEA (Finite Element Analysis) permet d'effectuer des simulations sur ordinateurs qui permettent d'optimiser beaucoup plus efficacement les formes des pièces mécaniques en réalisant une économie considérable d'essais sur prototypes qui sont très coûteux et longs à mettre en oeuvre.

Il est connu que la réponse des composants plastiques en grands déplacements n'est pas proportionnelle aux charges appliquées. Pour simuler de manière réaliste ce type de comportement, de nombreux logiciels sont capables aujourd'hui de réaliser des calculs en analyse non-linéaire. Toute la difficulté réside alors dans le choix du modèle, l'objectif étant de restituer de manière fidèle les capacités du matériau considéré.

L'une des approches les plus utilisées en élasticité non-linéaire est la théorie de l'hyperélasticité. Malheureusement, le choix des modèles hyperélastiques est limité à un petit nombre de lois de comportement prédéfinies où seuls quelques paramètres doivent être ajustés. Afin de permettre une modélisation moins limitative et donc plus précise on souhaite définir de nouvelles méthodes qui permettent d'identifier la loi de comportement d'un matériau quelconque en la recherchant dans l'espace entier des fonctions. Dans cette optique, une première approche par programmation génétique a été entreprise en 1995 par l'équipe du CMAP en collaboration avec le Laboratoire de Mécanique des Solides (LMS) de l'Ecole Polytechnique. Devant les résultats encourageants obtenus, il était intéressant de tester l'autre méthode classique utilisée en recherche fonctionnelle : l'apprentissage par réseaux de neurones. C'est justement l'objet de cette partie, qui est consacré à l'étude d'une approche par réseaux neuronaux entrainés par algorithmes évolutionnaires pour l'identification de loi de comportement.

Nous rappelerons dans un premier temps la théorie de l'élasticité et de l'hyperélasticité. Nous préciserons le problème inverse que constitue l'identification de loi de comportement. Nous montrerons comment exprimer ce problème sous la forme d'un problème de minimisation dans l'espace des fonctions. Nous justifierons le choix des algorithmes évolutionnaires pour entraîner les réseaux. Enfin, nous testerons sur deux exemples l'efficacité du calcul évolutionnaire sur des réseaux à structure fixe pour approcher des lois de comportement.

3.2 Le problème mécanique

3.2.1 Définition mathématique du problème de l'elasticité

Un matériau est dit élastique si ses déformations sous l'action de forces extérieures sont réversibles et si son état d'équilibre ne dépend pas de son histoire. Le problème de l'élasticité tridimentionnelle consiste à trouver l'état d'équilibre d'un corps occupant une configuration de référence $\overline{\Omega}$ en l'absence de forces appliquées. Sous l'action de forces extérieures, la configuration déformée est caracterisée par ses déformations représentées par une fonction $\varphi : \overline{\Omega} \subset IR^3 \rightarrow IR^3$.

L'équilibre de l'objet est alors défini mathématiquement sous la forme d'un problème aux limites par le système d'équation :

$$\left\{ \begin{array}{cl} -divT(x,\nabla\varphi\left(x\right)) = f\left(x\right), & x \in \Omega \\ \varphi\left(x\right) = \varphi_0\left(x\right), & x \in \Gamma_0 \\ T(x,\nabla\varphi\left(x\right)).n = g\left(x\right), & x \in \Gamma_1 \end{array} \right.$$

où T est le premier tenseur de Piola-Kirchoff, f et g sont respectivement les forces volumiques et les forces de pression appliquées et φ_0 sont les déplacements imposés.

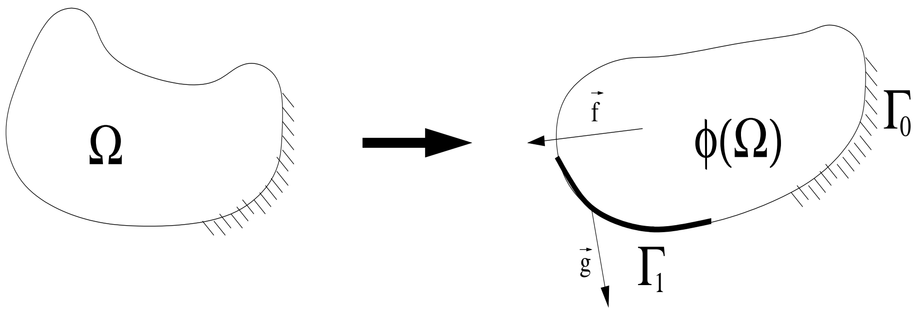


FIG. 3.1 – *Déformations sous l'effet de forces volumiques et surfaciques*

Dans le cas des petits déplacements, le modèle simple de l'élasticité linéaire, dans lequel le tenseur des contraintes s'exprime linéairement en fonction du gradient de déformation, constitue une bonne approximation. En revanche, lorsque le problème étudié traite de chargements plus importants et que par conséquent on se retrouve dans le cas des grands déplacements, l'approximation linéaire n'est plus réaliste. On a alors recours aux théories de l'élasticité non-linéaire. En particulier, il a été défini mathématiquement une catégorie de matériaux qui permet de modéliser ce type de comportements : les matériaux hyperélastiques.

3.2.2 L'hyperélasticité

Un matériau hyperélastique est un matériau dont le tenseur des contraintes dérive d'un potentiel W appelé densité d'énergie de déformation :

$$T\,(x,M) = \frac{\partial W}{\partial M}\,(x,M)\,,\,\forall x \in \overline{\Omega,}\,\forall M \in IM_3^+$$

Connaissant W on peut donc calculer le tenseur des contraintes T et le système de l'élasticité n'a alors plus que pour inconnues les déformations φ. Le comportement du matériau hyperélastique est donc entièrement determiné par sa densité d'énergie de déformation W encore appelée loi de comportement. De plus, si le matériau est isotrope et homogène, sa loi de comportement peut s'exprimer comme une fonction des invariants principaux $(\iota_1,\iota_2,\iota_3)$ de la matrice définie positive $C = \nabla\varphi^T\nabla\varphi$, c'est-à-dire $\iota_1 = trC$, $\iota_2 = tr\,(cofC)$ et $\iota_3 = detC$.

Malheureusement on connait très peu de choses sur les lois de comportement et seuls quelques modèles, basés sur des propriétés mathématiques et physiques, ont été définis. Parmi ces modèles on peut citer :

– la loi de Saint-Venant-Kirchhoff :

$$W = \mu(\iota_3 - 3) + \frac{\lambda - 2\mu}{8}(\iota_1 - 3)^2 - \frac{\mu}{2}(\iota_2 - 3)$$

où μ et λ sont les coefficients de Lamé.

– la loi de Mooney-Rivlin :

$$W = a_1\iota_1 + a_2\iota_2 + a_3\iota_3 - (a_1 + 2a_2 + a_3)\log\iota_3$$

– la loi d'Ogden :

$$\left\{\begin{array}{l} W = a_1\iota_1 + a_2\iota_1^2 + b_1\iota_2 + b_2\iota_2^2 + c_1(\det\nabla\varphi)^2 - c_2\ln(\det\nabla\varphi) \\ 2a_1 + 12a_2 + 4b_1 + 24b_2 + 2c_1 - c_2 = 0, \quad a_1,a_2,b_1,b_2,c_1,c_2 > 0 \end{array}\right.$$

3.2.3 Résolution numérique par éléments finis

Connaissant la loi de comportement du matériau considéré, le système de l'élasticité non linéaire peut être résolu numériquement grâce à la méthode d'approximation par éléments finis.

Pour pouvoir effectuer un calcul sur ordinateur il est nécessaire de réecrire le système variationnel de l'élasticité en le discrétisant dans un espace de dimension finie : l'objet est maillé c'est-à-dire découpé en petits éléments polyhédriques sur lesquels sont définis des noeuds. C'est ensuite sur ces noeuds que sont interpolées les fonctions de base polynômiales du nouvel espace. Le problème ainsi discrétisé peut alors s'écrire sous la forme d'un système linéaire $Ax = b$ qui peut être résolu numériquement.

3.2.4 Le problème inverse

Nous avons vu qu'il était possible de résoudre numériquement le problème de l'élasticité non linéaire à condition que l'on connaisse a priori la densité d'énergie de déformation du matériau :

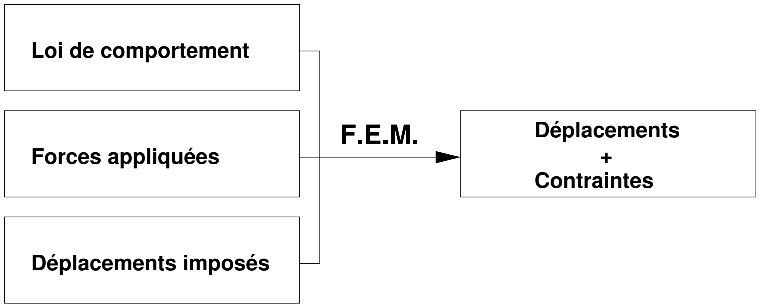


FIG. 3.2 – *Problème direct de l'élasticité*

Pour pouvoir tirer le maximum d'informations de la simulation numérique, le soucis de l'ingénieur est de trouver la loi qui va permettre de modéliser avec la meilleure approximation le comportement du matériau. Ce problème d'identification est en fait un problème inverse qui consiste à remonter à la loi de comportement du matériau étudié sachant que l'on sait résoudre le problème direct :

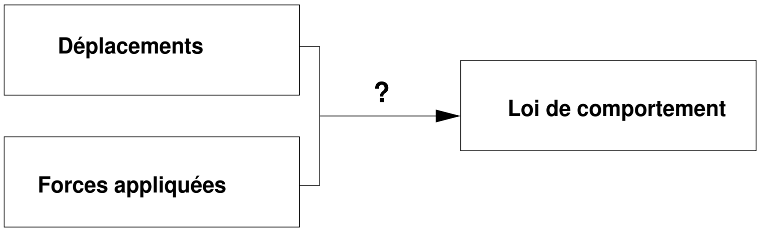


FIG. 3.3 – *Problème inverse de l'identification de loi de comportement*

Habituellement, pour identifier la loi d'un matériau, le mécanicien procède de façon assez rudimentaire : il commence par effectuer un certain nombre d'essais réels. Il sélectionne ensuite un modèle parmi le choix restreint des modèles standards qui existent en hyperélasticté. Enfin il ajuste les paramètres "à la main" jusqu'à obtenir une loi qui se rapproche suffisamment des déplacements mesurés au cours des essais.

Cette méthode est peu satisfaisante car elle oblige à retenir un modèle parmi ceux qui existent déjà alors que la véritable loi du matériau n'a aucune raison d'être de la même forme que l'un de ces modèles. C'est pour essayer de ne pas restreindre autant l'espace de recherche de la loi qu'on se propose d'étudier une approche par réseaux neuronaux.

3.3 L'algorithme évolutionnaire

3.3.1 Redéfinition du problème de l'identification

Considérons une éprouvette à laquelle on a fait subir un certains nombre d'essais.

Le problème de l'identification de loi de comportement peut s'écrire sous la forme d'un problème de minimisation dans l'espace fonctionnel. En effet, rechercher la loi de comportement du matériau constituant l'éprouvette est équivalent à chercher la fonction W^* qui minimise l'erreur commise par la simulation par rapport aux déplacements effectivement mesurés, ce qui s'écrit mathématiquement :

$$trouver\ W^* = \min_{W} \sum_{cas\ de\ charge} \|Deplacements_W - Deplacements_{mesures}\|_{L^2}$$

3.3.2 Pourquoi une approche évolutionnaire ?

La première solution que l'on envisage pour résoudre ce type de problème d'optimisation est une méthode de descente de gradient. Celle relative aux perceptrons est la Back-Propagation. Malheureusement cette dernière nécessite de savoir calculer la dérivée de l'erreur par rapport aux poids du réseau, ce qui est en fait un problème théorique extrêment compliqué qui reste ouvert encore aujourd'hui.

Tout comme pour la plupart des problèmes inverses délicats à aborder par une méthode déterministe, les algorithmes évolutionnaires constituent dès lors la seule alternative. De plus, les algorithmes évolutionnaires ont démontré au cours de ces dernières années qu'ils étaient capables d'obtenir d'excellents résultats sur de nombreux problèmes inverses de ce type. C'est la raison pour laquelle il a été choisi d'essayer une telle méthode stochastique sur le problème de l'identification de loi de comportement.

3.3.3 La Fitness

Les individus de la population sont des réseaux de neurones qui ont pour objectif d'approcher la loi de comportement du matériau constituant l'éprouvette. Pour évaluer la valeur de fitness, c'est-à-dire la qualité d'approximation d'un réseau, on compare simplement les déplacements obtenus par simulation avec le réseau avec les déplacements réels de l'éprouvette.

L'éprouvette considérée est un parrallélépipède rectangle maillé en 56 éléments Q_2. Elle possède 697 noeuds disposant chacun de 3 degrés de liberté. Le système linéaire associé que doit résoudre le solveur est donc un système à 697×3=1911 inconnues.

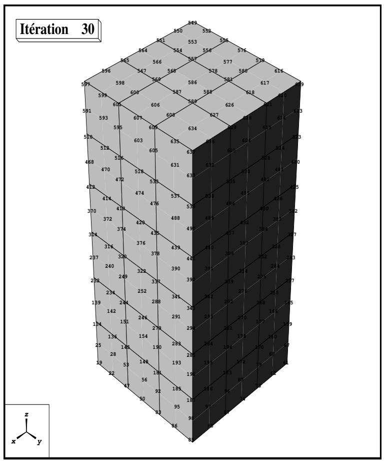


FIG. 3.4 – *L'éprouvette dans sa configuration de réference*

Les conditions aux limites sont l'encastrement au niveau de sa base. Dans un souci de généralisation, trois cas de charge de natures différentes lui sont appliqués au niveau de face supérieure: une traction, une compression et un basculement. Ces trois cas de charge ont été choisis de manière à ce qu'ils soient représentatifs des différents types de contraintes auquels les corps sont susceptibles d'être soumis dans la réalité.

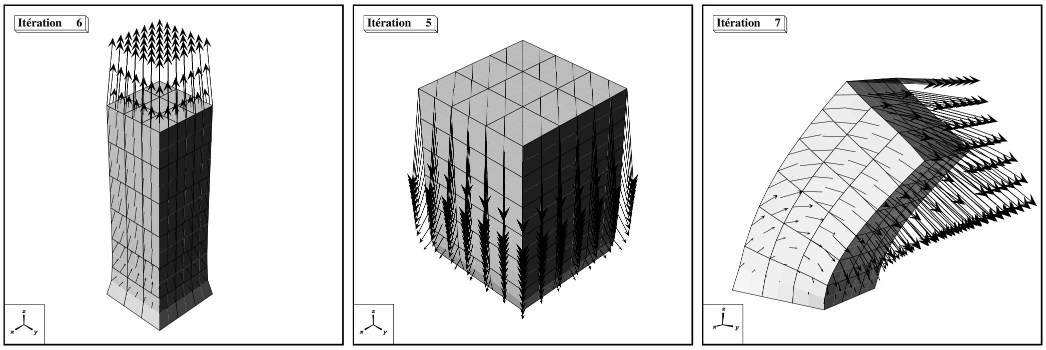


FIG. 3.5 – *Les trois cas de charge appliqués à l'éprouvette : traction, compression, basculement*

L'évaluation de la fitness consiste alors simplement à calculer l'erreur L^2 commise sur les déplacements :

$$Fitness = \sum_{cas\ de\ charges} \sum_{i \in noeuds} \left\| \overrightarrow{Depl^i_{NN}} - \overrightarrow{Depl^i_{reels}} \right\|^2$$

3.3.4 Le solveur éléments finis

Pour résoudre numériquement le problème de l'élasticité linéaire, le solveur éléments finis utilisé est un solveur mis au point par Francois Jouve, chercheur au CMAP. La résolution du système variationnel y est basé sur une méthode de Newton modifiée. Il s'agit d'une méthode itérative efficace présentant en outre l'avantage de pouvoir démarrer les itérations à partir de n'importe quel champ de déplacements initial. Dans le cas de notre problème direct qui implique un calcul en grands déplacements, on profite de connaître les déplacements solutions pour les prendre comme point de départ des itérations de Newton. De cette facon, à mesure que la population se rapproche de la loi cible, le nombre d'itérations nécessaire pour faire converger l'algorithme de Newton diminue, ce qui réduit le coût calcul des évaluations.

Il faut par ailleurs remarquer que l'algorithme de Newton est susceptible de ne pas converger si la loi considérée ne représente pas une loi de comportement. On a donc fixé le nombre d'itérations maximum à 30. De cette façon, si l'algorithme ne converge pas au bout de 30 itérations, on considère que le réseau n'est pas valide et on lui attribue une fitness infinie.

3.4 Résultats

3.4.1 Le modèle du perceptron

Dans un premier temps, le type de réseau que j'ai considéré est le classique perceptron à une couche cachée :

Les neurones d'entrée recoivent les invariants principaux et sont donc au nombre de trois avec un neurone supplémentaire de biais. Le seul neurone de sortie représente l'image des invariants par la loi de comportement. Compte-tenu des propriétés d'approximation des dérivées, j'ai muni les neurones cachés de fonctions de transfert sigmoidales avec un $\beta = 1$. Enfin j'ai repris le modèle de Kolmogorov en implantant 7 neurones cachés.

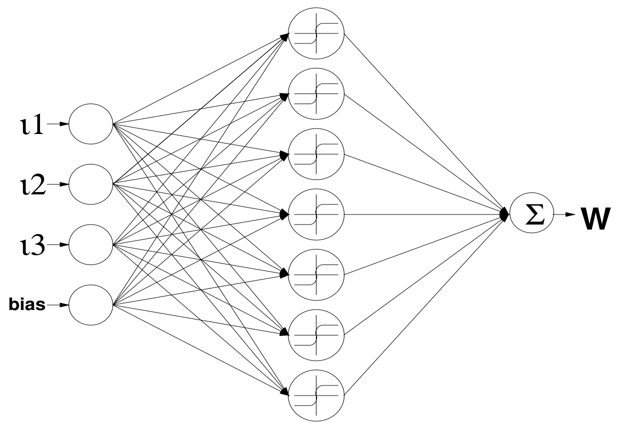


FIG. 3.6 – *Modèle du perceptron pour simuler une loi de comportement*

Malheureusement ce modèle s'avère être totalement inadapté puisque la quasi-totalité des perceptrons générés aléatoirement font en fait à chaque fois diverger l'algorithme de Newton à l'infini et ce même pour des chargements d'intensité très faible. En fait, on verra dans la section suivante que le modèle du perceptron ne permet pas de vérifier certaines propriétés cruciales des lois de comportement qui conditionnent très fortement la convergence de l'algorithme de Newton.

3.4.2 Prise en compte de certaines propriétés des lois de comportement

Si on s'intéresse de plus près à la théorie de l'hyperélasticité, on s'aperçoit que les lois de comportement doivent vérifier un certain nombre de propriétés :

– on a d'abord :

$$\lim_{\iota_k \to \infty} W(\iota_1, \iota_2, \iota_3) = +\infty$$

et

$$\lim_{\iota_3 \to 0^+} W(\iota_1, \iota_2, \iota_3) = +\infty$$

Comme j'ai pu le remarquer au cours de mes essais, ces deux propriétés jouent un rôle fondamental car elles décrivent les réactions des matériaux réalistes face à des déformations extrêmes. Elles peuvent en effet s'interpréter physiquement de la façon suivante : "des déformations très importantes en extension ou en compression produit une énergie de contrainte très forte". Si ces propriétés ne sont pas vérifiées, cela peut conduire à une absurdité physique qui se traduit par une divergence des déplacements à l'infini au cours des itérations de l'algorithme de Newton. C'est ce phénomène qui est sans doute à l'origine de l'échec du modèle du perceptron qui en tant que somme pondérée de sigmoides est borné et ne peut donc pas tendre vers l'infini.

- ensuite les lois de comportement satisfont une condition de non chargement quand la structure est dans son état initial ($\varphi(x) = x$, $\nabla\varphi = Id$ *et* $(\iota_1,\iota_2,\iota_3) = (3,3,1)$) :

$$\frac{\partial W}{\partial \iota_1} + 2\frac{\partial W}{\partial \iota_2} + \frac{\partial W}{\partial \iota_3} = 0$$

En se basant sur ces propriétés il a été défini un nouveau modèle de réseau :

- pour permettre au réseau de tendre vers l'infini quand l'un des invariants tend vers l'infini, un terme linéaire est ajouté en connectant les entrées $(\iota_1,\iota_2,\iota_3)$ à la sortie.
- pour assurer la deuxième et la troisième propriété, la solution consiste à introduire le modèle d'Ogden en rajoutant un terme en logarithme. Ainsi si la sortie du réseau est noté $\widetilde{W}$, la loi que l'on va effectivement transmettre au solveur est en fait :

$$W = \widetilde{W} - \alpha \log \iota_3$$

où le logarithme force W à tendre vers l'infini losrque ι_3 tend vers 0 et où le coefficient α est calculé de façon à vérifier la condition de non chargement, c'est-à-dire :

$$\alpha = \frac{\partial W}{\partial \iota_1}(3,3,1) + 2\frac{\partial W}{\partial \iota_2}(3,3,1) + \frac{\partial W}{\partial \iota_3}(3,3,1)$$

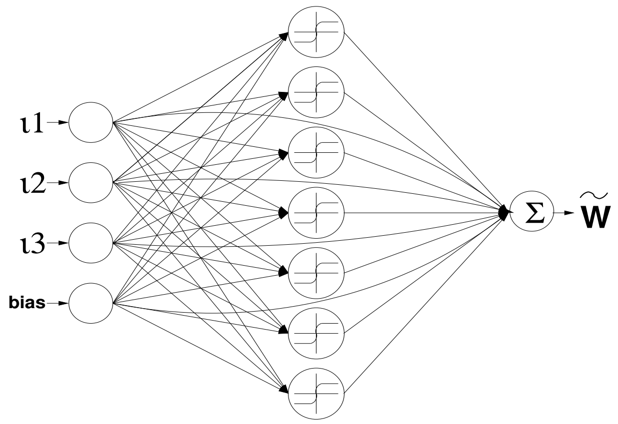


FIG. 3.7 – *Nouveau modèle de réseau pour simuler une loi de comportement*

Le nouveau modèle qui incite les réseaux à vérifier les propriétés des lois de comportement a permis de récupérer suffisamment de réseaux valides pour permettre à l'algorithme évolutionnaire de fonctionner. Dans la suite, c'est ce type de réseau qui a été utilisé.

3.4.3 Identification de la loi de Mooney-Rivlin

La première loi que j'ai tenté d'identifier fut la loi de Mooney-Rivlin qui possède un premier terme $\widetilde{W}$ linéaire en ι_1,ι_2,ι_3 :

$$W_a = \iota_1 + \iota_2 + \iota_3 - 4 \log \iota_3$$

L'algorithme évolutionnaire utilisé est un ES(20,80) sans crossover. Les intervalles d'initialisation des poids et des variances de la mutation adaptative sont respectivement [-10,10] et [0,5]. L'initialisation de la population est réalisée en générant des réseaux aléatoirement jusqu'à en obtenir 20 qui font converger l'algorithme de Newton sur les trois cas de charge. L'intensité des charges est restée constante tout au long des essais (de l'ordre de 0.1 N). Etant donné le coût très important des calculs en grands déplacements (100 générations sur un ES(20,80), c'est-à-dire 8000 évaluations de fitness demandent trois jours de calcul sur un Pentium 800 MHz), je n'ai réalisé qu'un seul essai. La courbe suivante présente le pourcentage d'erreur réalisée sur les déplacements par le meilleur réseau de la population :

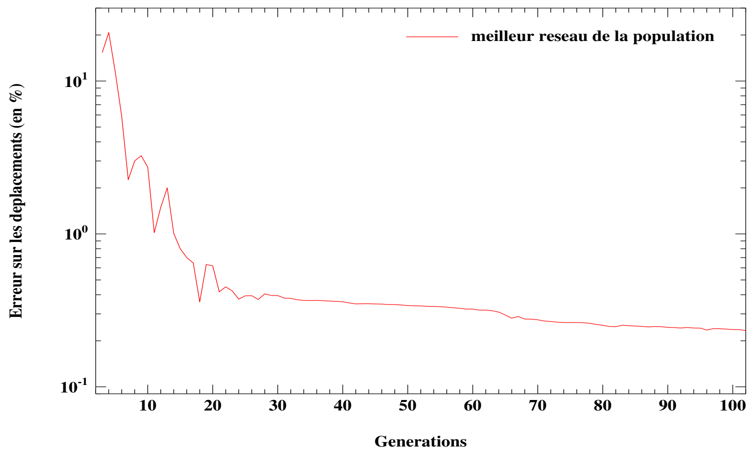


FIG. 3.8 – *Evolution de l'erreur sur le problème de l'identifiation de la loi de Mooney-Rivlin*

On constate que l'on obtient au bout de 100 générations un réseau qui approche la loi de Mooney-Rivlin sur les trois cas de charge avec une erreur de l'ordre de seulement 0.2%.

On peut remarquer néanmoins que le modèle de réseau utilisé possède un terme linéaire en ι_1,ι_2 *et* ι_3 qui peut avantager l'identification de la loi de Mooney-Rivlin. Pour tester ce type de réseaux sur un exemple plus général et moins adapté, j'ai essayer d'identifier une loi avec un premier terme $\widetilde{W}$ non-linéaire.

3.4.4 Identification d'une deuxième loi de comportement

La nouvelle loi à identifier est la loi :

$$W_b = \iota_1^2 + \iota_2^2 + \iota_3^2 - 8 \log \iota_3$$

L'ES utilisé est identique au précédent. L'idée est ici de voir si la couche cachée contribue à compenser la non-linéarité de $\widetilde{W_b}$. Pour ce faire, j'ai comparé le réseau utilisé précédemment à 7 neurones cachés avec un réseau sans couche caché, ce qui réduit l'espace de recherche de $\widetilde{W}$ à l'espace des fonctions linéaires. Toujours pour les mêmes raisons de temps calcul, je n'ai réalisé qu'un essai sur chacun des modèles. Le graphique suivant présente les résultats obtenus :

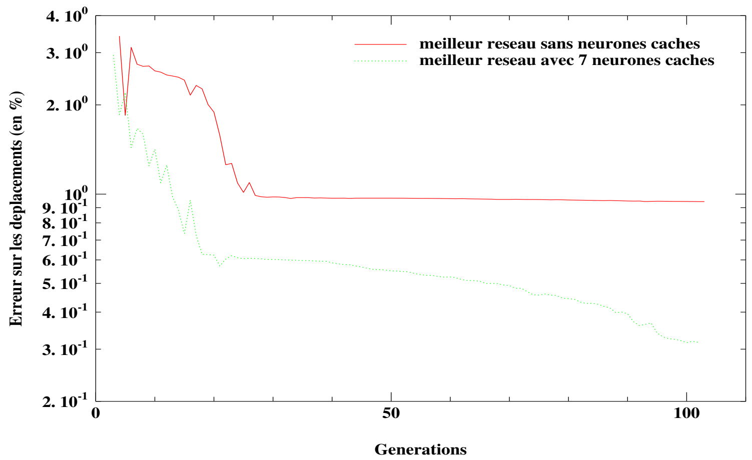


FIG. 3.9 – *Evolution de l'erreur sur le problème de l'identifiation de la loi de comportement quadratique*

On observe que l'dentification dans l'espace linéaire permet d'obtenir des résultats a priori tout à fait satisfaisant avec seulement 1% d'erreur. Il s'agit d'un résultat plutôt décevant car il aurait été plus logique qu'une simple fonction linéaire ne puisse pas se rapprocher autant d'une loi quadratique. L'explication provient probablement du fait que l'intensité des forces appliquées est trop faible et ne permet donc pas de bien distinguer les différentes formes de loi de comportement.

Le modèle de réseau muni de neurones cachés obtient une courbe de descente meilleure et apparemment moins bloquée, ce qui laisse espérer que les neurones cachés contribuent à adapter le réseau à la non-linéarité de la loi. Cela reste toutefois une interprétation tout à fait hypothétique et de nombreux essais supplémentaires doivent être entrepris pour éventuellement la confirmer.

3.4.5 Capacités de généralisation

L'autre aspect important à vérifier est la capacité des réseaux, qui ont appris sur un nombre fini de cas de charge, à être robuste, c'est-à-dire faire preuve de bonnes capacités d'approximation sur des cas de charge différents. Pour évaluer cette faculté de généralisation, j'ai appliqué aux meilleurs réseaux de chaque génération une torsion, autrement dit un cas de charge de nature différente des trois cas de charges d'entraînement.

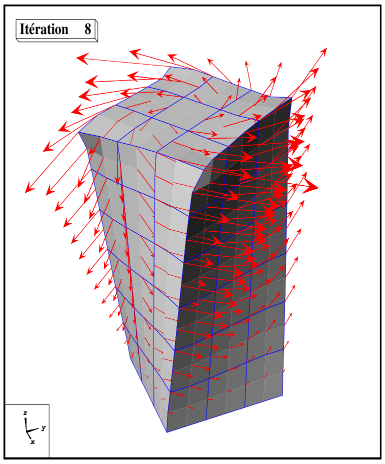


FIG. 3.10 – *Déformations de l'eprouvette sous l'effet d'une torsion*

Les courbes suivantes présentent les résultats obtenus :

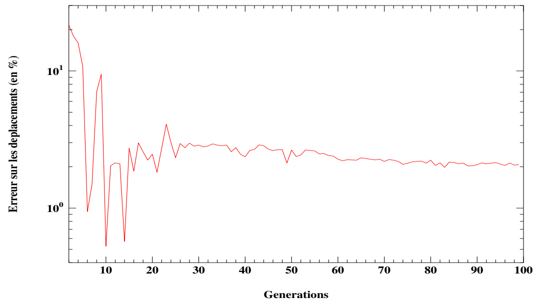


FIG. 3.11 – *Erreur sur les déplacements par rapport à la loi de Mooney-Rivlin*

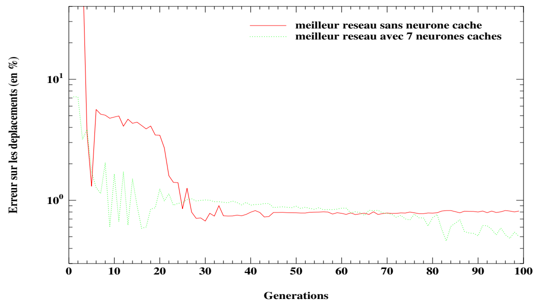


FIG. 3.12 – *Erreur sur les déplacements par rapport à la loi quadratique*

On constate que les meilleurs réseaux obtenus après apprentissage sont capables de bien modéliser la torsion avec des erreurs du même ordre que celles concédées sur les cas de charge d'apprentissage.

Conclusion

Au cours de la première partie de mon stage, l'étude du problème de contrôle sur module lunaire m'a permis de me familiariser avec les réseaux de neurones et les algorithmes évolutionnaires. Elle a par ailleurs été l'occasion de constater l'efficacité de la mutation adaptative sur la résolution d'un problème d'optimisation sur chaînes de réels et d'observer que le croisement entre réseaux neuronaux peut se réveler destructif et donc nuire à l'efficacité de l'algorithme.

Concernant l'identification de loi de comportement, la résolution du problème direct nécessite un temps calcul conséquent qui ne m'a pas permis de faire autant d'essais que je l'aurais souhaité. Néanmoins à partir des quelques simulations effectuées, on peut tirer les conclusions suivantes :

- les réseaux neuronaux ne sont apparemment pas le modèle le mieux adapté à ce problème. En effet, le perceptron, qui simule une fonction nécessairement bornée, est incapable de vérifier les propriétés de tendance vers l'infini des lois de comportement. Ce défaut nous a contraint à modifier fortement le modèle en rajoutant un terme linéaire pour obtenir suffisamment de réseaux valides, qui sont indispensables au fonctionnement d'un algorithme évolutionnaire.
- les résultats obtenus sur les deux identifications montrent que l'on peut trouver facilement des réseaux qui simulent avec une bonne approximation non seulement les déplacements exacts sur les cas de charge d'entraînement mais également les déplacements avec un chargement de nature différente, ce qui traduit de bonnes capacités en généralisation.

Les perspectives que je suggère sont les suivantes :

- le fait qu'une fonction linéaire parvient à approcher à 1% près les déplacements obtenus avec une loi quadratique indique que l'intensité des forces doit être augmentée pour mieux isoler les différents types de loi.
- des essais supplémentaires doivent être entrepris pour éventuellement montrer que les neurones cachés permettent effectivement au réseau de s'adapter aux non-linéarités.
- d'autres moyens de vérifier les propriétés en l'infini moins radicaux que de connecter les entrées à la sortie doivent être étudiés.

Quoiqu'il en soit l'approche par programmation génétique semble offrir une base de recherche mieux adaptée avec l'espace des polynômes. Celui-ci permet en effet de vérifier plus naturellement les propriétés asymptotiques des lois de comportement. De plus, le résultat d'une identification par programmation génétique s'écrit sous la forme d'un polynôme que le mécanicien peut interpréter plus facilement qu'un réseau de neurones qui n'a pas d'expression simple.

Bibliographie

[1] Annema, A. J., Feed-forward neural networks, Kluwer Academic Publishers, 1995

[2] Back, T., Fogel, D. B., Michalewicz, Z., Handbook of Evolutionary Computation, IOP Publishing Ltd and Oxford University Press, 1997

[3] Ciarlet, P. G., Elasticité tridimensionnelle, RMA-Masson, 1986

[4] Hecht-Nielsen, R., Neurocomputing, Addison-Wesley, 1990

[5] Hornik, K., Stinchcombe, M., et White, H., Universal Approximation of an Unknown Mappings and Its Derivatives Using Multilayer Feedforward Networks, Neural Networks, Vol. 3, pp 551-560, 1990

[6] Jouve, F., Modélisation de l'oeil en élasticité non linéaire, RMA-Masson, 1993

[7] Michalewicz, Z., Genetic Algorithm + Data Structures = Evolution Programs, Springer Verlag, 1992

[8] Ogden, R. W., Non-linear elastic deformations, Ellis Horwood Ltd

[9] Ronald, E. et Schoenauer, M., Genetic Lander: An Experiment in Accurate Neuro-Genetic Control, PPSN94

[10] Shoenauer, M., Cours d'optimisation évolutionnaire pour le DEA d'Analyse Numérique de Paris VI, Version 2000

[11] Shoenauer, M., Sebag, M., Jouve, F., Lamy, B., Maitournam, H., Advances in Genetic Programming II, pp 467-488, P. J. Angeline and K. E. Kinnear Jr editors, MIT Press, 1996

[12] Schoenauer, M. et Ronald, E., Neuro-Genetic Truck Backer-Upper Controller, ICEC94

[13] Vemuri, V., Artificial Neural Networks : Theoretical Concepts, The Computer Society of the IEEE, 1988

[14] Yao, X., A review of evolutionary artificial neural networks, Int. Jour. of Intelligent Systems, 8, pp 367-384