THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS-SUD

SPÉCIALITÉ : INFORMATIQUE

présentée par

Fei JIANG

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS-SUD

Sujet de la thèse :

Optimisation de la Topologie de Grands Réseaux de Neurones

Soutenue le 16/12/2009, devant le jury composé de :

Mr	Hugues Berry	Directeur de these (Chargé de Recherche, INRIA Rhone-Alpes)
Mr	Guillaume Beslon	Rapporteur (Professeur, INSA-Lyon, Université de Lyon)
Mr	Abdel Lisser	Examinateur (Professeur, Université Paris Sud)
Mme	Hélène Paugam-Moisy	Examinatrice (Professeur, Université de Lyon)
Mr	Marc Schoenauer	Directeur de these (Directeur de Recherche, INRIA Saclay)
Mr	Darrell Whitley	Rapporteur (Professeur, Colorado State University)

Abstract

In this dissertation, we present our study regarding the influence of the topology on the learning performances of neural networks with complex topologies. Three different neural networks have been investigated : the classical Self-Organizing Maps (SOM) with complex graph topology, the Echo State Network (ESN) and the Standard Model Features(SMF). In each case, we begin by comparing the performances of different topologies for the same task. We then try to optimize the topology of some neural network in order to improve such performance.

The first part deals with Self-Organizing Maps, and the task is the standard handwritten digits recognition of the MNIST database. We show that topology has a small impact on performance and robustness to neuron failures, at least at long learning times. Performance may however be increased by almost 10% by artificial evolution of the network topology. In our experimental conditions, the evolved networks are more random than their parents, but display a more broad degree distribution.

In the second part, we proposes to apply CMA-ES, the state-of-the-art method in evolutionary continuous parameter optimization, to the evolutionary learning of the parameters of an Echo State Network (the Readout weights, of course, but also, Spectral Radius, Slopes of the neurons active function). First, a standard supervised learning problem is used to validate the approach and compare it to the original one. But the flexibility of evolutionary optimization allows us to optimize not only the outgoing weights but also, or alternatively, other ESN parameters, sometimes leading to improved results. The classical double pole balancing control problem is used to demonstrate the feasibility of evolutionary reinforcement learning of ESN. We show that the evolutionary ESN obtain results that are comparable with those of the best topology-learning neuro-evolution methods.

Finally, the last part presents our initial research of the SMF - a visual object recognition model which is inspired by the visual cortex. Two version based on SMF are applied to the PASCAL Visual multi-Object recognition Challenge (VOC2008). The long term goal is to find the optimal topology of the SMF model, but the computation cost is however too expensive to optimize the complete topology directly. So as a first step, we apply an Evolutionary Algorithm to auto-select the features used by the systems. We show that, for the VOC2008 challenge, with only 20% selected feature, the system can perform as well as with all 1000 randomly selected feature.

Acknowledgements

I would like to sincerely thank :

My supervisors : Hugues Berry and Marc Schoenauer for their integrity, kindness, sagacity and unremitting efforts in scientific research.

My reviewers and jury : Professor Guillaume Beslon, Professor Abdel Lisser, Professor Hélène Paugam-Moisy and Professor Darrell Whitley for taking time to review this report and for giving very detailed comments.

My teams leaders, colleges and friends : Oliver Teman, Michele Sebag, Oliver Teytaud, Cédric Hartland, Sylvain Gelly, Xiangliang Zhang, Jun Liu etc. for their helping and beneficial discussion.

And my parents for their support.

Résumé

Nous sommes maintenant au 21ème siècle, et depuis les années 1950, l'Intelligence Artificielle est devenue un sujet indépendant d'études au sein des sciences informatiques, et son influence sur notre vie quotidienne n'a cessé d'augmenter.

L'intelligence est le résultat de l'évolution via la sélection naturelle. Dans les années récentes, l'étude de ce qu'on appelle les mécanismes *bio-inspirés*, qui tentent d'imiter les processus naturels, a soulevé beaucoup d'intérêt en recherche. En particulier, deux de ces volets de recherche ont attiré beaucoup d'attention et donné naissance à de nombreux travaux.

D'un point de vue macroscopique, la nature a produit un ensemble riche et diversifié d'espèces, et toutes les espèces ayant survécu sont apparues à la suite de modifications aléatoires lors de la reproduction, et ont été sélectionnées selon le principe naturel de la "*survie du plus apte*". Les Algorithmes Evolutionnaires Artificiels (AEs) sont des algorithmes puissants d'optimisation inspirés par ces mécanismes de variations aveugles et sélection naturelle, et ont été appliqués avec succès dans de nombreux problèmes du monde réel [231].

Du point de vue microscopique, la base matérielle de l'intelligence est basée sur des ensembles de neurones, organisés en réseaux à grande échelle. Les réseaux neuronaux artificiels (RNAs) sont des modèles puissants inspirés par leurs homologues biologiques pour le traitement des connaissances et l'analyse de données. Originaire du début des années 50 [182], la recherche dans le domaine des RNAs est encore très active [105, 143, 93, 198].

Au carrefour de ces deux domaines, la recherche utilisant des algorithmes évolutionnaires pour optimiser les réseaux de neurones artificiels est en cours depuis de nombreuses années [230] : au-delà la "simple" optimisation des poids d'un réseau avec une topologie fixe, la flexibilité des algorithmes évolutionnaires les a rendus attrayants quand il s'agissait également d'optimiser la topologie des réseaux neuronaux pour une tâche donnée.

Pendant la même période, l'étude des réseaux complexes s'est fortement développée et a vu émerger de nouvelles sources d'instpiration et orientations de recherche. Inspiré à la fois par les réseaux naturels (e.g., les réseaux de régulation génétiques, les réseaux d'interaction protéine-protéine, ...) et les réseaux artificiels (e.g., le World Wide Web, les réseaux de connexions des

compagnies aériennes, les réseaux sociaux tels que les réseaux des co-auteurs, ...), de nouvelles connaissances ont été acquises dans les topologies de réseaux. Plus particulièrement, les recherches sur les réseaux “petit-monde” [220] et les réseaux à invariance d’échelle [19] ont apporté un nouveau point de vue pour notre compréhension de la complexité des réseaux.

Le Chapitre **Contexte** de cette thèse passera en revue ces trois domaines de recherche plus en détail, mettant en évidence les différents paradigmes qui ont été utilisés dans le cadre de cette thèse. Dans ce contexte, le présent travail est centré autour de la relation (éventuellement complexe) entre la topologie d’un réseau de neurones donné et ses performances en tant qu’unité de calcul. Deux approches seront envisagées et expérimentées dans différents domaines d’application.

Le problème direct consiste en l’étude de la performance d’un type de réseau de neurones, dans un environnement donné, en fonction de sa topologie. Considérant une classe donnée de topologies paramétrées, nous mesurerons soigneusement les performances des réseaux en fonction des paramètres controllant la topologie et en dégagerons quelques tendances concernant l’influence de tel ou tel paramètre de description de la topologie sur le comportement du réseau.

Nous examinerons ensuite le **problème inverse**, qui consiste à optimiser la topologie afin de maximiser la performance du réseau de neurones. L’outil d’optimisation sera les Algorithmes Evolutionnaires, et l’action sur les topologies se fera soit au travers de certains paramètres macroscopiques, soit directement en laissant l’AE gérer la topologie lui-même (e.g., en agissant au niveau des connexions).

Le premier domaine d’application est celui des Cartes de Kohonen [124] (ou *Self-Organizing Maps* (SOMs)), détaillé dans le **Chapitre SOMs**. Les SOMs sont principalement utilisées pour l’apprentissage non-supervisé, pour lequel la topologie ordinaire standard (grille régulière) est la plus utilisée. Cependant, il n’existe pas de mesure de performance universellement reconnue permettant de comparer la performance des algorithmes d’apprentissage non-supervisé. Nous avons donc opté pour une mesure de la performance d’une topologie SOM donnée au travers de l’exactitude du classement pour un problème d’apprendissage supervisé, la reconnaissance de chiffres manuscrits de la célèbre base MNIST. Le problème inverse sera abordé ici en manipulant directement les connexions entre les neurones.

Le deuxième domaine d’application, présenté dans le Chapitre **ESNs**, est celui des *Echo State Networks* [105] qui rentre dans le paradigme récent du “Reservoir Computing”. Les ESNs sont généralement utilisés pour des tâches de régression, et le problème se ramène alors à un problème d'optimisation quadratique des poids sortants. Par contre, pour autant que nous le sachions, les ESNs n'ont pratiquement pas encore été utilisés pour des tâches d'apprentissage par renforcement. Nous allons utiliser dans ce cadre un algorithme évolutionnaire, le célèbre CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [88, 86], afin de pallier l'absence de gradient pour l'optimisation des poids de sortie du réseau. Un effet secondaire de ce choix est que le même algorithme peut être utilisé pour optimiser simultanément le poids de sortie et quelques hyper-paramètres définissant la topologie du réservoir. Nous allons ainsi étudier l'influence des différents hyper-paramètres définissant la topologie du réservoir sur une des tâches de référence en apprentissage par renforcement, l'équilibre du double pôle.

Enfin, le Chapitre **SMFs** présentera une première étude impliquant un modèle de reconnaissance d'objet inspiré par le cortex visuel [198]. En particulier, nous analyserons si, en utilisant seulement un sous-échantillon des nombreuses fonctionnalités conçues par l'algorithme original, nous pouvons améliorer le taux de reconnaissance global, tout en accélérant la phase d'apprentissage. Les résultats seront démontrés sur les données du Challenge VOC2008.

Comme d'habitude, le Chapitre **Conclusion** conclura cette thèse et donnera quelques pistes pour de nouvelles recherches. Nous allons maintenant donner un peu plus de détails sur les trois types de réseaux de neurones que nous avons étudiés.

Self-Organizing Maps (SOM)

Dans cette partie, nous utilisons les réseaux de Kohonen (ou SOM, Self Organizing Maps) pour la reconnaissance des chiffres manuscrits. Les SOMs sont des réseaux de neurones dont les relations de voisinage entre neurones sont définis par un réseau complexe, La théorie de la Self-Organizing Map (SOM) a été introduite par Kohonen [122, 123]. Il décrit une projection d'un espace d'entrées de grande dimension sur un espace de sortie de dimension bien inférieure. Cela rend possible l'utilisation de SOMs pour la visualisation des données de grandes dimensions [162]. Le but utilisé dans ce travail est la reconnaissance / classification de chiffres manuscrits, en utilisant la base de données bien connue MNIST. Le MNIST [130] a un ensemble d'apprentissage de 60 000 exemples, et un ensemble de test de 10 000 exemples. Les chiffres ont été normalisés en fonction de leur taille, centrés dans une taille fixe (28×28), et représentés par 28×28 pixels sur 256 niveaux de gris. Les SOMs seront donc utilisés ici en apprentissage supervisé. Les neurones sont répartis dans un espace

2D, et à chaque neurone est associé un vecteur de poids de taille 28×28 $(\mathbf{w_i})$ qui sont initialisés aléatoirement et seront ajustés pendant la phase d'apprentissage.

Expérience : Dans l'algorithme classique de SOM, les neurones sont disposés sur une grille régulière en 2 dimensions. Mais deux distances peuvent être définies entre les neurones : la distance euclidienne et la distance liée au graphe de voisinnages (le nombre minimal de neurones qu'il faut visiter pour joindre les 2 neurones donnés. Ces 2 distances sont équivalentes sur une grille régulière. Toutefois, lorsque la topologie se détourne de la grille régulière (par exemple, des liens sont ajoutés ou supprimés), la situation change radicalement. Comme le but ici est d'évaluer l'influence de la topologie du réseau sur ses performances d'apprentissage, la distance entre deux neurones sera leur distance 'graphique', définie par le graphe des connexions). L'apprentissage est un processus itératif. Quand un exemple $\mathbf{I}(t)$ est présentée au réseau, pour chaque neurone i, sa distance d_i à $\mathbf{I}(t)$ est calculé par : $d_i = \sum_{j=1}^{M} (I_j - W_{ij})^2$. La Meilleure Unité Adéquate (MUA) est le neurone dont le vecteur des poids est le plus proche (en norme L^2) de $\mathbf{I}(t)$. Les poids de la MUA k sont mis à jour et rapprochés de ceux de l'exemple courant par : $\mathbf{w_k}(t+1) = \mathbf{w_k}(t) + \eta(t) \times (\mathbf{I}(t) - \mathbf{w_k}(t))$,, où η est le taux d'apprentissage défini par l'utilisateur. Le poids de tous les voisins de la MUA sont mis à jour de façon similaire, mais le taux d'apprentissage η diminue selon une fonction gaussienne de la distance à la MUA. Ce processus est répété pour chaque exemple des données d'apprentissage pour un nombre (généralement grand) de cycles N_{max}. Les clusters des données d'apprentissage s'auto-organisent progressivement sur la carte. Dans le cadre de l'apprentissage supervisé, des étiquettes peuvent de plus être attachées aux neurones, et le SOM peut être utilisé pour classer les exemples de test. La fonction objectif utilisée est alors classiquement le taux d'erreur de classification. Il convient de noter que le but d'ici n'est pas d'atteindre le meilleur rendement possible pour le problème MNIST (SOM ne rivalise pas avec les meilleurs résultats publiés à ce jour) mais de comparer les performances relatives des différentes topologies sur le même problème.

Problème direct : Le but des premières expériences est de comparer les performances de classification de SOM construites sur des topologies différentes, allant des topologies régulières aux topologies aléatoires en suivant le modèle de Watts et Strogatz. Les résultats montrent que pour un apprentissage long, la performance du réseau est clairement indépendante de la topologie. Cela n'est pas surprenant puisque le rôle de la topologie diminue avec le rayon R. En effet, le nombre des voisins dans un rayon R d'un neurone augmente lorsque la probabilité de recâblage p augmente. Toutefois, cette différence disparait quand R diminue. Des différences importantes sont évidentes pour un temps d'apprentissage court ou moyen : plus le réseau est aléatoire, moins efficace il sera pour ces échelles de temps. Plus aléatoire est le réseau, plus petite est le chemin le plus court moyen. Par conséquent, une interprétation possible est que, pour de grandes valeurs de p, l'influence d'un neurone pour une période d'apprentissage courte s'étend sur la totalité de l'espace 2-D, à presque tous les autres neurones. Ainsi, pour des échelles de temps courtes, presque tous les neurones sont mis à jour chaque fois qu'une nouvelle image est présentée, ce qui interdit effectivement tout apprentissage dans le réseau. Cette interprétation est validée par des expériences dans le cas où le rayon initial est cinq fois moindre.

Problème inverse : Les algorithmes évolutionnaires [46] ont été choisis pour optimiser la topologie de SOM, ceci étant motivé par leur flexibilité et leur robustesse face aux minima locaux. Le but de la deuxième expérience est de savoir si l'optimisation aura tendance à pousser la topologie vers des réseaux aléatoires, ou vers d'autres topologies, en dehors des modèles de Watts et Strogatz. La performance du taux de reconnaissance peut être augmentée de près de 10% par l'optimisation évolutionaire de la topologie du réseau. L'erreur de classement de la meilleure topologie dans la population diminue au cours de l'évolution, à partir de $0,355$ jusqu'à $\approx 0,325$. Le plus court chemin moyen entre les neurones diminue en même temps que l'index de clustering, ce qui signifie que la topologie devient plus aléatoire. L'écart-type σ_k de la distribution de connectivité Pk (où Pk est la probabilité qu'un neurone choisi au hasard a k voisins) a par contre presque triplé au cours de l'évolution. Cela signifie que la distribution de connectivité des réseaux s'élargit En d'autres termes, l'évolution artificielle rend les réseaux plus hétérogènes. Toutefois, il convient de garder à l'esprit que ce résultat est largement tributaire de la topologie des données elles-mêmes (ici la base de données MNIST), et pourrait être différent avec d'autres données.

Ce résultat a été présenté à la conférence ECCS 2007 [110](European Conference on Complex Systems 2007 – Dresden) et à la conférence GEC 2009 [111] (Genetic and Evolutionary Computation 2009 – Shanghai).

Echo State Networks (ESN)

Il est connu depuis longtemps qu'une bonne conception de la topologie du réseau est un ingrédient essentiel pour une application réussie des réseaux de neurones. Sur le plan théorique, les études récentes sur les représentations profondes ont prouvé que certains types de topologies nécessitaient un nombre exponentiel d'unités cachées afin d'être en mesure de réaliser une tâche d'apprentissage donnée, tandis que les topologies de profondeur pourrait ne nécessiter qu'un nombre linéaire de couches pour la même tâche. Les Echo State Networks

[105], qui ont été récemment proposés pour l'apprentissage supervisé de séries temporelles, peuvent être considérés comme une approche alternative : au lieu de l'optimisation d'une topologie pour une tâche donnée, il propose d'utiliser un large réservoir de neurones dont les connexions sont tirées au sort (et à faible densité). Seuls les poids des connexions sortantes sont à apprendre, ce qui transforme le processus d'apprentissage en un simple un problème d'optimisation quadratique qui est facilement résolu par une méthode basée sur le gradient ..., du moins dans le cas de l'apprentissage supervisé.

Notre construction : Nous avons proposé d'utiliser l'algorithme évolutionnaire état de l'art pour l'optimisation continue, CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [87, 88, 85] pour remplacer la méthode du gradient dans le cadre non supervisé. CMA est un algorithme évolutionnaire reconnu et a obtenu de bons résultats sur un large éventail de problèmes dans le domaine de l'optimisation continue. La flexibilité de l'optimisation par l'évolution nous permet en outre d'optimiser non seulement les poids sortant, mais également, ou alternativement, d'autres paramètres de l'ESN (le rayon spectral, les pentes à l'origine des fonctions de transfert des neurones actifs). En ce qui concerne le rayon spectral, il est unanimement reconnu comme un paramètre critique des ESNs. En particulier, la valeur maximale autorisée pour le rayon spectral afin d'assurer la propriété "Echo State" est de 1. Cependant, des valeurs différentes ont été proposées dans la littérature pour différents problèmes. Il semble donc légitime de ne pas fixer a priori le rayon spectral mais de le considérer un paramètre libre optimisé par CMA-ES : cela ne fait qu'ajouter une dimension au problème. Pour les pentes des fonctions d'activation des neurones, comme l'Echo Etat Network est un ensemble de systèmes dynamiques qui sont combinés linéairement pour produire le résultat désiré, il semble plausible que la modification des pentes de tous les neurones de manière indépendante puisse permettre à l'ESN de mieux répondre à la tâche-cible. La fonction de transfert dans le neurone interne devient $\tanh_a(x) = \dfrac{2}{1+e^{-a*x}} - 1$. A l'origine la fonction sigmoïdale de Jaeger était tanh, correspondant au cas $a = 2$.

Supervised Learning : Dans cette partie, nous prenons un problème standard de prédiction de séries temporelles. Un signal mono-canal d'entrée sinusoïdal est donnée par $u(n) = sin(n/5)$. L'objectif est de d'apprendre au réseau à produire une sortie mono-canal, $y_{teach}(n) = \frac{1}{2}u^7(n)$. Le premier résultat, pour une taille de réservoir de 100, confirme simplement que CMA-ES peut être aussi précis que la méthode du gradient citée dans [105], en nécessitant toutefois un effort de calcul beaucoup plus important. Les résultats obtenus correspondent à la même précision lors de l'optimisation des poids de sortie que les résultats originaux obtenus à l'aide d'une optimisation quadratique. Avec un réservoir plus petit, cependant, c'est l'optimisation des pentes des fonctions de transfert de tous les neurones qui permet d'atteindre la meilleure précision de prédiction, pour un coût de calcul toutefois assez élevé. De plus, il existe une corrélation intéressante entre la valeur finale du rayon spectral à la fin de l'algorithme et la performance correspondante obtenue. Dans le cas de $= 100$ neurones, les meilleurs résultats sont obtenus lorsque le rayon spectral est compris entre 0,6 et 0,8, et dans le cas de 30 neurones, le rayon spectral optimal est centré autour de 0,96.

Apprentissage par renforcement : Le problème de l'équilibre des barres (ou encore du double pendule inversé) est une tâche de contrôle classique étudiée depuis plus de 40 ans – voir pour les plus récents travaux [225, 78, 71, 204, 101, 45, 70, 117]. Le système se compose de deux parties. La première partie est un charriot dont la masse est de 1 kg et qui a un degré de liberté le long de l'axe x, la deuxième partie comprend une ou deux barres de longueurs différentes ($(l_1 = 1m, l_2 = 0.1m)$) et de masses différentes ($m_1 = 0.1kg, m_2 = 0.01kg$)) qui sont connectés au charriot par des charnières. Chaque barre a un degré de liberté, à savoir l'angle de l'articulation (θ_1, respectivement θ_2) avec la verticale. La commande est une force F_x ($F_x \in [-10N, 10N]$) qui est appliquée au charriot, et le défi est de garder les pôles dans des limites données pour les angles des articulations aussi longtemps que possible. Nous montrons que l'ESN évolutionnaire obtient des résultats qui sont comparables à ceux des meilleurs algorithmes faisant l'apprentissage de la topologie des réseaux de neurones. Les meilleurs résultats sont ici obtenus en n'optimisant que les poids sortants. En outre, il semble y avoir une forte dépendance des résultats par rapport à la topologie du réservoir, au moins pour les petites tailles expérimentées ici.

Modèle de Reconnaissance d'Objets Visuels Inspiré par le Cerveau

Enfin, le chapite **SMF** est la dernière partie qui présente nos travaux initiaux de recherche sur la SMF ("Standard Model Feature"), un modèle de reconnaissance d'objets qui est inspiré par le cortex visuel. Deux version basées sur SMF ont participé au Challenge PASCAL - Reconnaissance de la multi objets visual (VOC2008). Le but à long terme est de trouver la topologie optimale du modèle SMF, mais le coût de calcul est cependant trop important pour optimiser la topologie complète directement. Donc dans un premier temps, nous appliquons un algorithme évolutionnaire pour sélectionner automatiquement les caractèristiques utilisée par le système pour la classification finale. Nous montrons que, pour VOC2008, avec seulement 20% caractèristiques bien choisies, le système peut fonctionner aussi bien qu'avec l'ensemble des 1000 caractèristiques choisies au hasard.

Pour les tâches de reconnaissance immédiate d'objets dans une scène, il a été suggéré que le cerveau utilise des propriétés d'invariance de l'objet. Le modèle SMF est inspiré par cette remarque. La SMF est fondamentalement un réseau de neurones à propagation directe (“feed-forward”) hiérarchique. Dans ce manuscrit, nous avons utilisé une simplification du modèle de référence [194], donné dans [198], qui se compose de quatre couches de réseaux de neurones non récurrents. Le modèle comporte deux types de neurones : ceux qu'on appelle simples unités, ou S, et les unités complexes, ou C. Les unités S combinent leurs entrées avec une “Bell-Shaped tuning function” pour augmenter la sélectivité. Les unités C combinent leurs entrées avec une fonction maximum (MAX) pour augmenter l'invariance. Par conséquent, en réglant les paramètres du système, le modèle peut obtenir un bon équilibre entre la sélectivité et l'invariance.

Dans ce modèle initial, la sortie du réseau est composée de caractéristiques qui sont passées au travers d'un algorithme de combinaison linéaire pour la tâche de classification elle-même, c'est à dire qu'il calcule la confiance de VOC 2008 de chaque image pour chaque classe d'objet. Dans notre étude, nous appliquons une optimisation évolutionnaire utilisant CMA-ES (Covariance Matrix Adaptation Evolution Strategy [88, 103] pour calculer la confiance de VOC2008 de chaque image pour chaque classe d'objet. Premièrement, nous utilisons CMA-ES afin d'optimiser le poids d'un combinateur linéaire. Comme la dimension d'optimisation est au moins aussi grande que 1000, nous testons également un algorithme multi-évolutionnaire pour sélectionner de façon optimale 200 dimensions parmi 1000, avant d'optimiser les 200 dimensions choisies.

Nous montrons qu'en sélectionnant 200 caractéristiques optimales le système peut garder presque les mêmes performances dans le défi VOC2008 qu'en utilisant 1000 caractèristiques choisies au hasard, tout en diminuant le coût calcul de 2 ordres de grandeur. La robustesse du système semble dépendre des caractéristiques sélectionnées. Même si nos résultats sur VOC2008 ne sont pas parmi les meilleurs du challenge, en considérant la simplicité du modèle que nous avons appliqué (nous utilisons seulement la mise en modèle par défaut et il y a beaucoup de paramètres qui peuvent être ajustés), nous estimons qu'il y a encore de la place pour des améliorations significatives. Les futurs travaux de recherche seraient d'optimiser la topologie des connexions entre les couches du modèle de SMF, et également d'optimiser les paramètres de réglage qui ont effectivement été définis par la recherche bio-inspirée.

Contents

1 Introduction **1**

2 Background **5**

2.1 Artificial Neural Networks 5

2.1.1 History 6

2.1.2 Learning Methods 9

2.1.3 Network Topology 11

2.1.4 Recent research 12

2.2 Evolutionary Computing 15

2.2.1 Key Issues 16

2.2.2 Historical Trends 20

2.2.3 An Adaptive Evolution Strategy: CMA-ES 22

2.2.4 Applications 26

2.3 Evolving Artificial Neural Network 28

2.3.1 Evolving Connection Weights 28

2.3.2 Evolving Network Topologies 31

2.4 Topology of Complex Networks 43

2.4.1 Small World Topology 43

2.4.2 Scale Free Topology 45

2.4.3 Applications to ANNs 47

2.5 Research Questions 47

3 Evolving the Topology of Self-Organizing Map **49**

3.1 Introduction 49

3.2 Topology of Self-Organizing Map 49

3.2.1 Kohonen Maps 49

3.2.2 An Example with Color Data 51

3.2.3 Quantitative measure of topology preservation in SOM . . 52

3.2.4 Discussion 55

3.3 Method and Experiments 56

3.3.1 A Simple Experiment with Classical Q H Measure 56

3.3.2 Experiments With MNIST database 59
3.3.3 Learning 60
3.3.4 Distance 62
3.3.5 Labeling 62
3.3.6 Classifying 65
3.4 Direct problem 67
3.4.1 Influence of the radius 67
3.4.2 Robustness against noise 69
3.5 Inverse problem 71
3.5.1 The Algorithm 71
3.5.2 Results 73
3.5.3 Generalization w.r.t. the Learning Process 73
3.6 Conclusion 76

4 Evolutionary Optimization of Echo State Networks 79
4.1 Introduction 79
4.2 Reservoir Computing Model 81
4.2.1 A Chaotic Time Series Prediction by ESN 83
4.2.2 Researches on RC 83
4.2.3 Discussion 86
4.3 Supervised Learning of ESN 87
4.3.1 Jaeger’s Original Settings 87
4.3.2 Which parameters to optimize? 89
4.3.3 The experiments 90
4.3.4 Comparative Measures 91
4.3.5 Results 91
4.3.6 Discussion 95
4.4 Reinforcement Learning of ESN 95
4.4.1 The Pole-balancing Benchmark 95
4.4.2 Fitness(es) 99
4.4.3 Experimental conditions 100
4.4.4 Results and Discussion 101
4.5 ESN vs Developmental Methods 110
4.6 Conclusion 111

5 Feature Selection in a Cortex-Like Object Recognition Model 113
5.1 Cortex-Like Object Recognition 114
5.1.1 Visual Object Recognition in the Cortex 114
5.1.2 Standard Model Features (SMFs) 115
5.1.3 Model Details 116
5.1.4 The Perspectives of the Model 121

5.2 PASCAL Visual Object classes Challenge (VOC08) 122
5.2.1 Classification/Detection Image Sets 123
5.2.2 Classification Task 125
5.3 Experiments 128
5.3.1 Using EC Algorithms with SMF Model 128
5.3.2 Pre-training of SMFs model 128
5.3.3 Linear Combination using CMA-ES 129
5.3.4 Multi-Evolutionary Optimization 130
5.3.5 Results 138
5.3.6 Result Analysis 138
5.4 Conclusion 142

6 Conclusion **145**
6.1 Summary of Results 145
6.1.1 Evolving SOM Topology 145
6.1.2 Evolving Parameters and Hyper-parameters of ESN 146
6.1.3 Feature Sub-Sampling in SMF Bio-Inspired Approach . . 147
6.2 Perspectives 148
6.2.1 Fundamental Issues 148
6.2.2 Practical Issues 149

Chapter 1

Introduction

We are now in the 21st century, and since the 1950s, Artificial Intelligence has become an independent subject of study of Computer Science, and its influence on our daily life has been ever increasing.

Intelligence is the result of an evolutionary process based on natural selection. In recent years, studying the so-called bio-inspired mechanisms, i.e. mechanisms that try to somehow mimic natural processes, has raised a lot of research interest. In particular, two of these research streams have attracted great attention and lead to a large body of work.

From a macroscopic point of view, nature produced a rich and diverse set of species thanks to two principles: blind variations, that created the diversity in the first place, and natural selection, that selected some individuals and populations for survival according to the natural principle of "survival of the fittest". Artificial Evolutionary Algorithms (EAs) [47, 43] are powerful optimization algorithms inspired by this natural selection mechanism, and have been successfully applied in lots of real world problems [231].

From the microscopic point of view, the material basis for intelligence is based on large-scale ensembles of neurons, organized in networks. Artificial Neural Networks (ANNs) are a powerful model inspired by their biological counterparts for knowledge processing and analysis. Originated in the early 50s [182], ANN research is still very active [105, 143, 93, 198].

At the crossroad of both research areas, research using Evolutionary Algorithm to optimize Artificial Neural Networks has been going on for many years [230]: Beyond the "simple" optimization of the weights of a network with a fixed given topology, the flexibility of Evolutionary Algorithms made them appealing when it came to optimize also the topology of candidate neural networks for a given task.

During the same period (the last 20 years), the study of complex networks has brought this area into new research directions and inspiration. Inspired both by natural networks (e.g. Gene Regulatory Networks, protein-protein interaction networks, ...) and artificial ones (e.g. the World Wide Web, airline connection networks, social networks such as co-authoring networks, ...), new insights have been gained into network topologies. More particularly, researches on small-world [220] and scale-free [19] networks have produced a new point of view for our understanding of network complexity.

Chapter 2 will survey those research areas in more detail, highlighting the different paradigms that have been used for this thesis.

In this context, the present work is centered around the relationship between the (possibly complex) topology of a given Neural Network and its performance as a computing unit. Two approaches will be considered, and experimented with in different application domains. The **direct problem** is the study of the performance of a given type of neural network, in a given environment, with respect to its topology. By controlled variation of the topology, within a given class of parameterized topologies, and careful measure of the resulting network performance, some hints can be gathered regarding the influence of this or that topology description parameter. We will then consider the **inverse problem**, i.e. optimize the topology in order to maximize the performance of the neural network. Evolutionary Algorithms will be the optimization tool, as they can handle either continuous optimization, in the case where the topology is described by some macroscopic description parameters, or directly handle the topology itself (e.g. through modifications of the connections) to fine-tune the topology.

A first application domain will be that of Self-Organizing Maps (SOMs) [124], detailed in Chapter 3. SOMs are mainly used for unsupervised learning, and the standard regular topology is generally used without much questioning. However, there is hardly a universally acclaimed performance measure for comparing the performance of unsupervised learning algorithms. Hence, the performance of a given SOM topology will be assessed through the classification accuracy on the well-known MNIST database for digit recognition. The inverse problem will be addressed here by directly manipulating the connections between neurons.

A second application context, presented in Chapter 4, is that of Echo State Networks [105], the recent paradigm pertaining to the Reservoir Computing family. While ESNs are generally used for regression tasks, and the optimization problem then amounts to the quadratic optimization of the outgoing weights, using ESNs for reinforcement learning tasks has hardly been addressed in the literature. We will investigate the influence of different hyper-parameters defining the reservoir topology on the well-known benchmark task of balancing the double-pole. The optimization algorithm for the output weights will be the well-known CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [88, 86], as the problem is not quadratic any more. But the good side of this drawback is that the same algorithm can be used to simultaneously optimize the output weights and some hyper-parameters defining the topology of the reservoir.

Finally, Chapter 5 will present a first study involving a Cortex inspired Visual Object Recognition model [198]. In particular, we will investigate whether using only a sub-sample of the many features designed by the original algorithm can improve the global recognition rate, while accelerating the learning phase. Results will be demonstrated on the complex VOC2008 challenge.

And as usual, Chapter 6 will conclude this dissertation, summarizing and discussing the results, and opening the way for further research.

1. INTRODUCTION

Chapter 2

Background

This chapter will set up the background for all work to be presented in this dissertation. First, brief introductions on Artificial Neural Networks (ANNs) and Evolutionary Computing (EC) will be given. Then we will focus on Neuro-Evolution, the coupling of both techniques, i.e., the specific EC methods that have been developed for evolving Artificial Neural Networks. Further, recent advances regarding the Topology of Complex Networks will be surveyed. Finally, in the light of this state-of-the-art methods, we will introduce the research questions addressed in the present research and their motivations.

2.1 Artificial Neural Networks

The human brain is a complex biological system composed of a large number of highly interconnected processing elements (neurons) [137]. It is formed by about 10^{11} neurons [104]. Each neuron connects to about 10^4 other neurons on average [155, 30]. Figure 2.1 shows an example of a biological neuron.

The physiological research about the brain and other biological neural systems is the foundation for artificial neural networks (ANNs). The artificial neural model is an abstraction of biological neurons (shown in figure 2.2) , i.e. an interpretation of our understanding of brain operations, applied to build an artificial intelligence system.

Artificial Neural Networks (ANNs) are parallel information processing systems in which a number of artificial neurons (or units) are interconnected by synapses. Usually the output of each unit is a nonlinear function of the sum of its inputs, weighted by synaptic weights (see figure 2.2). The main idea is then to find algorithms or heuristics for adjusting the synapses and the synaptic weights

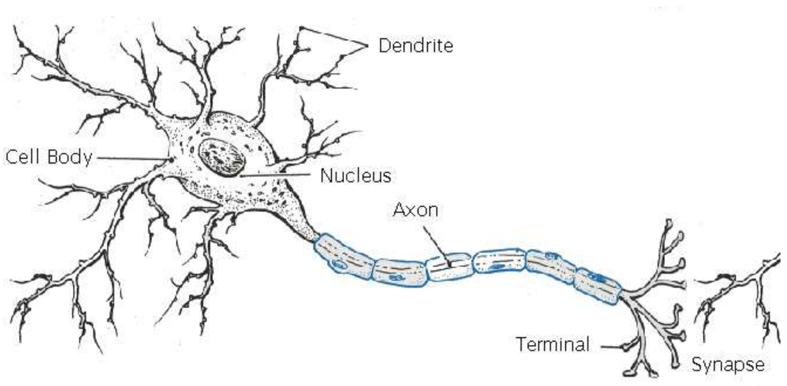


Figure 2.1: A neuron is an excitable cell in the nervous system that processes and transmits information. It exists in a number of different shapes and sizes. This figure from [32] shows a scheme of a spinal motor neuron, consisting of a cell body and a long thin axon. Around the cell body is a branching dendritic tree that receives signals from other neurons. The end of the axon has branching terminals (axon terminal) that release neurotransmitters between the terminals and the dendrites of the next neuron, which process the information transmission.

in such a way that the network fulfills the desired information processing task.

The success of Artificial Neural Networks in computer science and machine learning is mainly based on the following strong points:

- ANNs can discretionarily approximate any linear or complex non-linear function [98, 11];
- All quantitative or qualitative information is stored in a distributed manner in the connections of the network, so it takes on strong character of robustness and tolerance [182];
- The use of parallel distributed processing approach enables parallel computing [210];
- Self-adaptive and self-organizing ability endows ANNs with the capability to learn even when only limited knowledge about the system is available [122].

2.1.1 History

The history of ANNs is as long as that of Artificial Intelligence, but it is also more tortuous.

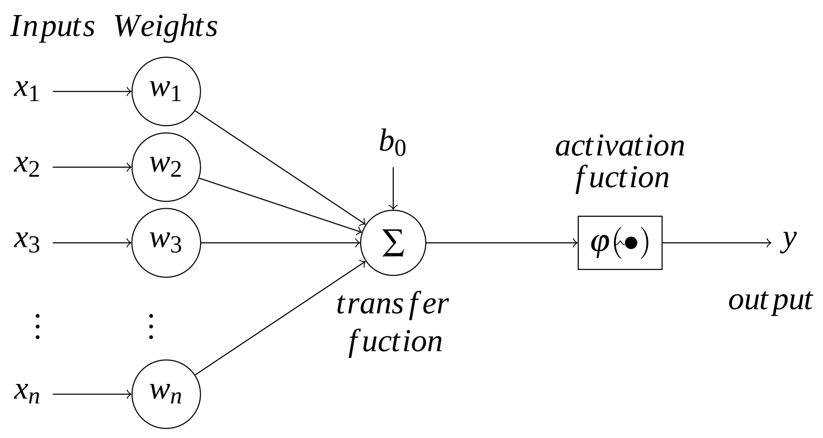


Figure 2.2: The artificial neural model is an abstraction of biological neurons. The artificial neuron receives one or more inputs (representing one or more dendrites) and sums them. Usually each input x_i is weighted in the sum by the corresponding synaptic weight w_i, and the resulting weighted sum is passed through a non-linear function known as the activation function or transfer function. This mimics the nonlinear input-output relationship observed in real biological neurons. The transfer functions usually have a sigmoidal shapes, but they may also take the form of other non-linear functions, piecewise linear functions, or step functions. They are also generally monotonically increasing, continuous, differentiable and bounded.

Creation: In 1943, psychologist McCulloch and symbolic logician Pitts built the first mathematical neuron model [147], known as the MP model. They gave a formal mathematical description: For a given artificial neuron k, let there be $m+1$ inputs with signals x_0 through x_m and weights w_{k0} through w_{km}. The output y_k of neuron k is: $y_k = \varphi\left(\sum_{j=0}^{m} w_{kj}x_j\right)$, where φ is the transfer function.

They proved that a single neuron can perform logic functions, thereby creating the era of artificial neural networks. In 1949, the psychologist Hebb proposed a conception that the synaptic contacts could have variable intensities, depending only on the activation of the pre- and postsynaptic neurons [91]: “When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased”.

This so-called Hebb rule of neural network is an underlying basis for the learning algorithm of ANNs. In 1958, Rosenblatt built the Perceptron model [177]. The Perceptron is a binary classifier that maps its input x_k (a real-valued vector) to an output value y_k (a single binary value) across the matrix of weights $(w_{i,j}$:

$$y_k = \begin{cases} 1 & \sum_{j=0}^{m} w_{kj}x_j > 0 \\ 0 & \text{else} \end{cases}$$

The perceptron model is a specifical case of MP model and the Hebb learning algorithm can also be used to tune their connection weights (see section 2.1.2).

The Neural Network Winter: After analyzing the function and the constraints of artificial neural networks, typically represented by Perceptron, Minsky and co-author published the «Perceptron» book in 1969 [151]. In this book they pointed out the limit of the perceptron model. For instance, they reported that no single-layer perceptron can solve higher-order classification problems, including nonlinearly separable problems such as the XOR function. Their arguments greatly impacted the study of ANNs. At the same epoch, serial computers and artificial intelligence progressed rapidly. These two reasons caused the lack of the necessity and urgency to develop new methods for computing. Therefore the research on ANNs was in depression during more than ten years afterwards. However, some researchers still continued to develop the field and proposed the theory of self-organizing map (SOMs) [122, 123] in 1972, and established a strong mathematical foundation for ANNs [3].

The Return of Spring: Three years after Minsky’s book, Grossberg published the paper introducing multi-layer networks capable of modeling XOR functions [74]. But the research on ANNs did not return to Spring until the 1980s, helped by the progresses in CPU power. In 1982, physicist Hopfield proposed what is today known as the Hopfield Neural Network model [96]. A Hopfield net is a form of recurrent artificial neural network. Hopfield nets serve as content-addressable memory systems with binary threshold units. The introduction of the “calculational energy” determined the stability of the neural network. They are guaranteed to converge to a local minimum, but convergence to one of the stored patterns is not guaranteed.

In 1983, Hinton and Sejnowski gave the name of Boltzmann machine to a type of stochastic recurrent neural network [94]. Boltzmann machines can be seen as the stochastic, generative counterpart of Hopfield nets. They were one of the first examples of a neural network capable of learning internal representations. They are able to represent, and, given sufficient time, to solve difficult combinatorial problems. If the connectivity is constrained, the learning can be made efficient enough to be useful for practical problems. In 1985, the statistical thermodynamics simulated annealing techniques, which applied in the Boltzmann model, helped to prove that the whole system will eventually converge toward a global stability point [1]. In 1986, by studying the microstructure of cognition, Rumelhart and the PDP Research Group proposed the theory of Parallel Distributed Processing [38]. The famous Back-Propagation (BP) algorithm for multi-layer feedforward ANNs was proposed in 1986 too [181].

2.1.2 Learning Methods

One major difference between classical programming techniques and ANNs is that the later are not strictly speaking programmed, but must be trained before they are used. A number of learning methods have been developed over the years, that can be divided into Supervised Learning, Non-supervised Learning and Reinforcement Learning.

Supervised Learning

In the supervised learning case, the training examples are given together with the expected outputs or labels (in classification case) called teacher’s dates. By comparing the margins between the expected dates and ANN’s outputs, the connection strengths are adjusted and converge to a stable position. When the environment changes after the training, the network is retrained and adapted to the new environment. The Back-propagation algorithm [181] is a widely used algorithm in the feedforward multi-layer ANNs. It is a supervised learning method, and is an implementation of the Delta rule.

The Delta rule is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. For a neuron j, with activation function $g(x)$, the delta rule for j's i_{th} weight w_{ji} is given by $\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i$, where α is a small constant called learning rate, $g(x)$ is the neuron's activation function, t_j is the target output, h_j is the weighted sum of the neuron's inputs, y_j is the actual output, and x_i is the ith input. It holds $h_j = \sum x_i w_{ji}$, and $y_j = g(h_j)$.

The Back-propagation algorithm can be summarized as follows:

1. Present a training sample to the neural network.
2. Compare the resulting output with the desired output for the given input. This is called the error.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error by applying the Delta rule.
5. Assign "blame" for the local error to neurons at the previous level, back-propagate the local error to the neurons of the previous level according to their connection weights with the neurons of current level, adjust the weights by applying the Delta rule too.
6. Repeat from step 3 on the neurons at the previous level, using each "blame" has its error.

In the feed-back ANNs domain, the Reservoir Computing model [105] attracted a lot of attention in recent years, in particular for the Supervised Learning of time series. It will be introduced in detail in chapter 4.

Unsupervised Learning

In the Unsupervised Learning case, there is no expected output associated with the input, so that, the ANNs is expected to find the correct input-output matching by using self-organized methods based on its unique network architecture and learning rules. In the learning process, ANNs continue to accept the learning samples and auto-organize the data in the pattern of connection weights among neurons. Self-organizing map [122, 123] network is typically used in this kind of learning model, to create low dimensional view of high dimensional data.

Reinforcement Learning

In a way, Reinforcement learning can be considered an intermediate between supervised and non-supervised learning. In reinforcement learning, the teacher does not provide the correct output associated with the input, but a simple reward (or punishment) when the predicted output is correct (false). This reward can be delayed. In all robotic experiments where the robot must find its way out of a maze. The system tries to find a policy for maximizing cumulative rewards over the course of the problem. Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control (chapter 4), elevator scheduling, telecommunications, backgammon and chess [209, 114].

2.1.3 Network Topology

Based on connection topology, neural network models are traditionally divided into feedforward networks and feedback (or recurrent) networks. However, recent developments in the so-called complex network field lead to contemplate the use of other classes of topology. We will present these aspects in section 2.4.

Feedforward networks

Feedforward networks are structures in which there is no loop in the connection graph of the network. A specific case is that of layered feedforward architectures, where the neurons are organized in layers, and where each neuron takes its inputs from the previous layer and sends its output to the next one. Because the connection graph is loop-free, there are no feedbacks in the network (Figure 2.3). Consequently, the networks has no real dynamics: information just flows from one layer to the other. Such a network realizes the projection from the input space to the output space, and its information processing capacity comes from a multiple compound of simple nonlinear functions. This kind of network structure is simple and easy to implement (compared to networks with loops). Backpropagation is an especially well-suited supervised learning method for feedforward networks, and even easier to implement within multi-layer feedforward networks.

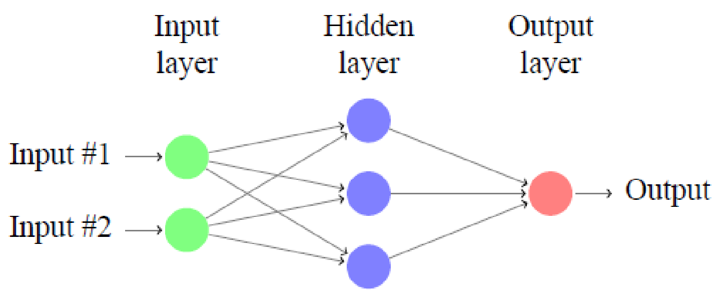


Figure 2.3: A sample feedforward ANN with one hidden layer. In feedforward network, each neuron receives its inputs from the previous layer and sends its outputs to the next one. As there is no feedback in the networks, it is a loop-free map.

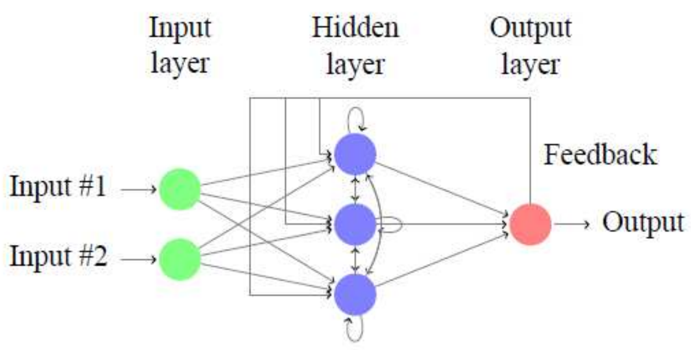


Figure 2.4: A sample feed-back ANN with one hidden layer. There exists feedbacks within the same layer or/and from the next one.

Feedback networks

In a feedback network, there exists recurrent connections, i.e. feedbacks within the same layer or from a given layer to a preceding one. It can be figured by a map of loops (Figure 2.4). Its information processing capacity comes from the transformation of network states. Because of the feedback connections, these networks are expected to undergo complex dynamics, that can be approached with dynamical systems theory. Hopfield networks, Boltzmann machines and reservoir computing models are feedback networks.

2.1.4 Recent research

Recent Neural Network research can be divided into theoretical research and applied research.

Theoretical Research:

Theoretical research can be divided into two categories:

1. Application of the scientific research in neurophysiology and cognitive science to understand the mechanisms of intelligence [21].

2. Improvements of the performance of ANN in various domains: Using the research results of basis neural theory, the goal is to explore the performance of neural network model, to study in-deep of network algorithms and performance and to develop new mathematical theory of Neural Networks [105, 93, 194].

Applied research:

Application can also be divided into two categories:

1. ANN's software simulation, as demonstrated and disseminated with the following software platforms: Stuttgart Neural Network Simulator (SNNS), JavaNNS, Neural Lab, etc, and corresponding hardware implementations [136].

2. Application of Neural Network in various fields, such as pattern recognition, signal processing, knowledge engineering, expert systems, optimization, robot control, etc.

Recent hot research topics

Below are some hot points in ANNs research area in recent years:

Self Adaptation:

"How can an ANN self-adapt to its environment?" is always an important issue. The most famous self-adaptive ANN model is Kohonen Map (aka Self-Organizing Maps) which has been wildly used in classification and representation of information. SOMs will be presented and studied in chapter 3. The search for further unsupervised learning methods is still an active domain of research.

Complex Systems Theory:

Since the 1990's, theoretical studies of Complex System have deeply changed our view of many fields [192, 232]. "Complex systems are systems where the collective behavior of their parts entails emergence of properties that can hardly, if not at all, be inferred from properties of the parts. Examples of complex systems include ant-hills, ants themselves, human economies, climate, nervous systems, cells and living things, including human beings, as well as modern energy or telecommunication infrastructures." – The Complex Systems Society (CSS). ANNs are present in many areas of Complex Systems, and can be seen as one example of complex systems, and recent research [199] shows that the performance of an ANN can be influenced by its topology, just like what is well-known in other complex systems (social networks, biological network, internet etc) [28].

Reservoir Computing and Deep Belief Network:

ANNs still suffer from (at least) some crucial issues. The difficulty of learning increases with the number of neurons (and synapses). Furthermore, learning feedforward networks is also still very difficult when the network is made of numerous layers. Finally, there is still no training algorithm for recurrent networks. Reservoir computing and Deep Belief Networks are, respectively, two elements of answer to these issues.

The Reservoir Computing model, proposed independently by Jeager [105] (Echo State Networks) and Maass and colleagues [143] (Liquid State Machines), uses a large recurrent (feedback) network of neurons that are randomly and sparsely connected (the "reservoir"). The weights and topology of this reservoir are not optimized nor adapted and are kept fixed during training. The reservoir neurons are connected to output neurons that are used as a readout of the reservoir states. The major idea behind reservoir computing is that only outgoing weights (those from the reservoir to the output neurons) are optimized, which amounts in the supervised case to a simple quadratic optimization problem that is easily solved by any gradient-based method. Hence, reservoir computing can use a large number of neurons and weights (within the reservoir) connected as a feedforward network with rich dynamics, but learning is restricted to the lower number of feedforward outgoing weights. The promises of Reservoir Computing have been fulfilled for several kinds of problems. See chapter 4 for further details.

The Deep Belief Network model was proposed by Hinton in 2006 [93].

It has been known for a long time that if the initial weights of a feedforward network that has multiple hidden layers are close to a good solution, gradient descent works well. Conversely, learning is poorly efficient whenever the initial weight values are far from a good solution. But finding such favorable initial weights is very difficult. In Hinton's model, each layer is trained just like in a single-layer network, one after the other and independently from the others. So a multilayer ANNs is divided into a stack of Restricted Boltzmann Machines (RBMs) and each hidden layer of the lower RBMs is the input layer of the higher RBM. After this pre-training, the model is unrolled and backpropagation of error derivatives can be used to fine-tune the ANNs to get good learning results.

The seminal work in Deep Belief Natworks [93] shows that the pre-training procedure works well for a variety of data sets. And inspired by this model, Jaeger proposed a hierarchical multilayer ESN model in 2007 [107].

Bio-inspired System:

The best current machine vision systems are still not competitive with human and primate natural vision in visual recognition field, especially for objects in cluttered and natural scenes of real world, despite decades of hard works [99, 166, 219, 198]. So taking inspiration from real biological systems is an important method to improve the performance of artificial systems.

Inspired by the biology of visual cortex, Serre and Poggio [198] proposed a model for invariant visual object recognition in 2007. The model gets the state of the art results in a series of visual benchmark tests on complex image data bases, such as CalTech5 [57, 221], CalTech101 [53], MIT-CBCL [92]. More details about this model will be discussed in chapter 5.

2.2 Evolutionary Computing

This section quickly surveys the bases of Evolutionary Computing, adopting the modern point of view popularized by [47, 43], and borrowing most ideas from [188].

Darwin's evolution theory is based on the two basic principles of natural selection and blind variations. Natural selection describes how individuals within a given population have better chances to survive and to reproduce if they are adapted to the current environment. Blind variations describes how the genetic material of the parents is randomly modified when transmitted to the children (even though Darwin had no precise idea about what such genetic material could be).

Evolutionary Algorithms (EAs) are stochastic optimization algorithms loosely inspired by this crude view of Darwinian theory. Candidate solutions to the optimization problem at hand, called individuals, are represented by their chromosome. The target function of the optimization, aka fitness, plays the role of the environment. Within this environment, a population, or set of individuals, is first randomly initialized, and the fitness of all individuals is computed. This population then undergoes a succession of generations. First, the population is subject to parental selection: some individuals are selected, based on their fitness, in such a way that fitter individuals are more likely to be selected than individuals with poor fitness. The selected individuals generate offspring individuals, thanks to the application of variation operators, by crossover (or recombination) operators, that involve 2 or more parents to generate each offspring, and mutation operators, where a single parent is randomly modified into an offspring. The newborn offspring are then evaluated, i.e., their fitness is computed. Finally, the loop is closed by applying survival selection, globally to both parents and offspring, in order to select the starting population for next generation; survival selection still favors fitter individuals.

Algorithm 1 Pseudo-code of Evolutionary Algorithm

```
Begin
    INITIALIZE population with random individuals;
    EVALUATE each individual;
    REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
        1 SELECT parents from population;
        2 RECOMBINE pairs of parents;
        3 MUTATE the resulting offspring;
        4 EVALUATE newborn offspring;
        5 SELECT individuals for the next generation;
    OD
END
```

2.2.1 Key Issues

The different components of the basic algorithm above will now be discussed in turn, and the key issues to consider when designing an EAs for a specific problem will be highlighted.

Fitness

How an individual adapts to the environment is measured by its fitness. Improving the fitness function should result in improving the objective function of the optimization problem at hand. On the other hand, modifying the fitness function by some linear transformation only impacts the selection steps of the algorithm. Hence we will consider in general that the fitness function **is** the original objective function, postponing the discussion about possible transformations to the discussion about selection procedures.

A crucial remark about the fitness is that it usually represents most of the computational cost of the EA. Indeed, in most real-world situations, selection and variation are generally very fast operations (though sorting huge populations can become time consuming when the population size increases) compared to the evaluation, that can involve heavy simulations of possibly non-linear physical, mechanical, chemical phenomenon. This also explains why the goal of the EC designer will be, beyond finding the optimum of the objective function at hand, to minimize the number of fitness function evaluations. And the usual measure for comparing different EA or EA settings is the number of evaluations needed to reach a predefined optimization goal (e.g., a given fitness value).

Representation

For a given problem, the representation of the candidate solution, or individuals, is crucial. The original optimization problem is posed in a given search space, and solutions must ultimately be given in this space. However, the EC designer faces two goals that are somehow contradictory. On the one hand, he needs to find the best possible solution in the original search space; On the other hand, exploration depends on the variation operators, and some structure of the search space is mandatory to make the exploration efficient. From this point of view, the choice of a representation and the design of the variation operators are the two faces of the same coin, and should be done together.

It is hence often useful to search an auxiliary space, where efficient variation operators can be designed, rather than the search space of the original optimization problem. Still crudely inspired from biology, the original search space is termed phenotypic space[1], while the space where variation operators are applied is called the genotypic space. The phenotypes (or potential solutions of the optimization problem at hand) are encoded into the genotypic space, where the actual search takes place through the application of the variation operators. The individuals

[1]or sometimes behavioral space as this is where the behavior of the individual is evaluated when the fitness is computed.

referred to in the previous Section 2.2 and in Algorithm 1 are genotypes, that are decoded into the corresponding phenotypes for evaluation.

'Natural' Selection

The two selection steps are the parental selection, that selects which individuals are allowed to reproduce, and the survival selection, that chooses which individuals will survive to the next generation. The main difference between both selections is that an individual can be selected several times as a parent, while all individuals are selected at most once for survival, and disappear forever if they are not selected. However, similar selection procedures can be used in both cases.

Another important remark is that both selection steps only depend on the fitness of the individuals in the population. In that respect, selection is **problem independent**, and can be viewed as a technical issue when designing an EAs for a specific problem. Many toolboxes exist indeed where the EC designer can pick the selection method of his choice, without the need to be creative.

Selection procedures can be either deterministic, or stochastic. Deterministic selection acts globally on the population, and selects the best (fitness-wise) individuals in the population up to the required number.

Two categories of stochastic selections can be distinguished, whether they are based on the fitness **values** or simply on the **ranks** of the individuals. The most popular value-based selection is the so-called roulette wheel selection [95, 68], where each individual is selected with a probability that is proportional to its fitness. Roulette-wheel, like all value-based selection procedures, suffers from being very sensitive to the actual distribution of the fitness values: whereas optimizing f or $\exp(f)$ are equivalent optimization problems, the corresponding roulette-wheel selection procedures will produce very different results on both settings, favoring much more the best individuals in the latter case: the selection pressure (the probability to select the best individual divided by the probability to select the average individual) is hard to control efficiently. This is why, in spite of many trials to cope with this defect, and in spite of some early theoretical studies on Schema Theory that are based on proportional selection [95, 68], value-based selection procedures are hardly used nowadays.

Rank-based selections (including deterministic selection) are insensitive to monotonous transformations of the fitness function. Many variants have been proposed, including some roulette-wheel on the ranks of the individuals, rather than on their fitness values (aka ranking). However, the most popular selection procedure today is the tournament selection: it is a rank-based selection, and the selection pressure can be easily controlled through a single parameter, the tournament size T. In order to select one individual, tournament selection first picks up T individuals, and returns the best of those T. The selection pressure is proportional to T. When selection pressure smaller than 2 are required, a stochastic variant of the tournament of size 2 is used, where the best of the two uniformly chosen individuals is returned with probability $t \in]0.5, 1]$. The selection pressure is then $2t$.

Variation Operators

Variation operators are stochastic transformations defined on the genotype space. One distinguishes two types of variation operators based on the number of parents that are required to generate an offspring.

Crossover: The basic idea of crossover operators is to be able to recombine some good parts of the parents' genotypes to possibly generate offspring that have better fitness than their parents. Crossover involves two (or more [48]) parents, and implicitly assumes some sort of linearity of the fitness function with respect to (parts of) the genotype.

The debate about the usefulness of crossover has been going on for long: crossover can be seen as just a macro-mutation [113, 4], or even be considered harmful [61]. Early theoretical work like the Schema Theory [95, 68] only give hints about the way crossover works, though introducing the notion of Building blocks. Indeed, the crossover is only analyzed there by bounding its destructive effects. Their extension of Forma Theory [172, 173] is tentative to also take into account the positive effects of crossover, but did not result in more practical conclusions. More recently, the theory of mixing [211] was revisited in [67] in another effort to better understand the Building Block Theory and the effects of Crossover. In any case, the debate has nowadays cooled down, and the dominant point of view is that of pragmatism: if an efficient crossover can be designed for the representation at hand, then using crossover does improve the performance of the EAs (but this is a tautology!). Otherwise, a mutation-only EAs is the best choice, and many examples of such situations exist, as will be detailed for instance in Section 2.3.

Mutation: Whereas there can be pros and cons for using crossover operators, there has hardly been any discussion about mutation. Of course, like all very general statements, this one has its exceptions: Koza's seminal work on Genetic Programming [125] did not use mutation, but provided sufficient provision of Building Blocks by using very large populations; the particular CHC algorithm uses an exploratory crossover, and hence does not need additional mutation [49]; same remark applies to the SBX crossover [41]. But almost all practitioners have experimented that in general the lack of mutation largely degrades the efficiency of any EAs, and early theoretical results [180] as well as more recent ones [6] prove that the ergodicity of the mutation operator (i.e., its ability to visit the whole search space with positive probability) is mandatory to ensure any convergence result of EAs.

The probability of mutation that should be used is related to which variation operator (crossover or mutation) is considered the major drive of evolution. When crossover plays this role, like in traditional GAs, mutation is simply a background operator ensuring ergodicity, and should be applied sparsely: only a few individuals should be mutated (generally after crossover has been applied). When mutation is the main (or the only!) driving force, it should be applied to all individuals. When relevant, the issue of the strength of mutation is also important, and mutation is generally defined such that 'small' mutations are more likely than 'large' ones. But the idea of small and large mutations in fact refers to the resulting modification of the fitness, and relates to the idea of Strong Causality [174]. Such considerations lead to the idea of adaptive operators, responsible for the great successes of Evolution Strategies (see Section 2.2.3 below). Unfortunately, such efficient adaptive strategies could not as far as we know be generalized to other domains than that of continuous optimization.

Exploration Vs Exploitation

An important concept to keep in mind when designing an Evolutionary Algorithm is the Exploitation Vs Exploration dilemma. At any time during evolution, the algorithm must choose whether to look around the best individuals found so far, exploiting the results of previous steps, or whether it should try to explore new regions of the search space that have not yet been searched. Parameter tuning should permanently take this balance into account. For instance, increasing the selection pressure will undoubtly favor exploitation. On the opposite, increasing the mutation probability (or the mutation strength) is very often the best way to increase the exploration. The case of crossover is less clear: whereas crossover is an exploration operator at the beginning of evolution, it becomes more and more an exploitation operator as the population diversity decreases.

2.2.2 Historical Trends

The main 3 historical trends of Evolutionary Algorithms were initially proposed in the mid 60s, though several seminal ideas in fact appeared in the late 50s (gathered 10 years ago in [60]). Independently, J. Holland in Michigan, I. Rechenberg and H.-P. Schwefel in Germany, and L. Fogel in California proposed to use artificial evolution as a model for problem-solving procedure, publishing thereafter the seminal books that grounded the field into reality.

John Holland [95] modeled adaptation in natural systems into what he called Genetic Algorithms, and the first PhD student in the field, Ken DeJong, used his ideas for function optimization [42]. Ingo Rechenberg and Hans-Paul Schwefel, two engineers in Berlin, optimized the shape of a nozzle using what was to become Evolution Strategies. Their ideas were generalized later in [174, 191]. Larry Fogel optimized Finite State Automata to predict time series [63]. The lack of CPU power of the computer at that time, resulting in the lack of possible real-world applications was responsible for what could be called the “EC winter” (see Section 2.1.1). In the 80s, however, Holland’s student David Goldberg applied GAs to optimize a gas pipeline system and found better than state-of-the-art solutions to this very complex problem. He later published his seminal book [68] “Genetic Algorithms in Search, Optimization and Machine Learning”, probably the most cited one in the field. This was the beginning of an extraordinarily revival of those ideas, and though still considered separate fields, the distinction between the 3 branches rapidly started to vanish, thanks to the second generation of pioneers like Z. Michalewicz [150], T. Bäck [13] and D. Fogel [62].

In the meantime, born as a particular case of application of Genetic Algorithms [37], Genetic Programming was introduced and popularized by John Koza [125] and rapidly became the fourth wheel of the Evolutionary Computation truck.

As of today, though those four streams should remain mainly as history, the terms are still used to distinguish some specific aspect of particular EAs, and are sometimes referred to as ’dialects’. The following attempts to summarize the communities and differences between those dialects.

Genetic Algorithm [95, 68]

Representation:	Bit-string representation
Parental Selection	Roulette-wheel (and tournament, later)
Survival Selection	Generational (or Steady-State)
Crossover:	1-point, 2-points, uniform (with given probability P_c)
Mutation:	Bit-flip (applied to every bit with probability p_m)

Evolution strategies [174, 191]

Representation:	Real-valued vectors
Parental Selection	No selection
Survival Selection	$(\mu \overset{,}{+} \lambda)$ strategies
Crossover:	Intermediate (exchange) or arithmetical (linear combination)
Mutation:	(Self-)adaptive Gaussian mutation (see below)

Evolutionary programming [63, 62]

Representation:	Finite State Automata, evolved into using any representation
Parental Selection Survival Selection	No selection (1 parent creates one offspring) Survival tournament
Crossover:	No crossover
Mutation:	Ad hoc – self-adaptive Gaussian mutation for real-valued genotypes

Genetic programming [125, 18]

Representation:	Parse-trees of LISP-like expressions
Selections	"inherited" from GAs, with preference for Steady State with tournaments
Crossover:	Sub-tree exchange
Mutation:	No mutation originally, point mutation, or node- or -leaf-mutations generally

2.2.3 An Adaptive Evolution Strategy: CMA-ES

Evolution Strategies

Evolution Strategies are continuous optimization algorithms, i.e., they work on a real-valued search space, say a subset of $\mathbb{R}^n$, for some integer n.

The main operator of ES is the Gaussian mutation, that generates offspring from a normal distribution centered around the parent. The most general Gaussian distribution in $\mathbb{R}^n$ is the multivariate normal distribution $\mathcal{N}(m, C)$, with mean m and covariance matrix C, a $n \times n$ positive definite matrix. It has the following Probability Distribution Function

$$\Phi(X) = \frac{\exp(-\frac{1}{2}(X-m)^t C^{-1}(X-m))}{\sqrt{(2\pi)^n |C|}}$$

where $|C|$ is the determinant of covariance matrix C.

The mutation of a vector $X \in \mathbb{R}^n$ is generally written as

$$X \rightarrow X + \sigma N(0, C)$$

distinguishing a scaling factor σ, also called the step-size, from the principal directions of the Gaussian distribution, as given by the covariance matrix C.

In the simplest case, C is the identity matrix. In this case, mutating vector X amounts to mutate independently all coordinates of X using a 1D-Gaussian mutation with variance σ^2. In this case, the Gaussian mutation is called isotropic.

Tuning an ES amounts to tuning the step-size and the covariance matrix – or simply tuning the step-size in the isotropic case.

Two simple functions have been considered by early analytical works on ES, corresponding to typical cases of the two extreme situations the algorithm can encounter: when far from the optimum, the isolines of the fitness will locally look like straight lines orthogonal to the direction of the optimum: the fitness function will look linear, with optimum at infinity; when close to the optimum, if the fitness function is smooth enough, it will look like its second order approximation in Taylor expansion, i.e., it will look quadratic. The study of these two very simple cases (the 'linear' function where $f(x) = x_1$, and the 'sphere' function where $f(x) = ||x^2||$) lead to the successive improvements in the way the parameters of the Gaussian mutation were adapted.

Adapting the Step-size

The step-size of an isotropic Gaussian mutation determines the scale of the search. Suppose a 1D situation and a (1+1)-ES (one parent gives birth to one offspring, and the best of both is the next parent) with a fixed step-size σ. In the linear case, the average distance between parent and successful offspring is proportional to σ: it should be increased as much as possible. In the quadratic case, the best precision one can hope is proportional to σ. Those arguments naturally lead to the optimal **adaptive** setting of the step-size: σ should be proportionally resized according to the distance to the optimum. See early work [191], completed with studies of the progress rate [26], and a more recent proof [7]. However, such algorithm is indeed impractical, as the distance to the optimum is unknown!

But further analytical derivations on linear and sphere function lead to the first practical method to adapt the step-size, the so-called one-fifth rule. Indeed, another piece of information is available to the algorithm, namely the proportion of successful mutations, in which the offspring is better than the parent. This success rate does indirectly give information about the step-size: if the success rate over some time window is larger than the success rate when the step-size is optimal (0.2, i.e., one-fifth), the step-size should be increased; on the opposite, if the success rate is smaller than 0.2, the step-size should be decreased. Though formally derived from studies on the sphere function and the corridor function (a bounded linear function), the one-fifth rule has been extrapolated to any function.

There are however many situations where the one-fifth rule fails. Moreover, it does not handle the case of non-isotropic (even if quadratic) functions: there, a full covariance matrix is mandatory. Nevertheless, after having been abandoned when Self-Adaptive ES was proposed (see below), the simplicity of the one-fifth rule makes it appealing when simple adaptation rules are needed. It is for instance used for fast step-size adaptation within the multi-objective CMA-ES algorithm [102].

Self-Adaptive ES

The idea of Self-Adaptive ES (SA-ES) is fairly simple: the parameters of the mutation (both the step-size and the covariance matrix) are attached to each individual, and are adapted by mutation, too. Three variants have been proposed: the isotropic case uses a single step-size per individual; the non-isotropic mutation uses a vector of n “standard deviations” σ_i (the resulting covariance matrix is equivalent to a diagonal matrix C with σ_i^2 on the diagonal); and the correlated mutations attaches a full covariance matrix to each individual. Mutating an individual is then a two-steps process: first, the mutation parameters are themselves mutated; then the design variables are mutated using the new mutation parameters. Details can be found in [15, 12].

The rationale for SA-ES are that even though the selection is based on fitness, an individual with ’poor’ mutation parameters will on average generate offspring that will be overpassed by offspring of individuals with ’correct’ mutation parameters, where ’poor’ and ’correct’ refer to the local fitness landscape (for instance the step-size should be small if the landscape is steep).

SA-ES have often been said to “optimize mutation parameters for free” through the evolution itself. And indeed, SA-ES have long been the state-of-the-art in parametric optimization [15].

But do SA-ES work as expected? Whereas, it has been experimentally demonstrated for the step-size [13, 27], it is not true for the covariance matrix. Indeed, when replacing the sphere function with a quadratic function (min $\frac{1}{2}X^tHX$ for some positive definite matrix H), the mutation should progress slower along the directions of steepest descent of H: the covariance matrix should be proportional to H^{-1}. And there are some evidences that the covariance matrix that is learned by the correlated SA-ES is not any close from the actual inverse of the Hessian [5].

Back to Adaptation: CMA-ES

Another issue when using SA-ES is the slow adaptation of the mutation parameters: even for the simple case of the step-size, if the initial value is not optimal, it takes some time to the algorithm to reach that optimal value and start being efficient.

This observation lead Hansen and Ostermeier to head back to an **adaptive** method for parameter tuning by proposing deterministic schedules to adapt the parameters of the Gaussian mutation. They first addressed the step-size adaptation [89], and later that of the full covariance matrix [87]. The full Covariance Matrix Adaptation (CMA-ES) algorithm was detailed after default values for its parameters had been carefully designed, in [88]. Later, an improvement for the update of the covariance matrix was proposed in [86].

The basic ideas in CMA-ES for step-size adaptation is to observe the previous moves of the algorithm. When successive moves are in collinear directions the step-size should be increased, in order to allow larger steps – and similarly, when the total path-length is very short after several iterations, then the step-size should be decreased, as the algorithm is hovering around the optimum.

Regarding the covariance matrix, the idea of the update rule is to increase the probability in the direction of previous successful moves by adding matrices with the corresponding eigenvectors to the current covariance matrix.

Following Hansen and Ostermeier [88], in the (μ_I, λ)-CMA-ES the λ offspring of generation $g+1$ are computed by

$$x_k^{g+1} = \langle x \rangle_\mu^{(g)} + \sigma^{(g)} \mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g)}, k = 1, \ldots, \lambda,$$

$$\mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g)} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}^{(g)})$$

where

$$\langle x \rangle_\mu^{(g)} = \frac{1}{\mu} \Sigma_{i \in I_{sel}^{(g)}} x_i(g)$$

represents the center of mass of the selected individuals of generation g , and $i \in I_{sel}^{(g)}$ is the set of indices of the selected individuals of generation g with $|i \in I_{sel}^{(g)}| = \mu$. $\sigma^{(g)}$ is the global step size.

The random vectors z_k are $\mathcal{N}(\mathbf{O}, \mathbf{I})$ distributed (n -dimensional normally distributed with expectation zero and the identity covariance matrix) and serve to generate offspring for generation $g+1$. We can calculate their center of mass as:

$$\langle Z \rangle_\mu^{(g+1)} = \frac{1}{\mu} \Sigma_{i \in I_{sel}^{(g)}} z_i(g+1)$$

The covariance matrix $\mathbf{C}^{(g)}$ of the random vectors $\mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g)}$ is a symmetrical positive $n \times n$ -matrix. The improvement equation for the update of the covariance matrix proposed in [86] is as follow:

$$\mathbf{C}^{g+1} = (1-c_{cov}) \cdot \mathbf{C}^{(g)} + c_{cov}(\alpha_{cov} \cdot \mathbf{p}_c^{g+1}(\mathbf{p}_c^{g+1})^{\mathbf{T}} + (1-\alpha_{cov}) \cdot \mathbf{Z}^{g+1})$$

where $\mathbf{p}_c^{g+1}$ is the evolution path, c_{cov} is the learning rate.

For adapting the global step size σ, the evolution path p_σ^{g+1} is computed in analogy to the evolution path $p^(g+1)$, and the new step size is determined by comparing the length of the evolution path to that of a random walk:

$$\sigma^{g+1} = \sigma^g \cdot exp(\frac{1}{d_\sigma} \frac{\| p_\sigma^{g+1} \| - \hat{X}_n}{\hat{X}_n})$$

where $\hat{X}_n = \mathrm{E}[\| \mathscr{N}(\mathbf{0},\mathbf{I}) \|]$ is the expected length of a $(\mathbf{0},\mathbf{I})$, i.e., a normally distributed random vector that would be the average length of a random walk, and $d_\sigma > 1$ is the damping parameter.

It is important to note that CMA-ES is almost a parameterless algorithm. Only the number of offspring λ (that greatly impacts on the actual computational complexity of the algorithm) has to eventually be modified to account for the ruggedness of the fitness landscape at hand. The default value, as set in [88] increases logarithmically with the dimension d of the problem (number of unknown parameters) as $\lambda = 4 + 3\ln(d)$. It should be increased for highly multi-modal objective functions, as demonstrated by the recent restart-CMA-ES [9]. Furthermore, some particular landscapes that are only mildly multi-modal, like Whitley's funnel landscapes [139], cannot be successfully solved by enlarging the population size: such types of landscapes motivated the bi-pop-CMA-ES [83].

CMA-ES can be considered the state-of-the-art in continuous optimization today. The restart-CMA-ES was demonstrated to outperform most other stochastic algorithms for parametric optimization (e.g. variants of PSO, DE, EDAs), as witnessed by its success in the 2005 contest that took place at CEC'2005 [51]. More recently, some comparative results [10] obtained for the 2009 GECCO Workshop on Black-Box Optimization Benchmark [50] demonstrated that the bi-pop-CMA-ES also outperforms the best-performing methods in classical numerical optimization (e.g. the standard Levenberg-Marquard algorithm, or the very recent NEWUOA algorithm by Powell [170, 171]).

2.2.4 Applications

It is widely acknowledged that in Evolutionary Computation, Theory lags far behind practice. Indeed, lessons from successful applications are one of the main driving forces of EAs research today. Several edited books are devoted to Applications of EAs (see e.g. the recent [231]), and almost every event dedicated to EAs has its own Special Session dedicated to real-world applications.

Application areas can be distinguished according to the type of search space they involve.

Regarding **continuous optimization**, Section 2.2.3 has introduced in detail the CMA-ES algorithm as the state-of-the-art algorithm for optimization of real-variables. An impressive list of applications of CMA-ES is maintained on N. Hansen's Web page [84].

Another niche of EAs is **Multi-objective optimization**. Basically, Multi-Objective Evolutionary Algorithms (MOEAs) are the only algorithms to-date that can produce a set of best possible compromise (the Pareto set). MOEAs use the same variation operators than standard EAs, but the Darwinian components are modified to take into account the multi-valued fitness [40, 35]. Though prominent application results have been obtained in the area of multi-objective coutinuous optimization area (see e.g. [159, 160]), the methods can be used with any search space [179, 80].

Combinatorial problems are another area where EAs have proven to be very efficient. However, as far as benchmark problems are concerned, it is commonly acknowledged that EAs alone cannot compete with OR methods (see e.g., the poor results in [16]). However, in the last 15 years, hybrid algorithms, also termed Memetic Algorithms, coupling traditional OR methods with EAs, have obtained the best-so-far results on a number of such benchmark problems (e.g. from [64] to [149]).

However, for real-world combinatorial problems, "pure" OR heuristics generally don't directly apply, and OR methods have to take into account problem specificities. This is where EA flexibility is crucial: some specific EAs, carefully tuned to the problem at hand, have been very successful solving real-world combinatorial problems, as for instance in the broad area of scheduling [164, 193].

When it comes to **mixed search spaces** (involving a fixed number of variables that can be floating points, integers or discrete), again the flexibility of EAs allow to tailor them to the problem at hand. For instance, constructing variation operators for mixed individuals is straightforward, and can simply be done by applying to each variable some variation operator defined for its type. Many problem have been easily handled that way (see e.g. the optical filter optimization in [14, 145]). Furthermore, some platforms now exist that help the non-EC expert to implement generic EAs for a given problem involving different types of mixed search spaces [36].

Ultimately, the ability of EAs to handle almost any search space can allow engineers and/or artists to unveil their wildest ideas in Design. Indeed, the idea of **component-based representations** can boost innovation in structural design [66, 81], architecture [178] as well as in many other areas including art [25, 24]. Furthermore, the recently emerged area of **developmental representations** [76, 126] might lead to tremendous applications. The idea is to optimize the program that builds the solution, rather than the solution itself. The pioneering work of Gruau (see Section 2.3.2) was used to design analog circuits [126] – though exploring a huge search space (a space of programs) implies a heavy computational cost. However, it seems that those approaches can bring a breakthrough in Evolutionary Design [201].

2.3 Evolving Artificial Neural Network

This Section surveys existing work that use Evolutionary Computation to optimize Neural Networks for a given task (aka Neuro-Evolution), as such context will be that of all original works presented in this dissertation. Two types of approaches will be considered, whereas only the weights of a given NN are being optimized, or in case the topology of the NN is being optimized as well. An orthogonal classification regards the nature of the task at hand, whether supervised, reinforcement or unsupervised (see Section 2.1.2). More details can be found about the early work on Neuro-Evolution in two well-known surveys that have been written by Schaffer, Whitley, and Eshelman [184] in 1992 and by Yao [230] in 1999.

2.3.1 Evolving Connection Weights

In the 80s, great successes of NNs were obtained in the supervised learning framework (see Section 2.1.2) using Back-Propagation (BP) algorithm to optimize the connections weights [128, 56]. However, the BP algorithm also showed a number of limitations. It demonstrated slow convergence for large-scale NNs, and too frequent premature convergence to some local minima close to the starting point, thus missing the global minimum [208, 226]. A workaround was to use different initializations, but without much guarantees of convergence. Furthermore, because it requires the differentiability of the objective function, BP algorithm is unable to handle discontinuous neural transfer functions (e.g., the Heaviside function – Unit Step fuction).

On the other hand, in unsupervised or reinforcement learning context, where direct gradient information is either unavailable, or costly to obtain, no gradient-based method is available, so even learning the weights of a given NN was considered a difficult task. In that context, because of the lack of a general theory and also because the optimal output is unknown, the traditional manual approaches have been designed by the experts, e.g. Hopfield networks [96] and Adaptive Resonance Theory (ART) [75]. But such methods have two shortcomings: their design has many degrees of freedom that must be fixed manually; furthermore, they have to be tuned anew for each given task, and the results hardly generalize from one task to another.

Therefore, even though ANNs were considered a good choice for unsupervised and reinforcement learning, it seemed that a new approach was required to actually perform the optimization of the neural network, starting with the optimization of the weights of a NNs with a given topology.

At the same period, Evolutionary Algorithms, and more precisely Genetic Algorithms, started to encounter some success in many areas. Because of the flexibility of Evolutionary Algorithm, and in particular because they do not require derivative information, the initial work of evolving ANNs began at this stage [152, 226, 39] with the easiest problem, that of optimizing the weights of a given NN with fixed topology.

The research of what was to become known as Neuro-Evolution hence started with optimization of the connection weights of single-layer or multi-layer feed-forward neural networks (aka perceptrons). Starting with the input and output sizes, the programmer has to manually determine the number of hidden layers, and the number of neurons in each layer. Such topology remains fixed during the whole learning phase, and the genotype is then simply the vector of all connection weights. Depending on the learning framework, any optimization algorithm can then be used, as the problem then amounts to standard continuous parameter optimization.

Early attempts dealt with supervised learning: In 1989, Montana and Davis [152] learned the weights of a simple perceptron for an ocean echo detection, and achieved better results than the back propagation algorithm; also the other works such as Caudell et al. [34]; Whitley et al. [226]

However, because there didn't exist any other unanimously recognized method for reinforcement learning tasks, many works in Neuro-Evolution have been devoted to such context, from the pioneering work of DeGaris [39] where the task was to control some "walking sticks" and Whitley et al. [225] that first introduced the pole balancing problems in the EC community (see Section 4.4.1), to handling other classical benchmark problems like lunar lander [176], truck backer-upper [190], to recent work using standard CMA-ES and solving the double-pole balancing problem using a perceptron with very few neurons [101].

Another distinction can be made here whether the weights are encoded as binary strings or as real numbers. Most of the early works used binary string encoding, that was considered in the GA community almost as a universal encoding. Hence lots of researchers followed this approach at that time [224, 34, 226, 39]. In this method, each connection weight is represented by a binary string of given length, and all these strings are concatenated to form the chromosome. Classical GA variation operators can then be applied directly (such as single-point or uniform crossover). Binary representation is also conducive to the realization of digital hardware.

However, the precision of the encoding then depends on the string length that is chosen to encode each weight, and a compromise has to be made between the precision and the total chromosome length. If the length is too short, the connection weights cannot be optimized with sufficient precision; On the other hand, when the code length is too large, the optimization process might become too slow to converge, degrading the efficiency of the whole process. There are however well-known exceptions, such as the work in Evolutionary Robotics performed by Floreano and co-authors [59, 158] that repeatedly used coarse binary encoding (8 bits per weight) to evolve successfully robot controllers for various tasks.

Using real encoding (and appropriate variation operators) is a way to overcome this difficulty. Used by Montana and Davis as early as 1989 [152], and followed by Whitley et al. [225] for reinforcement learning tasks, it became more and more popular in the 90s with the growing popularity of Evolution Strategies (see e.g., [176, 190], and more recently [101]). Our work in Chapter 4 builds on those ideas.

An issue that has been raised about the evolutionary learning of the weights of a given NN is the "structural/functional mapping" problem [225] (aka "competing convention" problem): The mapping from genotype to phenotype is here a many-to-one mapping, because two NNs that only differ by the order of the hidden neurons are de facto identical, whereas the corresponding chromosomes will be completely different. In particular, any crossover operator (and for any representation of the weights) might be very inefficient. However, in practice, and at least for not too large populations, it seems that this problem is not an issue, as evolution rapidly chooses one of the possible orderings of the neurons, and hardly has to crossover NNs that only differ by a permutation of the hidden neurons [82].

On the other hand, using evolutionary algorithm allows the programmer to optimize not only the connection weights, but also the parameters of the transfer functions, sometimes leading to improved results, as advocated in [189]. The work presented in Chapter 4 is another demonstration of this idea.

However, even though multi-layer perceptrons are universal approximators [97], it is well-known that the topology of a NNs is an important issue for large-scape applications. For instance, many man-years of effort have been devoted to building the topology of the well-known digit recognition NN for the US Post [129, 31]. And again, thanks to their flexibility, EAs are perfect optimizer candidates for the task of optimizing the topology of NNs for a given task.

2.3.2 Evolving Network Topologies

Tuning the topology of Neural Networks, even simple feed-forward networks like perceptrons, has received a lot of attention in the Machine Learning community. Both bottom-up and top-down approaches have been explored: Cascade-correlation algorithms for instance iteratively add neurons to a very specific architecture [55], while pruning methods start with a very large network and gradually remove neurons, using from brute force [222] to iterative algorithms [33]. Hence, quite naturally, early works in Neuro-Evolution trying to optimize the topology of NNs followed similar paths [90, 226, 206]. But here again, the flexibility of EAs rapidly lead researchers to generalize such methods. First, there is no need any more to work either bottom-up or top-down, as the complete connectivity matrix can be optimize, adding and removing either neurons or connections, optimizing the weights either using traditional BP algorithm, in case of supervised learning, or within the EAs itself, in case of reinforcement learning – and here again, not only the weights but any parameter of the transfer function can be optimized [54]. Those methods (see [230] for a more complete surveys, as most of these methods were published prior to 1999), that use different representations of the topology to directly manipulate and transform it through the variation operators, are termed direct representation methods.

However, the poor scaling-up behavior with respect to the size of the problem of direct encoding lead to a change of paradigm, that resulted in the indirect methods, starting with developmental methods, where evolution is concerned with evolving some program such that, when the program is executed on some pre-defined embryo, the result is a fully operational Neural Network. This approach has been illustrated by the seminal works of Kitano [119] and Gruau [77]. However, such developmental approaches have not met their expectations, and recent trends use other types of indirect representation, going back to biological inspiration, and borrowing some basic principles to Gene Regulatory Networks or Protein Interaction Networks to define artificial networks [58, 132, 45, 157].

We will now detail in turn the some recent direct representation methods, ultimately focusing on NEAT [204], that is today considered the state-of-the-art for the evolution of small NNs. Interestingly, NEAT uses an approach very close from a pure bottom-up approach, in that it keeps adding neurons and connections to a minimal embryo NN.

We will then survey some indirect representations, from the seminal work of Gruau to the recent approaches inspired by biological networks. Through this presentation, we will de facto survey part of the research progress in Neuro-Evolution in the last decade (i.e. posterior to Yao's 1999 survey [230]).

Some Direct Representations: from SANE to NEAT

An important issue when using direct representations to evolve both the weights and the topology of NNs is the following: when the topology is modified, the performance of the resulting new topology is generally rather low until the weights have been adjusted to the new topology. Several methods have been proposed in the recent years in the realm of direct representations for evolving NNs try to address this issue, beside taking into account the scaling-up issue.

Coevolution Methods

In 1997 Moriarty and Miikkulainen proposed a co-evolutionary model named Symbiotic Adaptive Neuro-Evolution (SANE) [154]. The basic idea in SANE is to evolve two populations. The first population is made of neurons, and the second population is made of network structures with one hidden layer (aka blueprints) pointing to neurons from the first population for the hidden neurons. The blueprints are instantiated, then evaluated on the task at hand; the neurons are evaluated based on how good are the networks they have participated to.

SANE obtained good results at diverse reinforcement learning tasks: First, because new topologies are using neurons that are proved good with the parent topologies, preventing too poor performance of offspring; Second, because the algorithm also implicitly maintains some diversity at the level of the neurons: no neuron can overtake the whole population as networks using mostly copies of the same neuron are likely to perform poorly. However, SANE proved poor at evolving recurrent NN, probably because in recurrent nets the dynamics of the neurons are much more dependent on the other neurons it is connected to – evolving neurons separately becomes unviable.

In the following years, Miikkulainen and his students modified the SANE model into the Enforced Sub-Populations (ESP) model [69]. Like SANE, ESP evolves the topology of NNs with one hidden layer. However, the number of hidden neurons must be specified, and ESP evolves as many populations of neurons. Complete Neural Networks are built by randomly choosing one neuron from each population, repeatedly until each neuron has been used on average a given number of times (e.g. 10). Each neuron is evaluated based on how good on average the networks it has been part of performed. Because the clustering of neurons is explicit (neurons with different sub-tasks belong to different populations), crossover only recombines neurons that are already specialized for the same sub-task. ESP hence obtained better results than SANE on the same reinforcement tasks. Furthermore, recurrent NN can be evolved with ESP, by including the weights of the recurrent connections into the genotypes of the neurons. However, the main limitation of ESP is that it can only evolve one-layer architectures.

Neuroevolution of Augmenting Topologies: NEAT

Building upon SANE and ESP, Miikkulainen and his student Stanley proposed NEAT (*Neuroevolution of Augmenting Topologies*) [204, 203], a new direct representation to evolve any feed-forward or recurrent NN architecture in order to address the following issues:

1. How to optimize both the weights and the topology, i.e. explore a huge search space efficiently?
2. How can crossover be implemented when structures do not match?
3. How can innovative topologies survive while having initially low fitness?

The original features of NEAT are

- The bottom-up approach to network topology: the optimization process starts with a minimal NN (one neuron per input, all being connected to the output neurons), and can only add connections and neurons.
- The representation, that can handle any type of connectivity, while being easily and smoothly evolvable through variation operators. Figure 2.5 shows an example of genotype to phenotype mapping (from [204]). The Node Genes describe the neurons and their role in the network (input, output, or hidden), and the Connection Genes describe the connections between the neurons: the input and output neurons, the corresponding weight, an Enabled/Disabled flag, and an important and original feature, the innovation number.
- The innovation number stores for each gene the moment it appeared in the history of evolution (a global counter is incremented every time a new gene is created). It allows the crossover operator to answer issue 2 above: when two individuals have to be recombined, they are first aligned with respect to innovation numbers, and only genes with matching innovation numbers are exchanged. They are genes that are descents of the same initial gene, and

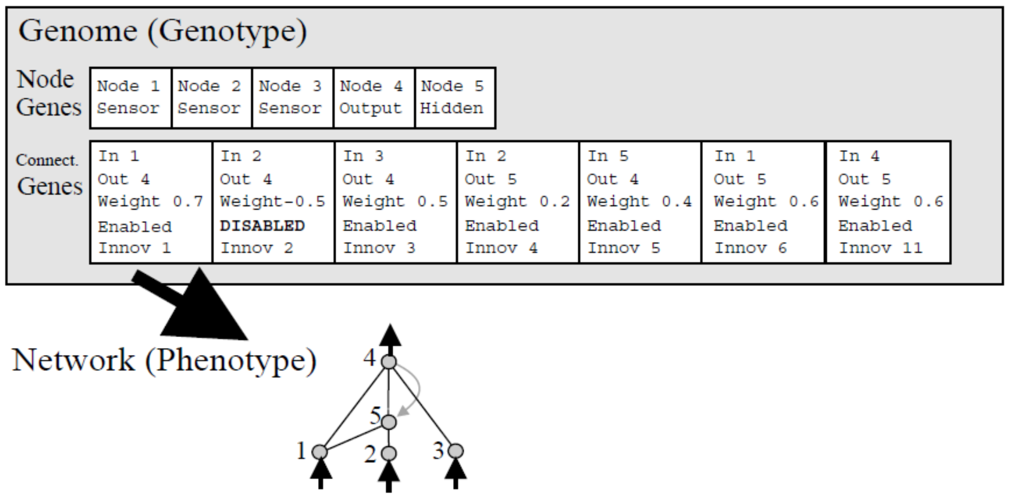

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 2.5: This example (from [204]) shows a genotype to phenotype mapping used by NEAT. There are 3 input, one hidden, and one output nodes, and seven connections, one of which is recurrent. The second gene is disabled, so the connection that it specifies (between nodes 2 and 4) is not expressed in the phenotype

hence should play a similar role in the structure, as only weight mutation might have happened since they were separated).

- Structural mutation operators that modify the network as little as possible: Add Connection mutation and Add Node mutation are described in Figure 2.6. Note that weight mutation adds some random Gaussian noise with fixed standard deviation.

- Explicit fitness sharing mechanism, based on some distance defined using gene alignments based on the innovation numbers: this prevents innovative structures to disappear immediately because of their poor performance, thus answering issue 3 above.

Because of its praised successes, there are already a series of different versions of NEAT, including C++ (Linux, Windows, Real-time), JAVA, Delphi, Matlab, C♯. More details as well as news about NEAT and the growing NEAT community can be found in the NEAT dedicated Web site[2] maintenaned by K. Stanley.

NEAT is indeed considered today the state-of-the-art of methods based on direct representations to evolve both the topology and the weights of a NN. It is hence the baseline algorithm to which any new approach that aims at generating

[2]http://www.cs.ucf.edu/ kstanley/neat.html

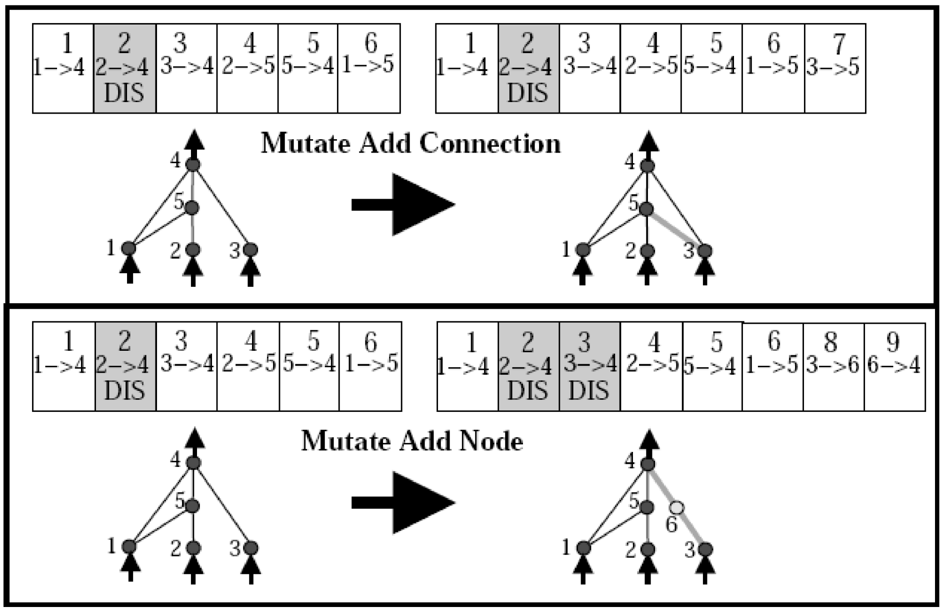


Figure 2.6: Structural mutations in NEAT (from [204]). In the Add Connection mutation, two previously unconnected neurons are chosen randomly and connected, and the connection is assigned a small random weight; In the Add Node mutation, an existing connection is chosen and replaced by a new neuron, that received two connections, with the two neurons which were previously connected by the disabled connection. The weight of input connection of the new neuron is set to 1 and the weight of output connection is set to the same weight as the old connection, for minimal destructive effect.

NNs should be compared. However, it also suffers from a severe drawback: it has a lot of parameters that the user needs to tune, and remains rather difficult to master for a completely new task. Furthermore, whereas, when correctly tuned, it can evolve diverse structures (both feedforward and recurrent) of reasonable size (a few dozens of neurons), it does not scale-up very well - in particular it does not address modularity or code-reuse issues, that seem mandatory to tackle large-scale problems.

Indirect representations have been proposed in order to address these crucial issues, and next Sections will detail the most popular ones, including one of the early ones, Hyper-NEAT, a follow-up of NEAT that tries to address the short-comings of NEAT, and some recent representations borrowing ideas to biological networks.

Developmental Representations

There has been several proposal for developmental representations, i.e. representations of a program that actually builds the solution to the problem at hand rather than directly evolving the solution itself (such representations are not specific to Neuro-Evolution, see e.g., [126]). The works by Kitano [119], using some grammar-based representation close to L-systems, and Gruau [77, 76] using Cellular Encoding, a representation close to the parse trees of Genetic Programming have pioneered this research direction. We have chosen to detail the latter here, as it seems more likely to be generalized to other tasks than the former. Finally, we will briefly mention the recent HyperNEAT [202], that can be viewed as a developmental representation even though it is based on different paradigm than the previous approaches.

Cellular Encoding

Cellular Encoding is one of the very first instances of developmental representation. Cellular Encoding is uses a langauge describing local graph transformations, in order to control the division of cells that grow an artificial neural network [77, 76].

In Cellular Encoding, individuals are represented as grammar trees with ordered branches. The nodes of the grammar tree are labeled with names of program symbols. The neural network starts with a single 'embryo' cell, and performs a sequential development process by reading the grammar tree and applying the operation described in the current node of the tree to the current cell it points to. Basic operations include Parallel Division and Sequential Division of the cell. When a cell meets the End instruction, the cell stops its development process and becomes a neuron. Optimization proceeds on the trees, using crossover and mutation operators that are very similar to those independently proposed for Genetic

Programming [125].

Early work with Cellular Encoding addressed only boolean neural networks [76]. It was later applied to NNs with real valued weights [227, 78]. In 1996, Gruau, Whitley, and Pyeatt solved the double poles balancing without velocity problem with Cellular Encoding [78] (see also Section 4.4.1).

Importantly, Cellular Encoding addressed the scalability issue by adding a Recursion node to the initial representation, allowing to tackle some amount of modularity. The proof of concept of increased scalability was made by evolving solutions to the multiplexer problem [76]. It was later demonstrated on a real-world problem by Khozabadjian [121] that successfully evolved a controller for an hexapod robot.

However, it seems that the evolvability of Cellular Encoding resulted in a computationally heavy algorithm, and to the best of our knowledge, it has not been used ever since. Note that such very heavy costs were also reported for experiments using similar ideas borrowing to embryogenies made by J. Koza [126].

HyperNEAT

A completely different approach, that can nevertheless be viewed as some developmental representation for Neuro-Evolution, is that of Hyper-NEAT [202]. Hyper-NEAT builds on NEAT by using the idea of Compositional Pattern Producing Networks (CPPNs) proposed by Stanley [201] to design patterns with regularities. The goal of Hyper-NEAT is to design large NNs where each neuron is materialized as a point on a 2D grid. The basic idea is then to build (using NEAT) a top-level Neural Network that will take as input two pairs of coordinates (those of two neurons), and output one real value, that of the weight between both neurons. Doing so, Hyper-NEAT is able to take into account the geometry of the target network (e.g., to design symmetric networks). It also addresses the scalability issue, as the optimization is always made on the small NNs (with 4 inputs and 1 output). For instance, Stanley and co-authors report some success building a visual discrimination network containing over eight millions connections [202].

Bio-inspired Indirect representations

Several models of Genetic Regulatory Networks (GRNs) have been proposed as a basis for an indirect representation for Neuro-Evolutoin. They differ in several ways, and actually model only parts of what is known about GRNs. This Section will quickly detail 3 of them, Analog Genetic Encoding [58, 146], RBF-Gene [131, 132], and finally the so-called Banzhaf model [17, 127] that, though not applied to Neuro-Evolution yet, seem relevant for the perspectives it opens, relative to the work presented in this dissertation.

The concept of Gene Regulatory Network (GRN) comes from the Biological Sciences. The expression of the genes in a genome is regulated by special proteins produced by other genes which can enhance or inhibit the production of their target protein, aka Transcription Factors. And the GRN describe the interaction between the genes and the Transcription Factors.

Analog Genetic Encoding

The Analog Genetic Encoding (AGE) was initially proposed for the synthesis of analog circuits [58, 146] (hence its name), and the same ideas were used later for Neuro-Evolution [45]. The neurons in Neuro-Evolution, or the components in the analog circuit application, are referred to as ’devices’ in AGE.

The genome of AGE is constituted by a sequence of characters from a finite genetic alphabet (eg. “A” - “Z”). Each device is encoded by a specific ’start’ token (e.g. “NE”) followed by a number of terminal sequences (e.g. “TE”). In the case of neuro-evolution, the substring between the start sequence and the first terminal sequence is the “input” string for the neuron, and that between the first and the second terminal sequence is the “output” string for the neuron.

Such genome encodes a fully connected Neural Network, and each connection weight between 2 neurons is the result of some alignment score between the output string of the first neuron and the input string of the second neuron. This alignment score in an integer value that depends on a user-defined matrix giving some score to any pair of characters. This integer value is then bounded (e.g. in $[1, 37]$), and converted into a floating point number by a logarithmic mapping into an interval that determines the minimum and maximum precision (e.g. $[0.001, 1000]$).

Similarly, all neurons are connected to the input and output of the network through the same mechanism, where the “output string” of every input of the network and the “input string” of every output of the network are predefined strings.

The variation operator used for evolution are [45]: Character deletion, insertion, and substitution, Fragment deletion, transposition and duplication, Device insertion (a random complete neuron is inserted), Homologous Crossover (crossover based on syntactic alignment of both parent genomes), and Genome duplication.

The initial population is created by generating random genomes in which a given number of different neurons with random terminal sequences are inserted. Such bootstrap procedure is necessary in order to ensure a minimum number of neurons.

In [45], AGE for Neuro-Evolution is applied to the double-pole balancing

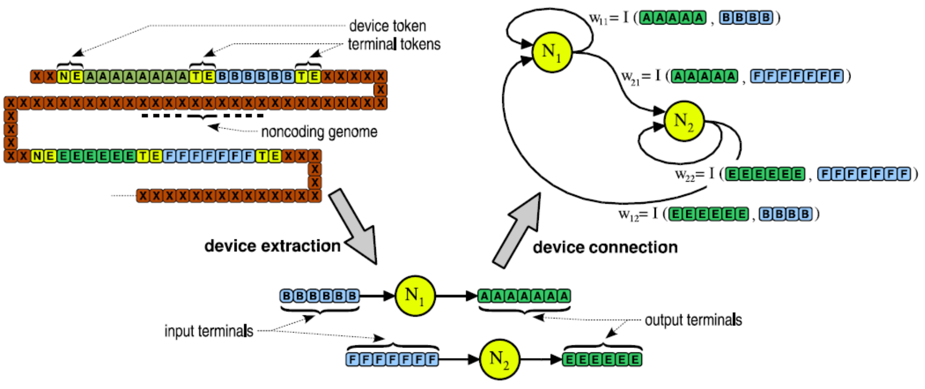


Figure 2.7: Sample genome in AGE (from [45]) (a 'device' is here a neuron). One device is extracted for each 'start' token ("NE") that is followed by the required number of 'end' tokens ("TE") – here 2 substrings are needed, the input and the output string for each device. The weights are then computed, for the inputs/outputs of the network, and for all connections between pairs of devices.

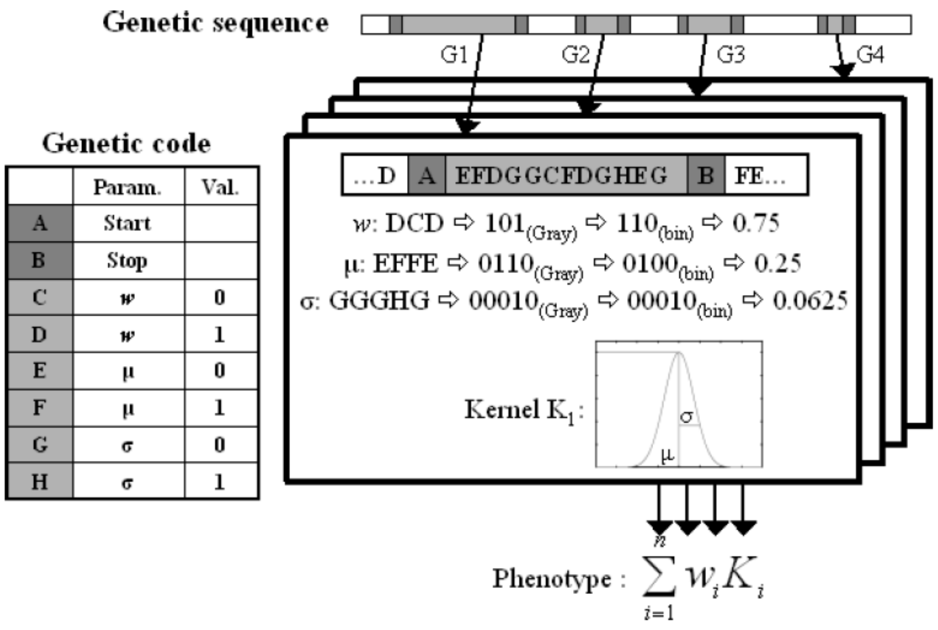
problem. The NN model is a Continuous-Time Recurrent Neural Net (CTRNN): the neurons are sigmoidal neurons, but their dynamic is governed by a continuous differential equation. The results presented in [45] are better than previously published results for Neuro-Evolution methods (see Section 4.4.1).

However, there are a number of design decisions that have been made in order to obtain those results that do not seem to be clearly straightforward: the substitution matrix used to compute the alignment score, the mapping from the alignment score into a floating point value, the sequences for the inputs and outputs of the networks (they are made of identical characters, by why are they of size 7, 5, 6 and 4 for respectively θ_1, θ_2, x and the bias?), not to mention the different rates for the variation operators. The authors are well aware of the problem, as they suggest that many of those parameters can be encoded in the genome, in order to be self-adapted by evolution – but no analysis is given about this part of the work.

RBF-Gene

In 2004, Lefort and co-authors proposed RBF-Gene [131, 132], yet another indirect encoding to evolve both the structure and the parameters (weights and parameters of the transfer functions) of an RBF netowork, i.e. a NNs with one hidden layer and Radial Basis Functions as transfer functions. The model is loosely inspired from genetic regulatory network: each gene encodes one neuron: one RBF, and the output weights (one per output of the network). Each RBF requires a mean vector (one coordinate per input variable) and a standard deviation. Each gene is thus defined by $n+m+1$ real values, where n is the input dimension, and m the number of outputs.

The representation is a variable length string built over an alphabet. Each gene is the sub-string between two special characters of the alphabet, the 'start' and 'end' characters (and there is hence a variable number of genes/neurons). Within a given gene, each required parameter is encoded by 2 characters, one representing a 0 and one representing a 1. The alphabet size is hence $2+2(n+m+1)$. Each parameter is encoded as a binary sequence defined by the values of the corresponding characters within the gene. This binary sequence is decoded as the Gray value of an integer and transformed into a real number. Figure 2.8 (from [132]) shows an example of decoding of such a genome.


Genetic sequence
G1
G2
G3
G4
Genetic code
Param. Val.
A Start
B Stop
C w 0
D w 1
E μ 0
F μ 1
G σ 0
H σ 1
...D A EFDGGCFDGHEG B FE...
w: DCD ⇨ 101(Gray) ⇨ 110(bin) ⇨ 0.75
μ: EFFE ⇨ 0110(Gray) ⇨ 0100(bin) ⇨ 0.25
σ: GGGHG ⇨ 00010(Gray) ⇨ 00010(bin) ⇨ 0.0625
Kernel K1 :
σ
μ
Phenotype : $\sum_{i=1}^{n} w_i K_i$

s

Figure 2.8: A simple example of the mapping from the chromosome to the neurons. One input value (n = 1) and one output value (m = 1) leads a genetic code of 2 + 2(1 + 1 + 1) = 8 letters [132]

The evolution is done using variation operators that are adapted from their biological counterparts: beside the 'traditional' crossover (for variable-length strings) and mutation (modification of a character), the authors have used both small (local) and large duplication, translocation and deletion operators.

Good results are obtained on some difficult regression problem [132]. However, this representation is limited to RBF networks: even though they are universal approximators, they have been hardly used for reinforcement learning task. Moreover, this representation does not seem to scale up well with the dimension of the input space, as the size of the alphabet increase linearly with the input dimension.

The promises of GRN-based Representations: Banzhaf model

In 2003, Wolfgang Banzhaf proposed a Artificial Gene Regulatory Networks in terms of computer science [17, 127]. In this model, all genes are both protein producers and Transcription Factors that regulate the production of all other genes (including their own) instantaneously.

A genome is represented as a bit string. As in AGE and RBF-Gene, a specific 32-bit sequence is used as 'promoter site', i.e. to identify the starting location of a gene (see Figure 2.9 for an illustration). Upstream from the promoter site, two 32-bits sequences represent one enhancer and one inhibitor sites for this gene. The 160 bits (5×32) downstream represent the gene information, encoding the type of protein that the gene will be responsible for creating. A protein is a 32-bits sequence, each bit is the result of a majority vote between the 5 corresponding bits at the same position in the 5×32 bits representing the gene information.

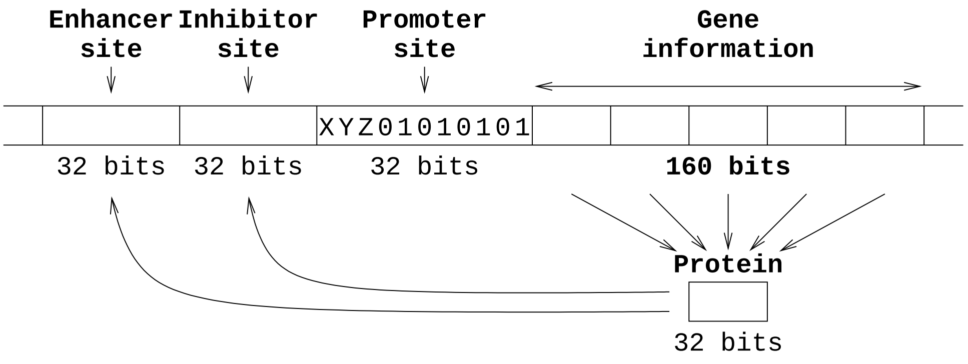


Figure 2.9: A gene of the Artificial Gene Regulatory Networks model proposed by Banzhaf [17]

All proteins regulate the production of all genes: the strength of the regulation (that is also the weight of the connection between both genes) is computing by OR-ing the protein code and the enhancer site (resp. the inhibitor site) of the corresponding gene. Hence the regulation network is a priori a fully connected network, i.e. without any possibly interesting topology. However, by putting a threshold on the regulation term, some connections will disappear, and interesting topologies do appear. Nicolau and Schoenauer has demonstrated [156, 157] that the evolution using quite standard variation operators, of populations of such GRNs that have been initialized using the duplication-divergence method proposed in the initial model by Banzhaf [17], can be lead toward networks exhibiting very specific topological statistical properties, such as small-world or scale-free characteristics (see next Section 2.4).

Discussion

All representations, whether direct or indirect, that have been used for evolving both the topology and the weights of NNs are variable length representations. They are all able to generate a variable number of neurons. However, only the bio-inspired representations have non-coding segments.

Regarding the topologies they can generate, though both AGE and RBF-Gene can indeed generate different topologies, those topologies remain within a very restricted set, and only differ by the number of genes: AGE creates fully connected NNs, and RBF-Gene generates 3-layer RBF networks. In that sense, NEAT, and, of course, the developmental methods, can create more diverse types of topologies - and so does the Banzhaf model thanks to the threshold trick that removes many connections. Note that it could be possible to apply this trick to AGE too.

Furthermore, only the developmental representations are actually able to scale-up to huge numbers of neurons without requiring a similar increase of the representation size. The direct representations clearly need to manipulate (create, tune, connect) each gene one by one: this can be an advantage when only a small network is needed, but will not be sufficient for large networks. On the other hand, the GRN-inspired representations, though they need to express all genes of the network, can be considered as intermediate with respect to the scaling-up issue: because they use global variation operators like duplication, they can succeed in generating networks of moderate size, up to a few hundreds neurons [156, 157].

Regarding the evolution of the weights, in AGE and in the Banzhaf model, the weights are based on some distance between labels/affinity sites, while in direct representations, and in RBF-Gene, they are directly encoded as binary strings. The consequence of the indirect encoding of the former methods is that by modifying one of the labels/sites, all weights issued from one neuron are modified at once, strengthening those connecting a certain type of neurons, and weakening others. A direct encoding of the weights requires, for the same result, that several weights are modified in the same direction, a very unlikely event. Whether one approach or another is best for evolution is however unclear: being able to modify all weights of a given neuron in a single mutation can indeed help for the coarse adjustment of the NNs, as witnessed by the improvements brought by adding the coefficient of the sigmoid function to the genome even when only optimizing the weights [189, 54]; On the other hand, fine-tuning a Neural Network often goes through fine-tuning its weights independently.

In developmental representations, the causality for the evolution of the weights is less clear: in Cellular Encoding and other grammar-based representations, the weights have to be described directly, but will be used whenever some code re-use happens in the morphogenetic process. In Hyper-NEAT, the weights are the result of the application of the top-level neural net, a complex process not easily related to the variation operators acting over this network.

In summary, there is of course no method that clearly outperforms all others. Simple approaches, i.e. evolving the weights of a robust topology (e.g. multi-layer perceptron) with a robust optimizer (e.g. CMA-ES) are clearly sufficient for many problems (e.g. as witnessed by Igel's results on the pole-balancing problem [101] – see also Section 4.4.1). But the influence of given characteristics of the topology like statistical measures that have recently been proposed for complex networks, has not been systematically studied – and this is part of what has motivated the present work (see also Section 2.5).

2.4 Topology of Complex Networks

Up to recent years, the major types of studied network topology were totally random networks and regular networks. Since the end of the 1990's, several real life networks have started to be known with enough precision so as to analyze them in quantified details. The results were that real life networks have actually several properties that exclude them from the regular or totally random classes. The so-called complex networks field has mainly been created as a response to this issue. In the last decade, complex network topologies, e.g. small-world or scale-free, have attracted a great amount of interests (for a review, see [28]). In this section we briefly introduce some basic concepts.

2.4.1 Small World Topology

The small-world phenomenon originates from social network studies. According to this theory, each person can be seen as a node of a graph, and connected with a large number of paths with the others. The connections between nodes mean that the peoples know each other. The hypothesis is that each person needs only a few intermediaries (an average of 6) to establish a link with anybody else in the world. It is an issue of sociology, mathematics and computer science. The hypothesis is suggested by the social psychologist Milgram [141] for an experiment in the 1960's: "Track the shortest path of the social network in United States". He asked each participant to send a letter to a "target person" who lives beside Boston, and each participant can only transmit the letter to another person he or she knows. Milgram found that on average a complete chain passed through 6 persons only (note that this statistics is biased by the fact that the large number of unsuccessful trials, ie broken chains, was not factored in.)

Similarly, in 2007, Microsoft researchers Leskovec and Horvitz [133] filtered the MSN messages of a single month of the year 2006. Using 30 billion communication messages between 180 million users in this month, they found that any user can be connected with anyone else among the 180 million users through an average of 6.6 people. Up to 87% of users can be connected within a connection trough 7 people.

In 1998, Watts and Strogatz [220] gave a formal model of such a small-world network. According to this famous algorithm, small-world networks are intermediate topologies between regular and random ones (see figure 2.10). Their properties are most often quantified by two key parameters [207]: the clustering coefficient ($< C >$) and mean-shortest path (λ). The clustering coefficient quantifies the extent to which the neighbors of a given network node (i.e. the nodes to which it is connected) are, on average, interconnected together. It reflects the network capacities of local information transmission. The graph distance between two nodes of the network is the smallest number of links one has to travel to go from one node to the other. The *MSP* is the average graph distance of the network and indicates the capacities of long distance information transmission.

Figure 2.10 illustrates an example of small-world network formation according to Watts and Strogatz. Starting from a regular network (a ring in the figure, where each node is connected to its four nearest neighbors), one re-wires each link with (uniform) probability p. A regular network ($p = 0$) has a high clustering coefficient but its *MSP* is long. At the other extreme, totally random networks, obtained with $p = 1$, display a small clustering coefficient but a short *MSP*. For small-to-intermediate values of p (for instance $p \in [0.004, 0.100]$ for rings with 1000 nodes and 10 links per node [220]), the obtained networks preserve a high clustering coefficient while their *MSP* is already small (Figure 2.11). Such networks have optimal information transmission capacities, both at the local and global scales [120] and are called small-world networks. Many "real-world" networks, either technological (the Internet, electric power grids), social (collaboration networks) or biological (neural networks, protein or gene networks), have been found to display such a small-world topology [28].

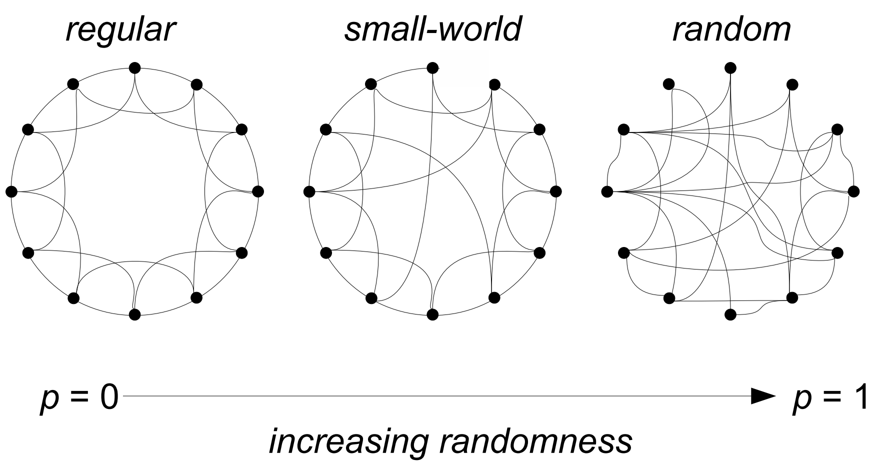


Figure 2.10: Small-world networks according to Watts and Strogatz. Starting with a regular ring network ($p = 0$, left), and each link is rewired to a randomly-chosen destination node with probability p. When $p = 1$ (right), the network is a purely random one. At small-to-intermediate p values (depending on the network size), the network displays small-world properties (center). Adapted from [220].

2.4.2 Scale Free Topology

Not all characteristics of real-world networks can be captured by the small-world networks. For instance, real-world networks often possess some highly-connected nodes ("hubs"), connecting almost disconnected sub-networks. Besides small-world networks, other types of networks, such as the scale-free networks [19], have been studied. In scale-free networks, the degree distribution $P(k)$ (that gives the probability that a node, chosen at random in the network, connects with k other nodes) follows a power law relationship: $P(k) \sim k^{-\gamma}$ (it usually decays much quicker, e.g. exponentially, in totally random networks). This power law indicates that in such networks, there exists a large number of nodes which have only a few connections (eg. the computers in internet often connect with a few neighbour in the same sub-net), but there is also a significant number of nodes with a huge number of connections (eg. the root routers of internet which are connected with almost every computer). The “preferential attachment algorithm” [19] can be used to build such topologies, reflecting also the dynamical aspect of those network, whose size can increase over time. The “Preferential attachment algorithm” starts with a small number of nodes connected by an edge. At each step of the algorithm, a new node is added to the network. To connect it to the existing network, one picks a node at random in the existing network, but with a bias that is proportional to the number of connections of this node. As more connected nodes are more likely to be chosen, thus to get more connections,

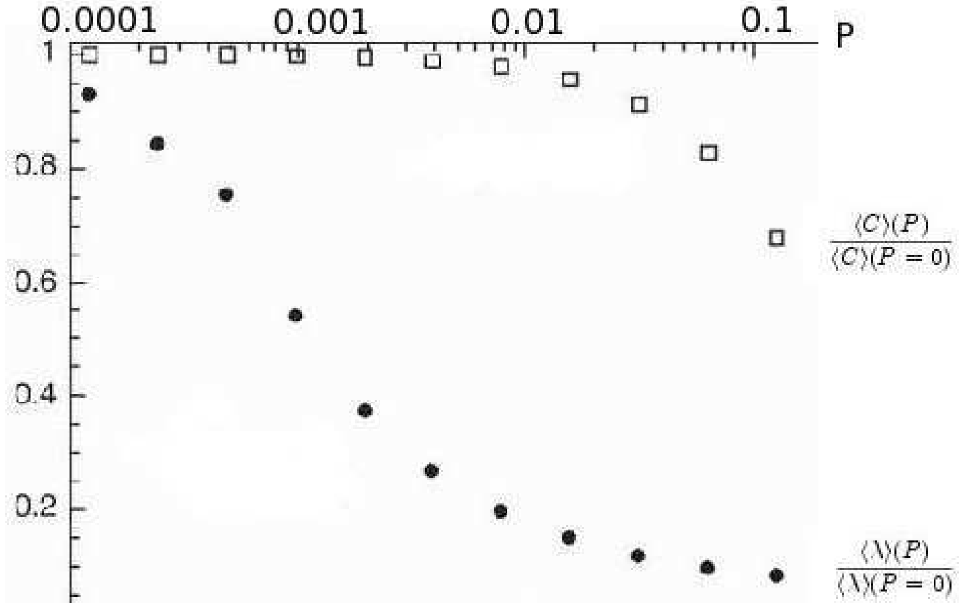


Figure 2.11: Mean Shortest Path length $\langle \lambda \rangle (p)$ and clustering coefficient $\langle C \rangle (p)$ for the family of randomly rewired graphs described in Figure 2.10. Here $\langle \lambda \rangle (p)$ (or $MSP(p)$) is defined as the number of edges in the shortest path between two vertices, averaged over all pairs of vertices. The clustering coefficient $\langle C \rangle (p)$ is defined as follows. Suppose that a vertex v has k_v neighbours; then at most $k_v \times (k_v - 1)/2$ edges can exist between them (this occurs when every neighbour of v is connected to every other neighbour of v). Let C_v denote the fraction of these allowable edges that actually exist. Define $\langle C \rangle$ as the average of C_v over all v. The data shown in the figure are averages over 20 random realizations of the rewiring process described in Figure 2.10, and have been normalized by the values $\langle \lambda \rangle (0)$, $C(0)$ for a regular lattice. All the graphs have $n = 1,000$ vertices and an average degree of $k = 10$ edges per vertex. A logarithmic horizontal scale has been used to resolve the rapid drop in $\langle \lambda \rangle (p)$, corresponding to the onset of the small-world phenomenon. During this drop, $C(p)$ remains almost constant at its value for the regular lattice, indicating that the transition to a small world is almost undetectable at the local level. [220].

this method reflects a "richer get richer" principle. With this method, the created network shows a $P(k) \sim k^{-\gamma}$ connection distribution. Many “real-world” networks fall into this category (the Internet, airplane routes, metabolic networks, social collaboration networks...) [28].

2.4.3 Applications to ANNs

Importantly, the connectivity structure of such complex networks (i.e. their topology) is a crucial determinant of their information transfer properties [28]. Hence, the computation made by complex neural networks, i.e. neural networks with complex connectivity structure, could as well be dependent on their topology. For instance, recent studies have shown that introducing a small-world topology in a multilayer perceptron increases its performance [199, 29]. However, other studies have inspected the performance of Hopfield [118, 153, 148, 205] or Echo state networks [44] with small-world or scale-free topologies and reported more contrasted results.

2.5 Research Questions

As noted above, the connectivity structure of complex networks (i.e. their topology) is a crucial determinant of information transfer in large networks (internet, social networks, metabolic networks...). For instance, information, virus or epidemic spreading in Small-world or Scale-free networks is much more efficient/faster than in comparable random or regular networks. Other crucial properties of these systems are topology-controlled, such as tolerance to faults/defects (robustness), vaccination or the existence of critical thresholds for epidemic spreading [28].

Several studies have applied such complex networks tools to neural networks. Here again, several functional properties of neural networks seem to depend on the network (complex) topology. For instance, a recent study has shown that introducing a Small-world topology in a monolayer perceptron increases the learning rate of the network [199].

Symmetrically, evolutionary algorithms are commonly used to modify the topology of neural networks so as to optimize ANNs performance. But, in most cases, the studied topologies are quite simple and the number of connections/neurons is low. Furthermore, the evolutionary mechanisms used in most of these studies do not modify the topology in an intensive manner.

Hence, the optimization of large neural networks through artificial evolution has hardly been studied. However the optimization of complex topologies in related systems has recently begun to be inspected. For instance, Tomassini and collaborators [142] have used evolutionary algorithms to optimize the topology of 1D-cellular automata networks (majority problem) and reported that their optimal topologies were systematically close to Small-world ones. More recently, Oikonomou & Cluzel [161] have optimized the topology of boolean networks and found that the evolution of networks with random topology was very different, even quantitatively, from networks with Scale-free topology.

Here, we hope to study the interaction between the topology of large neural networks and their learning capacities. Our approach tackles both the direct and inverse problem:

Direct problem: Given a network with a fixed topology, we study how the network learns to perform its target task through local (backpropagation, Hebb's rules) or global (artificial evolution) rules. An important aspect of this part of the study is that the quantification of the network efficiency can be defined on the basis of its pure performance, but it may also be based on its robustness to failures, noise, or attacks.

Inverse problem: Given a set of local learning rules and a given evolutionary optimization algorithm acting on the network topology (and, possibly, on the global parameters of the local rules themselves), we want to study what kind of topology the networks evolves to. In other words, we want to know if there exists such a thing as an optimal topology, for given local learning rules and a given task to perform. An interesting part of this study will be to compare the obtained topologies with real biological neural networks. Here again, network optimization can concern the pure performance of the network, but, alternatively, it may as well concern its robustness to noise or defects.

Chapter 3

Evolving the Topology of Self-Organizing Map

3.1 Introduction

In this chapter, we use classical Self-Organizing Maps (SOMs) to classify handwritten digits. But unlike classical SOMs where the neuron neighborhood relationships are defined using a simple topology (e.g. regular square or hexagonal lattices), we define neighborhood relationships using complex networks. We referred to this kind of ANNs "complex ANNs". We show that the topology of neural network has small impact on the performance and robustness of neuron failures, especially in the case of long learning times. However, the performance of recognition rate can be increased (by almost 10%) by evolutionary optimization of the network topology. In our experimental setting, the evolved networks are more random than their parents, whereas they have a more heterogeneous degree distribution.

3.2 Topology of Self-Organizing Map

3.2.1 Kohonen Maps

The theory of Self-Organizing Map (SOM) was first proposed as an Artificial Neural Networks (ANNs) by Kohonen [122, 123]. It is therefore also called Kohonen Maps [124]. The current version of the SOM bibliography contains around 8000 entries [116, 162].

The Self-Organizing Map usually describes a mapping from a higher dimensional input space to a lower dimensional map space. This makes SOM able to be used for visualizing high-dimensional data by low-dimensional views. For example SOM has been widely applied for feature extraction [144, 216].

Self-Organizing Map consists of components called nodes or neurons. Each neuron is associated with a weight vector of the same dimension as the input data vectors (high-dimension) and with a location (low-dimension) in the 2D map space. Classically, SOM neurons are linked to each other by hexagonal- or square-lattice connections, which defines the neighborhood of the neurons. When an example from data space is presented, the neuron who has the closest weight vector to the example is firstly found, then the weights of the neuron itself and of all its neighbors' are adjusted by the training process as follow:

Training a SOM is a competitive learning process. When a training example $\mathbf{I}(t)$ is presented to the network, for each neuron i, its distance d_i to $\mathbf{I}(t)$ is computed by:

$$d_i = \sum_{j=1}^{M} (I_j - w_{ij})^2$$

where w_{ij} stands for component j of the weight vector of neuron i, $\mathbf{w_i}$ and I_j is the j-th component of $\mathbf{I}$. The corresponding Best Matching Unit (BMU) is the neuron whose weight vector is the closest (in L^2-norm) to $\mathbf{I}(t)$. The weight of the BMU k are updated by:

$$\mathbf{w_k}(t+1) = \mathbf{w_k}(t) + \eta(t) \times (\mathbf{I}(t) - \mathbf{w_k}(t)),$$

where η is a user-defined learning rate. The weights of all BMU neighbors are updated similarly. In the simplest form, the learning rate η is 1 for all neurons close enough to BMU and 0 for others. However, Gaussian decay is a more common choice: the learning rate η decays according to a Gaussian law with respect to the distance to the BMU. The variance of the Gaussian law is also called the radius of the neighborhood. Regardless of the functional form, the radius always decreases with time. At the beginning when the neighborhood is broad, the self-organizing takes place on the global scale. When the neighborhood shrinks to only a couple of neurons, the weights are converging to local estimates.

This process is repeated for each input data example for a (usually large) number of cycles N_{max}. The patterns in the input data set is self-organized on the map. If these patterns can be labeled, the labels can be attached to the associated neurons.

3.2.2 An Example with Color Data

Let us illustrate with a toy example how Self-Organizing Maps works.

Visible lights are composed of three basic colors - Red, Green and Blue. For any color, it can be viewed as a data in a three-dimensional space. Figure 3.1 presents the color data set used as the training set for SOM learning.

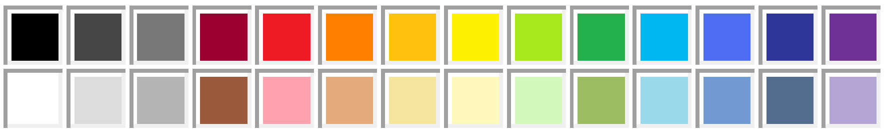


Figure 3.1: Random colors chosen as the training data for the SOM learning.

As introduced in the previous subsection, SOM is constructed in a 2-dimension grid, where each neuron is associated with a vector (R,G,B), as shown in Figure 3.2.

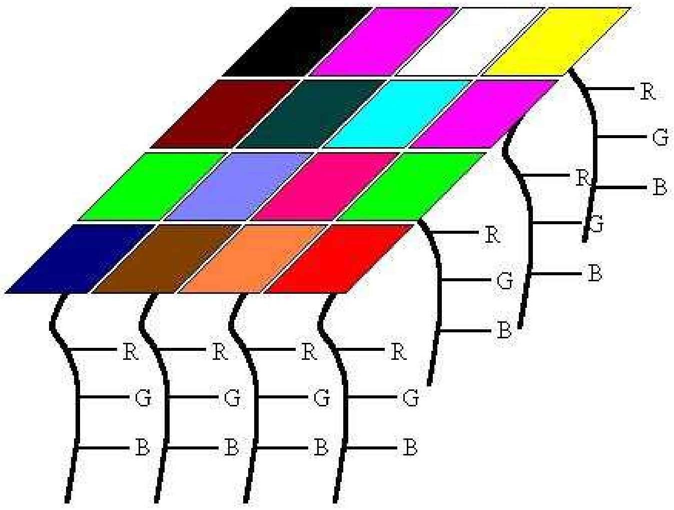


Figure 3.2: SOM in a 2D grid, with neurons associated with (R,G,B) vectors.

After the training process, the colors from the training set self-organize in the map. Figure 3.3 shows the learning results with rectangular, hexagonal and octagonal topology. The left one has rectangular connections, the middle one has hexagonal connections and the right one has octagonal connections. For every topology, similar colors are found in close proximity to each other in the 2D map: all those images display local similarity. This feature can be clearly viewed in this two dimension grid.

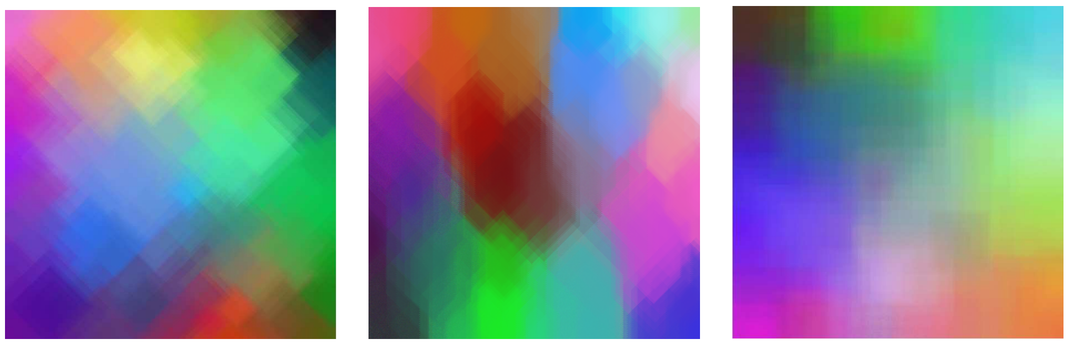


Figure 3.3: The learning results with different regular topologies, the left map has rectangular connections, the middle map has hexagonal connections and the right map has octagonal connections. All others learning parameters were the same: the map size was 60×60, the initial learning radius was the half of the map size (30) and decreased with time, the initial learning rate was 0.1 and also decreased with time, the training set consisted of 200 randomly chosen color, and the number of learning step was 10,000. The distance used is the Graphic distance (not Euclidian distance).

3.2.3 Quantitative measure of topology preservation in SOM

In the previous Section, we found out that the learning results are different for different topologies, even when that all the learning parameters are the same, as shown in Figure 3.3 with three regular topologies. It is an open issue to study these differences in a quantitative way, especially when the topology is not regular. In this subsection we will briefly present the quantitative studies of SOM topology in the literature.

In Self-Organizing Maps (SOMs), the role of network topology has been studied for several years under the perspective of the relationship between data topology and network topology. Several criteria, such as Topographic Product, Topographic Function, have been proposed for the quantitative measure of topology preservation. We will first introduce these criteria and then discuss the articles which compared the results with different methods.

Topographic Product

The topographic product is a measure for the preservation of neighborhood relations in maps between spaces of different dimensionality. It has first been introduced in the context of nonlinear dynamics and time series analysis [135] and then used to measure the preservation of neighborhood relations in SOM by

Bauer and Pawelzik in 1992 [20]. The topographic product is a quantification method which compares the distance between neuron i and its k_{th} nearest neuron in both input space and output space. For computing the topographic product, $Q_1(i,j)$ is the distance between point i in the output space and its k_{th} nearest neighbor j in the output space, divided by the distance between point i in the input space and its k_{th} nearest neighbor j in the input space. $Q_2(i,j)$ gives analogous information where i and j are both points in the output space. In Ref. [20], Bauer proposed several ways to combine $Q_1(i,j)$ and $Q_2(i,j)$ to produce a single number P, the "Topographic Product" which can define the quality of the preservation of neighborhood.

The method has been tested in [20] by a simple experiment where a 2-D space is embedded into 1, 2 and 3 dimensions spaces, and a more complex example of the speech recognition. The result is quantitative, but the process is hugely time-consuming.

Topographic Function

The topographic function has been introduced by Villmann et al in 1995 [217]. The notion of " Voronoi polyhedron" was applied to determine the receptive field of each neuron. Each neuron defines a Voronoi cell, that consists of all points closer to this neuron than to any other. The segments of the Voronoi diagram are all the points that are equidistant to the two nearest neurons. The Voronoi nodes are the points that are equidistant to three (or more) neurons. The main idea is to observe the neighborhood relations between receptive fields in input space. If only the nearest lattice neighbors of a neuron i have receptive fields which are adjacent to the receptive field of neuron i, the SOM is considered doing a perfect preservation. The authors have also given a method to determine the adjacency of two receptive fields.

Q, H and QH measures

Polani introduced several quantitative measures of SOM quality, such as the energy function measures and Hebbian measures [167] and the Q and QH measures [168]. He also compared the results among these measures by testing 6 different topologies (linear, hexagonal, square, crocodile, cubic and tree). We will briefly present these works below.

The Q measure: Q is the mean quantization error between a large set of points from the input space and its discrete approximation in the output space. It is a rather canonical quantity measure. For calculating Q, a large set of points $x_1 \ldots x_q \in \mathbb{R}$ of the input space are chosen randomly according to the probability data distribution (if known) on $\mathbb{R}$. $w_i(x_k)$ is the nearest approximation of x_k in the output space defined by the distance between $w_i(x_k)$ and x_k ($w_i(x_k)$ is the Best Matching Unit (BMU) of x_k). Q is defined as:

$$Q = \sqrt{\frac{1}{q}\sum_{k=1}^{q} d_R(w_i(x_k), x_k)^2}$$

where d_R define the distance between $w_i(x_k)$ and x_k. With respect to Q measures, small quantization errors denote good performance.

H measure: The H measure is the Hebbian measure developed in [167]. As the Q measure, a large set $x_1 \ldots x_q \in \mathbb{R}$ in the input space are chosen randomly according to the probability data distribution (if known) on $\mathbb{R}$. For every point x_k, one searches for the neurons i and j, whose weights w_i and w_j are closest (in the $L2$-norm sense) and second closest to x_k. An (Hebbian) edge is then created between neurons i and j, with strength 1 if it was not present before or increased by +1 if it already existed. The Hebbian measure essentially determines the Kohonen edges that do not match Hebbian edges and vice versa. Let A_K denotes the edge set of the Kohonen graph and A_H denotes the edge set of the Hebbian graph, $\hat{c} = (\sum_{(j,k)\in A_H} c_{jk})/|A_H|$ is the average strength of the Hebbian graph edges. Measure H is calculated as:

$$H = 1 - \frac{\hat{c}.|A_K \backslash A_H| + \sum_{(j,k)\in A_H \backslash A_K} c_{jk}}{\hat{c}.|A_K| + \sum_{(j,k)\in A_H} c_{jk}}$$

where $A_K \backslash A_H$ denotes the edge set which are in A_K but not in A_H and the $A_H \backslash A_K$ denotes the edge set vice versa.

The Hebbian measure essentially determines the Kohonen edges that do not match Hebbian ones, if values of H close to 1 indicate a good topology preservation and vice versa.

QH measure: The QH measures is a hybrid measure, given by H/Q^2

Comparing these indicators

Many articles have proposed to compare the different methods of measuring topology preservation in SOM.

Goodhill compared results obtained with some of the methods that have been proposed before 1995 [72] and showed one important caveat, namely that for the same learning result, different measures could give different assessments.

Therefore, it is difficult to decide which one is the best. In this paper[72], the author also gave two suggestions to define a mapping that perfectly preserves the neighborhood structure of the original data in the mapping space. The first suggestion is that the mapping must preserve similarities, which means for each pair of points in one space, its similarity should be equal to the similarity of the images of those points in the other space. The second suggestion is that the mapping must only preserve similarity orderings, which means, rather than comparing the absolute values of the similarity between pairs of points in one space and the similarity between their images in the other, it is only concerned that their relative orderings within the two sets of similarities are the same. This weak condition in fact imposes strong constraints on the mapping.

In [169], Polani compared the 6 measure proposed before 1997, including the H, Q and QH measures introduced above. In the experimental conditions of this work, the QH measure performed best, but no canonical measure can be given as in the Goodhill article. The author also advise [169] that: “The results show that in the current state of research one must proceed with great care when applying some organization measure to new situations with no a priori knowledge of the structure of the training data.”

3.2.4 Discussion

The article of Villmann in 2001 [218] was devoted to the development of network topologies that preserve the structure of data. In this data-driven context, the network topology is thus constrained by the data under study.

In the context of complex networks however, a key issue concerns the general performance of complex network classes: considering a given data set, do the different complex network topology classes (regular, small-world, scale-free) yield significant differences with respect to performance?

Furthermore, for complex data sets, defining the similarity (distance) among the data in the input space can be very difficult. Indeed, two similar data may represent different things as in the following example: Let us consider for instance a handwritten digit data set where every digit is represented as a bitmap. The similarity between the data is usually computed by the Euclidean distance. However in many cases, images which are very similar with respect to their bitmap representation can have different labels (i.e. represent different digits). Therefore, if the SOM keeps these similarities in its preservation of neighborhood relations, it may cause confusion for the following classification process. In such a complex dataset, do the different complex network topology classes (regular, small-world, scale-free) yield significant differences?

Another important issue is the following. In real-world large-scale neuron networks, noise of neurons may appear and neurons may languish at any time. Could the different complex network topologies be more robust when some part (eventually large) of the neurons do not function correctly i.e. can robustness to noise or defect of the neural network depend on its topology?

In the following sections we investigate both issues through an experimental study on the relationship between complex topology following 2D-Watts and Strogatz models and the performance of the corresponding SOMs on a supervised learning problem (handwritten digit classification), and on its robustness with respect to noise. After introducing the context in Section 3.3, Section 3.4 is devoted to the direct problem, i.e. observing the performances of networks with different topologies. The inverse problem is addressed in Section 3.5: what topology class emerges from the evolutionary optimization of the classification accuracy of a class of networks?

3.3 Method and Experiments

3.3.1 A Simple Experiment with Classical Q H Measure

Before the experiments for hand written digits, a first experiment is performed using classical Q H measures described in section 3.2.3.

One thousand real numbers are picked randomly in a 2D space within $[0,1]^2$. A 30×30 size SOMs is then created for learning of this data set. During the learning process, we use Q, H and QH measures (described in section 3.2.3) to monitor the evolution of the SOM with various topologies (regular, small-world, random).

Distance

In the classical SOM algorithm, the N neurons are regularly scattered on a regular $2d$ grid, that can be quadrangular, hexagonal or octogonal. The two distances that can be naturally defined between neurons, the Euclidian distance and the graph distance (the minimum number of hops between two neurons following the graph connections), are completely equivalent. However, when the topology diverts from that of a regular grid (e.g. links are added or suppressed), the

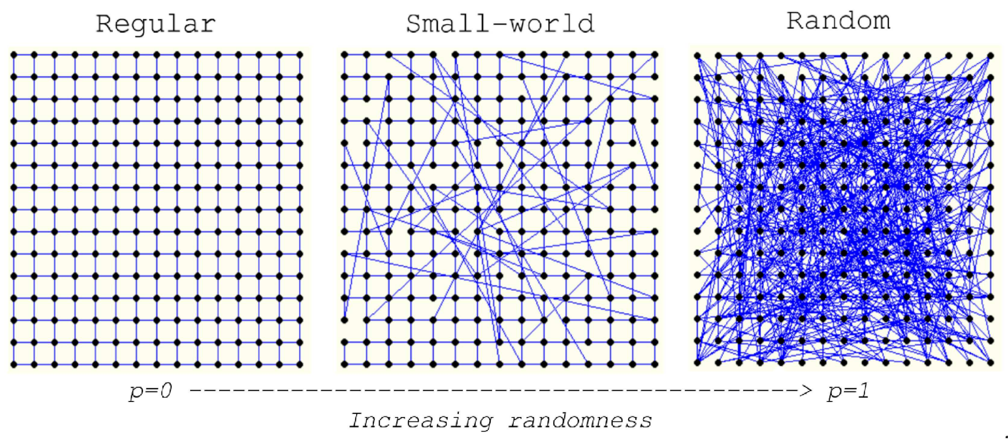


Figure 3.4: Illustration of 15×15 Kohonen maps with various interconnection topologies, ranging from regular (left), to small-world with rewiring probability $p = 0.04$(center) and random (right) depending on a rewiring probability p.

situation changes dramatically. Because the goal here is to evaluate the influence of the topology of the network on its learning performance, the distance between two neurons will be their graph distance. In other words, while classical SOM algorithms use regular grid networks for neuron positions and Euclidean distance for learning, we define the distance between the neurons as the graph distance as given by their complex interconnection network.

Figure 3.4 illustrates three kinds of interconnection topologies for 2D SOM networks. In analogy with Watts and Strogatz algorithm (Figure 2.10), neurons are first positioned on a square grid and each neuron is connected to its 4 nearest neighbors on the grid. This defines a regular topology (Figure 3.4, left). Each link is then rewired with probability p : its destination neuron is changed to a uniformly randomly chosen neuron. Depending on the value of p (0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.99 in our test), the neuron interconnection network thus varies from regular (left) to small-world (center) and totally random (right).

Figure 3.5 shows the evolution of the Q Measure during the learning process. The results indicate that in such a learning case, the regular topology gives the best topology preservation with Q measures.

Figure 3.6 shows the evolution of the H measure during the learning process. The situation is the same as with the Q measure above, i.e. the regular topology gives the best topology preservation .

Figure 3.7 show the fitness by combining the Q measure and the H Measure during the learning process.

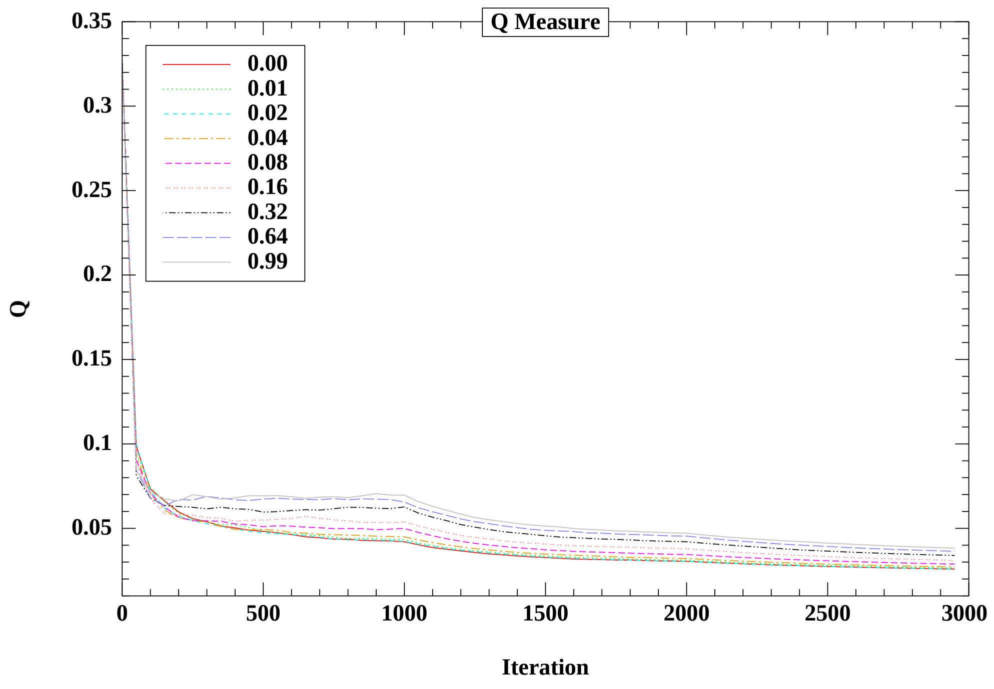


Figure 3.5: The fitness using Q Measures during the learning process. Plots for different values of the rewiring probability *p*.

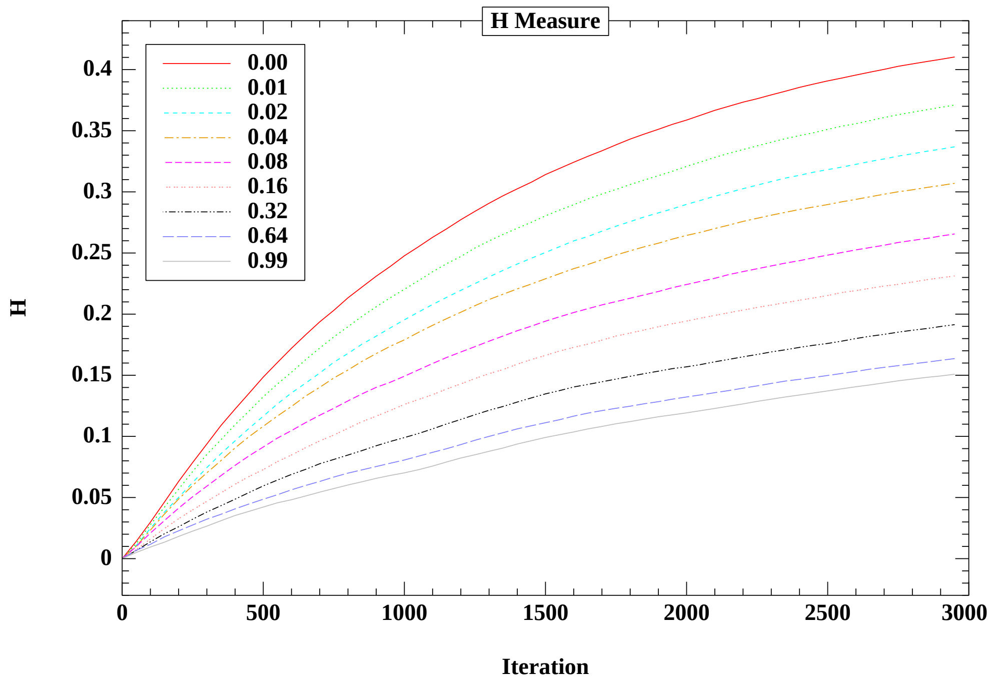


Figure 3.6: The fitness using H Measures during the learning process

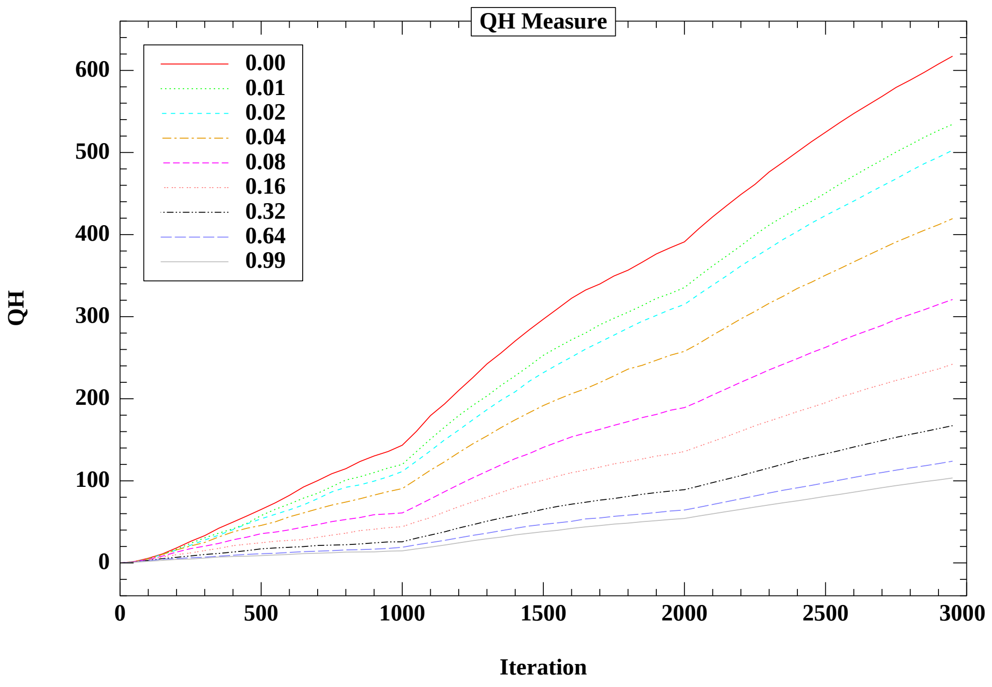


Figure 3.7: The fitness using QH Measures during the learning process

This first experiment shows that when the topology of the training set is very simple, the SOMs with the most simple topology can give the best performance as judged by classical topology preservation measures. But if the topology of the training set is more complex, e.g. with handwritten digits, so that the classical topology measures cannot not be applied (since the topology of the input data set is too complex or hierarchic), the situation may be different. This issue is treated in the following section, with our work on the MNIST database of handwritten digits.

3.3.2 Experiments With MNIST database

SOM are usually used for unsupervised learning tasks and produce low-dimensional representations of high-dimensional data [162]. They are thus useful for visualization purposes. This section will detail the supervised learning procedure that has been used here. To sum up, a label must be given to each neuron of the network after the standard SOM unsupervised learning. To this aim, we'll used a supervised learning process. The classification of an unknown example is then achieved by finding the best matching neuron (BMU) of the network. The guessed class of the unknown image is then the label of this BMU. More details are given below.

The task considered here is the recognition/classification of handwritten digits, using the well-known MNIST database: The MNIST database of handwritten digits [130] has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size (28×28) image. SOM will hence be used here with partly supervised learning, in order to give an unambiguous performance measure and estimate the corresponding topology/performance relation. It should be noted that the goal here is not to reach the best possible performance for the MNIST problem (and indeed SOM cannot compete with best-to-date published results) but to compare the relative performances of different topologies on the same problem [130].

Each digit in the data base is described by a $M = 28 \times 28$ matrix of pixels (integer gray level in [0,255]). The N neurons of the SOM are scattered on a $2d$ space. Each neuron i has an associated M-dimensional weight vector $\mathbf{w_i}$ that is initialized randomly and will be adjusted while learning the training set. The different phases of the learning process go as follows.

3.3.3 Learning

This phase is the classical unsupervised SOM learning process which has been introduced before (section 3.2.1). It classically go through the following steps:

1. Randomly initialize the weight vectors of all neurons
2. Randomly fetch an input image from the training set
3. For each node in the map
    (a) Use Euclidean distance formula to find similarity between the input image and the neuron's weight vector item Track the BMU, i.e., the node that produces the smallest distance
4. Update the BMU and the neurons in the neighborhood of the BMU by pulling them closer to the input vector
5. Increment t and repeat from 2 while $t < N_{max}$

In the present work, the total number of learning steps N_{max} was varied between 10^4 and 10^6, and the radius is decreased along learning iterations (see e.g. Figure

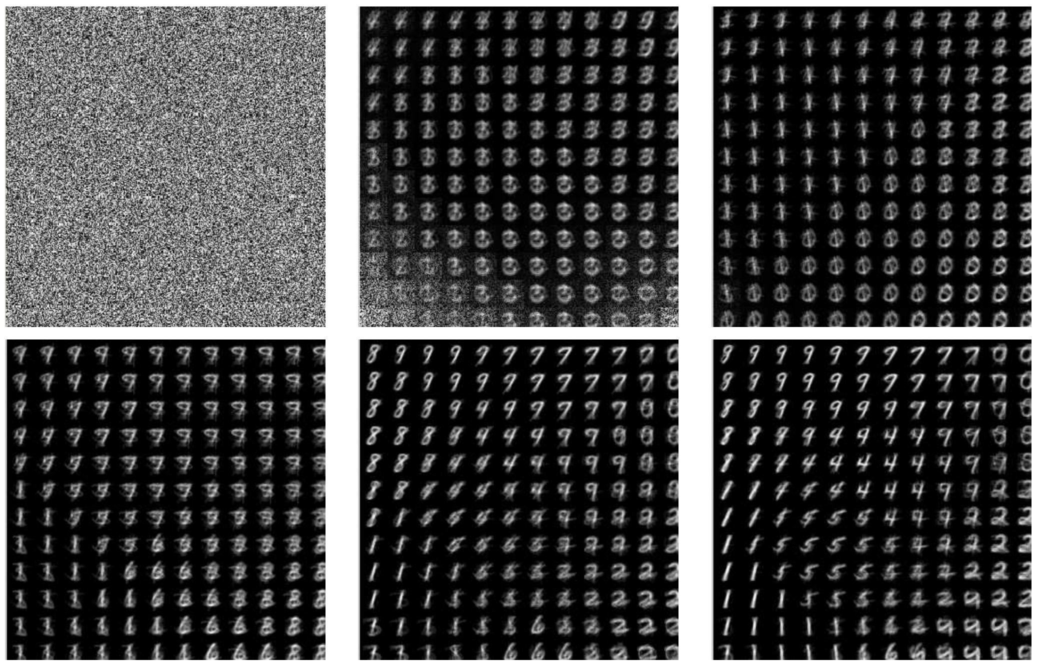


Figure 3.8: The visualization of SOM learning result during the learning process, from the top left to top right, and bottom left to bottom right.The weight vectors of every neuron was represented as a 28×28 image centered on the neuron position. It was just a demo example with only 3000 learning steps. The numbers of iteration for the six images were 0, 200, 500, 1000, 1800, 3000.

3.12 B or E).

If we visualize the learning result as a image (the weight vectors of every neuron is represented as a 28×28 image centered on the neuron position) during the learning process, we can get a more direct understanding of how the SOM learning works. Figure 3.8 shows a 15×15 size SOM, form the top left to top right, and bottom left to bottom right are the visualizations of learning result during the learning process. We can see at the beginning, since the weight vectors of every neuron are randomly initialized, the visualization of SOM present a complete fuzzy image; then with the SOM learning process, the neuron begin to adjust it's weights vectors from the training example according to the classical SOM learning rule. The visualization of SOM become finally clear and the cluster appear in the end of learning process.

3.3.4 Distance

The three figures in Figure 3.9 show the results of the learning phase (described above) using the three topology classes of Figure 3.4. In these figures, the weight vector $\mathbf{w_i}$ of each neuron i is represented as a small 28×28 image, centered on the neuron position in the 2d grid. In the regular network, the original 784-dimensional data images of handwritten digits have been projected on the 2d map so that the visual proximity between two digits is well rendered by the Euclidean distance between the network nodes. When the rewiring probability p increases, the image representations of the neuron weights become increasingly fuzzy. Furthermore, the visual proximity between two images becomes less and less correlated to the Euclidean distance between the neurons, because it is correlated to their graph distance.

Figure 3.10 illustrates an example of the influence of the topology on the size of the neighborhood of the neurons. The three top figures show a regular topology while the three bottom figures show a small word topology. From the left to right, the three figures give the neighborhood of a randomly chosen neuron (left: one step in graph distance, middle: four steps in graph distance, right: eight steps in graph distance). This figure illustrates that in small-world or random topologies, the neighborhood at a given radius of a neuron is composed of a larger number of neurons than with regular topology. That means that the influence of a BMU neuron is wider in a small-world topology. This property will be crucial for the results shown thereafter.

For a given connectivity density, although the Random topology has the smallest Mean Shortest Path (MSP), the influence of a BMU neuron may even extend to every neuron, and thus forbids any learning (as demonstrated by our next experiments). Thus one target of this study is to investigate if there exists a optimal trade-off between the regular topology and the random topology – such as small world topology for instance – for the given task.

3.3.5 Labeling

The major issue here however is that measuring the quality of the results obtained with SOMs such as those presented in figure is not trivial. Here, we will use an additional supervised training level to this aim.

The aim of this phase is thus to prepare the map obtained from the previous unsupervised learning phase described above for the recognition/classification of

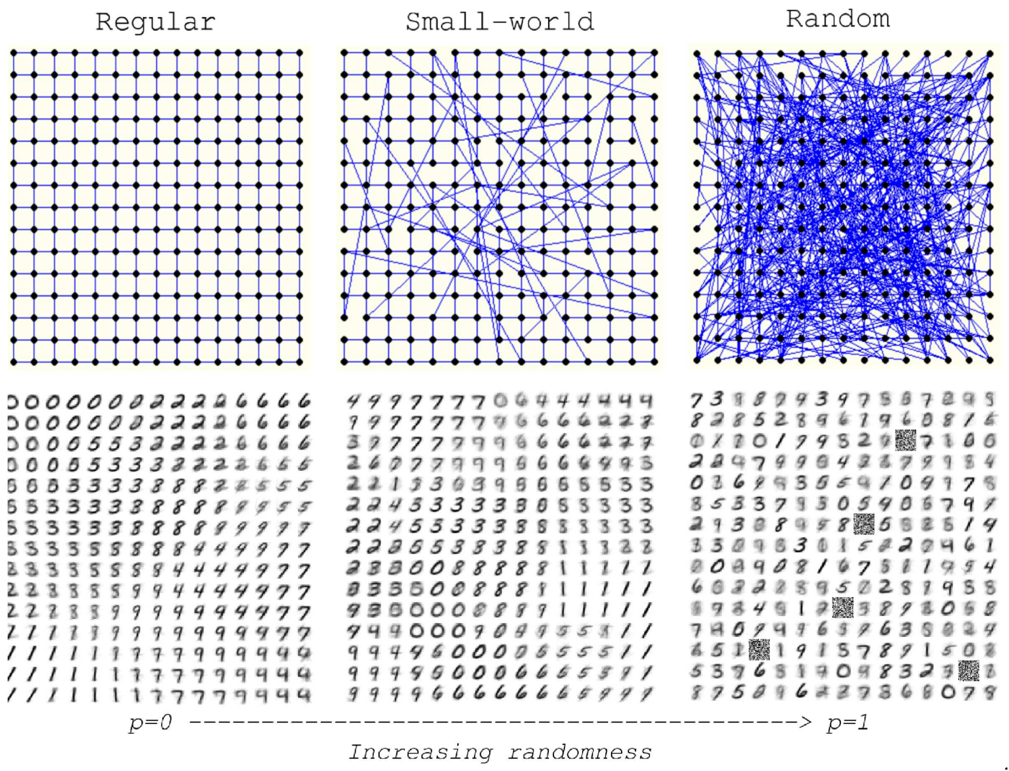


Figure 3.9: Results of the learning phase using the three topology classes shown in Figure 3.3.5. The weight vector $\mathbf{w_i}$ of each neuron i is represented as a 28×28 image, centered on the neuron position in the $2d$ grid. For the random network, some neurons are disconnected from the network, and thus do not learn from the examples during the learning phase. Their weight vectors are thus kept random, yielding the random images on the figure.

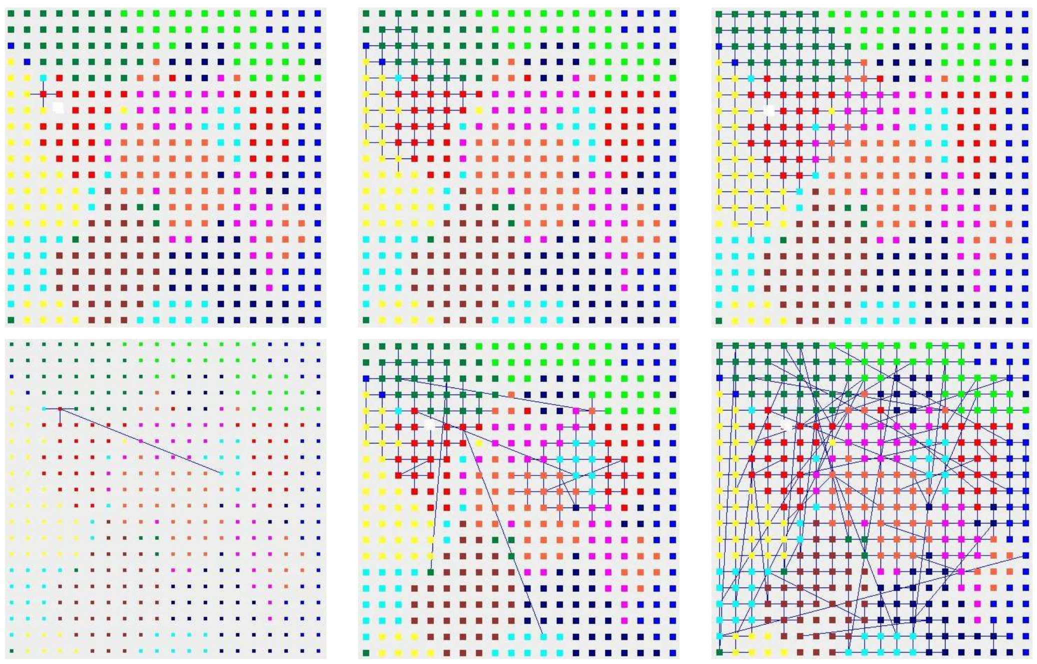


Figure 3.10: The three top figures show a regular topology (square) while the three bottom figures show a small word topology with rewiring probability $p = 0.04$. From the left to right, the three figures give the neighborhood of a randomly chosen neuron.(left: one step in graph distance, middle: four steps in graph distance, right: eight steps in graph distance). We can find in small-world topology, there are more neurons which are connected with the same radius. It means that the influence of a BMU neuron is more wilder in a small-world topology, with a smaller MSP (mean shortest path).

handwritten digits, i.e., to be able to classify unknown examples. The label of each neuron is assigned in the following way. Firstly, the BMU of each example of the training set is computed (see Section 3.3.3). Secondly, for each neuron of the network, a probability vector is computed, describing the different votes of each class for this neuron as BMU. For example, if neuron i is the BMU of 20 (labeled) examples, out of which 16 are labeled “1” and 4 labeled “7”, the probabilities attached to this neuron are computed as $p_i(1) = \frac{16}{20}$, and $p_i(7) = \frac{4}{20}$ (and $p_i(l) = 0$ for other values of l). The *basic* label for the neuron is then defined as the class with higher probability, e.g. “1” in the preceding example. Neurons that never were BMUs are given the basic label of the class from which they are at shortest distance in L^2 norm (this case only concerns a very small number of neurons and does not impact the classification accuracy).

Figure 3.11 illustrates an example of the influence of the topology. In the three figures, the different color indicates the different labels of the neurons. The left figure is a SOM with regular topology, the middle one has a small-world topology with low rewiring probability ($p = 0.02$, close to regular topology) and the right figure has a topology with high rewiring probability ($p = 0.32$ which is no longer small-world by the definition, but close to random topology). We can see that in the regular topology, the neurons with same label are close to each other and the clusterings can be clearly identified. But in the right figure, the different clusterings are mixed with others.

3.3.6 Classifying

Using either the whole probability vector, or the basic label, two strategies for classifying unknown examples can be designed. In both cases, the test example is presented to the network, then the distance between the example and each neuron is computed, and finally the N nearest neurons from the examples are recorded (N is a user-defined parameter).

Majority by numbers: The class given to the unknown example is the basic label most often encountered among the N nearest neurons. Preliminary experiments (for maps of $3,600$ neurons) with N ranging from 1 to 500 showed that the best performances are obtained for $N = 1$.

Majority by probability: Here, the probability vector p_i attached to each neuron i is used to compute the probability that the test image belongs to class k ($k \in [0,9]$) using the following equation:

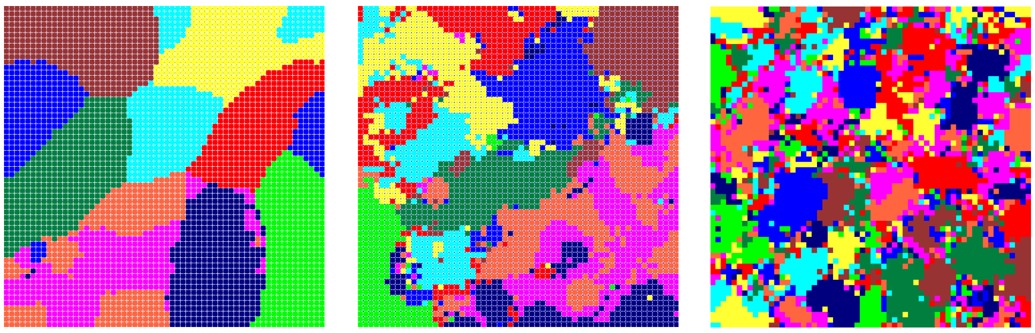


Figure 3.11: In the three figures, the different color indicate the different labels of the neurons. The left figure is a SOM with regular topology, the middle one has a small-world topology with low rewiring probability ($p = 0.02$, close to regular topology) and the right figure has a topology with high rewiring probability ($p = 0.32$ which is no longer small-world by the definition, but close to random topology). We can see that in the regular topology, the neurons with same label are close to each other and the clusterings can be clearly identified, except in the right figure, where the different clusterings are mixed with others.

$$P_k = \frac{1}{N}\sum_{i=1}^{N} p_i(k)$$

The label of the test image is given by the highest probability P_k. For this strategy, preliminary experiments reported that the best performance is obtained with $N \in [1-8]$.

The "Majority by probability" strategy is more computationally expensive than the former. Moreover, the same preliminary experiments mentioned above showed that its performance is not significantly better. Hence, all following experiments will use the first strategy ("Majority by numbers") with $N = 1$: the class given to an unknown test example is simply the basic label of its BMU.

The performance (or fitness) F of the network can then be computed as the misclassification error over the whole test set:

$$F = n_{err}/N_{test},$$

where n_{err} is the number of incorrectly classified test examples and N_{test} the size of the test set.

3.4 Direct problem

The goal of the first experiments is to compare the classification performance of SOM built on different topologies, namely ranging from regular to random topologies according to the Watts and Strogatz model (see Figure 2.10).

Figure 3.12-A shows plots of the evolution of the classification performance F during the (unsupervised) learning phase for networks of 1024 (32×32)neurons with regular (rewiring probability $p = 0$, bottom curve) to small-world (intermediate curves) to fully random ($p = 1$, top curve) topologies. The initial learning rate is $\eta(0) = 0.008$ (adjusted after a few preliminary runs) and the total number of learning steps is 10^6. The radius of each neuron (i.e. the variance of the learning rate) was varied as shown in figure 3.12-B. The full MNIST database was used for those experiments, i.e. the size of training set is 60000 and the size of test set is 10000.

3.4.1 Influence of the radius

First of all, Figure 3.12-A shows that, at long learning times, the network performance is clearly independent on the topology. This is not surprising since the role of the topology decreases with the radius R. Indeed, the number of neighbors within a radius R of a given neuron increases when the rewiring probability p increases. However, this difference decays as R decreases. Important differences are however obvious at short to intermediate learning times: the more random, the less efficient the network at this time scale. This remark deserves further analysis. Indeed, the performance of these random networks evolves in a piecewise constant fashion. Comparing this evolution to the simultaneous decrease of the neighborhood radius (Figure 3.12-B) uncovers that performance plateaus are synchronized to radius plateaus.
The more random the network, the lower its Mean Shortest Path. Hence, a possible interpretation is that, for high p values, the influence of a given neuron at short learning times extends over the entire $2d$ space, to almost every other neuron. Thus, at short time scales, almost all neurons are updated each time a new image is presented, which actually forbids any learning in the network. This interpretation is supported by Figure 3.13-D, where the initial radius is five times smaller than in Figure 3.12-A, everything else being equal. Here, the differences in short time behaviors observed above have vanished.

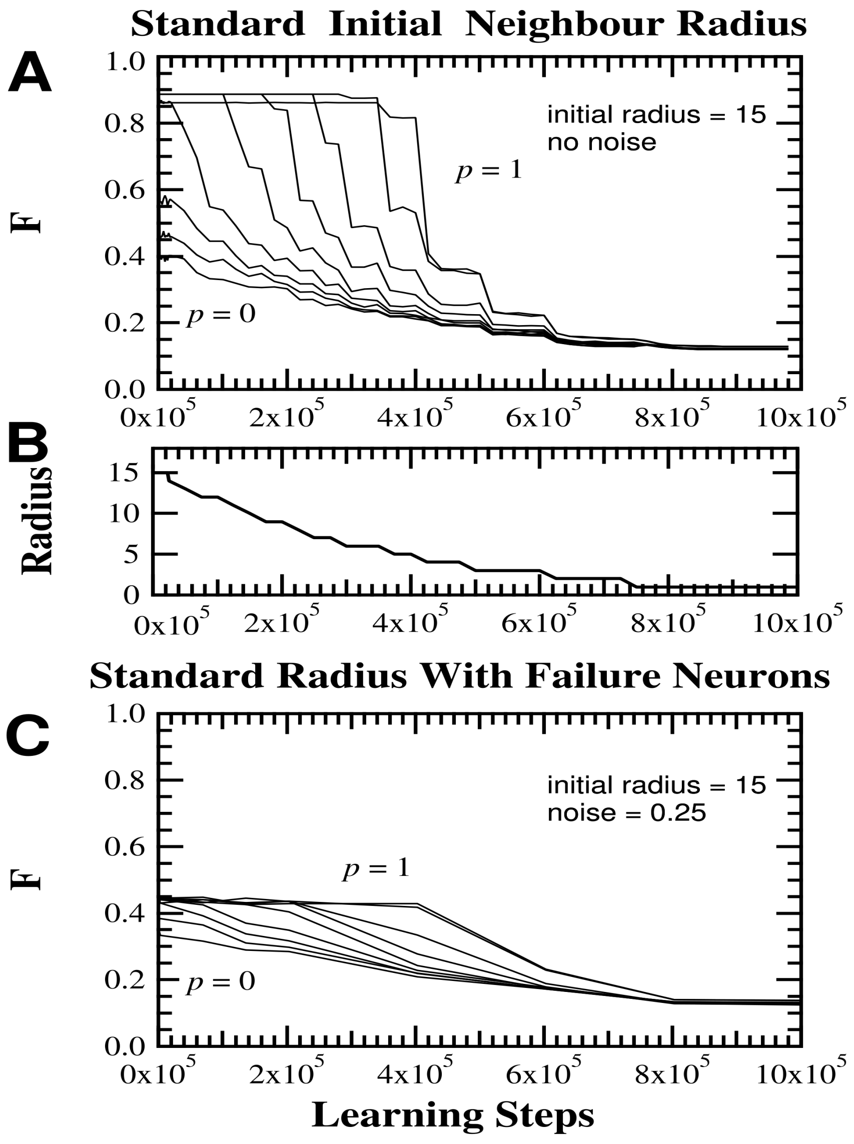


Figure 3.12: Evolution of the performance F during learning for SOM on complex neighborhood network with large initial neighborhood radius. Neighborhood networks are constructed positioning neurons on a square grid, and linking each neuron to its 8-nearest neighbors on the grid (Moore neighborhood). Each link is then rewired to a (uniformly) randomly-chosen destination neuron with probability $p = 0, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.256, 1.000$ (from bottom to top). Panels $\underline{A}$, $\underline{C}$ show the evolution of the fitness F for different noise levels (as indicated on each panels). Panels $\underline{B}$ displays the evolution of the neighborhood radius. Other parameters: map size $N = 1024$ neurons, initial learning rate $\eta(0) = 0.080$, training and test sets of 30,000 and 10,000 examples, respectively.

3.4.2 Robustness against noise

Unlike most of computer-simulated ones, real-world networks (including neural networks), involving a large number of computing units/neurons, will be subject to noise and defects. In particular, we were interested in studying now noise affecting the topology, and its impact on the computing performances of the network (here, its classification performance). Noise is modeled here by deactivating a fraction ν of the neurons at each learning step (the list of the $N\nu$ deactivated neurons is chosen uniformly for each learning step). All neurons are however considered active for the evaluation phase (Section 3.3.6).

Figure 3.12-C shows the performance of the same networks during learning with $\nu = 0.25$ noise level (i.e. 25% of the neurons are insensitive to learning, at each step) and the same large initial radius as in Figure 3.12-A. The differences between both figures can be explained by looking again at the radius. Clearly, because the deactivated neurons are protected from update, the effect of large radius that is described above is strongly attenuated. In other words, the presence of noise (here random node failures) actually improves the performance of these complex random networks at short learning times. That this effect is effectively related to large radius sizes is confirmed by inspection of Figure 3.13C, which shows that with small initial radius, this 'beneficial' effect of noise is not observed (compare with Figure 3.13D).

Another result from Figure 3.12 is that the effects of noise are restricted to short-to-average learning times and almost disappear with long learning times, where the performances of all networks are similar (whatever the topology randomness or initial radius). Hence, for long learning times, the SOM are robust to neuron failure rates as high as 25%, and this robustness does not seem to depend on their neighborhood topology.

Finally, Figure 3.14 shows the effects of network size on its performance. Each point is averaged over 11 independent runs. While large SOM ($N > 2,000$) perform better with regular neighborhood networks, the situation is just the opposite with small ($N < 200$) SOM, where random networks perform better than regular ones. Small-world topologies are intermediate (not shown). Note however that even for the extreme sizes, the difference of fitness between regular and random topologies, though statistically significant (see caption), remains minute.

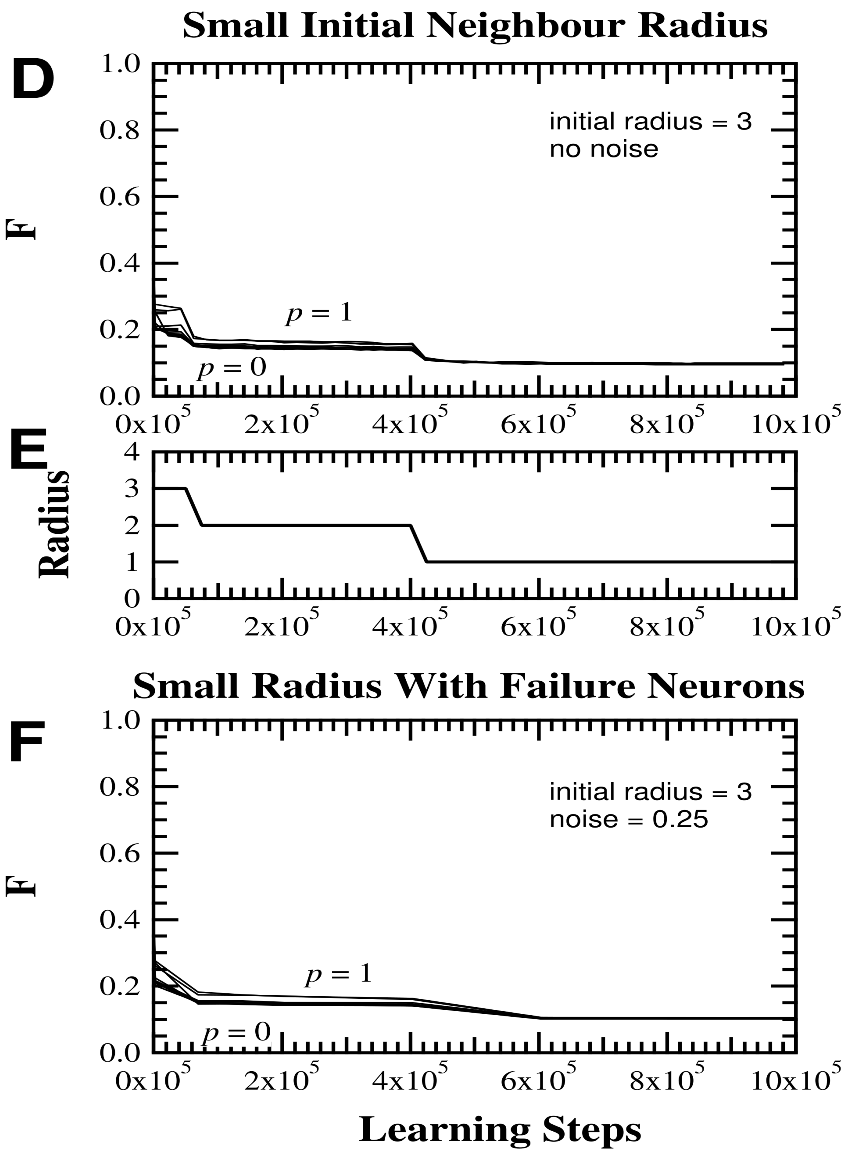


Figure 3.13: Evolution of the performance F during learning for SOM on complex neighborhood network with small initial neighborhood radius. Other caption details are the same as in Figure 3.13.

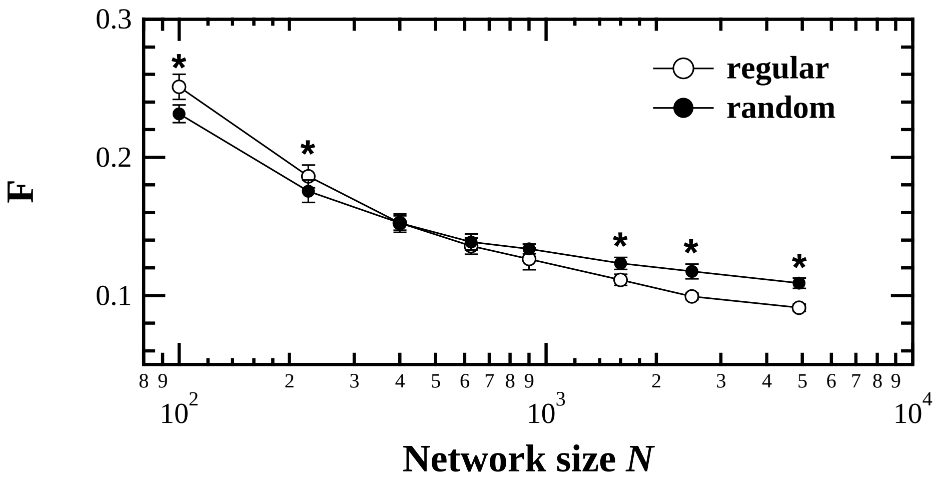


Figure 3.14: Performance F vs number of neurons N after 10^6 learning steps for regular (rewiring probability $p = 0$, white circles) or random ($p = 1$, black circles) topologies. Each point is an average over 11 random initial weight and topology realizations. Vertical bars are standard deviations. Stars indicate statistically significant differences (unpaired t-test, $p < 0.010$). Other parameters as in Figure 3.12-A.

3.5 Inverse problem

The inverse problem consists in optimizing the topology in order to minimize the classification error. Evolutionary Algorithms [46] have been chosen for their flexibility and robustness with respect to local minima. However, due to their high computational cost, only SOM with $N = 100$ neurons could be tested: according to the results of previous section, the best topology among the Watts and Strogatz models for this size of network and this task is that of a random network (see Figure 3.14). The purpose of the following experiments is to find out whether optimization will tend to push the topology toward random networks, or if other topologies, outside the Watts and Strogatz model, will appear.

3.5.1 The Algorithm

Evolutionary Engine: The algorithm used here is a Steady-State Genetic Algorithm with 2-tournament selection and 6-tournament replacement: at each generation, the best of two uniformly drawn networks undergoes variation (see below), and replaces the worse of 6 uniformly drawn networks for the population.

The initial population is composed of 100 different small-world networks (initial with rewiring probability $p = 0.050$).

Variation operators: Evolutionary algorithms typically use two types of variation operators: crossover, involving two or more parents to generate one offspring, and mutation, that uses a single parent to create an offspring. However, because no meaningful crossover operator could be designed here since we want to have the total number of connections fix during the optimization, only mutation was used (no crossover operator is better than a poor crossover operator).

The mutation consists in random rewiring of C% of uniformly chosen links. C decreases exponentially during evolution ($C(g) = 30\,(102.6)^{-g/g_{max}}$ where g is the generation number and g_{max} is the total number of generations). Here, $g_{max} = 200,000$. In these conditions, $C(g)$ decreases from 102 rewired links in total ($g = 0$) down to 1 single rewiring ($g = g_{max}$).

Fitness: The fitness is computed as the average misclassification error F (see Section 3.3.6) over 5 learning phases, starting from 5 different initial weights.

Each network contains 10×10 neurons, each neuron having 8 neighbors (Moore neighborhood). The initial learning rate is set to $\eta(0) = 0.35$ for a fast learning. However, in order to further decrease the computational time, the learning algorithm is run during only 10000 learning steps (see discussion below), using only 2000 examples from the training set, and the fitness is computed using 5000 examples from the test set.

This algorithm is summarized below :

- Initialize K networks: rewire 5% of a regular network (8 links per neuron with its neighbors, 342 global)

- For M "generations":

  1. Draw uniformly 2 networks, select the best one (N)
  2. $N' = mutation(N)$ (rewire C% of its links)
  3. Compute n times fitness(N'), get the mean
  4. Draw uniformly 6 networks, replace the worst by the mutated offspring N'

3.5.2 Results

As already mentioned, considering the small size of the SOM involved, one may expect random networks to perform slightly better than regular ones (Figure 3.14). The main statistics of the best networks obtained during 9 evolution runs are plotted Figure 3.15.

The first remark from Figure 3.14-A is that indeed, the classification error of the best topology in the population decreases along evolution, from 0.355 to ≈ 0.325, i.e. a $> 9\%$ improvement.

But the most interesting results can be seen when looking at the characteristics of the best topologies that have emerged during evolution: Figure 3.15-B shows an important decrease of the Mean Shortest Path, while Figure 3.15-C demonstrates a clear collapse (more than fourfold reduction) of the Clustering Index. In other words, the topology evolves towards more randomness – as could be expected from Figure 3.14.

Interestingly, there is another important change in the topology along evolution, concerning the network connectivity distribution. Indeed, the standard deviation σ_k of the connectivity distribution $P(k)$ (where $P(k)$ is the probability that a neuron chosen at random has k neighbors) almost triples during evolution (Figure 3.15D). This means that the connectivity distribution of the networks broadens (becomes less sharply peaked). In other words, artificial evolution yields more heterogeneous networks. However, it should be kept in mind that this result is highly dependent on the topology of the data themselves (here MNIST database), and could be different with other data. Future works will be necessary to investigate this interesting question.

3.5.3 Generalization w.r.t. the Learning Process

During the evolution process, the networks at each generation were trained during $10,000$ learning steps mainly for computational cost reasons. But how do the evolved networks perform with learning protocols of different lengths (e.g. one million steps)? In order to investigate this generalization ability, the 6 best networks from the initialization phase and the 6 best networks obtained after evolution were trained during respectively $10,000$ (Figure 3.16) and one million (Figure 3.17) learning steps. Note that the results obtained with $10,000$ learning steps are not a simple zoom-in of the results obtained with one million learning steps, because the radius R decays at different rates in these two cases (as shown

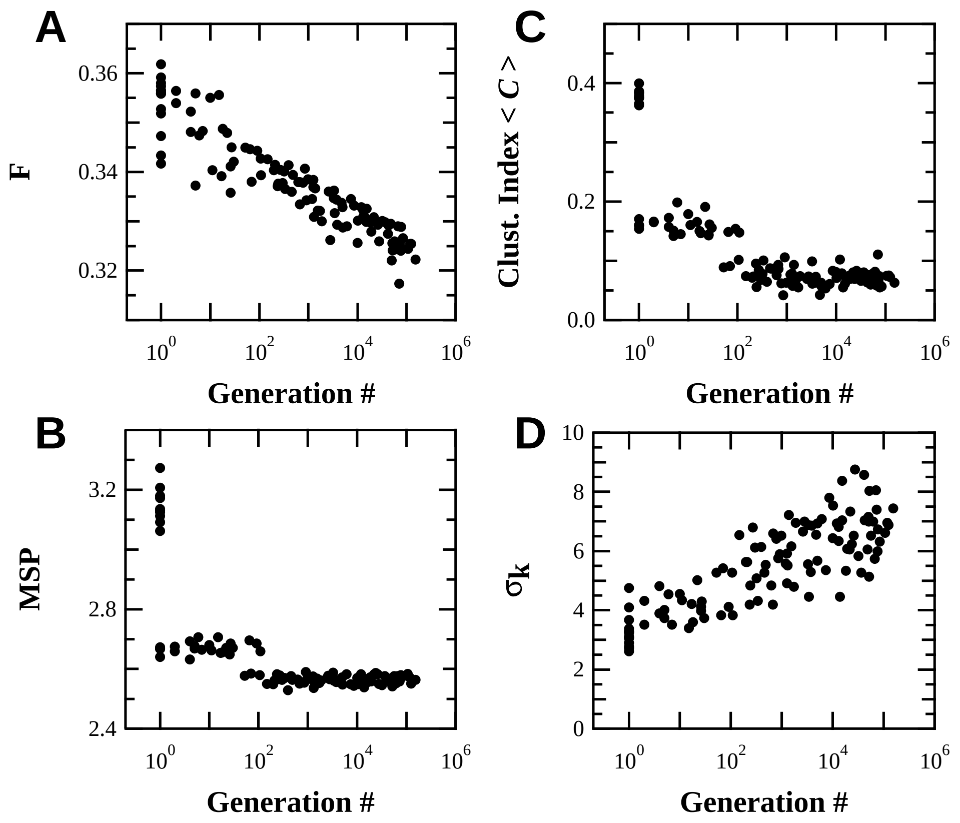


Figure 3.15: Time courses of the main network statistics during artificial evolution. Each time a mutation gives rise to a topology with a better fitness than the best one in the current population, its fitness (A), average mean shortest path (B), average clustering index $\langle C \rangle$ (C) and the standard deviation of its connectivity distribution σ_k (D) are plotted against the current generation number. Each panel groups the results of 9 evolution runs.

in the bottom plots).

With $10,000$ learning steps, the fitness obtained at the end of the learning phase by the evolved networks are slightly better than those obtained with the initial networks. Surprisingly, this improvement of the fitness is much clearer when learning was performed over one million learning steps (Figure 3.17) - albeit during evolution, these networks were trained using 100-fold less training steps-. At the end of the learning protocol, the average fitness of the 6 best evolved networks is $> 4\%$ better than that of the 6 best initialized networks (note that this figure is lower than the $> 9\%$ improvement above, because the

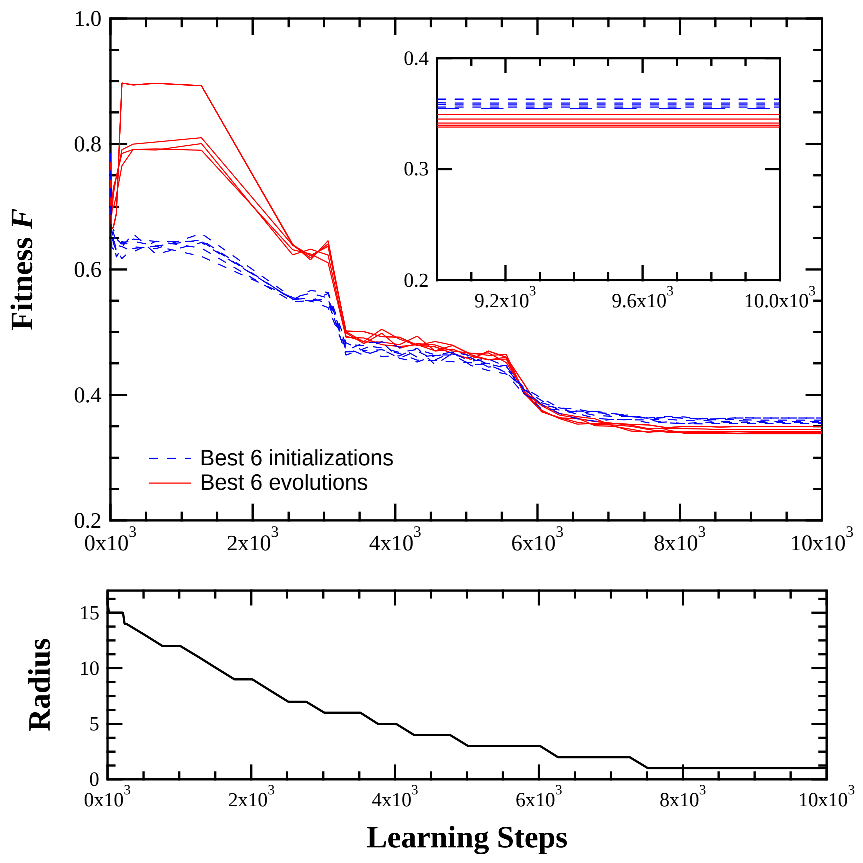


Figure 3.16: Evolution of the fitness during learning of the 6 best networks from the initialization phase (dashed lines) and the 6 best networks obtained after evolution (full lines). The learning protocol consisted of 10^4 learning steps. The inset shows magnified views of the results at the end of the learning phase. The evolution of the neighborhood radius is also given in each case for comparison purposes. Each curve is an average over 11 initial realizations of the neuron weights.

12 networks were selected from 3 evolution runs only). In the case of one million learning steps, this difference increases up to $\sim 8\%$. Finally, note that, at the end of the learning period, the difference between the two populations is statistically significant ($p < 0.01$, unpaired t-test) for both learning conditions (10000 and one million steps). Hence, the networks selected using 10^4 learning steps also outperform the initial networks for very different learning processes (here 100-times longer). Further investigations are required to better understand this phenomenon.

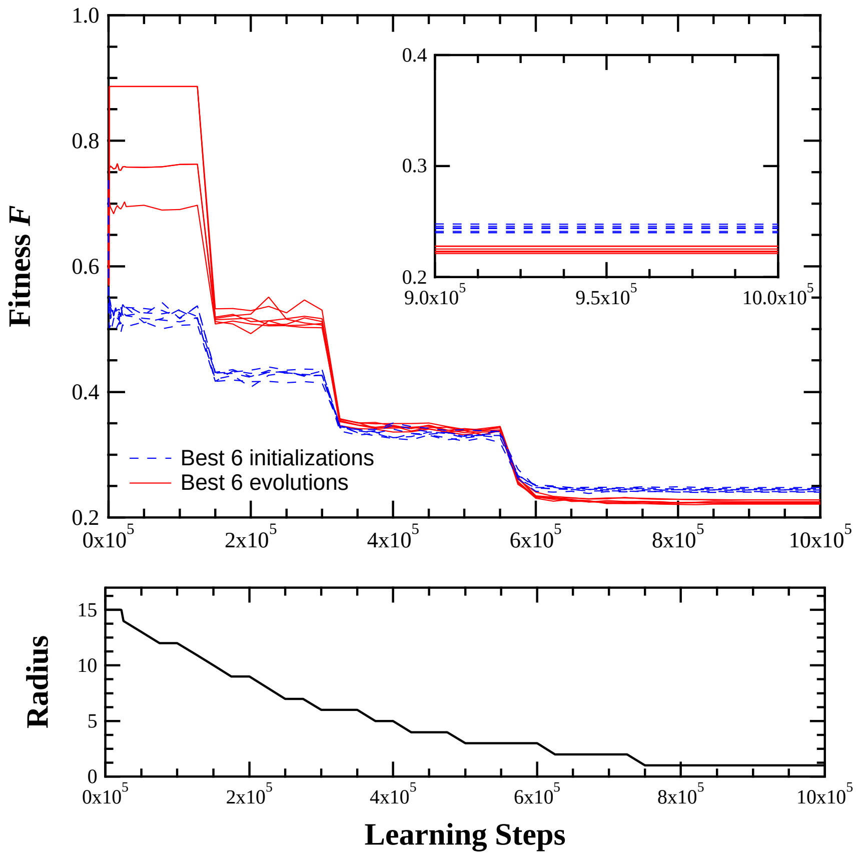


Figure 3.17: Same simulations as those in 3.16 except that the learning protocol used during evolution consisted of 10^6 steps.

3.6 Conclusion

The objective of this chapter was to study the influence of topology in a case of neural network defined on a complex topology. On the limited experiments presented here, it seems that the performance of the network is only weakly controlled by its topology. Though only regular, small-world and random topologies, have been presented, similar results have been obtained for scale-free topologies. This suggests that for such learning task, the topology of the network is not crucial.

Interestingly, though, these slight differences can nevertheless be exploited by evolutionary algorithms: after evolution, the networks are more random than the initial small-world topology population. Their connectivity distribution is also more heterogeneous, which may indicate a tendency to evolve toward scale-free topologies. Unfortunately, this assumption can only be tested with large-size networks, for which the shape of the connectivity distribution can unambiguously be determined, but whose artificial evolution, for computation cost reasons, could not be carried out.

Chapter 4

Evolutionary Optimization of Echo State Networks

A possible alternative to fine topology tuning for Neural Network (NN) optimization is to use Echo State Networks (ESNs), recurrent NNs built upon a large reservoir of sparsely randomly connected neurons. The promises of ESNs have been fulfilled for supervised learning tasks. But reinforcement learning tasks, such as control problems, require more flexible optimization methods – such as Evolutionary Algorithms.

This chapter proposes to apply CMA-ES, the state-of-the-art method in evolutionary continuous parameter optimization, to the evolutionary learning of some ESN parameters (see Section 2.2.3). First, a standard supervised learning problem is used to validate the approach and compare it to the standard quadratic approach. Further, thanks their flexibility, evolutionary optimization allow one to optimize not only the standard outgoing weights, but also, or alternatively, some other internal parameters of the ESN, sometimes leading to improved results, as will be demonstrated here. The classical double pole balancing control problem (Section 4.4.1) is then used to demonstrate the feasibility of evolutionary reinforcement ESN learning. Note that special care must be taken when using CMA-ES for this specific problem, in order to lead the evolutionary ESN toward results that are comparable with those of the best topology-learning methods.

4.1 Introduction

It has long been known to Neural Networks practitioners that a good design of the topology of the network was an essential ingredient to a successful application of Neural Networks, for a given learning task. In the framework of supervised learning, the famous US Postal application [129, 31] demonstrated the need to carefully craft the topology for the problem at hand (see also Section 2.3.2). On the theoretical side, the recent studies on deep representations, as developed in [22] for instance, proved that indeed, some types of topologies (e.g. shallow one-layer perceptrons) require an exponential number of hidden units w.r.t. the input dimension, in order to be able to achieve a given learning task, while deep topologies might require as few as a linear number of layers for the same task. The critical issue then becomes that of learning the appropriate weights, and specific methods have to be used, as the standard backpropagation algorithm becomes inefficient [23].

Echo State Networks [105], that was recently proposed for supervised learning of time series, can be seen as an alternative approach: instead of optimizing a topology for a given task, it proposes to use a large reservoir of neurons that are randomly (and sparsely) connected. Only the weights of the outgoing connections are to be learned, transforming the learning process into a simple a quadratic optimization problem that is easily solved by any gradient-based method . . . at least in the supervised learning case.

The situation changes dramatically when addressing reinforcement learning tasks, such as control tasks: no example of input-output of the network are available, and hence the learning problem can no longer be set as a quadratic problem. Even some tricks like BackPropagation Through Time [223] don't directly apply to recurrent neural network learning.

Evolutionary Computation provides a possible solution for such situations, as long as some fitness for the sought controller is available

This chapter addresses the following issues: are Evolutionary Algorithms (EAs) a viable method to train Echo State Networks in general, and for reinforcement learning tasks in particular – and how does it compare to the quadratic learning in the supervised context? Furthermore, are Echo State Networks an alternative to topology learning in the framework of Control Problem? Finally, as the flexibility of Evolutionary Algorithms allows them to learn to adjust more than just the weights of the outgoing connections of the ESN, does this improve the leaning power of ESNs?

This chapter will start by introducing in Section 4.2 the Echo State Networks and the standard supervised learning of their outgoing weights. To address the first research issue listed above, the same experimental setting as the original supervised learning of time series proposed in Jaeger's seminal paper [105] will be used in Section 4.3, and different reservoir sizes will be experimented with. But because EAs don't require gradient information, using an EA in lieu of a gradient method will offer more flexibility for ESN learning, allowing us to optimize more than just the outgoing weights of the network, also optimizing the spectral radius of the ESN, or the slope of the sigmoid at the origin for each neuron.

Moreover, Evolutionary Learning opens up the field of reinforcement learning to ESNs, that is optimizing the characteristics of the network when no direct input-output examples are available, but only some possibly delayed reward, as is the case in control problems.

In the reinforcement context, many works advocating different methods to evolve a complete neural network (i.e. both topology and weights) have dealt with the double pole balancing problem, as presented in Section 4.4.1, providing a basis for comparison without the burden of re-doing all experiments. The same control problem will be used in Section 4.4, the results of ESN evolutionary learning will be compared to those of the literature. Moreover, as in the supervised case, experiments will involve the optimization of different parameters of the ESN. Section 4.5 will further discuss the results in line with the developmental and generative approaches it was compared to. Finally, Section 4.6 will sum up the chapter and sketch directions for on-going and further researches.

4.2 Reservoir Computing Model

The paradigm underlying what is today known as Reservoir Computing (RC) made Recurrent NNs accessible for practical applications as never before, and outperformed classical fully trained RNNs in many tasks [138]. Echo State Networks (ESN) have been proposed by Jaeger in 2001 [105] with the objective of endowing a neural network with rich dynamics behavioral patterns while keeping learning complexity at a low level. An ESN is a discrete time, continuous state, recurrent neural network using a sigmoidal activation function for all neurons. A typical ESN is shown in figure 4.1: the input layer is totally connected to the hidden layer; the hidden layer, and possibly the input layer, are totally connected to the output layer. Moreover, though considered not essential, the output layer can be connected backward to the hidden layer. In this setup, the hidden layer, or reservoir, is randomly generated: N neurons are randomly connected up to a user-defined density of connections δ. The weights of those connections are randomly set uniformly in $[-1,1]$, and are scaled so

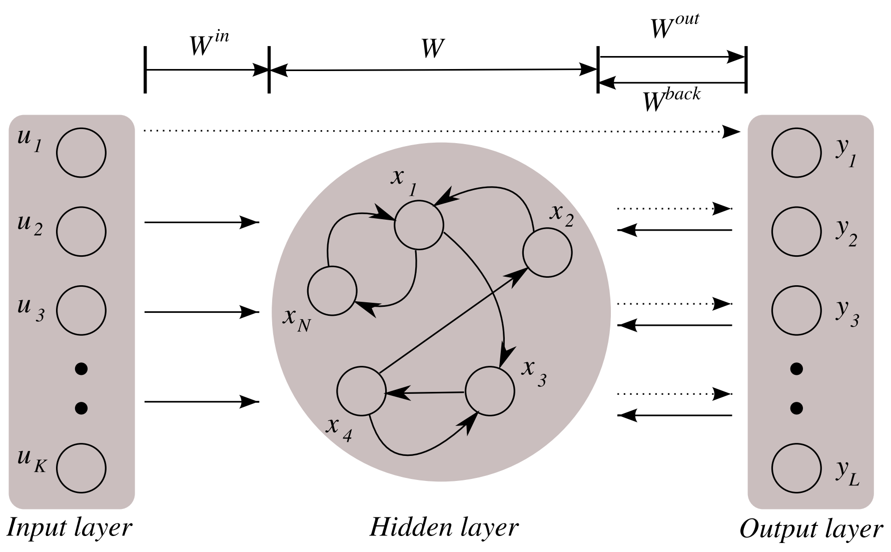


Figure 4.1: Schematic view of an Echo State Network. Plain arrows stand for weights that are randomly chosen and remain fixed, while dashed arrows represent the weights to be optimized: either $(K+N) \times L$ if direct connections from inputs to outputs are used, or simply $N \times L$ if not, where K, L and N are the number of inputs, of outputs, and of neurons in the reservoir.

that the spectral radius of the connection matrix is less than a given value $\rho < 1$, ensuring that the network exhibits the “echo state property”, i.e. stays out of the chaotic behavior zone whatever the input sequence (see e.g. [106]). The random construction of an ESN is thus determined by the 3 parameters N, δ and ρ.

The main point in ESN is that only the weights going from the input and hidden nodes to the output nodes are to be learned. If the problem has K inputs and L outputs and a reservoir of size N, this amounts either N or $(K+N) \times L$ free parameters (depending on whether or not the input layer is directly connected to the output layer). Moreover, any supervised learning problem using some MSE objective is then reduced to a quadratic optimization problem that can be efficiently and quickly solved by any deterministic optimization procedure, even for very large values of N. In some sense, an ESN can be seen as a universal dynamical system approximator, which linearly combines the elementary dynamics contained in the reservoir [163]. ESNs have been shown to perform surprisingly well in such context of supervised learning, in particular for problems of prediction of times series.

For instance, Reservoir Computing (RC) has starkly outperformed previous methods of nonlinear system identification, prediction and classification, in predicting chaotic dynamics (three orders of magnitude improved accuracy [108]), nonlinear wireless channel equalization (two orders of magnitude improvement [108]), the Japanese Vowel benchmark (zero test error rate, previous best: 1.8% [109]), financial forecasting (winner of the international forecasting competition NN3[1]), and in isolated spoken digits recognition (improvement of word error rate on benchmark from 0.6% for the previous best system to 0.2% [215]).

We will first introduce the state of arts work of ESN in supervised learning area.

4.2.1 A Chaotic Time Series Prediction by ESN

The first great success of ESN has been done with the Mackey-Glass system, a popular chaotic dynamic system that is defined by a single nonlinear time delay ordinary differential equation :

$$\frac{dy}{dt} = \alpha \frac{y_\tau}{1 + {y_\tau}^\beta} - \gamma y, \quad \gamma, \alpha, \beta > 0,$$

where $\alpha, \gamma, \tau, \beta$ are real numbers, and y_τ represents the value of the variable y at time $(t - \tau)$. Depending on the values of the parameters, this equation displays a range of periodic and chaotic dynamics. In the chaotic systems modeling community the parameters are often set to $\alpha = 0.2, \gamma = 0.1, \beta = 10$. When $\tau > 16.8$, the system is a chaotic attractor. Depending on the value of τ, the system varies form a mildly chaotic attractor ($\tau = 17$, used normally) to a more chaotic behavior ($\tau = 30$). Figure 4.2 shows 600 consecutive time steps obtained with $\tau = 17$ and $\tau = 30$

The task here is to create a system which can reproduce the given chaotic attractor as precisely as possible in a predefined number of iterations.

ESNs show excellent results in Jaeger's first technique report [105] with 400 units and state of art results with 1,000 units were also presented [108].

4.2.2 Researches on RC

Since the first report of Jaeger in 2001, Reservoir Computing (RC) attracted a lot of research interest. RC methods have indeed been successful when applied to several benchmarks, often outperforming classical fully trained RNNs. And as with all RC methods these good results were obtained with purely random (i.e.

[1]http://www.neural-forecasting-competition.com/NN3/index.htm

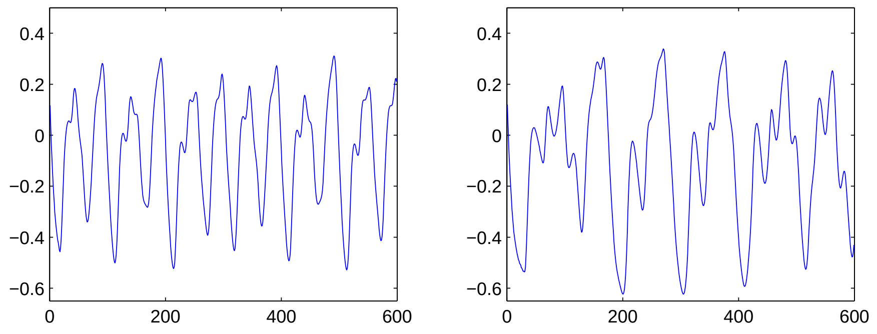


Figure 4.2: 600-step sections of the Mackey-Glass chaotic attractor system for delays $\tau = 17$ (left) and $\tau = 30$ (right).

non-optimized) reservoirs. However, the obvious success of random reservoir does not imply that they are optimal and that they cannot be improved/optimized further. Thus besides application studies, anther important stream of current RC research on reservoir methods is devoted to optimal reservoir design, or reservoir optimization algorithms.

Lukosevicius and Jaeger gave in 2007 an overview which presents most of the works in this stream [138]. We briefly present below these works, inasmuch as they are related to our own research.

Generic Reservoir Recipes

As mentioned in [105], in order to produce ’rich’ sets of dynamics, it is important to create ’big’ reservoirs (that can display rich varied dynamics), that are ’sparsely’ (reservoir neurons are only loosely coupled) and ’randomly’ connected (every generated reservoir is different from the other).

And the echo state property of the reservoir is needed to insure that the ESNs work. This condition makes sure that a previous state and a previous input should vanish as time passes in the future instead of persist or get amplified (via self-excitation). Jaeger proved that for most practical purposes, the echo state property is assured if the reservoir weight matrix W is scaled so that its spectral radius $\rho(W)$ (i.e., the modulus of the largest eigenvalue) satisfies $\rho(W) < 1$ in reservoirs using the tanh function as neuron active function, and for zero input. A rather conservative rigorous sufficient condition of the echo state property for any kind of inputs $u(n)$ (including zero) and states x(n) (with tanh nonlinearity) being $\sigma_{max}(W) < 1$, where $\sigma_{max}(W)$ is the largest singular value of W.

But this condition only ensures that the ESNs will not self-excite. The optimal value of $\rho(W)$ should be set depending on the amount of memory and nonlinearity that the given task requires, and varies with the task. Jaeger gives a rule stating that $\rho(W)$ should be close to 1 for tasks that require long memory and accordingly smaller for the tasks where a too long memory might in fact be harmful [105].

Small-world, scale-free and biologically-inspired connection topologies were tested and compared with sparsely randomly connected ones, but no significant differences were presented among there different topologies using the measure and the experiment methods of the article [134].

For a conventional ESNs, one of the shortcoming is that the activations are still coupled strongly even though the reservoirs are sparse. That makes ESNs poor performers when dealing with different time scales simultaneously. In [229], the reservoir was divided into partially decoupled sub-reservoirs, introducing inhibitory connections among all the sub-reservoirs. This system was shown to successfully resolve the multi-time scale problem at hand. Note moreover that other related works combine outputs from several separate reservoirs and will be discussed in the following section.

In conventional ESN, the activation function of the neurons is usually a Sigmoidal function, but others types of functions were also used for different proposes. For example, in Evolino [187] a Long Short-Term Memory type of RNNS were used to preserve memory for long periods of time. Jaeger [105] also suggested a version of a leaky integrator ESNs (LI-ESNs) which uses a leaky integrator active function. This version is in fact a discretized version of a continuous differential equation for a leaky integrator neuron that reads :

$$x(n) = (1-\alpha\Delta t)x(n-1) + \Delta t f(W_{in}u(n) + Wx(n-1))$$

When $\alpha = 1$ and $\Delta t = 1$, one recovers the classical (non leaky) simple ESN. The parameters α and Δt control the 'speed' of the reservoir dynamics. Small values of α and Δt result in reservoirs that react slowly to the input. By changing these parameters it is possible to shift the effective interval of frequencies in which the reservoir is working.

Readouts from the reservoirs

In conventional ESN, the readout always has a single layer. The optimization is a linear mapping form the reservoirs states $x(n)$ to target output $y_{target}(n)$ and there exists for this aim many linear regression algorithms that are very fast, even for a very large number of time steps n to forecast, as the objective is simply here to minimize the quadratic error between the $W_{out}x(n)$ and the $y_{target}(n)$.

Recently, however, some multilayer perceptrons (MLPs), trained by error backpropagation, are beginning to be used as readouts. Theoretically they are more powerful than a singer layer, but in practice, training by error backpropagation is much more difficult than linear regression with a single layer.

Another approach that has been proposed consists in combining several readouts by averaging the outputs coming from several instances of ESNs (e.g., to refine the prediction of chaotic time series [108]). There is also an ESN-based predictive classifier [200] which is reported to be much more robust to noise than a standard Hidden Markov Model for the task of spoken word recognition, by using a set of competitively trained predictors for each class of the training set, and then applying dynamic programming to find the optimal sequence of the a opoutput of these predictors.

Finally, though the reservoirs may have thousands of units, they usually have a single layer. It was suggested that for complex demanding tasks the adaptation of a single reservoir might not be enough and a hierarchical architecture of ESNs might be needed. Jaeger himself [107] proposed a first approach to this problem, but this research is still in its infancy.

4.2.3 Discussion

In our opinion, at least three aspects need further investigation in the context or Reservoir Computing in general, and ESN in particular.

Gradient Descent or Evolutionary Computing

As discussed in previous section, gradient decent algorithms have been used to optimize the readout layer in the supervised learning field. The idea of Evolutionary Learning for Echo State Networks amounts to replacing the gradient descent that is used to optimize the outgoing weights in Jaeger's approach by an Evolutionary Algorithm (EA). This makes it possible to apply ESN in Reinforcement Learning field. A first mandatory step in that direction is to validate this approach on a supervised learning task, comparing the results with those of the standard gradient method. This will be done in Section 4.3. Then a reinforcement learning task will be experimented with in Section 4.4

The Evolutionary Algorithm that has been chosen for the optimization here is the Covariance Matrix Adaptation Evolution Strategy, aka CMA-ES [87, 88, 85], which has been introduced in detail in 2.2.3. As already discussed, the only parameter of CMA-ES that needs to be set by the user is the number of offspring λ, depending on the possible ruggedness of the fitness landscape at hand. The default value, as set in [88], increases logarithmically with the dimension d of the problem (number of unknown parameters), as $\lambda = 4 + 3\ln(d)$. In the supervised case of Section 4.3, involving problems of dimensions 30, 60 and 100, the corresponding values of λ are respectively 12, 15, and 17.

Auto Parameters Turning

In many ESN applications, the ESN parameters were chosen or fine tuned by hand (most often through trial-and-error) and have fixed values. Jaeger [105] states that $\rho(W)$ should be close to 1 for tasks that require long memory and accordingly smaller for the tasks where a too long memory might in fact be harmful. But the optimal $\rho(W)$ is still unknown. One side effect of the Evolutionary Optimization of ESN weights is that we will also here use EC to determine the optimal $\rho(W)$.

Transfer Function

In conventional ESNs, the transfer function of all neurons is usually a Sigmoid function (or, equivalently, the tanhfunction) with fixed slope. But because we are using EC, and do not need any more we are also able to optimize the slopes of every neuron. And this gives another point of entry into ESN optimization. So in our study, we will also test this aspect on a supervised learning problem and on a reinforcement learning case study.

4.3 Supervised Learning of ESN

In order to validate the Evolutionary approach to ESN learning, a first experiment reproduces one of Jaeger's initial setting [105], but using an Evolutionary Algorithm in lieu of some gradient-based quadratic optimization procedure. The problem is that of a time series prediction.

4.3.1 Jaeger's Original Settings

In this toy example, a single-channel sinusoidal input is given by $u(n) = sin(n/5)$. The target is to train the network to produce a single-channel output,

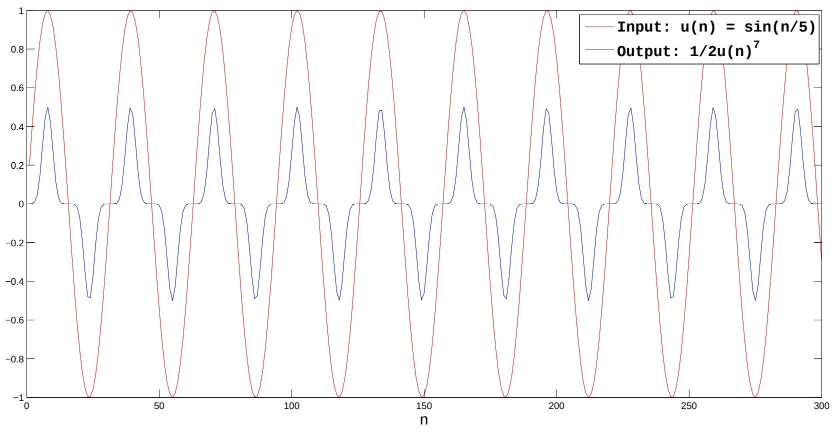


Figure 4.3: The input signal $u(n) = sin(n/5)$ in red and the target output $y_{teach}(n) = \frac{1}{2}u^7(n)$ in blue

$y_{teach}(n) = \frac{1}{2}u^7(n)$, as shown in Figure 4.3

The network output is given by the following equation:

$$y(n) = \tanh(\sum_{i=1}^{N} w_i^{out} * x_i(n))$$

where w_i^{out} denotes the weight of the i-th output connection, and $x_i(n)$ is the state of i-th neurone at time step n. The activity function is the standard sigmoid function with slope 2:

$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

The reservoir used in Jaeger's original paper has a size of 100 neurons. The neurons of the reservoir are randomly connected, and the weights are set to values of 0, +0.4 and -0.4 with probabilities 0.95, 0.025, 0.025 respectively, thus reaching a sparse connectivity of 5% (further works on ESNs rather used uniform initialization of the non-zero weights in $[-1,1]$). The weights are then scaled so that the spectral radius of the connection matrix of the reservoir is $|\rho_{max}| \approx 0.88 < 1$.

The input weights (from the input to all neurons in the reservoir) are set to $+1$ or -1 with equal probability. No direct links from inputs to outputs, and no backward links from outputs back into the reservoir are used here either.

As in Jaeger's work, the fitness to be minimized is the Mean Square Error of the network. The bootstrapping is done the following way: the states of all neurons in the reservoir is initialized to zero; the network is then run for 100 time steps without measuring the error; finally, the MSE is computed on the next 200 time steps as

$$MSE_{train} = \frac{1}{200} \sum_{n=101}^{300} (y(n) - \mathrm{arctanh}(\mathrm{y_{teach}(n)}))^2$$

4.3.2 Which parameters to optimize?

In Jaeger's original paper, only the output weights were optimized, leading to a quadratic optimization problem of dimension N, the size of the reservoir. The optimization was done using a gradient method (more precisely, the Fit function from Mathematica©). The reported result is a training error $mse_{train} \approx 3.3 * 10^{-15}$. When the trained network was tested on test data, the reported test error is $mse_{test} \approx 3.7 * 10^{-15}$.

However, as already mentioned, whereas gradient method can only solve reliably convex problems, CMA-ES can reliably find good global quasi-optima of non-convex problems, leading to different other option to choose which parameters to optimize, as already demonstrated beneficial in the case of feed-forward or small recurrent NNs [189, 54].

The Spectral Radius

In particular, a critical parameter in ESN tuning seems to be the maximal value allowed for the spectral radius. To ensure "echo state" property, this radius must be smaller than 1, but different values have been proposed in the literature for different problems. Hence it seems a good idea to use the spectral radius as a free parameter, to be optimized by CMA-ES: it only adds one dimension to the problem. The procedure goes as follows: when a set of parameters is to be evaluated, the weights of the recurrent connection inside the reservoir are first scaled so that the spectral radius of the connection matrix takes the value of the additional optimized parameter, the same way it was done during the initialization of the reservoir itself. Then the response of the ESN is computed as usual, using the scaled weights.

The Sigmoid Slopes

Because an Echo State Network is a set of dynamical systems that are linearly combined to generate the desired output, it seems plausible that modifying the slopes of all neurons independently might allow the ESN to better fit the target. The transfer function of in internal neuron becomes $\tanh_a(x) = \frac{2}{1+e^{-ax}} - 1$. Jaeger's original sigmoidal function was tanh, corresponding to the case $a = 2$.

However, if both the output weights and the sigmoid slopes are optimized, the dimension of the optimization problem is doubled. Hence, a first experiment was to optimize the slopes alone (and the problem still has as dimension the number of neurons in the reservoir), before running the complete experiment, optimizing both the slopes and the output weights.

4.3.3 The experiments

Four variants of evolutionary optimization of an ESN have been compared: the standard optimization of the output weights, denoted Std in the following, which has N unknown parameters, N being the reservoir size; the optimization of the output weights plus the spectral radius, denoted Rho, of dimension $N+1$; the optimization of the sigmoidal slopes denoted Slopes, of dimension N; and the optimization of both the output weights and the slopes, denoted Full, of dimension $2*N$.

Two reservoir sizes have also been chosen: $N = 100$ as in Jaeger's original paper, that then provides a basis for comparison, and $N = 30$, that allows more systematic comparison, in particular regarding the Full variant, as no meaningful better results could be obtained using that variant in the case $N = 100$.

However, because of the small reservoir size, the average connectivity was increased, to ensure that each neuron has the same average number of connections than in the $N = 100$ case, leading to setting the weights to values of 0, +0.4 and -0.4 with probabilities 0.864, 0.068, 0.068 respectively. The same maximal spectral radius of $|\rho_{max}| \approx 0.88 < 1$ was used for both reservoir sizes (except for the Rho variants of course).

4.3.4 Comparative Measures

One surprising issue in Jaeger's original paper is that there is no statistical analysis whatsoever: indeed, even if the optimization problem is quadratic, and has a unique global solution that the gradient method will reliably find, the network itself is randomly built, and different networks might have different optimal value for the MSE fitness.

In any case, because CMA-ES, like all EAs, is a stochastic optimization procedure, no strong conclusion can be drawn without doing a thorough statistical analysis of the performances. Hence 15 different networks have been used, and for each network, 5 runs of CMA-ES were launched with different random seeds (and hence starting points).

A first global performance measure is given by the now standard SP1 measure, an estimator for the success performance used in [85] and analyzed in [8]: it is defined as the number of evaluations needed to reach a given fitness level, divided by the proportion of runs that did reach that fitness value, and hence can be viewed as the computational effort that is required to reach a given performance level.

Moreover, the variability over the reservoir topology will also be assessed, by studying in detail the different results obtained by the 5 runs for each of the 15 networks.

4.3.5 Results

Figures 4.4 and 4.5 show the SP1 plots for reservoirs of 100 and 30 neurons, respectively, and all evolution variants, except the Full method that could not be applied for the reservoir size of 100. The first result, for a reservoir size of 100, simply confirms that CMA-ES can be as precise as the gradient method reported in [105], though undoubtedly requiring a much greater computational effort: indeed, a factor of 7 to 10 in the computational cost between CMA-ES and BFGS was reported in [88] for quadratic functions ... when using numerical derivation – whereas the analytical gradient is available here. In a few cases, though, CMA-ES was able to find better solution than our local gradient-based method, that was stopped by precision thresholds.

However, the two other plots on Figure 4.4 are rather disappointing, as increasing the search space didn't allow to improve the precisions: neither the

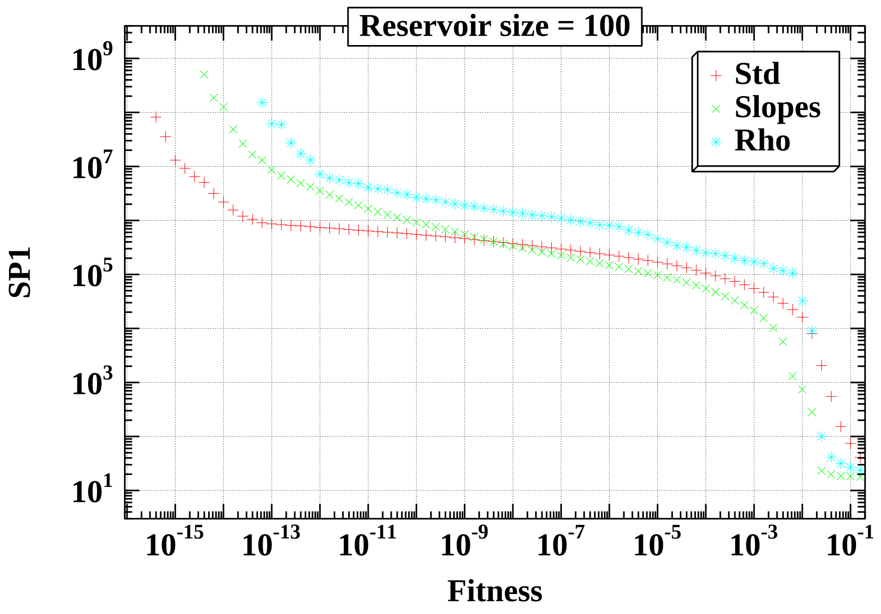


Figure 4.4: Comparative SP1 measures for the case $N = 100$, in log-log scale. The Full variant is not represented.

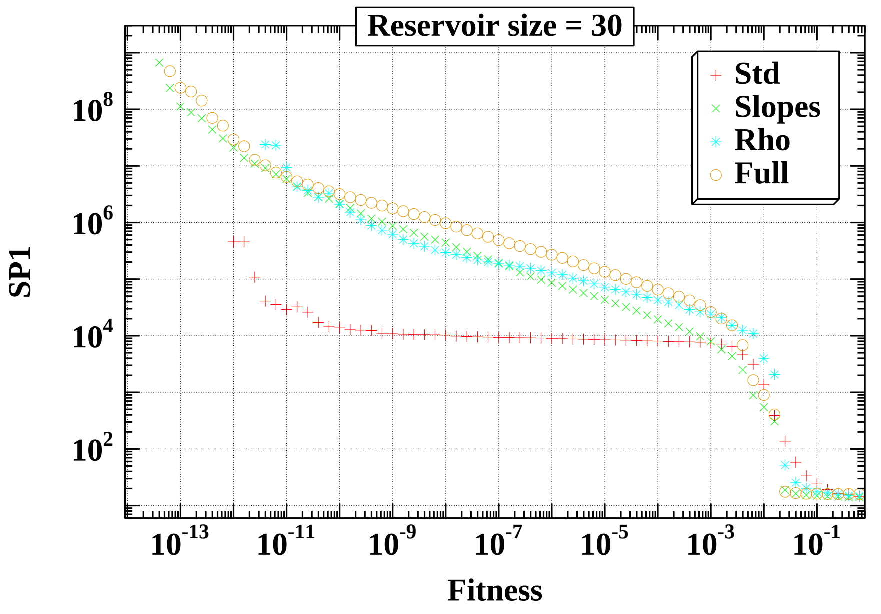


Figure 4.5: Comparative SP1 measures for the case $N = 30$, in log-log scale, for all 4 variants.

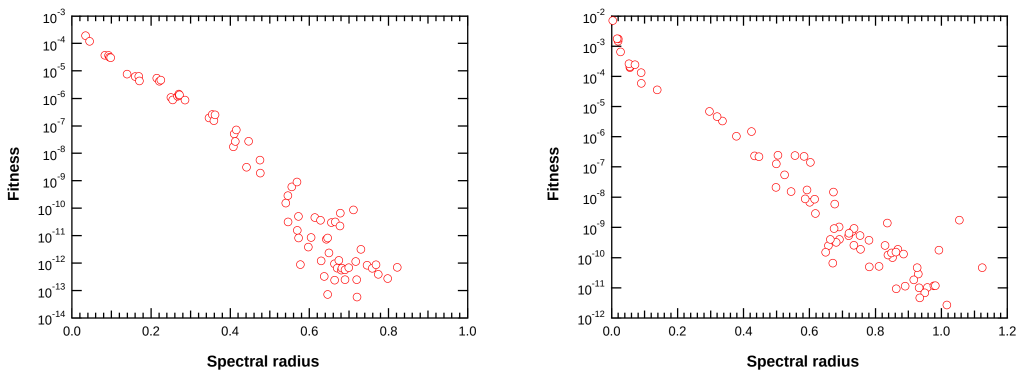


Figure 4.6: Correlation between the final value of the spectral radius at the end of a run of the Rho method and the corresponding fitness, in the case $N = 100$ (left) and $N = 30$ (right). The best fitness in the case $N = 100$(left) is found where the spectral radius is between 0.6 and 0.8, and in the case $N = 30$(right), the optimal spectral radius is centered around 0.96

Slopes nor the Rho variants are able to reach a better precision, the Rho method being the worse.

The situation is different on Figure 4.5: In the case of a reservoir size of 30, the Std method could not go below 10^{-12} while the Slopes and the Full variants were able to reach values below 10^{-13}. This, however, has a cost, and requires almost 100 times more evaluations when measured with SP1 (due to the fact that very few runs do find such low values). However, here again, the Rho variant didn't produce any good result, requiring about a 100 times larger computational effort than the Std method to reach its best value, slightly below 10^{-11}.

A possible explanation is that, though the Rho variant solves a problem that is only one dimension larger than that of the Std method, the additional variable is interacting intricately with all other variables, whereas the quadratic problem solved by the Std method is separable. Figure 4.6 confirms this hypothesis demonstrating a clear correlation in both $N = 100$(left) and $N = 30$(right) versions between the final value of the spectral radius at the end of a run of the Rho method and the corresponding fitness: low ρ have a poor fitness, indicating that CMA-ES had some hard time increasing variable ρ, and found an easier descent direction by increasing the output weights. This is confirmed by the average absolute values of the outgoing weights: in the case of 100 (resp. 30) neurons in the reservoir, the average of the absolute values of all outgoing weights over all runs of the Rho variant is 1.7×10^3 (resp. 6.3×10^3), while it is of order

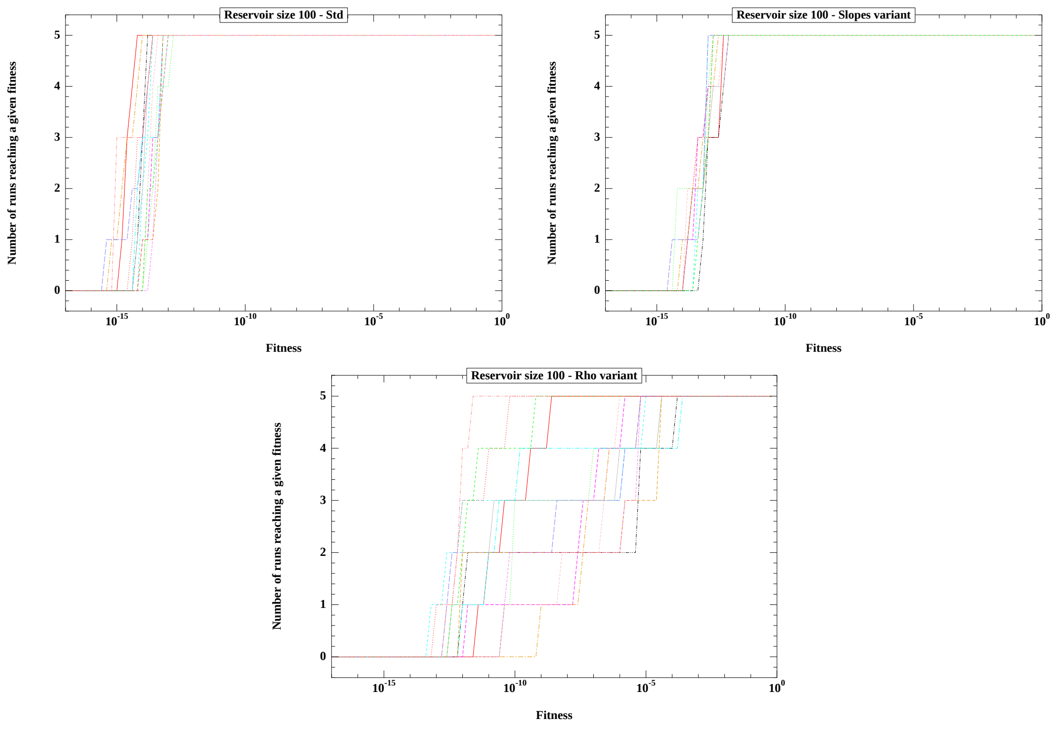


Figure 4.7: Cumulative figures with $N = 100$ neurons, each line of those plots represents the 5 runs for one network, the Y-axis being the number of runs that actually reached the X-axis fitness value. The top left figure is the Std variant, the top right is the Slopes variant and the bottom is the Rho variant

of magnitude 10 for all other variants and reservoir sizes.

This is confirmed by the other plots of Figure 4.7 and Figure 4.8 : each line of those plots represents 5 runs for one single network, the Y-axis being the number of runs that actually reached the X-axis fitness value. All variants (for both sizes of the reservoir) except the Rho method, show almost no dispersion (i.e. the performance of CMA-ES on each network is independent of the starting point) while the Rho method displays a very large dispersion. Another remark on those plots is that, for a given method, the range of optimal values reached for the MSE is larger in the case $N = 30$ than in the case $N = 100$, even for the Std variant. However, even in the case $N = 100$, the best MSE reached by the Std or the Slopes variants for different networks can vary by almost 2 orders of magnitude from the best network to the worst.

4.3.6 Discussion

The result show that:, for such a simple time series problem, the size of ESN is a crucial parameter for getting a better learning precision. When the size of ESN is large enough (as with $N = 100$), the dynamical states within the ESN are already rich enough for solving the problem, and the Std version can obtain the best results. But when N is small (e.g., $N = 30$), it is more important to create rich dynamic states by varying the spectral radius (Rho variant) or the slopes (Slopes variant). A large N may be not a problem for such supervised time series problem, but as will be detailed in next section, it will be in the case of reinforcement learning, when the simulation is very costly.

Another conclusion for this section is the interesting correlation between the final Rho value and the fitness. We also need to mention that in the case $N = 100$, the ρ used by Std version is taken from Jaeger's suggestion, and in that point it is already an optimized value. For choosing an optimal ρ for a specific task, Jaeger made a generic suggestion: if we need short memory in a dynamic environment, the ρ should be small, and vice versa. The optimal ρ is often chosen by trial and error. In our experiments, the optimal ρ in Rho version for $N = 100$ is very closed to the ρ value which is chosen by Jaeger. This suggests that EAs are a possible method for choosing the optimal ρ for a given task.

4.4 Reinforcement Learning of ESN

4.4.1 The Pole-balancing Benchmark

This Section introduces the Pole-balancing problem and briefly recalls comparative results that have been obtained by different Neuro-Evolution techniques in the recent years, for getting a more detailed comparison between the Neuro-Evolution techniques surveyed in section 2.3.2, and also for preparing the experiments on evolving ESN for reinforcement learning.

The pole-balancing problem (aka inverted pendulum) is a classical control task that has been used as a benchmark for reinforcement learning for more than 40 years [183, 2, 225, 78, 71, 204, 101, 45, 70, 117]. In the early research, there was only one pole connected. The car position (x), car speed ($\dot{x}$), the joint angle (θ_1) and the joint angle speed ($\dot{\theta}_1$) are normalized and become the inputs of the controllers. However, at the end of last century, this problem finally proved to be rather easy to solve. So in order to increase the difficulty, the speed information of car and pole were removed from the inputs. This did not add much to the difficulty, and people thus turned to the double pole problem, adding a second pole with different length to the system. In this case the two poles have to be

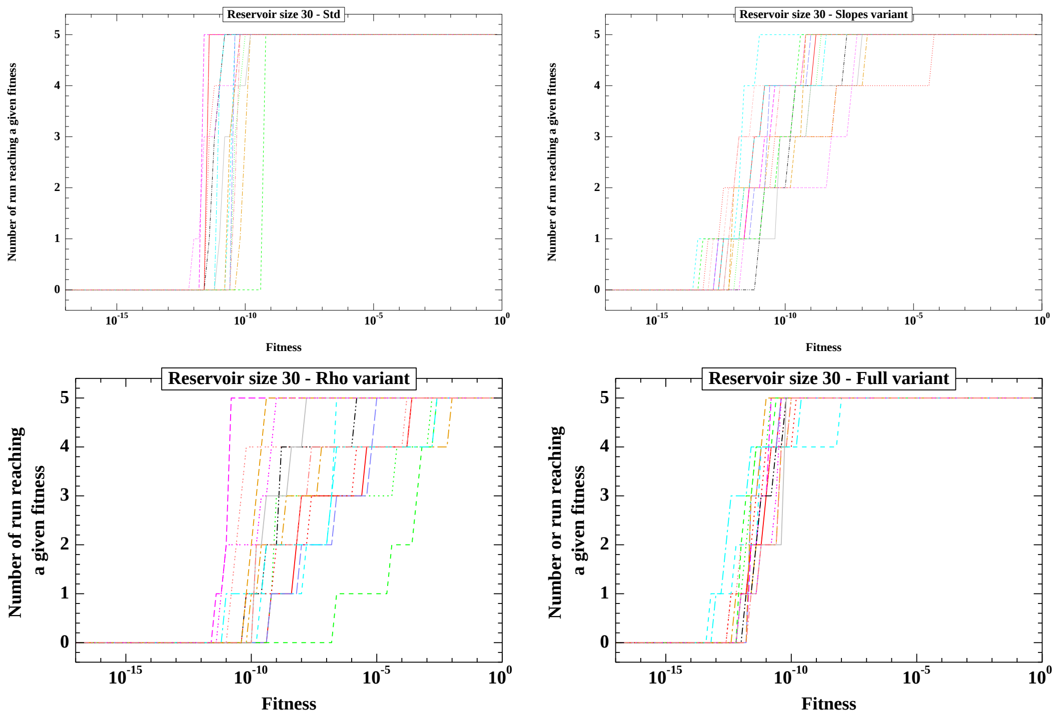


Figure 4.8: Cumulative figures with $N = 30$ neurons, each line of those plots represents the 5 runs for one network, the Y-axis being the number of runs that actually reached the X-axis fitness value. The top left figure is the Std variant, the top right is the Slopes variant, the bottom left is the is the Rho variant and the bottom right is the Full variant.

balanced simultaneously. Similarly, and again in order to increase the difficulty, the speed information are removed from the inputs. This final problem, also known as the double pole balancing without velocity problem, is the most difficult version of currently used pole balancing problems.

The double pole balancing problem without velocity problem is used by Stanley and Miikkulainen [203] to compare different neuroevolution methods that evolve both the topology and the weights of neural networks: Gruau's Cellular Encoding (CE, [78]), Gomez and Miikkulainen's Enforced Sub Populations (ESP, [71], Stanley and Miikkulainen's Augmenting Topologies (NEAT, [203]). More recently, Dürr, Mattiussi and Floreano have used the same test problem to compare their system AGE [45] with the previous approaches, while C. Igel [101] has evolved the weights of a totally connected recurrent neural network with fixed topology for the same task. A brief presentation of there methods has been given in section 2.3.2.

In all experiments (in this chapter as well as in previous works), the ordinary differential equations that model this mechanical system were integrated numerically using a fourth-order Runge-Kutta method with a constant step size of 0.01s.

The system consists of two parts, the first part is a car whose mass is 1kg and has one degree of freedom along the x axis, the second part consists of one or two poles of different lengths ($l_1 = 1m, l_2 = 0.1m$) and different masses ($m_1 = 0.1kg, m_2 = 0.01kg$) that are connected to the car by hinges. The poles have one degree of freedom, namely the joint angle (θ_1) (resp. θ_2) with the vertical direction. The command is a force F_x ($F_x \in [-10N, 10N]$) that is applied to the cart and the challenge is to keep the poles up (i.e. within given bounds for the joint angles) as long as possible.

But this criterion leads to some problems since some good solutions fall quickly when the initial conditions have changed. In order to study the generality of solution, a new evaluation criterion has been proposed. An individual passes the first test if it succeeds in keeping the system into the success domain during 100 000 time steps. It is then tested with a true generalization test, involving 625 different initial positions from where the controller must balance the system and stay within the success domain for 1000 time steps. The initial conditions are chosen such that the normalized values for x, $\dot{x}$,θ_1 and $\dot{\theta}_1$ are $\in 0.05, 0.25, 0.5, , 0.75, 0.95$ (the 625 values correspond to a full factorial Design Of Experiment). And if the best individual in the population succeeds for at least 200 of those 625 trials, the run is stopped, and the individual that passed those 200 trials successfully is returned as the solution. The number of trails passed by the solution is also called the Generalization.

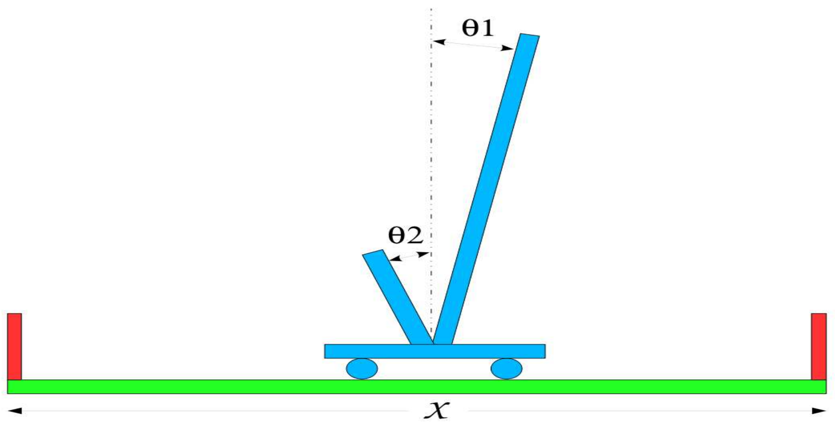


Figure 4.9: Balancing double poles

Comparative Results of Neuro-Evolution methods

For the double poles balancing without velocity problem, the optimizing results of different models are as following (different methods are compared by their average number of Evaluation for finding the success solution):

Table 4.1: Published performance results of evolving ANNs methods on the double pole balancing with no velocity information and evolving the weights and topology simultaneously. The number of neurons is fixed during the optimization (3 and 11 are the smallest and largest number of hidden neurons reported by Igel).

Method	# Eval.	Std Dev.	Generalization
CE [78]	840000	n.a	300
ESP [71]	169466	n.a	289
NEAT [204]	33184	21790	286
AGE [45]	25065	19499	317
Igel [101] $3N_{hidden}$	6061		250
Igel [101] $11N_{hidden}$	25254		226

4.4.2 Fitness(es)

The idea of the fitness function in the double pole experiment, as in all control problems, is that an individual that maximizes the fitness should have very good generalization capabilities. However, standard fitness in such situations usually average the performance of the controller over many trials from different starting positions, an extremely costly procedure. In order to avoid such heavy computational cost, many, if not all, previous works in the evolutionary literature addressing the double pole balancing problem [78, 71, 204, 101, 45] have used a simplified fitness where the controller performance only depends on its ability to maintain the poles up during a single trial. Only the best individual in the population (for this fitness) is then evaluated through generalization tests.

However, though this simplified fitness does save a lot of computational resources, we agree with [45] that it is a poor fitness with respect to the overall goal of the optimization, in that it is not clear that individuals maximizing this fitness will perform any good when it comes to generalization tests. We will hence propose another fitness that additionally captures the generalization ability of the controller.

We will now in turn introduce both fitnesses.

The “Cheap“ Fitness Function

A single trial is run for every individual in the population, starting from the same state ($\theta_1(0) = 4.5^o, \dot{\theta}_1(0) = \theta_2(0) = \dot{\theta}_2(0) = x(0) = \dot{x}(0) = 0$). The simulation is continued until either the system leaves the success domain $x \in [-2.4m, 2.4m]$ and $\theta_1, \theta_2 \in [-36^o, 36^o]$, or a maximum of 1000 time steps is reached.

The fitness function F_{cheap} is then a sum of two components:

$$F_{cheap} = 0.1\frac{t}{1000} + 0.9 f_{stable}, \text{ with}$$

$$f_{stable} = \begin{cases} 0 & \text{if } t < 100 \\ \dfrac{0.75}{\sum_{i=t-100}^{t}(|x^i| + |\dot{x}^i| + |\theta_1^i| + |\dot{\theta}_1^i|)} & \text{otherwise} \end{cases}$$

where t denotes the number of time steps the system remains inside the success domain.

The Generalization Fitness

In preliminary tests, we found that some controllers can obtain a very high fitness without ever passing the second test, while some controllers passed all 200 generalization tests with a rather low F_{cheap}. Hence we have also used a fitness that takes into account all 3 tests describe above. Of course, its computation involves all 3 tests, including the stabilization during $100,000$ times steps, and the 625 trials from different starting positions. The fitness value is then given by

$$F_{new} = F_{cheap} + \frac{\#Iter}{100,000} + 30\frac{\#Success}{625} \tag{4.1}$$

where $\#Iter$ is the number of iterations among the 100,000 where the system is maintained in the success domain, and *#Success* is the number of generalization trials that the controller passes. The value 30 was chosen by trial and error. Note that the latter means that individuals that are successful on more than $625/30 \approx 21$ generalization runs will have a high fitness now.

4.4.3 Experimental conditions

The size of the reservoir was fixed here to 20 neurons: initial experiments indicated that larger reservoirs did not improve the results – an assumption that will be revisited in Section 4.5.

To study the variability with respect to the reservoir topology, as in Section 4.3, 20 different reservoirs were generated, and 11 independent runs of CMA-ES were made for each reservoir.

Each reservoir was initialized as described in section 4.3.1, except for the fixed weights: here, the 10% non-zero weights of the reservoir as well as the in-coming weights were randomly initialized in $[-1,1]$.

At the beginning of each run, the activity of all neurons in the reservoir was zeroed, and the network was run for 20 iterations (the number of neurons) before the control actually began and the fitness started to accumulate.

As mentioned in Section 2.2.3, CMA-ES is almost a parameterless algorithm. Only the population size (in fact, number of offspring λ) is to be tuned [88]. Hence, after some preliminary experiments that seemingly demonstrated that a large population was needed, a first series of experiments was run, with $\lambda = 256$, and weird results were obtained (described below in section 4.4.4). A second series of experiments was then launched, following Igel's advice to give a lower bound of 0.05 to the step-size σ during the CMA-ES run. Indeed, the standard population size was then sufficient to obtain solutions that passed all 3 tests of the fitness function (see previous Section 4.4.2). Moreover, as explained also in that Section, both the "cheap fitness" and the new "generalization fitness" (denoted simply "new fitness" in the following) were tried for both CMA-ES settings. And for all (CMA-ES – fitness) pair, all 4 variants of the evolutionary ESN learning described in Section 4.3.2 were run – except the Slopes method with the default CMA-ES setting, for reasons that will be explained later.

4.4.4 Results and Discussion

Table 4.2: Experimental results for the double pole balancing. The five top lines refer to the ($\lambda = 256$, $\sigma_{min} = 0$) CMA-ES setting, the four bottom lines to the setting where $\lambda = 13$ takes its default value and $\sigma_{min} = 0.05$. The fitness used is the cheap F_{cheap}.

			Cheap	Fitness		
Method	Avg. Eval.	Std. Dev.	Genera-lization	% success	Avg. Force	BB Rate
Std - 0.95	16128	5127	246	8.2%	9.1	78.6%
Std - 0.60	16248	7121	250	13.2%	8.0	64.3%
Rho	35903	9695	265	25.0%	4.4	24.2%
Slopes	38214	8741	247	1.8%	**0.23**	**0**%
Std - Opt	17888	6671	264	11.8%	8.3	68.4%
Std - 0.95	**14960**	6291	234	6.8%	0.61	1.16e-5
Std - 0.60	16639	17037	225	6.8%	0.44	0.59%
Rho	23571	10175	241	**52.7%**	2.9	12.3%
Std - Opt	19168	21782	232	9.5%	0.62	3.83e-6

All results are summarized in Table 4.2 (the results using Cheap fitness) and in Table 4.3 (the results using New fitness). For each table, the CMA-ES settings is in horizontal and the corresponding fitness is in vertical, the five top lines refer to the ($\lambda = 256$, $\sigma_{min} = 0$) CMA-ES setting, the four bottom lines to ($\lambda = 13$ (default value), $\sigma_{min} = 0.05$) setting.

For each variant, the 220 runs (11 runs for each of the 20 different reservoir initializations) are here grouped together, and the four columns of each sub-table show the average number of evaluations **averaged over the successful runs**

Table 4.3: Experimental results as same as in Table 4.2 except the fitness used is the new one F_{new}. The five top lines refer to the ($\lambda = 256$, $\sigma_{min} = 0$) CMA-ES setting, the four bottom lines to the setting where $\lambda = 13$ takes its default value and $\sigma_{min} = 0.05$

			New	Fitness		
Method	Avg. Eval.	Std. Dev.	Genera-lization	% success	Avg. Force	BB Rate
Std - 0.95	27770	12522	234	44.5%	9.1	78.6%
Std - 0.60	27152	14696	236	62.7%	9.1	82.1%
Rho	35275	7481	233	**99.5%**	7.0	40.6%
Slopes	51712	13672	208	19.1%	2.0	0.18%
Std - Opt	26683	15395	243	64.5%	9.2	83.6%
Std - 0.95	16303	11511	209	82.3%	2.4	5.7%
Std - 0.60	16886	11073	211	87.3%	2.0	5.0%
Rho	19796	6770	224	91.4%	3.4	11.0%
Std - Opt	**15965**	11813	208	86.8%	2.0	4.3%

Table 4.4: Experimental results

	Standard	Fitness
Method	Success %	BB Rate
Std	8.2%	78.6%
Rho	**25.0%**	24.2%
Slopes	1.8%	**0**%
Std	6.8%	1.16e-5
Rho	**52.7%**	12.3%

Table 4.5: Experimental results

	New	Fitness
Method	Success %	BB Rate
Std	44.5%	78.6%
Rho	**99.5%**	40.6%
Slopes	19.1%	**0.18%**
Std	82.3%	5.7%
Rho	91.4%	11.0%

(column Avg Eval.), its standard deviation (Std Dev.), the number of tests (out of 625) passed during the third generalization test (Generalization), and, most importantly, the percentage of success (% success), i.e. of runs where the best individual did pass the 3 generalization tests. The other 2 columns will be detailed later.

First of all, it should be noted that only Table 4.2 (Cheap Fitness) can be compared to the published results of Table 4.4.1, because the experiments in the second column didn't use the same fitness.

The first striking result is the poor performance of the large population size with no bound on σ (first 5 lines of results of Table 4.2): only a small fraction of all runs did pass the 3 generalization tests, from 25% for the Rho method to as little as 1.8% for the Slopes method. Since the Avg. Eval. is only averaged over the successful runs and because of the poor success rate, compared to other published results (Table 4.4.1), this result is very disappointing. If we compare to other algorithms, the SP1 need to be computed to make a fair comparison. In the simplest way the SP1 can be cursorily computed as $SP1 = Avg.Eval./success\%$. (we need to notice that it is just a cursorily one because for the unsuccessful runs, their evaluation number could be more or less than the average evaluations number of success runs.). With this equation the best result is from the Std-0.60 version whose SP1 is $16248/0.132 = 123090$, it is almost 4 times larger by comparing with NEAT.

Things get a little better for the second CMA-ES setting with a lower threshold on σ (4 last lines of the Table), at least when looking at the performance of the Rho variant: more than half of the runs succeeded, with an average cost of 23571 evaluations, which amounts to an SP1 value of about 45300. This is still worse than NEAT and AGE, but within the same order of magnitude.

However, those weak success rates are to be related with the statement in [45] that "each run has to be restarted 10 times on average to find a solution". But on the other hand, NEAT [203], and Igel [101] at a very low cost, consistently found a solution that passed all tests.

As expected, the results really improve when using the new fitness, that takes into account the generalization ability of the network: with the first CMA-ES setting, the Rho variant almost always find a solution (except for one run out of 220), and the Std and the Slopes variants improve a lot over their results with the cheap fitness.

More importantly, using both the new fitness and a lower bound for σ allows all variants to reach performances at the level of those of NEAT, AGE, or the totally recurrent network with 9 neurons or more – though of course those results cannot really be compared, as they were obtained using a different fitness. Indeed, the SP1 values for Std-0.60, Std-0.95, and Rho variants respectively are 19342, 19808 and 21658, even though slightly more Rho runs fail to find a solution.

Controller Behaviors In order to try to understand those striking differences, the behavior of each controller was analyzed – and the results are summarized in the last 2 columns of each sub-table: column Avg. Force gives the average (over the 100000 times-steps of test 2 and the successful trials of 1000 time steps out of the 625 generalization test 3) of the absolute value of the force to be applied to the cart, (in Newton, in $[0, 10]$); the last column, BB rate gives the percentage of time steps where the controller gave a ”Bang-Bang“ command, i.e. a force whose absolute value is higher than 9 N.

It now clearly appears that all successful controllers that learned without a lower bound on σ exhibit a behavior that is very close to a complete bang-bang control – and this is even more so with the new fitness. On the other hand, the Rho variant manages to find intermediate behaviors, while the few successful runs of the Slopes variant can balance the double pole and pass all tests while spending an incredibly low energy (the average force is close to 0, which means the car is almost motionless during the simulation!). Note, however, that the new fitness, because it favors generalization more than decrease of movement (as does the cheap fitness alone), allows the Rho and Slopes methods to find many more solutions passing the generalization tests, but the corresponding controllers need larger forces and are “closer from bang-bang”.

When there is a lower bound on σ, however, no method whatsoever finds any bang-bang solution. Moreover, the Std method in fact exhibits controllers using smaller forces (on average) than the Rho method. Note that the Slopes variant was not tried (yet) for this setting.

Interestingly, the average squared weights reaches values as high as 10^{11} for the results of both Std variants with the new fitness and CMA-ES unbounded σ, and as low as 69 and even 7 for the Slopes and Rho variants respectively in the same conditions. But averaging here hides the most interesting phenomenon: as can be seen on Figure 4.10, that shows the average histograms of forces values: the Std variants are clearly bang-bang, the Slopes method demonstrates a very smooth behavior with all forces around 0, and surprisingly, the Rho method shows both modes, indicating that it found both kinds of solutions.

Those findings give some hints about the actual fitness landscapes, and might help to explain the results.

Fitness landscapes From the above study on the weights of the solutions, it seems that bang-bang solutions are frequent, deep and wide local optima w.r.t. the output weights: indeed, they correspond to regions of the search space with very high values of all weights, giving saturated output neurons, and small modifications of those weights are not likely to move the search away from such local optimum. This explains why CMA-ES, without a lower bound on σ, prematurely converges into one of those wells. Adding a lower bound on σ prevents a too early stagnation, and favors exploration, thus allowing the algorithm to find better optima for the fitness at hand …and in the long run, smooth optimizers are better optima than bang-bang commands.

The situation is different for the Slopes variant, that cannot reach any of the bang-bang commands, because its output weights are fixed (with low values). Hence it can only find smooth controllers, but its degrees of freedoms are too limited to make this search efficient. The most interesting case is that of the Rho method: it seems that adding this additional variable gives the algorithm sufficient flexibility to be able to discover both kinds of solutions (see Figure 4.10), thus favoring exploration increases its chances to find smooth controls. In some sense, even with a small σ, modifying ρ might be enough to find a shortcut toward a smooth solution later in the run than by modifying all weights individually.

Spectral Radius It has always been advocated by ESN pioneers that the upper bound on the spectral radius was an important feature for successful ESN use, and the results for both Std variants with different spectral radius seem to confirm this. In Table 4.2 of the version with Cheap fitness, if we don't set the low bound for σ, the success rate of Std-0.60 is better than the success rate of Std-0.95. (13.2 % vs 8.2 %) while in Table 4.3 when the new fitness is used, the success rate of Std-0.60 is also better than the success rate of Std-0.95. (62.7 % vs 44.5 %).

However, the most remarkable fact here is that for all settings (CMA-ES tuning and fitness), the Rho variant, that explicitly optimizes the Spectral Radius, almost always gives the best results. This is surprising when comparing to the situation in the supervised context (Section 4.3.5), where the Rho variant performed the worst of all.

Further experiments were run, using the Std variant but fixing the Spectral Radius to the final value found by the Rho method (see the lines "Std – Opt" in

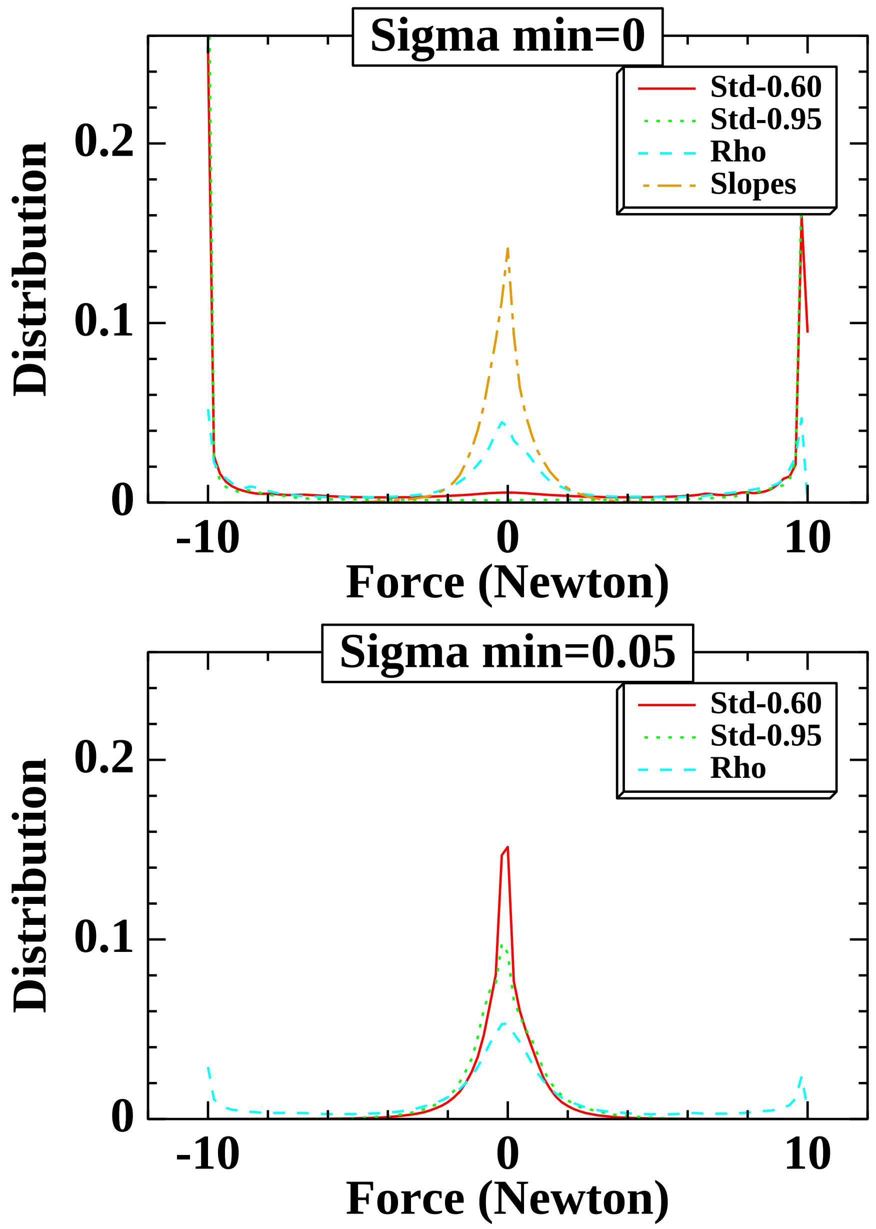


Figure 4.10: The distribution of the output force (averaged over the successful trials out of the 625 generalization tests) for the different variants. The high peaks at both ends of the domain illustrate a bang-bang behavior, while a peak around 0 demonstrate a smooth control. The Rho variant is the only one to find both behaviors (for different runs) whether or not σ is lower-bounded or not (see text for more details). The above are the results for the ”cheap fitness“, but the results of the new fitness exhibit the same tendencies.

Table 4.2). Though it generally slightly improves the results over an arbitrary value like 0.6 or 0.95, it does not allow to reach the same level of performance than the Rho method itself. The important feature is that ρ can be modified during the optimization, and not the final value it reaches.

Figure 4.11 show the final ρ value for the success runs of four Rho versions, the top two figures are the result with the cheap fitness, the left one with $\lambda = 256$ and $\sigma_{min} = 0.0$ and the right one with $\lambda = 13$ and $\sigma_{min} = 0.05$, each color represent an ESN. In this case there are not many runs can find the success solutions, but with $\sigma_{min} = 0.05$ the results is already much better.

The bottom two figures are the result with the New fitness, the left one with $\lambda = 256$ and $\sigma_{min} = 0.0$ and the right one with $\lambda = 13$ and $\sigma_{min} = 0.05$, we can find clearly that there are much more success runs than with the cheap fitness. This result may indicate that in our test, each randomly created ESN (with some constraints) has the ability to solve the double-pole balancing without velocity problem, if given enough information.

This result is proved by the following test: if evolution (using the new fitness) is continued after the first network has passed the 200-tests of test3: all resulting networks are able to successfully solve more than 500 out of the 625 test cases, with a peak at 555 for one network. This is a good advantage about the Rho variant, which seems to be able to provide controllers that generalize very well. Unfortunately, the other published results do not provide results of this kind, except for one sentence in [45] that mentions that one network successfully solved 525 test cases.

Finally, Figure 4.12 shows the final ρ and the number of evaluation of the best Rho version, where $\lambda = 13$ and $\sigma_{min} = 0.05$ by using New fitness. Each color represent a ESN, the dot is the average of the optimal ρ and the number of evaluation, the bars is the range of the different runs. One can see that each network has its specific optimal value for ρ, and that each network does behave differently (the range of optimal ρ, the range of number of evaluation, etc.) even though all the learning parameters are identical. This suggest that the topology of ESN might have an important role for such a learning task, at least when the size of ESN is small.

CMA-ES settings It has been said (Section 2.2.3) that CMA-ES was almost a parameterless algorithm. However, the experiments presented above are another example of the dangers of black-box use of any algorithm: because of the characteristics of the fitness landscape, the best results were obtained when using CMA-ES in a very atypical way, favoring exploration and forbidding

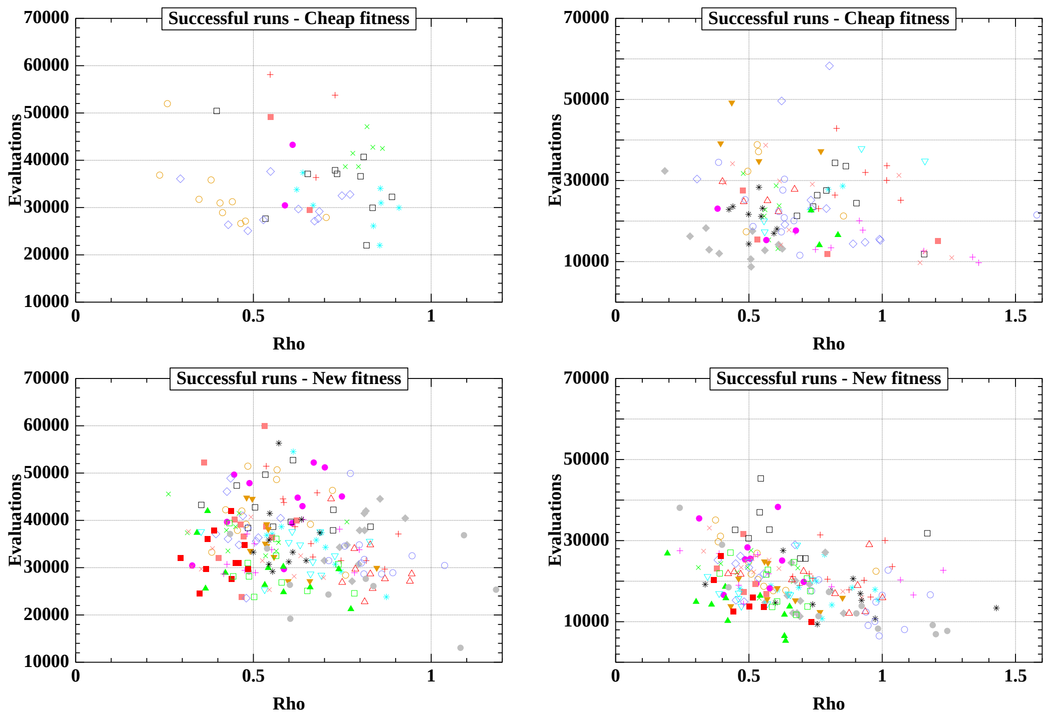


Figure 4.11: Rho variant , final values of ρ for the runs which successfully find the solutions, with $\lambda = 256$ and $\sigma_{min} = 0.0$, Cheap (top left) and new (bottom left) fitness, $\lambda = 13$ and $\sigma_{min} = 0.05$, Cheap (top right) and new (bottom right) fitness.

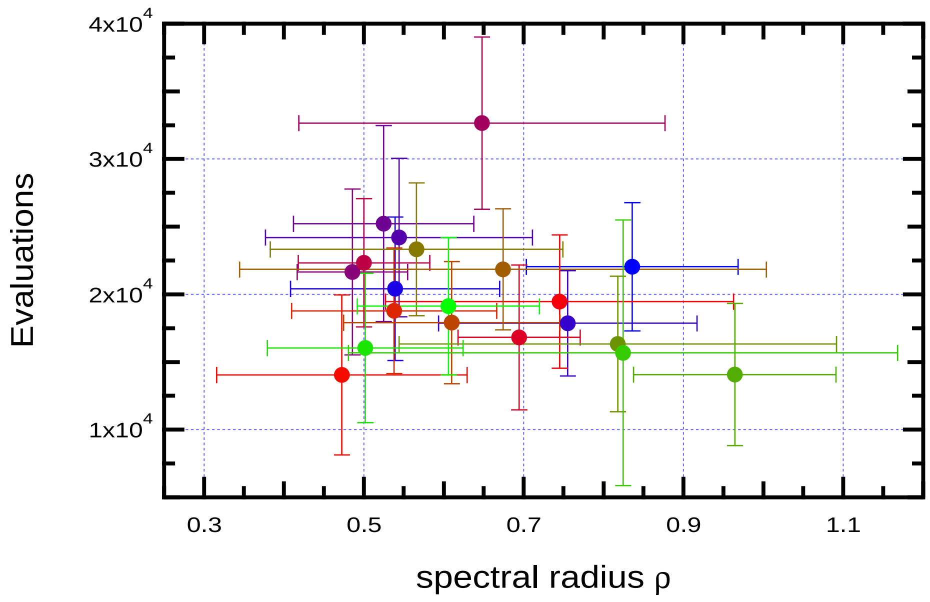


Figure 4.12: The final ρ and the number of evaluation of the best Rho version, where $\lambda = 13$ and $\sigma_{min} = 0.05$ by using New fitness. Each color represent a ESN, the dot is the average of the optimal ρ and the number of evaluation, the bars is the range of the different runs.

exploitation by keeping σ far from 0. But this relates to the fitness design, not to the optimization algorithm used.

Reservoir topologies The influence of the initialization of the topology of the reservoir is clear: for instance, among the 55 runs that passed all tests for the Rho variant with CMA-ES first setting (25% of the 220 runs), 40 were obtained from 4 reservoirs out of 20, and 10 reservoirs could not generate a single success during the 11 runs that were launched. Together with the differences noted in the supervised learning context (though their statistical significance could not be measured due to the too small number of experiments), this makes a clear picture that the topology of the reservoir matters. Why, and how to take advantage of this fact, is the object of further work.

4.5 ESN vs Developmental Methods

Quite different results could be observed on the double pole problem for different (random) settings of the connections (i.e. the non-zero weights), for the same value of the density of connection. The question is now open: whereas reservoir computing has been proposed as a possible alternative to fine tuning of the topology of Neural Networks, it might be the case that tuning the topology of the reservoir allows to obtain more efficient ESNs. Further work will address this research question, and two main directions can be imagined.

The network can be built using classes of topologies (e.g. small world, scale free, . . .); identifying classes of network that are efficient for a given type of problem (i.e. such that randomly built networks from this class have a very high probability to solve the problem at hand) would indeed relieve the programmer from the task of optimizing the topology, restricting the search space to a parameter space, where CMA-ES proved to be an efficient tool.

It might be the case, however, that for reservoir computing, problem-specific topology tuning is nevertheless required anew for each problem. The main difficulty will then be to design efficient techniques for tuning the topology of large networks, as most existing methods do not really scale up to hundreds of neurons or more. Some hints have been recently given with Hyper-NEAT [201] on the one hand, and with the different approaches based on Genetic Regulatory Networks, starting with AGE [45], though other GRN approaches can be envisioned, too (see [156, 157] and the discussion in Section 2.3.2).

But the benefit of using such approaches still remains to be uncovered, and in particular special care will need to be paid to comparing reservoir approaches with the straightforward fully connected recurrent network approach: as witnessed by Igel'results for the double pole balancing problem [101], the latter, thanks to CMA-ES, is able to solve quite a large class of control problems that were at some point thought to be difficult.

4.6 Conclusion

This chapter has demonstrated the feasibility of evolutionary learning of Echo State Networks. Moreover, the flexibility of EAs opens new paths for optimizing not just the outgoing weights of the ESN.

In a supervised context, the results on a standard time series prediction problem reach the same precision when optimizing the output weights than the original results obtained using quadratic optimization, and other optimization fail to improve this precision. With a smaller reservoir, however, optimizing also the slopes of the transfer functions of all neurons allows us to reach better prediction accuracy, at a high computational cost, though.

For reinforcement tasks, the good news is that the Evolutionary Learning of ESNs works. Moreover, optimizing more than just the outgoing weights does improve the results. Furthermore, there seems to be a high dependency of the results on the topology of the reservoir, at least for the small sizes experimented with here.

Hence, the results presented here do not satisfactorily answer the question of where ESNs stand between the two extremes of neuroevolution today: on the one hand, the evolutionary optimization of the weights of a fully recurrent neural network (as proposed in [101]), with which evolutionary learning of ESNs shares the ease of implementation (a straightforward application of the now state-of-the-art CMA-ES for parameter tuning); and on the other hand, the carefully crafted developmental systems that evolve the topology of highly efficient NNs for a given task [204, 45]. Further experiments using more reliable test problems, and using larger reservoir sizes, are needed to definitely address this issue.

Indeed, a side take-home lesson from this chapter regards the usefulness of the double pole balancing problem as a benchmark for evolutionary control in general: the answer is clearly negative for us now (but had been claimed by others before), at least with the kind of fitness used up to now to tackle the problem. The lack or correlation between that fitness and the generalization tests that the system is asked to pass to be declared successful introduces a too high random part in the evaluation of the results, as there is no promise that the good fitness will lead to the good generalization, and also as they are many ways to minimize the fitness, leading to unpredictable generalization properties.

Chapter 5

Feature Selection in a Cortex-Like Object Recognition Model

In this chapter, we will introduce our work regarding a visual object recognition model which is inspired by the biology of visual cortex. This model, originally proposed by T. Serre and T. Poggio [198], is basically a feed-forward hierarchical neural network that alternates between template matching stages and maximum pooling operations. The long term goal for us is to study the relationship between the model's topology and performance. But for the reason of high computation cost, the first step will be the question of how to decrease the computational cost while preserving the system performance, using a multi-evolutionary algorithm and applying the result in the framework of a PASCAL visual multi-object recognition challenge [140] (VOC2008).

In this model, the network output consists of standard features that are passed to a linear combination algorithm for the final classification task itself. The coefficients of this linear combination will be obtained by evolutionary optimization, using CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [87, 88, 103] detailed in Section 2.2.3.

The results show that optimizing the selection of 200 optimal features among 1000 randomly chosen ones, we are able to maintain the performance of the system on the VOC2008 challenge, while decreasing the computational cost by 2 orders of magnitude. Even if our results in VOC2008 are not very strong from the perspective of their performance on the classification tasks at hand, considering the simplicity of the model being applied shows that there still is a large room for progress in the future: we simply used the default model settings, and there are lots of parameters that could be adjusted.

5.1 Cortex-Like Object Recognition

Complex visual object recognition is an important domain in computer science, in both theory and practice. However, in spite of decades of hard work, the best machine vision systems are still not competitive with humans and primates [99, 166, 219, 198]. Therefore building a bio-inspired computer vision system is an attractive research direction.

Serre and Poggio proposed a cortex-like feed-forward model [194, 195] which was directly inspired by the image processing mechanisms of cortex and showed excellent results in a series of image classification tasks [196, 198]. In this section, we will briefly present their model.

5.1.1 Visual Object Recognition in the Cortex

The primary regions for the treatment of visual information by the cortex of primates (regions $V1$ and $V2$) are located in the occipital lobe, in the rearmost part of the skull [115]. Visual information is then transmitted to higher-level treatment regions through two distinct (albeit partly connected) neural routes : the ventral and dorsal streams. The dorsal stream ($V1 \rightarrow V2 \rightarrow MT \rightarrow$ posterior parietal cortex), sometimes called the “where pathway” is associated with object location in space and motion, while the ventral stream ($V1 \rightarrow V2 \rightarrow V4 \rightarrow$ inferior temporal cortex), sometimes called the “what pathway” is associated with form recognition and object representation. Being interested here in object recognition/classification tasks, we will emphasis the latter.

As a first approximation, basic processing of information in the ventral stream is feedforward, at least for the first stages of visual processing. This hypothesis is supported by the short time spans required for a selective response to appear in the inferior temporal (IT) cortex cells [165]. Indeed, as emphasized in [100]: “Through a variety of recognition tasks, the activity of small neuronal populations in monkey IT contains surprisingly accurate and robust information just over very short time intervals (as small as 12.5 ms) and only about 100 ms after stimulus onset”.

The ability to learn and categorize scenes as well as the objects within is an essential and important functionality of human. In this framework, the immediate recognition task consists in recognizing an object within a scene in a very short time delay. For this task in particular, it has been suggested that the ventral stream may have invariance properties for the object to be recognized, which means that if the same object is located at different positions in space, or zoomed in or out within the scene, the system is still able to recognize it. The processing is probably based on a rapid and parallel detection of disjunctive sets of unbound features of the target category [212, 228] : an object is first characterized by a set of line segments that compose it; the process then detects each line segment in parallel; the object is then recognized through the distribution in space of the detected line segments. More recently, a psychophysical experiment [52] found that during immediate recognition processing, spatial information on the location of the objects may be absent, which would mean that the dorsal stream is not implicated in such immediate recognition tasks. Moreover that means the human observers may correctly detect a target object in a rapid sequence of images, but not be able to recover even its approximate location [52].

Another fundamental property of the ventral pathway is that neuron tuning becomes increasingly complex along the feedforward path. Neuronal tuning designates the fact that any given neuron responds preferentially to a subset of stimuli within its receptive field. In the primary visual areas, neurons have simple tuning. For example, a neuron in V1 may fire to any vertical stimulus in its receptive field. In the higher visual areas, neurons have more complex tuning. For example, in the inferior temporal cortex (IT), a neuron may fire only when a certain object appears in its receptive field.

Based on these research results, Riesenhuber, Serre and Poggio elaborated a theory of the feedforward path of object recognition in cortex, that accounts for the first 100-200 milliseconds of processing in the ventral stream of primate visual cortex [175, 194]. Riesenhuber and Poggio proposed the model in 1999 and showed that it is able to duplicate the tuning properties of neurons in several visual cortical areas [175]. This first model's performance was only evaluated on simple artificial stimuli without real world image degradations such as change in illumination, clutter, etc. Hence, Serre and Poggio extended the model in 2005 and obtained successful results on a variety of large-scale real-world object recognition databases with performances that compare favorably with state-of-the-art systems, whether bio-inspired and not [194]. We will present this model in the following section.

5.1.2 Standard Model Features (SMFs)

The model itself attempts to summarize – in a quantitative way – a core of well-accepted facts about the ventral stream in the visual cortex:

1. Visual processing is hierarchical, and aims at building invariance to position

and scale first and then to viewpoint and other transformations.

2. Along the hierarchy stream, the receptive fields of the neurons (i.e., the part of the visual field that can potentially elicit a response from the neuron) as well as the complexity of their tuning properties (i.e., the set of stimuli that elicit the maximal response of the neuron) increases.

3. The initial processing of information is feed-forward (for immediate recognition tasks, i.e., when the image presentation is rapid and there is no time for eye movements or shifts of attention).

4. Plasticity and learning probably occur at all stages and certainly at the level of infratemporal (IT) cortex and prefrontal cortex (PFC), the top-most layers of the hierarchy.

In the present report, we used a simplification of the model from [194], given in [198] and that consists of four feedforward layers of neuron networks. Basically, the model has two types neurons : so-called simple S units and complex C units. The S units combine their inputs with a bell-shaped tuning function to increase selectivity. The C units pool their inputs through a maximum (MAX) operation to increase invariance. The main idea behind this model is that by carefully tuning the parameters, the model can get a good balance between selectivity and invariance.

Figure 5.1 shows a schematic illustration of the model, taken from Serre and Poggio 's article [198].

5.1.3 Model Details

S1 units:

Hubel and Wiesel first described the classical simple cells of the primary visual cortex (V1) almost 50 years ago [99]. These cells optimally respond to contrasted bars or gratings with a preferred orientation [115] . Gabor functions, described by the equation 5.1, have been shown to provide a good approximation for the response of cortical simple cells [112].

$$F(x,y) = \exp\left(-\frac{x_o^2 + r^2 y_o^2}{2\sigma^2}\right) \times \cos\left(\frac{2\pi}{\lambda} x_o\right) \tag{5.1}$$

$$\text{with } x_o = x\cos(\theta) + y\sin(\theta) \text{and } y_o = -x\sin(\theta) + y\cos(\theta) \tag{5.2}$$

In the model, the first S1 layer consists in a multidimensional array of simple S1 units, where each simple S1 unit is modeled using eq. 5.1. In the simplest model, Serre and Poggio adjusted the Gabor parameters (aspect ratio r, effective width σ,

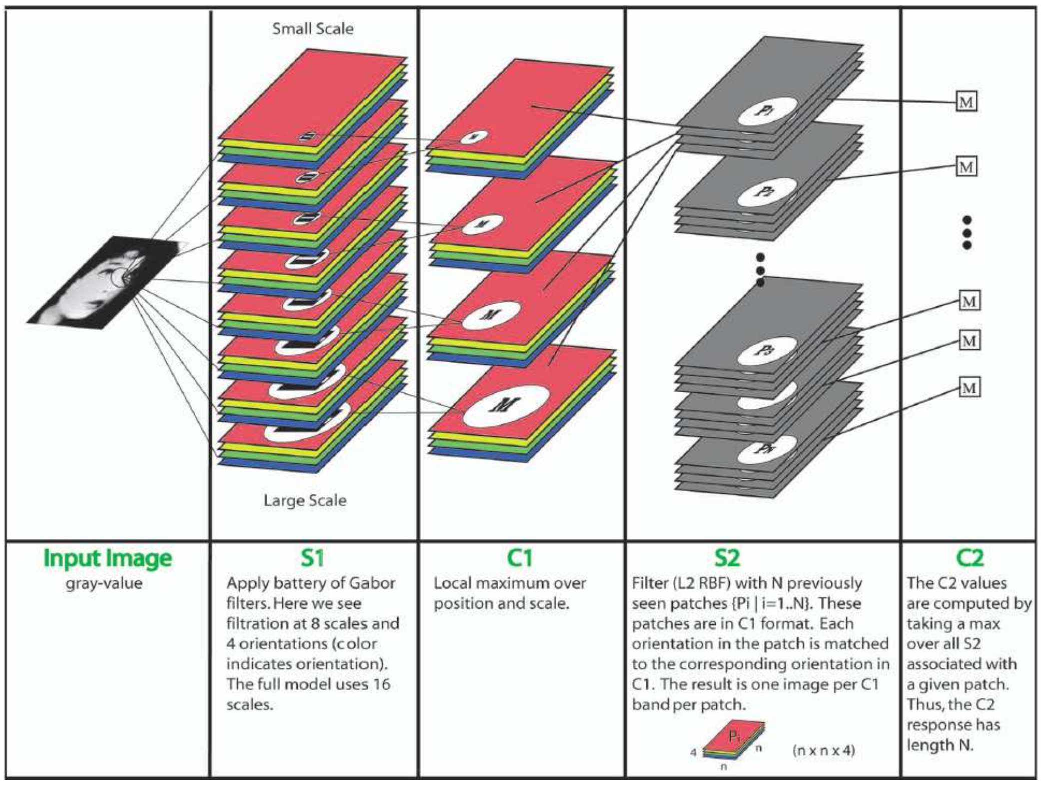

Small Scale
Large Scale
Input Image
gray-value
S1
Apply battery of Gabor filters. Here we see filtration at 8 scales and 4 orientations (color indicates orientation). The full model uses 16 scales.
C1
Local maximum over position and scale.
S2
Filter (L2 RBF) with N previously seen patches {Pi | i=1..N}. These patches are in C1 format. Each orientation in the patch is matched to the corresponding orientation in C1. The result is one image per C1 band per patch.
Pi
4
n
n
(n x n x 4)
C2
The C2 values are computed by taking a max over all S2 associated with a given patch. Thus, the C2 response has length N.
M

Figure 5.1: Scheme of the model. A gray-value image passes through the four layers - S1, C1, S2, C2, form left to right. S1 analyzes the image by orientations and 16 scales. Then at layer C1, the image is subsampled through a local MAX pooling operation over a neighborhood of S1 units in both space and scale, but with the same preferred orientation. S2 units are essentially RBF units, each having a different preferred stimulus cross all positions and scales. Finally the C2 layer performs a MAX pooling operation with the same selectivity of S2 units, yielding the C2 unit responses.

C_1 layer			S_1 layer		
Scale band $\mathcal{S}$	Spatial pooling grid ($N_S \times N_S$)	Overlap Δ_S	filter size s	Gabor σ	Gabor λ
Band 1	8×8	4	7×7	2.8	3.5
			9×9	3.6	4.6
Band 2	10×10	5	11×11	4.5	5.6
			13×13	5.4	6.8
Band 3	12×12	6	15×15	6.3	7.9
			17×17	7.3	9.1
Band 4	14×14	7	19×19	8.2	10.3
			21×21	9.2	11.5
Band 5	16×16	8	23×23	10.2	12.7
			25×25	11.3	14.1
Band 6	18×18	9	27×27	12.3	15.4
			29×29	13.4	16.8
Band 7	20×20	10	31×31	14.6	18.2
			33×33	15.8	19.7
Band 8	22×22	11	35×35	17.0	21.2
			37×37	18.2	22.8

Figure 5.2: Summary of the S1 and C1 SMFs Parameters

wavelength λ and preferred orientation θ) according to the tuning properties of V1 simple cells, that were determined experimentally by two different groups [213, 214, 186, 185]. The sizes of these Gabor filters ranged from 7×7 to 37×37 pixels by steps of two pixels. Four discrete preferred orientations ($0^o, 45^o, 90^o$, and 135^o) were used, thus leading to 64 different S1 receptive field types in total (16 scales $\times$ 4 orientations).

C1 units:

The next layer consists of C1 units. It corresponds to the so-called cortical complex cells, which already show tolerance to position and size experimentally. Because they pool the outputs from S1 cells that are neighbors in the previous layer, the receptive fields of C1 cells is twice as large as that of simple cells. Basically, the C1 cells implement a MAX operation of the S1 simple cells they are connected too. That means the response of a complex unit corresponds to the response of the strongest cell from the previous S1 layer with the same orientation and from the same scale band.

C1 units are organized in scale bands. Scale band 1 contains C1 cells connected to S1 cells with sizes 7×7 and 9×9, scale band 2 contains C1 cells connected to S1 cells with sizes 11×11 and 13×13 and so on.... Hence there are 8 scale bands in layer C1 for 16 scales in layer S1 (see figure 5.2). The response of a C1 unit in scale band 1, for instance, is computed by subsampling the output of the S1 cells of size 7×7 and 9×9 that are situated beneath the C1 cell, within a grid of size 8×8 pixels. The output of this C1 cell is just the max of these 64×2 S1 cells. Note that C1 responses are not computed at every possible pixel locations in the image: C1 units overlap by an amount of Δ_S pixels (Δ_S depends on the scale band considered, see figure 5.2). The overlap Δ_S diminishes the total number of cells in layer C1, thus avoiding too many redundant cells in this layer. This makes the computations at the next stage more efficient. Again, these arrangements were adjusted so that the tuning of the C1 units matches the tuning of actual complex cells as measured experimentally (see [197] for details).

Figure 5.3 is an example given by Serre [194] that schematizes how scale and position tolerances are obtained at the C1 level. Thanks to the MAX operation of the horizontal bar (that of the "A" letter), the C1 cell displays the same output whatever the location of the horizontal bar within its receptive fields (ie the underlying S1 cells it pools over). Tolerance to scale changes is obtained with the same mechanisms.

Learning stage:

Before using the S2 layer, N (e.g., 1,000 here) prototypes or patches are clipped and stored during a specific learning process. The learning process consists of presenting to the network to a large number of images from the database and computing the network response up to layer C1. A patch is defined as the response to a given image of all the C1 cells within a contiguous region within layer C1. During the learning stage, a large pool of patches corresponding to neighborhoods of various sizes and located at random positions, are extracted from each image. The patches are furthermore extracted across all 4 orientations, i.e., a patch P_o, of size $n \times n$ in fact represents the output of $n \times n \times 4$ C1 cells to a given image. In the following, we extracted patches of four different sizes: 4×4, 8×8, 12×12 and 16×16. The final step consists in selecting at random (uniform distribution) N patches amongst the stored ones.

Note that an important question for both neuroscience and computer vision regards the choice of the unlabeled images from which the patches are extracted as well as how to select the N patches among all the stored ones. In other words, the issue amounts to find how to learn in such an unsupervised way a vocabulary of elementary visual features. We will present below results on the use of an evolutionary method in this perspective.

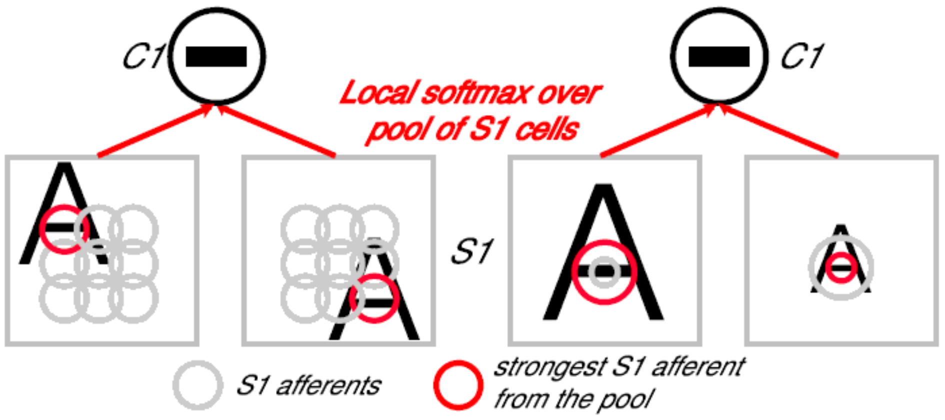

C1
C1
Local softmax over pool of S1 cells
S1
S1 afferents
strongest S1 afferent from the pool

Figure 5.3: An example given by Serre in [194] showing how scale and position tolerances are obtained at the C1 level: Each C1 unit receives inputs from S1 units at the same orientation (e.g., 0 degree) arranged in bands. For each orientation, a band S contains S1 units in two different sizes and various positions (grid cell of size $N^S_{C1} \times N^S_{C1}$). From each grid cell (see left hand side) one measurement is obtained by taking the maximum over all positions: this allows the C1 unit to respond to a horizontal bar anywhere within the grid, providing a translation-tolerant representation. Similarly, taking a max over the two sizes (see right hand side) enables the C1 unit to be more tolerant to changes in scale.

S2 units:

Each cell in layer S2 computes the Euclidean distance between one of the selected patch P_i and the response of Layer C1 to a new (unknown) image. To this aim, the response of S2 cell layer consists of a Gaussian Radial Basis Function (RBF) $\phi(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{P_i}\|^2\right)$, where $\mathbf{P_i}$ is the real-valued vector Patch i and $\mathbf{x}$ the response of layer C1 to the current image.

Each S2 cell pools over afferent C1 units from a local spatial neighborhood across all four orientations. Layer S2 thus contains $8 \times N$ maps, each representing the response of one of the 8 scale bands of Layer C1 compared to one patch P_i out of the N selected ones.

C2 units:

The final layer, layer C2, computes the global maximum of the all S2 units over all scales and positions. As explained above, one S2 map measures the match between a selected prototype P_i and the input image at every position for a given scale band. One C2 units only keeps the maximal value of the S2 maps across all scale bands for a given patch P_i. The final system output for a new image is thus a N-dimensional vector, where N is to the number of patches extracted during the learning stage. Note that one C2 unit expresses the maximum of a S2 map over the whole image, which implies that its response is not dependent on the object location within the image.

The Classification Stage:

Hence every new image passes the feedforward system and yields a N-value vector in the output of layer C2, whose elements are called Standard Model Features (SMFs). The SMFs are then further passed to a simple linear classifier (SVM or boosting in the original article). This part of the learning is a simple supervised learning, since all the labels are given for the training set, and the fitness to be optimized is just the misclassification rate.

5.1.4 The Perspectives of the Model

Serre et al. [198] list three major perspective directions that could be followed to further improve the performance of their architecture :

- Addition of extra layers (e.g., S3, C3, S4, etc.)
- Applying a standard feature selection technique.

- Auto parameters tuning through learning.

In our study (see above), we focused on the second item of this list, i.e. develop strategies to select the patches (standard features) in an optimal way: from the 1000 randomly chosen patches, we will try to select as few as possible while maintaining the classification rate. And it could significantly reduce the high computational cost of this model.

5.2 PASCAL Visual Object classes Challenge (VOC08)

The image database we used is from the PASCAL Visual Object classes Challenge 2008 [140]. The goal of this challenge is to recognize objects from a number of visual object classes in realistic scenes (i.e. not pre-segmented objects). There are twenty object classes (e.g. Figure 5.4):

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

Two main tasks are:

- Classification: For each of the classes, predict the presence/absence of at least one object of that class in a test image.
- Detection: For each of the classes, predict the bounding boxes of each object of that class in a test image (if any).

The VOC2008 database contains a total of 10,057 annotated images. The data is released in two phases:

1. Training and validation data with annotation are released with the first development kit before the competition;
2. Test data without annotation is released later. After completion of the challenge, annotation for the test data will be released.

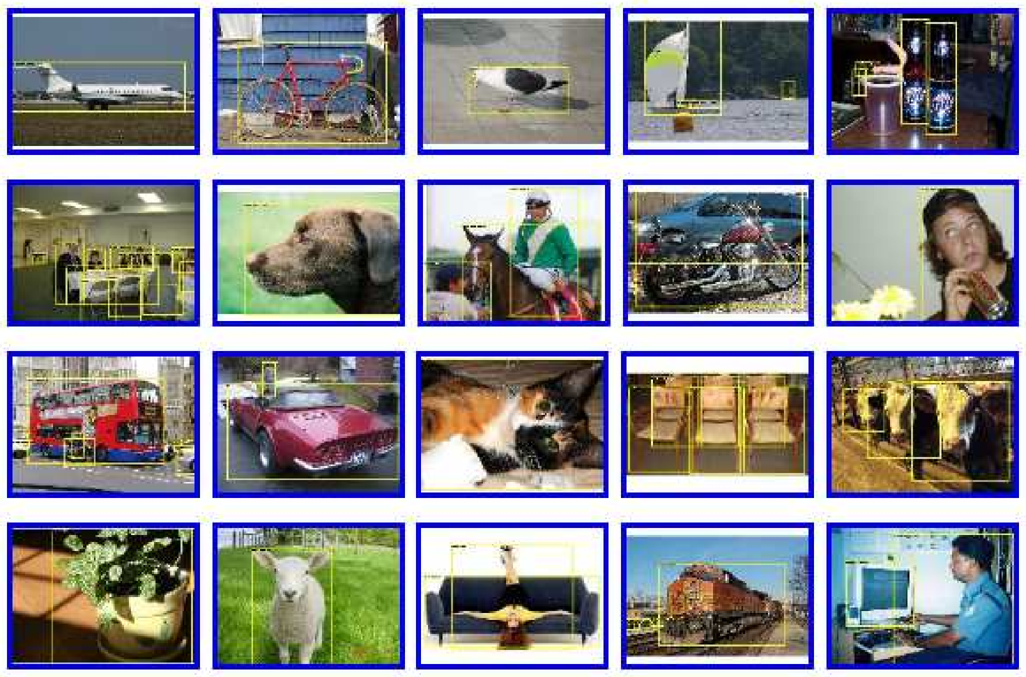


Figure 5.4: Examples of the 20 classes of VOC challenge

5.2.1 Classification/Detection Image Sets

Table 5.1 summarizes the statistics of the main image sets. Each line gives the statistic of each target class. The column “train” means training set and the column “val” means validation set, while the column “trainval” gives a sum number of both training and valid sets. The column “img” means how many images have the target class, while the column “obj” counts how many target objects exists in these images. The number of “obj” is always larger than the number of “img” since within the same image can exist more than one target object (e.g. a picture of family in the “Person” class will always have more than 2 persons (obj) in the same picture (img)).

Table 5.1 summarizes the number of objects and images for each class and image set. The data have been split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/ validation and test sets.

Table 5.1: Statistics of the main image sets. Object statistics list only the “non-diffcult” objects used in the evaluation.

	train		**val**		**trainval**	
	img	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	119	159	117	157	236	316
Bicycle	92	133	100	136	192	269
Bird	166	239	139	237	305	476
Boat	111	170	96	166	207	336
Bottle	129	229	114	228	243	457
Bus	48	61	52	68	100	129
Car	243	426	223	414	466	840
Cat	159	186	169	192	328	378
Chair	177	313	174	310	351	623
Cow	37	61	37	69	74	130
Diningtable	53	55	52	55	105	110
Dog	186	238	202	239	388	477
Horse	96	139	102	146	198	285
Motorbike	102	137	102	135	204	272
Person	947	1996	1055	2172	2002	4168
Pottedplant	85	178	95	183	180	361
Sheep	32	67	32	78	64	145
Sofa	69	74	65	77	134	151
Train	78	83	73	83	151	166
Tvmonitor	107	138	108	136	215	274
Total	2113	5082	2227	5281	4340	10363

5.2.2 Classification Task

We participated in the classification task of the VOC2008 challenge. In this task, for each of the twenty object classes, the goal is to predict the presence/absence of at least one object of that class in a test image. The output from the system should be a real-valued confidence of the object's presence so that a precision/recall curve can be drawn. And the result of the challenge was judged by the average precision (AP). The following Section explains how to compute the AP in detail.

For each image, the system is expected to output a real-value that represents the confidence of the object presence within the image and that we refer to as the VOC confidence. This VOC confidence could be interpreted as the probability that there is at least one target object with the image. For all classes, we first construct the vector $\mathbf{V_{confidence}}$, where the i-th element $V_{confidence,i}$ gives the VOC of image i. $\mathbf{V_{confidence}}$ is then sorted by decreasing order, so that $V_{confidence,i} > V_{confidence,j}\ \forall i > j$.

The next step consists in comparing the real-valued probabilities of target object in $\mathbf{V_{confidence}}$ to their real label. Two vectors, $\mathbf{V_t}$ and $\mathbf{V_f}$ are created to this aim. Each element $V_{t,i}$ indicates how many positive images (those images that indeed have the target object) exist between element 0 and element i of the sorted $\mathbf{V_{confidence}}$. $\mathbf{V_f}$ vector collects the same information for negative images (those images that do not have the target object). For instance, if there are 150 positive and 50 negative images before element 200 of $\mathbf{V_{confidence}}$, the 200th element of V_t will be 150 and the 200th element of V_f will be 50.

The final results are stored in vectors $\mathbf{V_{precision}}$ and $\mathbf{V_{recall}}$. $\mathbf{V_{precision}}$ vector registers the percentage of positive image before the same position element of $\mathbf{V_{confidence}}$: $V_{precision,i} = V_{t,i}/i$ (for the example above, $V_{precision,200} = 150/200 = 0.75$). $\mathbf{V_{recall}}$ is just $\mathbf{V_t}$ divided buy the total number of positive images in the image set. Therefore the first element of $\mathbf{V_{recall}}$ is close to 0 and the last one is always 1.

To investigate these final vectors, one can plot each element i of $\mathbf{V_{recall}}$ as a function of i. If the system can give good confidence, this curve will start from 0 and quickly increase to 1 in a very short time. $V_{precision}$ can be plotted the same way. A perfect system is expected to produce a curve starting from 1, then decreasing to the percentage of positive images.

Here we give an example of these results: let us say we have 1000 images and half of them have the target object. We illustrate in figure 5.5 how to interpret the quantifiers above with three kinds of classifiers. The first one is the best

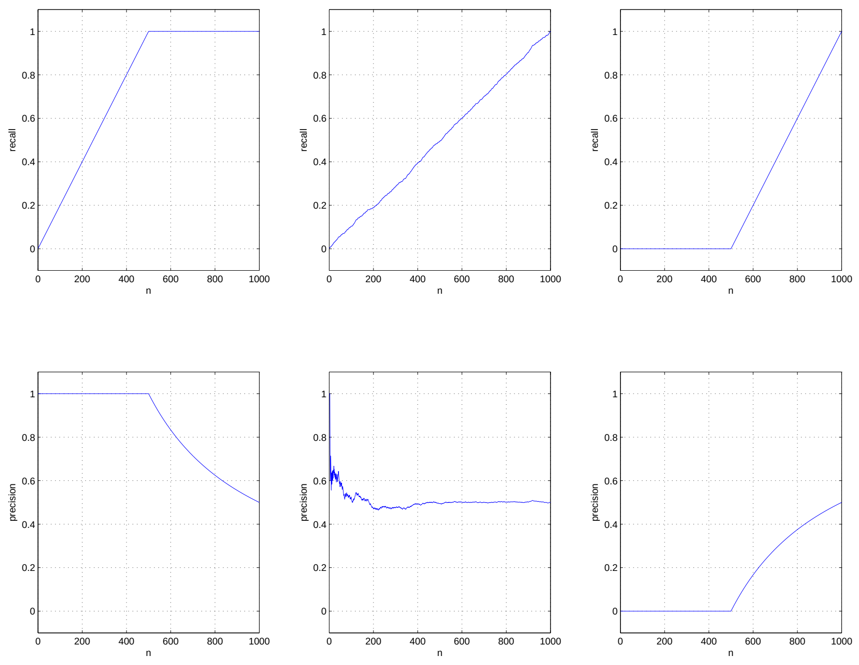


Figure 5.5: An illustration of the behavior of 3 synthetic classifiers. The top row shows the V_{recall} curves and the bottom one the $V_{precision}$ curves for the best possible classifier (first column), a random classifier (middle column) and the worst possible classifier (third column).

classifier ever possible. With this classifier, the 500 positive images are sorted in front of the list and are the first 500 elements of $\mathbf{V_{confidence}}$. In this case, $V_{recall,i} = i/500\ \forall i < 500$ and $V_{recall,i} = 1\ \forall i \geq 500$, while $V_{precision,i}$ decays rapidly from 1 to the total percentage of positive images in the set. The second classifier is the worst one possible and puts the 500 positive images at the end of the $\mathbf{V_{confidence}}$. The recall curve is zero up to 500 then increases linearly, while the precision curve is zero up to 500 then increases to the total percentage of positive images in the set. Finally, the last classifier (middle column in the figure) is just a random one, giving a random confidence to each image. The recall curve increases slowly to 1, reaching it only for the last image, while the precision curve decreases rapidly to the total percentage of positive images in the set.

An alternative representation consists in plotting the precision/recall in the same figure, i.e. plotting, for each i, $V_{precision,i}$ as a function of $V_{recall,i}$. Figure

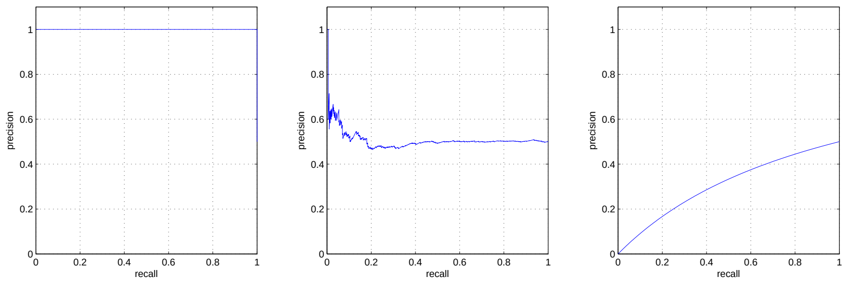


Figure 5.6: Precision/recall curve, the x-axis is the recall values and the y-axis is the corresponding precision value, the left is from best classifier, the middle is from the random classifier and the right is from the worst classifier

5.6 shows the result for the three cases illustrated above. The average precision (AP) is then computed from these precision/recall curves. In practice, the range of recall is divided into 11 sub-parts : $\{0\}$, $[0.1, 1.0]$, $[0.2, 1.0]$, ... $[0.9, 1.0]$ and 1.0. The AP is the average of the largest precision value in each sub-parts: $AP = \frac{1}{11}\sum_{i=0}^{10} \max(precision \in recall([\frac{i}{10}, 1]))$. The AP fitness is also called VOC fitness in this chapter.

For the three synthetic classifiers illustrated above, the AP fitness is :$AP_{best} = 1.0$, $AP_{random} = 0.5234$ and $AP_{worst} = 0.5$.

An example code for computing the precision/recall and AP measure is provided in the development kit [140].

Note that, in the system developed by Poggio and collaborators, the final classification stage results in a simple binary response : the input image is guessed to contain (output = 0) or not to contain (ouput=1) one object belonging to the image class under study (car, plane...). In our case however, the above mentioned definition of fitness is based on a real-valued confidence indicator and not a binary answer, e.g. whether the target is guessed to exists in the image scene or not. We thus first use an Evolutionary Algorithm to optimize a linear classifier that produces the VOC confidence for every images, then compute the AP (average precision) of the training set and validation set. For this optimization step, the fitness is simply the AP, to be maximized: the EA seeks to optimize the coefficient of the final linear classifier so as to maximize the AP. A second layer will be added on top of this EA, trying to select the patches in an optimal manner. A second evolutionary algorithm will wrap the system described above, resulting in a multi-evolutionary algorithm.

5.3 Experiments

5.3.1 Using EC Algorithms with SMF Model

As explained in the previous section, instead of giving an unambiguous Yes or No answer for the question whether there exists a target object in the test image, the VOC challenge requests an intermediate fitness (the VOC confidence) for the probability that the target object exists in the image, in order to be able to sort the ensemble of images and compute the final fitness (AP). This kind of fitness is useful for real-world applications (e.g. image search engines), but it cannot be directly applied to the conventional SMFs model with SVM classifier. Since the relationship between the VOC confidence and the AP fitness is nonlinear, and almost no prime knowledge about the relationships between them can be used directly, Evolutionary Algorithms seems to be a good choice for this kind of problem. The EC algorithm here is used to optimize a monolayer linear combinator to create VOC confidence for each image, and the final fitness used by EA is the AP fitness.

Therefore in our study, we apply evolutionary optimization using CMA-ES (see Section 2.2.3) to optimize the VOC confidence for all object classes. In a first step, we will use CMA-ES to optimize the weights of a linear combination (Section 5.3.3). But because there are 1000 features gathered from the random choice of patches in layer C1, and in order to try to reduce the complexity of the overall algorithm, we will also implement a multi-level Evolutionary Algorithm (Multi-Evolutionary algorithm for short) to optimally select a small subset (200 here) of those 1000 features while preserving the recognition accuracy. This will be presented in Section 5.3.4

5.3.2 Pre-training of SMFs model

We will first briefly present the pre-training process for the SMFs model before applying our optimization approach.

Because the VOCdevkit is released in Matlab code and the demo program released by Serre is also in Matlab code, we build our own program in Matlab on the basis of the Serre's version[1] and keep the original parameters settings. The pre-training process is as follows:

- First, all training and validation images have to be normalized to a gray

[1]The author gratefully acknowledges Thomas Serre, McGovern Institute for Brain Research, MIT, for kindly providing the Matlab code of his program

pixel image and resized so that the larger axis is 140 pixels while keeping the ratio between X and Y axes constant. This was done on all the 10,363 images (see Table 5.1).

- Second, the normalized images are processed through the S1, and C1 layers with the original parameters given in Figure 5.2. The final output for every images is saved for the next step.
- Third, for each image among the 20 classes, we select those images from the training set that contain at least one example of the class object (car, plane...). These images are used to generate 1,000 patches from the C1 outputs. With 4 patch sizes (4×4, 8×8, 12×12 and 16×16) this means that we select 250 patches per patch size at random positions of the output of Layer C1.
- Fourth, each image from both the training and the validation sets yields a S2 map by using the previous C1 patches. Since there are 1,000 patches for each class, the C2 layers will give a vector whose size is 1,000 for each object class. This vector gives the probability of the image having an object which is similar compared to the randomly chosen C1 patches.
- Finally, for every one of the 5082 training images and every one of the 5281 image from validation set, we have 20 $1,000$-dimensional vectors (20 object classes and 1000 randomly chosen patches) that can be used later for optimization.

5.3.3 Linear Combination using CMA-ES

In this first version (referred to as the CMA version in the following), each image in each object class is thus associated to a 1,000-dimensional vector, the output of the C2 layer. The goal is then to aggregate those 1000 scalar values into a single value, that will be given as the VOC confidence for the competition. The idea is to use optimize a linear combination of those 1000 values in order to optimize the VOC confidence, and an Evolutionary Algorithm is our preferred choice.

However, because a search space of dimension 1,000 is too large for a conventional CMA-ES, we will apply here a specific version of CMA-ES proposed in [103] and computationally less expensive (but also less robust), the $(1+1)$-CMA-ES. The $(1+1)$-CMA-ES generates one offspring from the unique parent and keeps the best of both – see Section 2.2.3). It uses an improved variant of the one-fifth success rule for the step-size adaptation in place of the usual path length control, and an incremental $o(n^2)$ Cholesky update of the covariance matrix replacing the original $o(n^2)$ covariance matrix update altogether with the iterative

$o(n^3)$ eigendecomposition of the covariance matrix. That make it possible to apply $(1+1)$-CMA-ES for a 1000 dimensions problem where the standard algorithm would reach its limit in terms of CPU cost.

The search space here is hence the 1000-dimensional space of the weights of the linear combinations of the original 1000 features from the C2 layer, and the fitness is the resulting AP (Average Precision) that is obtained when this linear combination is used as VOC confidence to rank the images.

In spite of using the modified $(1+1)$-CMA-ES, this CMA version has nevertheless a high computational cost because of the dimension of the search space. Hence, in order to cope with this cost, we only run one optimization for each object class (whereas CMA is a stochastic optimization algorithm, and should thus be ran several times on the same data in order to get reliably robust results).

During the optimization, the Average Precision was computed for both the training set and the validation set. Figures 5.7, 5.8 and 5.9 present the results of the first 10,000 iterations for each class (the fitness only changes very slightly after the first 10,000 iterations). The individuals with the best fitness on both the train sets and the valid sets are then used to classify the test set, and the obtained results were submitted to the VOC2008 challenge.

Looking at Figures 5.7, 5.8, and 5.9, we find that during the evolution, the Average Precision of the (unique) individual in the population increases, which supports that for such large dimensions (1,000), the $(1+1)$-CMA-ES can still find the way to optimize the solution. However, the red line of VOC fitness (AP) on validation set increases at the beginning, but then fails to continue this initial trend: the Average Precision is only marginally improved on the training set. Interestingly, the Average Precision does not decrease either: there seems to be no danger of overfitting, even though the generalization results are disappointing.

5.3.4 Multi-Evolutionary Optimization

Using and handling 1,000 standard features has a certain computational cost. In this Section, we present another approach to Average Precision optimization, that will be called the MEVO version, that tries to tackle this issue without degrading the recognition results. The basic idea is that most probably, because the 1000 features were chosen randomly at layer S1, only a few of them are useful for the recognition task. In order to try to validate our hypothesis that many features could be dropped without decreasing the recognition rate, we implemented a two-level MEVO, to optimally select 200 standard features (patches) (other trials with 500 and 900 features lead to similar results).

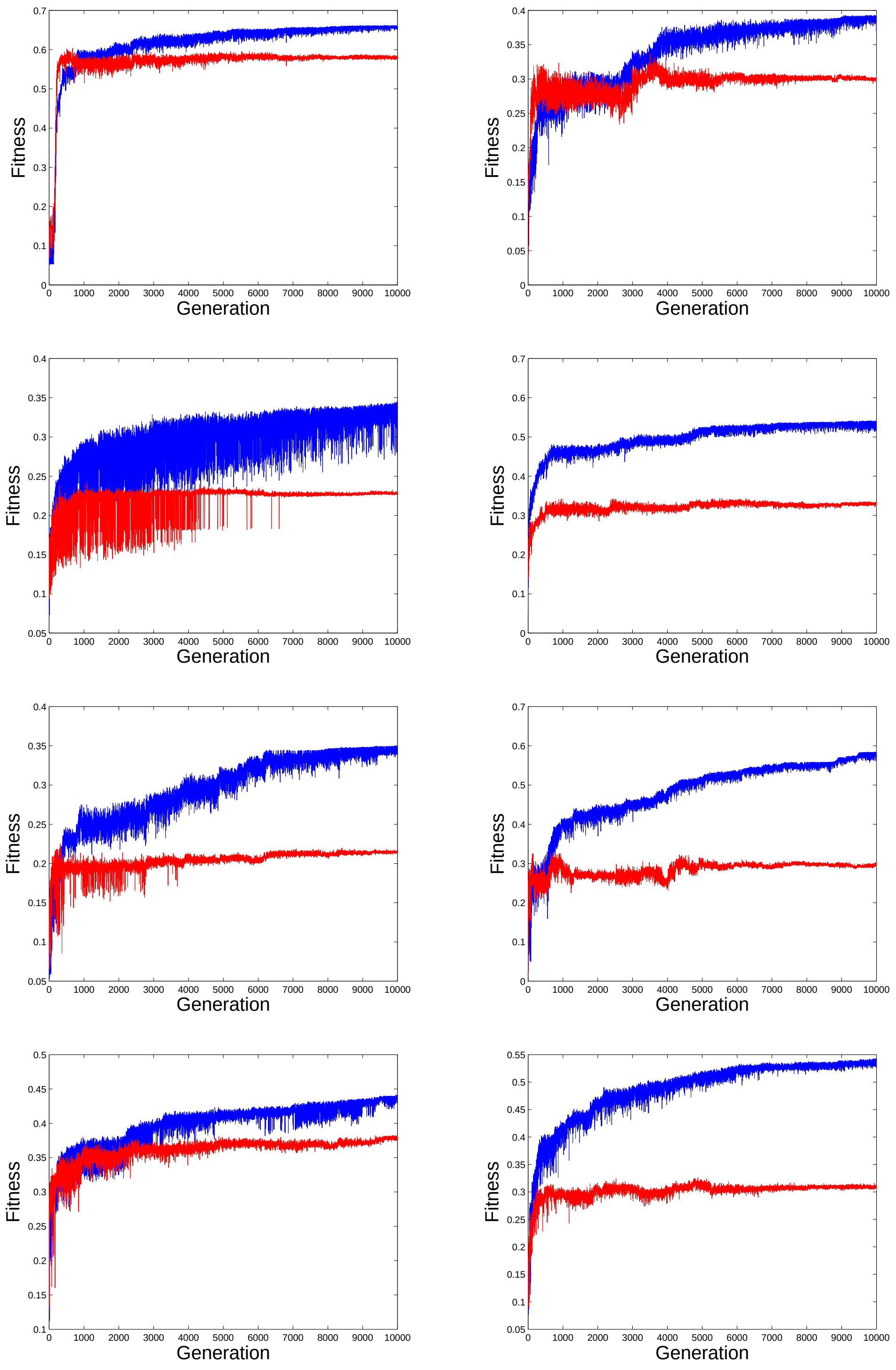


Figure 5.7: Behavior of the CMA version for classes 1-8 (aeroplane bicycle bird boat bottle bus car cat). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

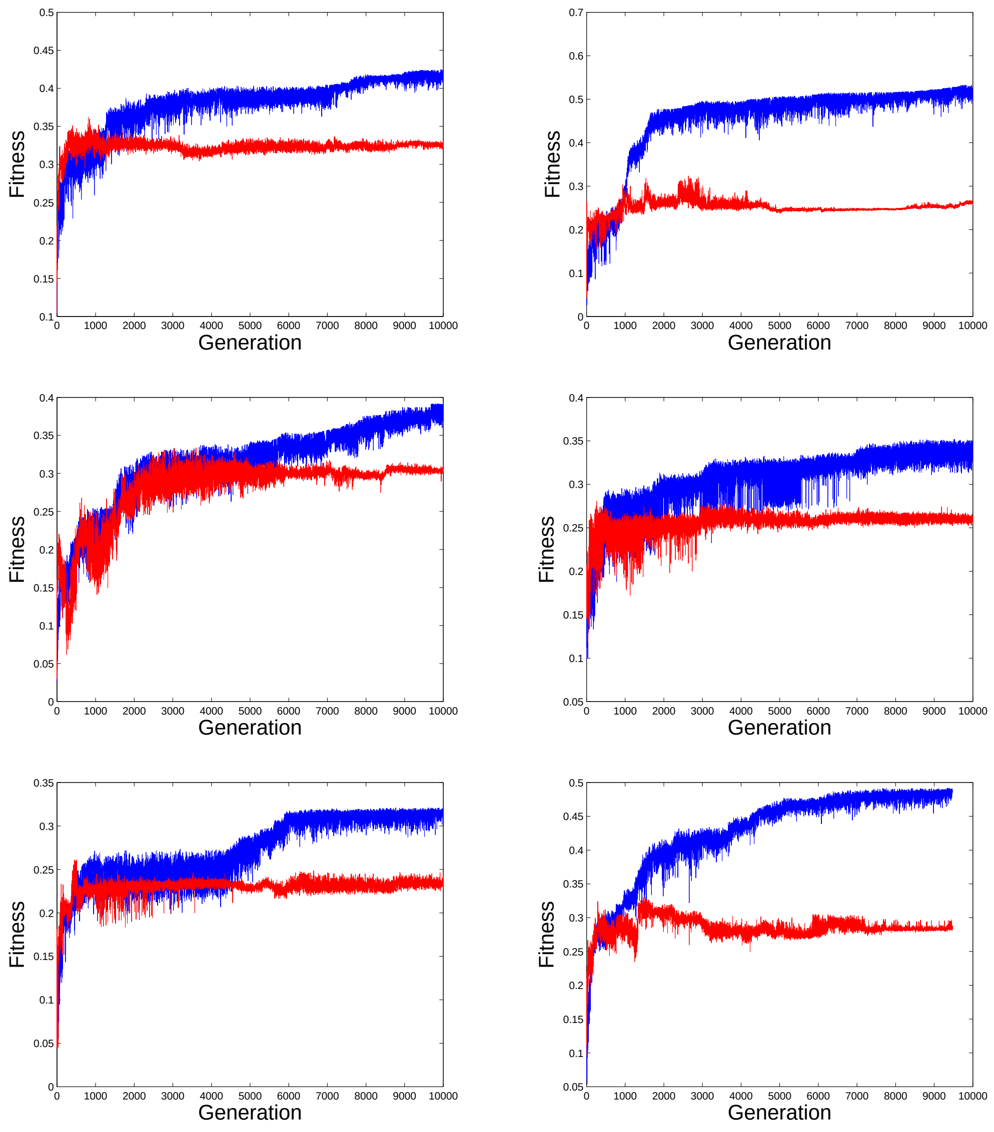


Figure 5.8: Behavior of the CMA version for classes 9-14 (chair cow dining-table dog horse motor bike). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

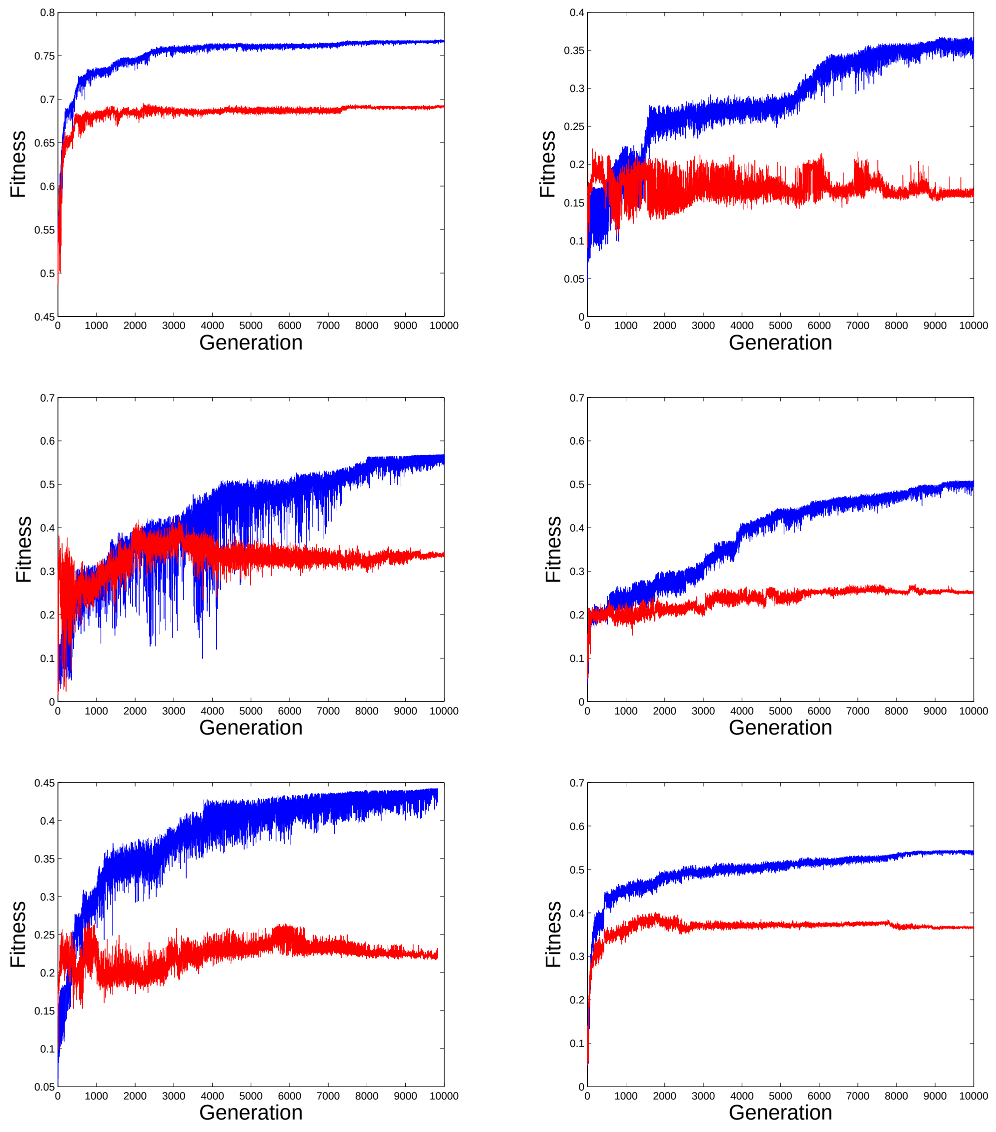


Figure 5.9: Behavior of the CMA version for classes 15-20 (person potted-plant sheep sofa train tv). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

The main optimization loop in MEVO consists in an Evolutionary Algorithm that optimizes the choice of those 200 features among the 1,000 available ones. In order to compute the fitness of any of such 200 subset of features, an inner evolutionary loop uses the same CMA-ES optimization for the CMA version to optimize the linear combination of these 200 chosen features with respect to the Average Precision. This optimized Average Precision is the fitness of the subset of 200 individuals for the outer EA.

The outer EA (the main optimization loop) searches the space of 1000-bits long strings that contain exactly 200 1's and 800 0's, a 1 indicating that the corresponding feature belongs to the subset of selected features. The evolution engine is a $(10+70)ES$: 10 parents generate 70 offspring, and the best 10 are retained to become the parents of the next generation. No crossover operator is used in our experiment. The only variation operator is the bit-exchange mutation operator which swaps a randomly chosen 1 with a randomly chosen 0 in the string. This operator makes sure that the total number of chosen features is always 200.

The fitness of a given subset (a bitstring) is computed using the CMA version above: a linear combination is obtained using the $(1+1)$-CMA-ES to maximize the Average Precision – and this optimized AP is returned as the fitness of the corresponding bitstring.

To each subset (or 1000-long bitstring) is associated a 1000-dimentional real-valued vector **a** which stores the optimal values for the weights of the linear coefficients giving the VOC confidence, result of the inner-loop CMA-ES optimization. The next application of CMA optimization on an offspring of this individual will use those stored values as the starting point of the next CMA-ES optimization in order to save computation times.

The Average Precision of the best individual of the offspring population (out of 70), both in training set (blue) and in validation set (red), are presented in Figure 5.10, 5.11 and 5.12. We find that the fitness in training set increases quickly at the beginning of the optimization, and then stops increasing. This may suggest that it is not a hard task to chose 200 optimal features from 1,000 randomly chosen features for the training set. But their generalization ability is poor as the fitness in validation set does not increase much. Furthermore, a lot of peaks are observed. This might correspond to the selection of special features, which are harmful or benefit only for part of the validation set.

Since there is no significant improvement for the best individual in validation set, the individual which has the best train+valid fitness were applied to the test set, and the results were submitted to the VOC2008

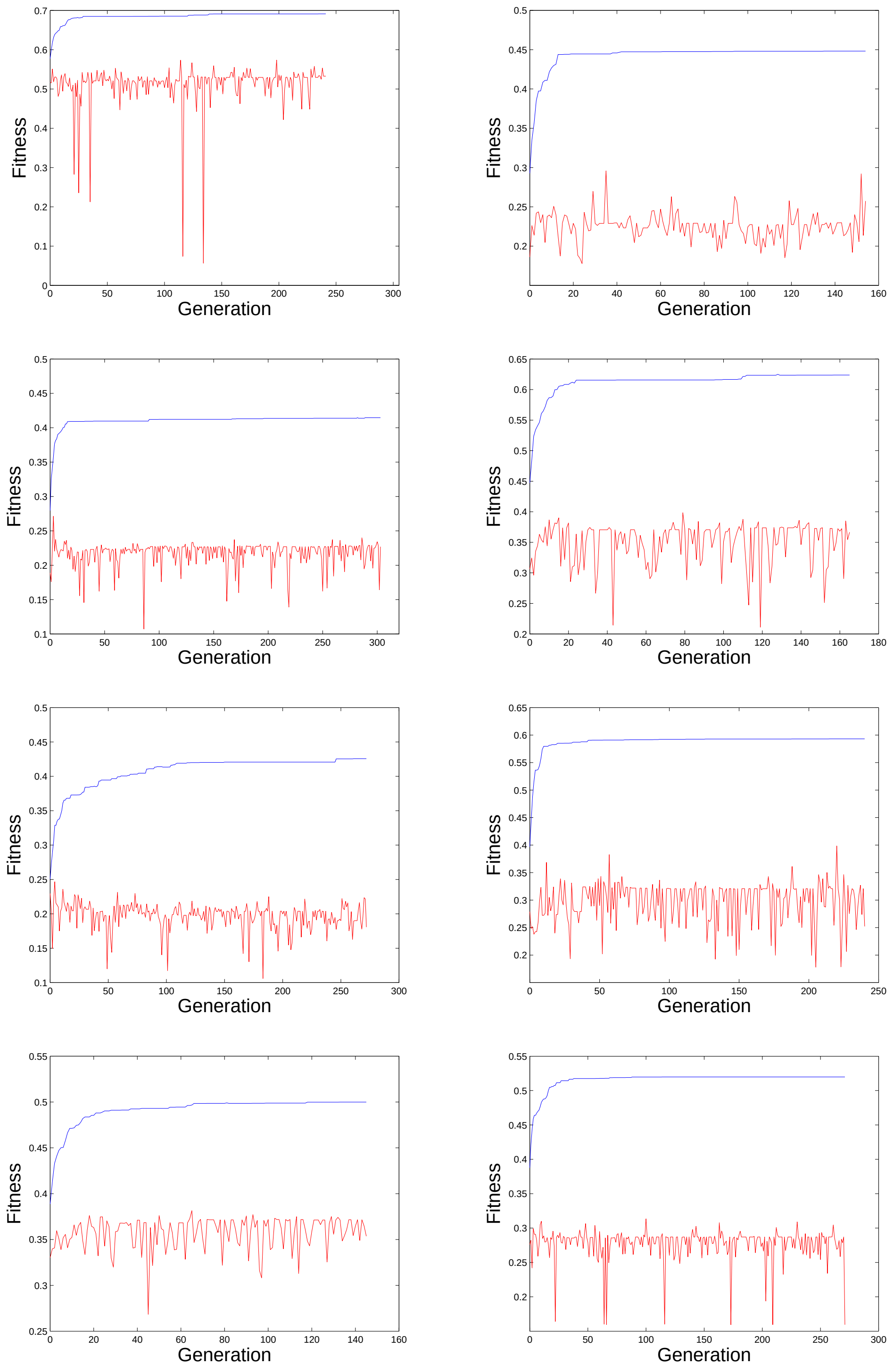


Figure 5.10: Behavior of the MEVO version for classes 1-8 (aeroplane bicycle bird boat bottle bus car cat). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

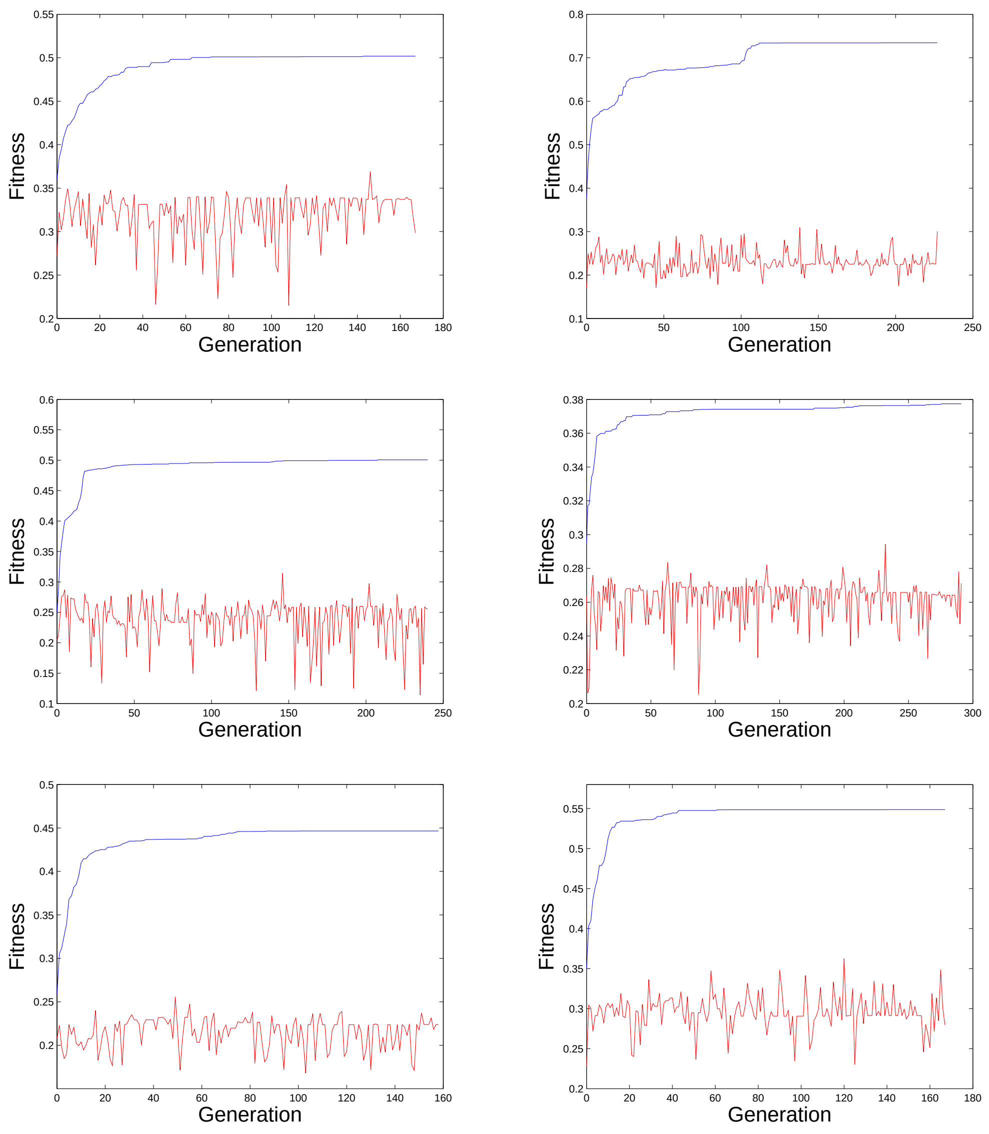


Figure 5.11: Behavior of the MEVO version for classes 9-14 (chair cow dining-table dog horse motor bike). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

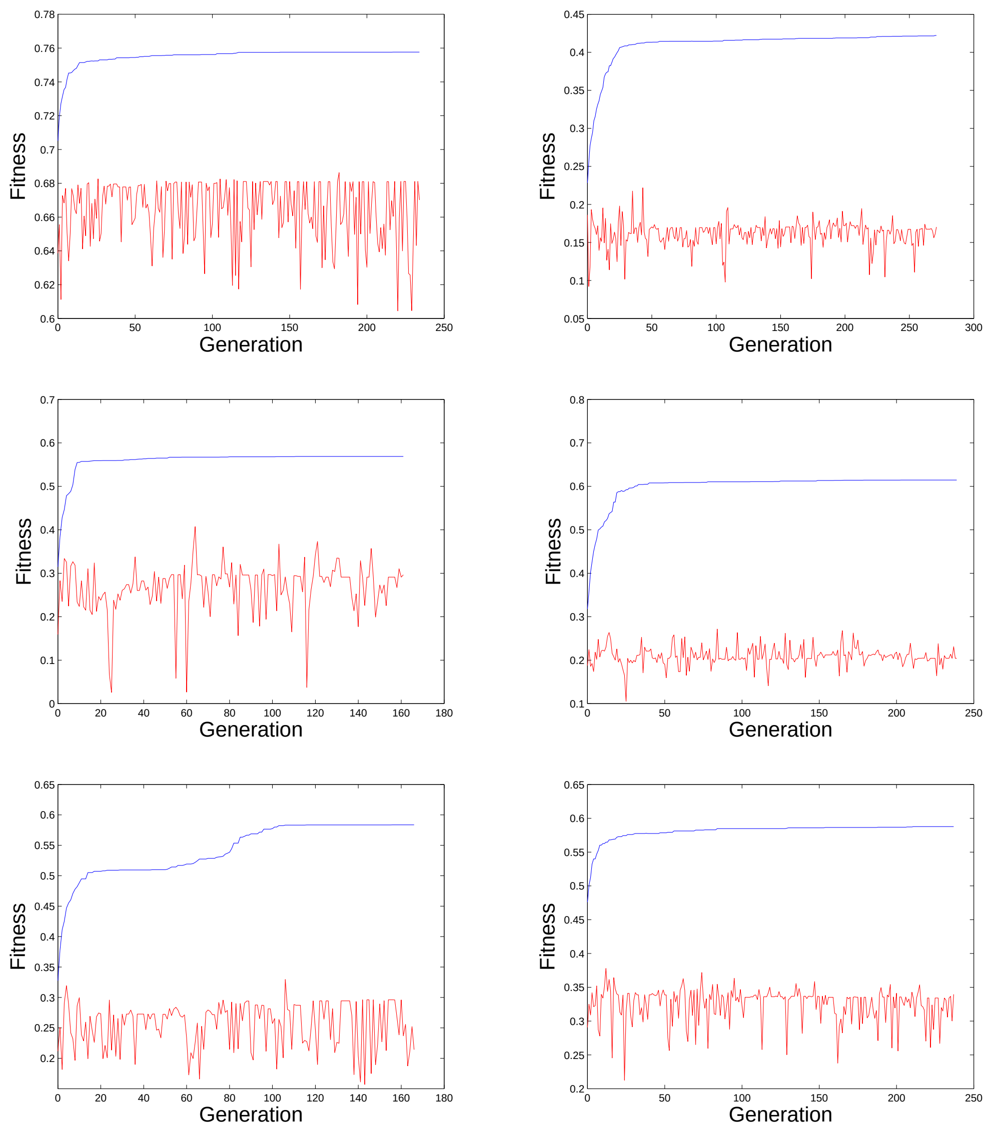


Figure 5.12: Behavior of the MEVO version for classes 15-20 (person pottedplant sheep sofa train tv). The blue line presents the Average Precision of current solution of the $(1+1)$-CMA-ES on the training set and the red line presents its Average Precision on the validation set.

We will give the final test results and then compare these two methods in the next section.

5.3.5 Results

Table 5.2 shows the final results for the 18 different participants' algorithms that participated to the VOC 2008 challenge (our two own methods are indicated with bold faces). We immediately note that our two methods do clearly not compete with the best methods for each class. Nevertheless, our results are comparable to those of several other methods participating in the challenge. Furthermore, one must keep in mind that all the other methods (even among those that show worse results than ours) are developed by research groups with a long-lasting expertise in computer image analysis. We only leveraged the self-training properties of the original neural network-based method and almost parameter-free evolutionary strategies, to reach, with no a priori expertise on computer vision, a similar level of performance than several of these groups. We think that this is an encouraging indication of the promises of this research direction. Moreover we think that our methods still offers a wide room for improvement. To this aim, let us first proceed in the following to a deeper analysis of our results.

5.3.6 Result Analysis

In Table 5.3, we summarize the VOC fitness results of the CMA version and the MEVO versions of our approaches, with separate comparison of the results on the training set, the validation set, and the test set. We also list the corresponding best fitness results among all participants of VOC 2008.

In Figures 5.13 and 5.14, we further give similar information with histograms of the performance of CMA and MEVO, comparing their fitness on the training set (deep blue), validation set (sky blue), and test set (yellow) to the best VOC 2008 fitness (brownish red).

From these tables and figures, we find that the results of MEVO on the training set are statistically better than that of CMA. On the other hand, comparing the results obtained on the validation set and on the test set, no statistical difference can be seen any more, but most of the time, the MEVO version seems to generalize better than the CMA version (with some noteworthy exceptions like the cow class!) : using only 200 features allows to slightly overfit the training set, without degrading too much the generalization results, even improving them on many classes.

However, when compared with the best VOC08 results, our test result look much worse. Furthermore, for several object classes, our results on the training set are better than the best VOC08 results (e.g. CMA version for classes 5, 6, 10,

	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv monitor
BerlinFIRSTNikon	72.4	37.4	51.1	57.4	24.5	38.5	53.9	44.7	46.2	25.6	28.6	40.3	57.0	53.5	83.0	21.0	21.4	28.6	66.2	50.2
CASIA_LinSVM	50.8	21.5	31.8	41.8	18.3	7.7	35.7	40.6	30.6	4.9	7.1	31.5	29.3	22.0	73.4	7.9	13.7	11.6	18.9	31.0
CASIA_NeuralNet	47.6	12.7	28.4	35.3	12.7	5.8	31.6	34.5	22.7	3.7	6.0	23.6	21.9	16.1	69.1	8.1	5.0	8.7	15.9	15.1
CASIA_NonLinSVM	35.1	19.7	24.2	40.5	13.7	3.8	30.5	37.1	30.2	5.8	6.2	31.1	20.4	26.3	74.5	5.1	14.4	9.3	10.3	21.8
ECPLIAMA	54.9	25.3	31.1	29.8	18.9	21.4	33.7	25.5	28.4	-	-	21.2	27.7	30.5	68.5	-	-	-	31.7	26.6
FIRST_SC1C	36.6	16.8	17.3	26.9	7.6	14.0	29.0	28.5	22.9	4.3	8.0	23.2	14.8	30.3	64.5	10.3	5.5	13.2	9.6	24.0
FIRST_SCST	36.6	16.8	17.3	26.9	7.6	14.0	29.0	28.5	22.9	4.3	8.0	23.2	14.8	30.3	64.5	10.3	5.5	13.2	9.6	24.0
INRIASaclay_CMA	**52.4**	**15.0**	**23.6**	**33.9**	**10.2**	**10.3**	**32.7**	**32.4**	**26.4**	**13.2**	**16.0**	**22.2**	**18.5**	**27.3**	**64.8**	**8.6**	**4.4**	**7.9**	**20.0**	**30.4**
INRIASaclay_MEVO	**50.2**	**19.5**	**17.9**	**32.1**	**13.0**	**14.9**	**34.2**	**30.6**	**23.1**	**4.4**	**14.8**	**21.0**	**13.1**	**26.1**	**65.7**	**7.6**	**7.4**	**17.5**	**16.5**	**30.6**
LEAR_flat	80.1	51.8	60.5	66.9	29.1	52.0	57.4	58.6	48.7	31.0	39.2	47.6	64.2	64.6	87.0	28.6	33.3	42.6	73.1	59.8
LEAR_shotgun	81.1	52.9	61.6	67.8	29.4	52.1	58.7	59.9	48.5	32.0	38.6	47.9	65.4	65.2	87.0	29.0	34.4	43.1	74.3	61.5
SurreyUvA_SRKDA	79.5	54.3	61.4	64.8	30.0	52.1	59.5	59.4	48.9	33.6	37.8	46.0	66.1	64.0	86.8	29.2	42.3	44.0	77.8	61.2
TKK_ALL_SFBS	77.9	47.3	52.4	61.0	27.9	45.5	53.5	55.5	47.6	26.8	40.8	46.1	58.6	58.3	83.5	26.4	24.3	39.2	70.3	56.9
TKK_MAXVAL	76.7	47.3	51.6	60.8	28.3	44.6	54.2	55.5	47.8	21.2	39.2	46.1	58.8	55.9	83.3	26.4	24.3	41.9	70.2	52.4
UvA_FullSFS	79.8	53.0	61.3	65.7	28.9	46.5	58.4	58.9	47.7	25.4	35.4	45.2	64.2	59.6	87.0	31.0	35.3	44.6	74.7	60.9
UvA_Soft5ColorSift	79.7	52.1	61.5	65.5	29.1	46.5	58.3	57.4	48.2	27.9	38.3	46.6	66.0	60.6	87.0	31.8	42.2	45.3	72.3	64.7
UvA_TreeSFS	80.8	53.2	61.6	65.6	29.4	49.9	58.5	59.4	48.0	30.1	39.6	45.0	67.3	60.4	87.1	30.1	41.5	45.4	74.3	59.8
XRCE	78.9	48.0	58.7	65.2	29.0	44.8	56.1	56.3	43.7	32.8	30.4	39.7	61.2	61.7	86.8	22.9	34.2	44.2	68.4	59.1

Table 5.2: Statistics of the main image sets. Object statistics lists only the “non-diffcult” objects used in the evaluation.

		aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow
C	Train	67.0	40.6	35.0	54.4	36.0	60.4	44.6	54.9	42.5	57.8
M	Val	58.4	30.3	22.8	33.3	21.8	30.8	38.0	31.1	33.6	30.4
A	Test	52.4	15.0	23.6	33.9	10.2	10.3	32.7	32.4	26.4	13.2
M											
E	Train	69.1	44.5	41.4	61.6	42.6	59.3	49.7	52.0	50.2	73.4
V	Val	57.4	29.6	24.0	39.9	22.4	39.9	38.2	31.4	36.9	30.9
O	Test	50.2	19.5	17.9	32.1	13.0	14.9	34.2	30.6	23.1	4.4
Best of VOC08		81.1	54.3	61.6	67.8	30.0	52.1	59.5	59.9	48.9	33.6

		dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv
C	Train	42.5	35.9	32.3	48.1	77.1	38.4	57.8	51.39	43.3	55.0
M	Val	32.1	26.5	25.2	30.4	69.3	18.7	35.3	25.6	26.0	37.0
A	Test	16.0	22.2	18.5	27.3	64.8	8.6	4.4	7.9	20.0	30.4
M											
E	Train	50.0	37.6	43.7	54.9	75.8	41.2	56.7	61.0	58.3	58.3
V	Val	31.4	29.4	25.6	36.3	68.6	22.2	40.7	26.2	33.0	37.2
O	Test	14.8	21.0	13.1	26.1	65.7	7.6	7.4	17.5	16.5	30.6
Best of VOC08		40.8	47.9	67.3	65.2	87.1	31.8	42.3	45.4	77.8	64.7

Table 5.3: Statistics of the main image sets. Object statistics lists only the "non-difficult" objects used in the evaluation.

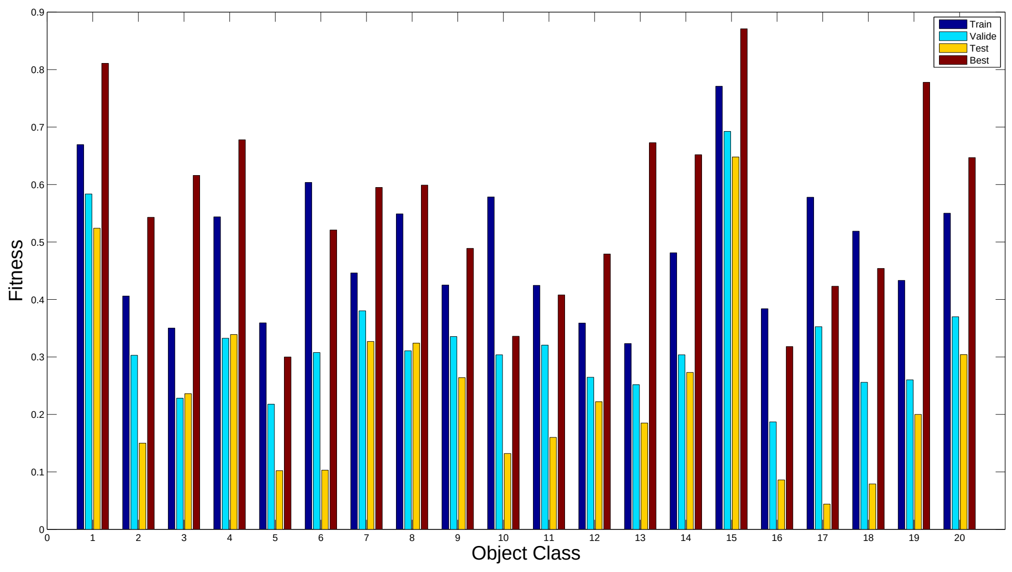


Figure 5.13: CMA version by comparing the fitness in training set (deep blue), valid set (sky blue), test set (yellow) and the best VOC 2008 fitness (brownish red). The X axis is the 20 index of object class and the Y axis is the VOC fitness.

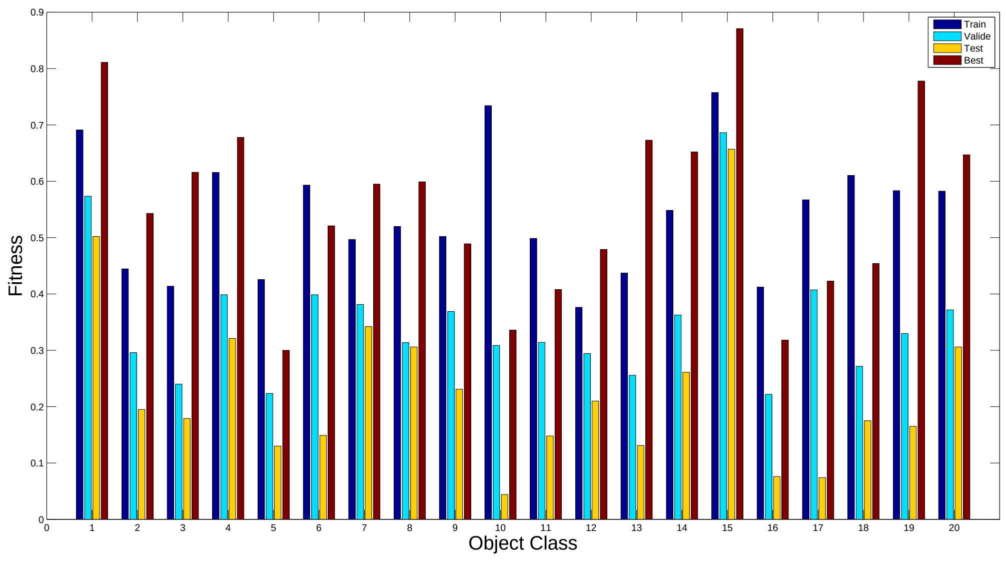


Figure 5.14: MEVO version by comparing the fitness in training set (deep blue), valid set (sky blue), test set (yellow) and the best VOC 2008 fitness (brownish red). The X axis is the index of 20 object class and the Y axis is the VOC fitness.

11, 16, 17 and 18, and MEVO version for class 5, 6, 9, 10, 11, 16, 17 and 18). But even for these classes, the results on the test set are really worse. This may indicate that the optimization processes do some sort of over-fitting after all, and the MEVO approach selects features that are too specific of the training set, thus failing to generalize well.

In agreement with the latter hypothesis, this generalization issue seems to depend on the object class, too. Indeed, Figure 5.13 and Figure 5.14 show that, for classes 1, 3, 4, 7, 8, 14, 15 and 20, the differences between the performance on the validation set and the fitness on the test set are much smaller or can even be reversed (yielding. better performance on the validation set).

Finally, we emphasize that the good news here is that using only 200 selected features/patches among the 1,000 randomly chosen ones, our results are as good as using all the 1,000 features. Since the CMA is much faster on 200-dimensional problems than with 1000-dimensional ones, decreasing the total number of patches by an optimal selection mechanism would enable us to develop strategies to optimize the topology of the SMF model itself, which is currently out of reach for computational cost reasons.

In theory, reducing the problem dimensionality from N_1 to N_2, the computational cost associated with CMA-ES can be reduced by $(N_1/N_2)^2$ [103]. In our case, with $N_1 = 1,000$ and $N_2 = 200$ this theoretical limit would represent an improvement by a factor 125 in the computation cost. In practice, this rate may however vary according to a lot of others experimental condition, such as the computation cost of simulation and the implementation of the program. We find that in our case, on an Intel(R) Xeon(R) CPU 2.50GHz computer, one evolution run of CMA with 200 dimensions takes 488*ms* on average, while the same task with 1000 dimensions needs 81150*ms* on average to complete. The ratio is thus ≈ 166. We emphasize that this ratio is just based on a few experiments and is expected to depends on a lot of other experimental condition. But it still means that the computation cost may be reduced by circa 2 orders of magnitude if we use the optimal features in this optimization problem.

5.4 Conclusion

In this chapter, we presented the SMF model for the recognition of visual object and applied this model to the VOC 2008 challenge. The results show that the auto-selection feature can reduce the computation cost and keep the performance of system. But the generalization capacity of the system seems to depend on the feature selected. Future work will try to optimize the topology of the connections between the layers of the SMF model, and to optimize the parameters whose values in the current work were taken without modification from ref. [198].

Chapter 6

Conclusion

This dissertation has introduced our study of three different artificial neural networks models: Self-Organizing Map (SOM, Chapter 3), Echo States Network (ESN, Chapter 4) and Standard Model Features (SMF, Chapter 5). We will now summarize our findings, and sketch some directions for further research.

6.1 Summary of Results

6.1.1 Evolving SOM Topology

The first part of our work, Chapter 3, directly addressed issue of the influence of the topology on Self-Organizing Maps. The task we considered for SOM networks was the classification of handwritten digits. A modification of the original SOM practice was to change the neighborhood relationship from Euclidian distance to graph distance for weight updates, so that topology modification resulted in modified dynamics.

In a first step, we empirically explored the influence of small-world-like topology modification of the standard regular SOM topology on the performance for the task at hand. The small-world topology is created by rewiring the connections of a regular SOM network with a probability ρ. By varying the rewiring probability, we tracked the performance of different topologies in the handwritten digits recognition task and found out that, in this class of topologies, topology has a small impact on the performance. Furthermore, it does not seem to influence the robustness to neuron failures, at least for long learning times. This result may indicate that the performance of the network is only weakly controlled by its topology, at least for such learning tasks.

We then addressed the inverse problem, and evolved SOM topologies of the same class with the goal of maximizing the performance on the handwritten digits recognition. The evolutionary algorithm used a direct representation of the network topology, where mutation directly manipulates the connections. Unfortunately, because of the high computation cost of such evolutions, we could only apply such evolutionary algorithm to find the optimal topology of a small network (100 neurons). At the end of optimization, however, the performance of the best network is improved by almost 10%. All evolved networks are ’more random’ than the initial small-world topology population, and display a more heterogeneous degree distribution, which may indicate a tendency to evolve toward scale-free topologies. Unfortunately, in order to confirm this tendency with less ambiguity, larger networks should be considered.

6.1.2 Evolving Parameters and Hyper-parameters of ESN

Reservoir Computing is an alternative to the fine tuning of NNs topology for each task, and has recently received a lot of attention. The original idea of Echo State Networks, as proposed by Jaeger, is to rely on random sparse connections within a very large network and to simply combine them linearly. The topology is governed by some hyper-parameters, and only the out-going weights are optimized, leading to a quadratic optimization problem in the supervised learning case. The question remains about how to tune those hyper-parameters for a given task - and whether the ESN approach can be efficient in other contexts, e.g., some reinforcement learning tasks.

In Chapter 4, we demonstrated the ability of CMA-ES, the state-of-the-art method in evolutionary continuous parameter optimization, to efficiently optimize ESN parameters and hyper-parameters for both the standard supervised learning context already used in previous studies and for the double-pole balancing, a typical reinforcement learning task. Indeed, the flexibility of Evolutionary optimization allowed us to optimize not only the outgoing weights but also, other ESN parameters (the Readout, Spectral Radius, Slopes of the neurons active function).

The combination of ESN and CMA-ES was first tested on a standard time series prediction problem, and validated against the quadratic optimization approach. With the standard parameter setting of an ESN with 100 neurons, the results show that CMA-ES can reach the same precision when optimizing the output weights than the original results obtained using quadratic optimization. With a smaller reservoir, surprisingly, optimizing only the slopes of the transfer functions of all neurons allowed us to reach even a better prediction accuracy, though at a higher computational cost.

In the context of the double-pole balancing, the results of CMA-ES applied to ESN optimization are compared to those of the best topology-learning methods in neuro-evolution. For such a reinforcement learning task, the good news is that Evolutionary Learning of ESN does work: our results are at the level of the best published results of neuro-evolution methods. Moreover, optimizing more than just the outgoing weights does improve the results. Furthermore, there seems to be a high dependency of the results on the topology of the reservoir, at least for the small sizes experimented with here.

6.1.3 Feature Sub-Sampling in SMF Bio-Inspired Approach

The issue of the topology of the information processing workflow has received many inspirations that are strongly influenced by the study of actual biological processes, when sufficient information has been gathered by biologists. This is the case for the biology of visual cortex, and Chapter 5 has concentrated of a bio-inspired model of the visual cortex proposed for object detection by Serre and Poggio. Here, the topology is specifically designed to match the recently researches of the biological process. However, several degrees of freedom remain in the proposed topology, leaving room for improvement.

We applied the Standard Model Features of Serre and Poggio to the PASCAL Visual multi-Object recognition Challenge (VOC2008), and concentrated on the possible selection of some of the random features that are used in the model, in an attempt to decrease its huge computational complexity.

First, a specific variant of the CMA-ES algorithm was used to optimize the final layer of the model: because of the non-linearity of the VOC confidence, the measure of quality proposed within the PASCAL Challenge, the problem of optimizing even a linear combination of the 1000 features for this specific criterion could hardly be solved by traditional methods. Then, a higher-level evolutionary algorithm was used to select a small fractions of those random features while maintaining the same quality of recognition results: the fitness used by this EA was obtained by running CMA-ES on the corresponding subset of features and measuring the resulting VOC-confidence.

The results of this Multi-Evolutionary algorithm demonstrated that using only with 20% of the original features, the system could obtain the same performance than with all the features, requiring less computational effort. This result opens the path for the longer-term goal about this model: optimize also the topology of the SMF model, as this was not possible with the current cost of the method when using all features.

6.2 Perspectives

We will here give some hints about possible research directions that emerged from our work, both at the fundamental and at the practical levels.

6.2.1 Fundamental Issues

One of the main difficulty we faced in Chapter 3 was to find an appropriate fitness measure for the performance of SOMs. Especially, when the input data set has a complex topology which may not be already clearly understood. How to measure the approximation between the topology of network and the topology of the data set is a difficult task. This is why we turned to using SOMs in a classification task, which is not what they have been designed for. A new more specifical quantitative measure method is required for unsupervised learning, that will allow us to hopefully evolve optimized SOM topologies, and to maybe confirm that some types of scale-free topologies are better suited than the regular one, at least in some cases. But it would also help to compare SOMs to other dimension-reduction algorithms.

Regarding our work on the ESN, more research is obviously needed to understand the relationship between the hyper-parameters and the performance of reservoirs, both from a theoretical and a practical viewpoint. Even the simple question of the reservoir size is still open: for what tasks are larger-size reservoirs helpful? On the practical side, we could not even try to solve the direct problem in reinforcement learning context – i.e., experimental investigate a wide range of reservoir sizes on the same problem.

The problem of optimizing the reservoir topologies is still open, too, and seems to be even more out to reach in terms of computational cost. Several issues remain open. We would like to investigate the generative approaches based on Genetic Regulatory Networks, as discussed at the end of Section 2.3.2. Two approaches can be envisioned. The straightforward approach would be to directly evaluate the evolved topologies on the reinforcement learning problem at hand, with the huge CPU cost this implies. But, if some statistical characteristics of efficient reservoirs have been previously identified, some topologies meeting those requirements could first be evolved, without the need for costly simulations, and only in a second phase the evaluation would be made on the reinforcement learning task. Recent experiments have demonstrated the feasibility of the first phase of this plan [156, 157], and CPU cost is the only barrier to the second phase (see below).

Another possible extension of ESN research, as pointed in [138], regards the single-layer architecture of current reservoirs: "Even if the reservoirs may have thousands of units, it has still a single layer. It suggests that for demandingly complex tasks the adaptation of a single reservoir might not be enough and a hierarchical architecture of ESNs might be needed.” [138] The possible combination between Reservoir Computing and Deep Network is another appealing future research direction . . . that will undoubtedly also require some orders of magnitude more CPU resources.

The evolutionary computing vision system presented in Chapter 5, the system may benefit from further advances in the biological knowledge of the natural visual system. In the SMF model presented here, the parameters and the topology of the system were completely hand-designed, and we have only accomplished a small step in the pruning of the set of random features. But the complete topology could be self-turned by an evolutionary process, especially if we can reduce the CPU cost even more. Evolving the network topology between the layers, self-turning the system parameters together with self-selection of the C1 feature is hence a promising direction for future research.

Another issue in object recognition research is that of the sample set of images: only raw images are available at the moment, whereas a universal object data base may be helpful for the future study of the SMF model.

6.2.2 Practical Issues

Balancing the Double Pole

As many published work used this problem as a benchmark, it made sense to keep the same problem settings and use the same fitness definition to compare one’s algorithms with previous work. However, one important issue concerns the adequacy of the classical fitness used up to now to tackle this problem. Considering the results presented in Chapter 4, the answer is clearly that it is not adapted to the goal (note that such an answer has already been claimed by others before). Indeed, the correlation between the fitness and the generalization tests that the system is demanded to pass to be declared successful, is possibly deceptive. It introduces a too high random part in the evaluation of the results: indeed, we often found that the best individual of the population cannot pass the test of generalization, but other individuals, with lower fitness, could successfully pass the test. Another inconvenience of the problem is that it doesn’t regard the control behavior of the successful solution. From the view of engineering, the solution with less consumption of energy is better in practice, as in the Figure 4.10 showing the distribution of the output force. Hence we propose to incorporate the total energy minimization into the fitness – possibly leading to a multi-objective problem.

Another way to go would be to use other benchmarks for reinforcement learning tasks, from the 2D pole balancing, that seems at the moment still challenging, to video-game role playing, that could not only be challenging, but could also attract many good students.

Distributed Computing

In a more general perspective, another issue that we have faced in all our studies is the computation cost for evolving large neural networks. For example, when we evolved the topology of SOM networks (Chapter 3), even though we used a Client-Server approach written in C++ and running on more than 20 recent CPUs, the whole evolutionary run took *several weeks* before giving the final results. Similar CPU costs were necessary for the work in other Chapters: One run of optimization of ESN could take serval days, which is untractable considering that several different networks have to be tested, each for at least 11 times in order to be able to draw statistically meaningful conclusions. This was the main limitation regarding the sizes of the networks that were used there.

At the end of the age of Moore's law, as the speed of single CPU has almost stopped to increase, it seems that the three following technologies may be useful to construct new ANN simulation frameworks – while requiring new algorithmic designs.

Multi-thread The tendency of CPU designers is to add more and more cores to the basic commercialized CPUs. But in order to directly benefit from those architectures, multi-threading requires some precautions in terms of memory access to avoid conflicts and/or endless locks.

Message Passing Interface Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is used in computer clusters and supercomputers, and ultimately can be used on GRIDS. MPI was created by William Gropp, Ewing Lusk and others [73]. The famous MOGO computer go program got a speedup of about 25 by using 30 computers in parallel with MPI [65]. In [79], the authors give a general guide for this new technique. For a Evolutionary Algorithms addressing reinforcement learning problems, there is always a big computation cost for the fitness simulation. Parallel simulation by using MPI with large computational grids might useful for such optimization task. However, here again care must be taken about the algorithm: it is known that increasing the population size of CMA-ES does not always result in improving the convergence speed. Specific studies are necessary to actually take advantage of a huge number of CPUs.

Graphical Processing Units (GPU) have been designed as specialized multi-cores SIMD architectures for graphical processes. Nevertheless, they can now be programmed with almost general-purpose languages (e.g., CODA$^{®}$ language by NVIDIA$^{®}$) and are hence becoming more and more popular in the EC community, e.g. the Computational Intelligence on Consumer Games and Graphics Hardware (CIGPU) workshop organized in GECCO 2009 [1]. Unfortunately, the SIMD architecture requires a complete rewriting of all algorithms. However, they seem well suited for the simulation of large Neural Networks (the most recent Geforce Gtx 295 has 480 processors) where they could speed up by several orders of magnitude the simulation time of very large networks.

Ideally, clusters of computers, each being equipped with some highly efficient multi-core architectures like the GPU could help fulfilling the research agenda that was set for this work, regarding how the topology of **large** Neural Networks influence their computational capacities, and whether it is possible to optimize their topologies to reach breakthrough results.

[1]http://www.sigevo.org/gecco-2009/workshops.html, http://www.gpgpgpu.com/gecco2009/

Bibliography

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. Cognitive Science, 9:147–169, 1985.

[2] C. W. Anderson. Learning to control an inverted pendulum using neural networks. IEEE Control Systems Magazine, 9(3):31–37, April 1989.

[3] J. A. Anderson. A simple neural network generating an interactive memory. Mathematical Biosciences, 14:197–220, 1972.

[4] P. Angeline. Subtree crossover: Building block engine or macromutation? In J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba, and R. Riolo, editors, Genetic Programming 1997, pages 9–17. Morgan Kauffman, 1997.

[5] A. Auger. Contributions théoriques et numériques À l'optimisation continue par algorithmes Évolutionnaires. PhD thesis, Université Paris 6, December 2004. in French.

[6] A. Auger. Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible markov chains. Theoretical Computer Science, 334(1-3):35–69, 2005.

[7] A. Auger, C. L. Bris, and M. Schoenauer. Dimension-independent convergence rate for non-isotropic $(1,\lambda)-es$. In E. Cantu-Paz et al., editor, Proc. GECCO'2003, pages 512–524. LNCS 2723 and 2724, Springer Verlag, 2003.

[8] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In Proc. CEC'05, 2005.

[9] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In Proc. CEC'05, pages 1769–1776. IEEE Press, 2005.

[10] A. Auger, N. Hansen, J. Perez Zerpa, R. Ros, and M. Schoenauer. Experimental Comparisons of Derivative Free Optimization Algorithms. In

Jan Vahrenhold, editor, 8th International Symposium on Experimental Algorithms, LNCS, pages 3–15. Springer Verlag, 2009.

[11] A. D. Back and T. Chen. Universal approximation of multiple nonlinear operators by neural networks. Neural Comput, 14(11):2561–2566, Nov 2002.

[12] T. Bäck. Evolution strategies: an alternative evolutionary algorithm. In J.-M. Alliot et al,, editor, Artificial Evolution, EA'95, number 1063 in LNCS, pages 3–20. Springer Verlag, 1995.

[13] T. Bäck. Evolutionary Algorithms in Theory and Practice. New-York:Oxford University Press, 1995.

[14] T. Bäck and M. Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In J. R. McDonnell et al., editor, Proc. EP'95. MIT Press, 1995.

[15] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation, 1(1):1–23, 1993.

[16] T. Baeck, D. Fogel, and Z. Michalewicz, editors. Handbook of Evolutionary Computation. Oxford University Press, 1997.

[17] W. Banzhaf. Artificial regulatory networks and genetic programming. In R. Riolo and B. Worzel, editors, Genetic Programming Theory and Practice, pages 43–52. Kluwer Publishers, 2003.

[18] W. Banzhaf, P. Nordin, and R. Keller. Genetic Programming: an Introduction. dpunkt, Heidelberg, 1998.

[19] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. Science, 286:509, 1999.

[20] H. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. IEEE Transactions on Neural Networks, 3(4):570–579, July 1992.

[21] M. F. Bear, B. Connors, and M. Paradiso. Neuroscience: Exploring the Brain (Third Edition). Lippincott Williams & Wilkins, February 2006.

[22] Y. Bengio, O. Delalleau, and N. L. Roux. The curse of highly variable functions for local kernel machines. In NIPS 18. MIT Press, 2006.

[23] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In NIPS 19, pages 153–160. MIT Press, 2007.

[24] P. Bentley and D. W. Corne, editors. Creative Evolutionary Systems. Morgan Kaufmann, 2001.

[25] P. J. Bentley, editor. Evolutionary Design by Computers. Morgan Kaufman Publishers Inc., 1999.

[26] H.-G. Beyer. The Theory of Evolution Strategies. Natural Computing. Springer Verlag, 2001.

[27] H.-G. Beyer and K. Deb. On the desired behaviors of self-adaptive evolutionary algorithms. In M. Schoenauer et al., editor, Proc. PPSN VI, pages 59–68. LNCS 1917, Springer Verlag, 2000.

[28] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang. Complex networks: Structure and dynamics. Physics Reports, 424:175–308, 2006.

[29] J. Bohland and A. Minai. Efficient associative memory using small-world architecture. Neurocomputing, 38–40:489–496, 2001.

[30] S. A. Braitenberg V. Anatomy of the cortex: Statistics and geometry. Berlin: Springer, 1991.

[31] C. Burges, J. Ben, J. Denker, Y. LeCun, and C. Nohl. Off line recognition of handwritten postal words using neural networks. Intl Journal of Pattern Recognition and Artificial Intelligence, 7(4):689–704, 1993.

[32] N. A. Carlson. Foundations of Physiological Psychology. Needham Heights,Massachusetts:Simon & Schuster, 1992.

[33] G. Castellano and A. M. Fanelli. An iterative pruning algorithm for feedforward neural networks. IEEE Transactions on Neural Networks, 8(3):519–531, 1997.

[34] T. P. Caudell and C. P. Dolan. Parametric connectivity: Training of constrained networks using genetic algorithms. In J. D. Schaffer, editor, ICGA, pages 370–374. Morgan Kaufmann, 1989.

[35] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont. Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, 2002.

[36] L. D. Costa and M. Schoenauer. Bringing evolutionary computation to industrial applications with. In G. Raidl et al., editor, Proc. GECCO'09. ACM Press, 2009.

[37] N. Cramer. A representation for the adaptive generation of simple sequential programs. In L. J. Eshelman, editor, Proceedings of the 6^{th} International Conference on Genetic Algorithms, pages 183–187. Morgan Kaufmann, 1985.

[38] D. E. Rumelhart and J. L. McClelland, editor. Parallel Distributed Processing, volume 1:Foundations. MIT Press, Cambridge, MA, 1986.

[39] H. de Garis. Genetic programming : building artificial nervous systems using genetically programmed neural networks modules. In R. Porter and B. Mooney, editors, Proceedings of the 7^{th} International Conference on Machine Learning, pages 132–139. Morgan Kaufmann, 1990.

[40] K. Deb. Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley, 2001.

[41] K. Deb and R. W. Agrawal. Simulated binary crossover for continuous search space. Complex Systems, 9:115–148, 1995.

[42] K. DeJong. The Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, Ann Harbor, 1975. Dissertation Abstract International, 36(10), 5140B. (University Microfilms No 76-9381).

[43] K. DeJong. Evolutionary Computation. A unified Approach. MIT Press, 2006.

[44] Z. Deng and Y. Zhang. Complex systems modeling using scale-free highly-clustered echo state network. In IJCNN'06, pages 3128–3135, 2006.

[45] P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with Analog Genetic Encoding. In Th. Runarsson et al., editor, PPSN IX, pages 671–680. LNCS 4193, Springer Verlag, 2006.

[46] A. Eiben and J. Smith. Introduction to Evolutionary Computing. Springer Verlag, 2003.

[47] A. E. Eiben and M. Schoenauer. Evolutionary computing. Information Processing Letters, 82(1):1 – 6, 2002.

[48] A. E. Eiben, C. H. M. van Kemenade, and J. N. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In Proc. ECAL, pages 934–945, 1995.

[49] L. J. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlins, editor, FOGA, pages 265–283. Morgan Kaufmann, 1990.

[50] G. R. et al., editor. Workshop on Black-Box Optimization Benchmarking at GECCO'09. ACM, 2009.

[51] N. H. et al. Comparison of evolutionary algorithms on a benchmark function set. CEC'05 Special Session, 2005.

[52] K. K. Evans and A. Treisman. Perception of objects in natural scenes: is it really attention free? J Exp Psychol Hum Percept Perform, 31(6):1476–1492, Dec 2005.

[53] L. F., R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In Proc. Conference on Computer Vision and Pattern Recognition Workshop, pages 178–178, 27–02 June 2004.

[54] A. Fadda and M. Schoenauer. Evolutionary chromatographic law identification by recurrent neural nets. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, Proc. 4^{th} Conf. on Evolutionary Programming, pages 219–235. MIT Press, 1995.

[55] S. Fahlman and C. Libiere. The cascade-correlation learning architecture. In D. Touretsky, editor, Advances in Neural Information Processing Systems 2, pages 524–532. Morgan Kaufmann, 1990.

[56] S. S. Fels and G. E. Hinton. Glove-talk: a neural network interface between a data-glove and a speech synthesizer. IEEE Trans Neural Netw, 4(1):2–8, 1993.

[57] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 2, pages II–264–II–271, 18–20 June 2003.

[58] D. Floreano and C. Mattiussi. Evolution of analog networks using local string alignment on highly reorganizable genomes. In Proc. Conf. on Evolvable Hardware, EH'04, 2004.

[59] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man, and Cybernetics, 26:396–407, 1994.

[60] D. Fogel. Evolutionary Computing: The Fossile Record. IEEE Press, 1998.

[61] D. Fogel and L. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. BioSystems, 32:171–182, 1994.

[62] D. B. Fogel. Evolutionary computation: toward a new philosophy of machine intelligence. IEEE Press, New York, 1995.

[63] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial Intelligence Through simulated Evolution. John Wiley and Sons, New York, 1966.

[64] B. Freisleben and P. Merz. New genetic local search operators for the TSP. In H.-M. Voigt et al., editor, Proc. PPSN IV, LNCS 1141, pages 890–899. Springer Verlag, 1996.

[65] S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of monte-carlo planning. In Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008), pages 198–203, 2008.

[66] J. Gero. Adaptive systems in designing: New analogies from genetics and developmental biology. In I. Parmee, editor, Adaptive Computing in Design and Manufacture, pages 3–12. Springer Verlag, 1998.

[67] D. Goldberg. The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, 2002.

[68] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA, 1989.

[69] F. Gomez. Robust Nonlinear Control through Neuroevolution. Tech. rep. ai-tr-03-303, University of Texas at Austin, 2003.

[70] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient nonlinear control through neuroevolution. In Proc. European Conference on Machine Learning (ECML, 2006.

[71] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuro-evolution. In IJCAI, pages 1356–1361, 1999.

[72] G. Goodhill, S. Finch, and T. Sejnowski. Quantifying neighbourhood preservation in topographic mappings. Technical report, Institute for Neural Computation Technical Report Series, No. INC-9505, 1995.

[73] E. Gropp, W. Lusk and A. Skjellum. Using MPI: portable parallel programming with the message-passing interface. MIT Press In Scientific And Engineering Computation Series, Cambridge, MA, USA. 307 pp. ISBN 0-262-57104-8, 1994.

[74] S. Grossberg. Contour enhancement, short term memory, and constancies in reverberating neural networks. Studies in Applied Mathematics, 52:213–257, September 1973.

[75] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance, pages 213–250. Ablex Publishing Corp., 1988.

[76] F. Gruau. Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm. PhD thesis, Ecole Normale Superieure de Lyon, 1994. L'universite Claude Bernard-lyon I and Of A Diplome De Doctorat and M. Jacques Demongeot and Examinators M. Michel Cosnard and M. Jacques Mazoyer and M. Pierre Peretto and M. Darell Whitley.

[77] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. Evolutionary Computation, 1(3):213–233, 1993.

[78] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. K. et al., editor, Proc. GP'96. MIT Press, 28–31 1996.

[79] L. Hablot, O. Glück, J.-C. Mignot, S. Genaud, and P. Vicat-Blanc Primet. Comparison and tuning of MPI implementations in a grid context. Research Report RR-6200, INRIA, 2007.

[80] H. Hamda, O. Roudenko, and M. Schoenauer. Multi-objective evolutionary topological optimum design. In I. Parmee, editor, Evolutionary Design and Manufacture, pages 121–132. Springer Verlag, 2002.

[81] H. Hamda and M. Schoenauer. Topological optimum design with evolutionary algorithms. Journal of Convex Analysis, 9:503–517, 2002.

[82] P. J. B. Hancock. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and

Neural Networks (COGANN-92, pages 108–122. IEEE Computer Society Press, 1992.

[83] N. Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In G. R. et al., editor, Workshop Proceedings at GECCO, pages 2389–2395. ACM, 2009.

[84] N. Hansen. References to cma-es applications, 2009+.

[85] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao et al., editors, PPSN VIII, pages 282–291. LNCS 3242, Springer Verlag, 2004.

[86] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation, 11(1):1–18, 2003.

[87] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolutionstrategies: the covariance matrix adaptation. In Proc. CEC'96, pages 312–317. IEEE Press, 1996.

[88] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 9(2):159–195, 2001.

[89] N. Hansen, A. Ostermeier, and A. Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In L. J. Eshelman, editor, Proc. 6th ICGA, pages 57–64. Morgan Kauffman, 1995.

[90] A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks. In J. Schaffer, editor, Proc. Third ICGA, pages 360–369. Morgan Kaufmann, 1989.

[91] D. O. Hebb. The Organization of Behavior: A Neuropsychological Theory. Wiley, New York, June 1949.

[92] B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio. Categorization by learning and combining object parts. In Advances in Neural Information Processing Systems 14, pages 1239–1245. MIT Press, 2002.

[93] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. Neural Comput, 18(7):1527–1554, Jul 2006.

[94] G. E. Hinton and T. J. Sejnowski. Optimal perceptual inference. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, page 448453, 1983.

[95] J. H. Holland. Adaptation in Natural and Artifical Systems. University of Michigan Press, Ann Arbor, MI, 1975.

[96] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In Proc. National Academy of Sciences, pages 2554–2558, 1982.

[97] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. Neural Networks, pages 359–366, 1989.

[98] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks, 3:551–560, 1990.

[99] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol, 160:106–154, Jan 1962.

[100] C. P. Hung, G. Kreiman, T. Poggio, and J. J. DiCarlo. Fast readout of object identity from macaque inferior temporal cortex. Science, 310(5749):863–866, Nov 2005.

[101] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In Proc. CEC'03, pages 2588–2595. IEEE Press, 2003.

[102] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. Evolutionary Computation, 15(1):1–28, 2006.

[103] C. Igel, T. Suttorp, and N. Hansen. A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 453–460, New York, NY, USA, 2006. ACM.

[104] G. Inc., editor. Glencoe Health 2nd Edition. Mission Hills, 1989.

[105] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical report, GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.

[106] H. Jaeger. Tutorial on training recurrent neural networks. Technical report, GMD Report 159, Fraunhofer Institute AIS, 2002.

[107] H. Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. Technical report, Technical report 10, School of Engineering and Science, Jacobs University, 2007.

[108] H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science, 304(5667):78–80, Apr 2004.

[109] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. Neural Network, 20(3):335–352, Apr 2007.

[110] F. Jiang, H. Berry, and M. Schoenauer. Optimising the Topology of Complex Neural Networks. In J. et al., editor, ECCS07, 2007.

[111] F. Jiang, H. Berry, and M. Schoenauer. The impact of network topology on self-organizing maps. In L. Xu, E. D. Goodman, G. Chen, D. Whitley, and Y. Ding, editors, GEC Summit, pages 247–254. ACM, 2009.

[112] J. P. Jones and L. A. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. J Neurophysiol, 58(6):1233–1258, Dec 1987.

[113] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, Proceedings of the 6^{th} International Conference on Genetic Algorithms, pages 184–192. Morgan Kauffman, 1995.

[114] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH, 4:237–285, 1996.

[115] E. R. Kandel. Principles of Neural Science. McGraw-Hill Education, June 2000.

[116] S. Kaski, J. Kangas, and T. Kohonen. Bibliography of self-organizing map (som) papers 1981-1997. Neural Computing Surveys, 1:1–176, 1998.

[117] Y. Kassahun, J. de Gea, M. Edgington, J. H. Metzen, and F. Kirchner. Accelerating neuroevolutionary methods using a kalman filter. In GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pages 1397–1404, New York, NY, USA, 2008. ACM.

[118] B. Kim. Performance of networks of artificial neurons: the role of clustering. Phys. Rev. E, 64:045101, 2004.

[119] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. Complex Systems, 4(4):461–476, 1990.

[120] J. M. Kleinberg. Navigation in a small world. Nature, 406(6798), 2000.

[121] J. Kodjabachian and J.-A. Meyer. Evolution and development of modular control architectures for 1-D locomotion in six-legged animats. Connection science, 10:211–237, 1998.

[122] T. Kohonen. Correlation matrix memories. IEEE Transactions on Computers, 21:353–359, 1972.

[123] T. Kohonen. An adaptive associative memory principle. IEEE Transactions on Computers, 4:444–445, April 1974.

[124] T. Kohonen. Self-Organization and Associative Memory. Springer-Verlag, 1989.

[125] J. R. Koza. Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, 1992.

[126] J. R. Koza and al. Genetic Programming III: Automatic Synthesis of Analog Circuits. MIT Press, 1999.

[127] P. D. Kuo, W. Banzhaf, and A. Leier. Network topology and the evolution of dynamics in an artificial genetic regulatory network model created by whole genome duplication and divergence. Biosystems, 85(3):177–200, Sep 2006.

[128] K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. Neural Netw., 3(1):23–43, 1990.

[129] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, NIPS'89. Morgan Kaufmann, 1990.

[130] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[131] V. Lefort, C. Knibbe, G. Beslon, and J. Favrel. A bio-inspired genetic algorithm with a self-organizing genome: The rbf-gene model. In K. Deb et al., editor, Proc. GECCO'04, number 3103 in LNCS, pages 406–407. ACM SIGEVO, Springer Verlag, 2004.

[132] V. Lefort, C. Knibbe, G. Beslon, and J. Favrel. Simultaneaous optimization of weights and structure of an rbf neural network. In E. Talbi et al., editor, Artificial Evolution'05. Selected papers, number 3871 in LNCS. Springer Verlag, 2005.

[133] J. Leskovec and E. Horvitz. Worldwide buzz: Planetary-scale views on an instant-messaging network. Technical report, Microsoft Research Technical Report MSR-TR-2006-186, Microsoft Research, 2007.

[134] B. Liebald. Exploration of effects of different network topologies on the esn signal crosscorrelation matrix spectrum. Master's thesis, Bachelor's thesis, Jacobs University Bremen, 2004.

[135] W. Liebert, K. Pawelzik, and H. G. Schuster. Optimal embeddings of chaotic attractors from topological considerations. EPL (Europhysics Letters), 14(6):521–526, 1991.

[136] B. Linares-Barranco, A. Andreou, G. Indiveri, and T. Shibata. Guest editorial - special issue on neural networks hardware implementations. IEEE Transactions on Neural Networks, 14(5):976–979, 2003.

[137] F. Lopez-Munoz, J. Boya, and C. Alamo. Neuron theory, the cornerstone of neuroscience, on the centenary of the nobel prize award to santiago ramun y cajal. Brain Res Bull, 70(4-6):391–405, Oct 2006.

[138] M. Lukosevicius and H. Jaeger. Overview of reservoir recipes. Technical report, Technical Report 11, School of Engineering and Science, Jacobs University Bremen, 2007.

[139] M. Lunacek, D. Whitley, and A. Sutton. The impact of global structure on search. In G. Rudolph et al., editor, Proc. PPSN X, number 5199 in LNCS, pages 498–507. Springer Verlag, 2008.

[140] E. M., V. G. L., W. C.K.I., W. J., and Z. A. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[141] S. M. The small world problem. Psychology Today 2, pages 60–67, 1967.

[142] C. D. M. Tomassini, M. Giacobini. Evolution and dynamics of small-world cellular automata. Complex Systems, 15:261–284, 2005.

[143] W. Maass, T. Natschllger, and H. Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput, 14(11):2531–2560, Nov 2002.

[144] J. Mao and A. K. Jain. Artificial neural networks for feature extraction and multivariate data projection. IEEE Trans Neural Netw, 6(2):296–317, 1995.

[145] S. Martin, J. Rivory, and M. Schoenauer. Synthesis of optical multi-layer systems using genetic algorithms. Applied Optics, 34:2267, 1995.

[146] C. Mattiussi. Evolutionary synthesis of analog networks. PhD thesis, n.3199, EPFL, Lausanne, 2005.

[147] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. Bull Math Biol, 52(1-2):99–115; discussion 73–97, 1943.

[148] P. McGraw and M. Menzinger. Topology and computational performance of attractor neural networks. Phys. Rev. E, 68:047102, 2003.

[149] P. Merz and J. Huhse. An iterated local search approach for finding provably good solutions for very large tsp instances. In G. Rudolph et al., editor, Proc. PPSN X, number 5199 in LNCS, pages 929–939. Springer Verlag, 2009.

[150] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, New-York, 1992-1996. 1st-3rd edition.

[151] M. Minsky and S. Papert. Perceptrons. MIT Press, Cambridge, MA, 1969.

[152] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In Proceedings of the 11th International Joint Conference on Artificial Intelligence, page 762–767, San Francisco: Kaufmann, 1989.

[153] L. G. Morelli, G. Abramson, and M. N. Kuperman. Associative memory on a small-world neural network. Eur. Phys. J. B, 38:495–500, 2004.

[154] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. Evolutionary Computation, 5(4):373–399, 1997.

[155] J. M. Murre and D. P. Sturdy. The connectivity of the brain: multi-level quantitative analysis. Biol Cybern, 73(6):529–545, Nov 1995.

[156] M. Nicolau and M. Schoenauer. Evolving Scale-Free Topologies using a Gene Regulatory Network Model. In IEEE Congress on Evolutionary Computation, pages 3748–3755. IEEE Press, 2008.

[157] M. Nicolau and M. Schoenauer. Evolving Specific Network Statistical Properties using a Gene Regulatory Network Model. In European Conference on Complex Systems'08, 2008.

[158] S. Nolfi and D. Floreano. Evolutionary Robotics. MIT Press, 2000.

[159] S. Obayashi. Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations. In D. Quadraglia, J. Périaux, C. Poloni, and G. Winter, editors, Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences, pages 245–266. John Wiley, 1997.

[160] S. Obayashi. Pareto solutions of multipoint design of supersonic wings using evolutionary algorithms. In I. Parmee, editor, Adaptive Computing in Design and Manufacture V, pages 3–15. Springer-Verlag, 2002.

[161] P. Oikonomou and P. Cluzel. Effects of topology on network evolution. Nat Phys, 2(8):532–536, 2006.

[162] M. Oja, S. Kaski, and T. Kohonen. Bibliography of self-organizing map (som) papers: 1998-2001 addendum. Neural Computing Surveys, 1:1–176, 2002.

[163] M. Ozturk, D. Xu, and J. Principe. Analysis and Design of Echo State Networks. Neural Computation, 19(1):111–138, 2007.

[164] B. Paechter, R. Rankin, A. Cumming, and T. C. Fogarty. Timetabling the classes of an entire university with an evolutionary algorithm. In T. B. et al., editor, Proc. PPSN V, LNCS 1498. Springer Verlag, 1998.

[165] D. I. Perrett, J. K. Hietanen, M. W. Oram, and P. J. Benson. Organization and functions of cells responsive to faces in the temporal cortex. Philos Trans R Soc Lond B Biol Sci, 335(1273):23–30, Jan 1992.

[166] D. I. Perrett and M. W. Oram. Neurophysiology of shape processing. Image Vision Comput., 11(6):317–333, 1993.

[167] D. Polani. On the choice of organization measures for self-organizing feature maps. Technical report, Technical report, 1995.

[168] D. Polani. Fitness functions for the optimization of self-organizing maps. In T. Bäck, editor, Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97), San Francisco, CA, 1997. Morgan Kaufmann.

[169] D. Polani and J. Gutenberg. Organization measures for self-organizing maps. In Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6, pages 280–285. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[170] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives, pages 255–297. Number 83 in Large-Scale Nonlinear Optimization. Springer Verlag, 2006.

[171] M. J. D. Powell. Developments of newuoa for minimization without derivatives. IMA Journal of Numerical Analysis, 2008.

[172] N. J. Radcliffe. Equivalence class analysis of genetic algorithms. Complex Systems, 5:183–20, 1991.

[173] N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, Foundations of Genetic Algorithms 3, pages 51–72. Morgan Kaufmann, 1995.

[174] I. Rechenberg. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. frommann-holzbog, Stuttgart, 1973. German.

[175] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. Nature neuroscience, 2(11):1019–1025, November 1999.

[176] E. Ronald and M. Schoenauer. Genetic lunar lander : an experiment in accurate neuro-control. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, Proc. PPSN III, pages 452–461. Springer-Verlag, LNCS 866, 1994.

[177] F. ROSENBLATT. The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev, 65(6):386–408, Nov 1958.

[178] M. Rosenman. Evolutionary case-based design. In C. Fonlupt et al., editor, Artificial Evolution'99, pages 53–72. Springer Verlag, LNCS 1829, 1999.

[179] O. Roudenko, T. Bosio, R. Fontana, and M. Schoenauer. Optmization of car front crash members. In Artificial Evolution'01, pages 202–214. Springer Verlag, LNCS 2310, 2002.

[180] G. Rudolph. Convergence of non-elitist strategies. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano, editors, Proceedings of the First IEEE International Conference on Evolutionary Computation, pages 63–66. IEEE Press, 1994.

[181] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. MIT Press, 1986.

[182] D. E. Rumelhart and J. L. Mcclelland. Parallel Distributed Processing: Explorations in the Microstructure of Cognition : Foundations (Parallel Distributed Processing). MIT Press, August 1986.

[183] Schaffer and Cannon. n the control of unstable mechanincal systems. In In Automatic and Remote Control III: Proceedings of the Third Congress of the International Federation of Automatic Control, 1966.

[184] J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In Proc. COGANN-92 Combinations of Genetic Algorithms and Neural Networks International Workshop on, pages 1–37, 6 June 1992.

[185] P. H. Schiller, B. L. Finlay, and S. F. Volman. Quantitative studies of single-cell properties in monkey striate cortex. ii. orientation specificity and ocular dominance. J Neurophysiol, 39(6):1320–1333, Nov 1976.

[186] P. H. Schiller, B. L. Finlay, and S. F. Volman. Quantitative studies of single-cell properties in monkey striate cortex. iii. spatial frequency. J Neurophysiol, 39(6):1334–1351, Nov 1976.

[187] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. Neural Comput, 19(3):757–779, Mar 2007.

[188] M. Schoenauer. Evolutionary computation: Basic algorithms. Notes from Master Course at University Paris-Sud.

[189] M. Schoenauer and E. Ronald. Genetic extension of neural net training: transfer functions and renormalisation coefficients. In J.-M. Alliot et al., editor, First Conference on Artifical Evolution. Cepadues, 1994.

[190] M. Schoenauer and E. Ronald. Neuro-genetic truck backer-upper controller. In Proc. 1st Intl Conf. on Evolutionary Computation. IEEE Press, 1994.

[191] H.-P. Schwefel. Numerical optimization of Computer models. John Wiley & Sons, Ltd., Chichester, 1981.

[192] Science Magazine. Complex Systems, volume 284, No 5411 of Special Issue. Science, 2 April 1999.

[193] Y. Semet and M. Schoenauer. On the benefits of inoculation, an example in train scheduling. In GECCO 06, pages 1761–1768. ACM Press, 2006.

[194] T. Serre, M. Kouh, C. Cadieu, U. Knoblich, G. Kreiman, and T. Poggio. A theory of object recognition: computations and circuits in the feedforward path of the ventral stream in primate visual cortex. In AI Memo 2005-036 / CBCL Memo 259, MIT, 2005.

[195] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio. A quantitative theory of immediate visual recognition. Prog Brain Res, 165:33–56, 2007.

[196] T. Serre, A. Oliva, and T. Poggio. A feedforward architecture accounts for rapid categorization. Proc Natl Acad Sci U S A, 104(15):6424–6429, Apr 2007.

[197] T. Serre and M. Riesenhuber. Realistic modeling of simple and complex cell tuning in the hmax model, and implications for invariant object recognition in cortex. Technical report, Massachusetts Institute of Technology, July 2004.

[198] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. IEEE Trans Pattern Anal Mach Intell, 29(3):411–426, Mar 2007.

[199] D. Simard, L. Nadeau, and H. Kroger. Fastest learning in small-world neural networks. Phys. Lett. A, 336:8–15, 2005.

[200] M. D. Skowronski and J. G. Harris. Automatic speech recognition using a predictive echo state network classifier. Neural Netw, 20(3):414–423, Apr 2007.

[201] K. Stanley. Compositional Pattern Producing Networks: A Novel Abstraction of Development. Genetic Programming and Evolvable Machines, 8(2):131–162, 2007.

[202] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. Artificial Life, Jan 2009.

[203] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In W. B. Langdon et al., editor, Proc. GECCO'02, pages 569–577. Morgan Kaufmann, 2002.

[204] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, 2002.

[205] D. Stauffer, A. Aharony, L. da Fontoura Costa, and J. Adler. Efficient Hopfield pattern recognition on a scale-free NN. Eur. Phys. J. B, 32:395–399, 2003.

[206] S. W. Stepniewski and A. J. Keane. Pruning back-propagation neural networks using modern stochastic optimization techniques. Neural Computing and Applications, 5:76–98, 1997.

[207] S. H. Strogatz. Exploring complex networks. Nature, 410(6825):268–276, 2001.

[208] R. S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In Proc. 8th Annual Conf. of Cognitive Science Society, pages 823–831, 1986.

[209] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

[210] Y. Takefuji. Neural network parallel computing. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[211] D. Thierens. Dimensional analysis of allel-mixing revisited. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, Proc. PPSN IV, pages 255–265. Springer Verlag, 1996.

[212] A. M. Treisman and G. Gelade. A feature-integration theory of attention. Cogn Psychol, 12(1):97–136, Jan 1980.

[213] R. L. D. Valois, D. G. Albrecht, and L. G. Thorell. Spatial frequency selectivity of cells in macaque visual cortex. Vision Res, 22(5):545–559, 1982.

[214] R. L. D. Valois, E. W. Yund, and N. Hepler. The orientation and direction selectivity of cells in macaque visual cortex. Vision Res, 22(5):531–544, 1982.

[215] D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir-based techniques for speech recognition. In Proc. International Joint Conference on Neural Networks IJCNN ’06, pages 1050–1053, 16–21 July 2006.

[216] J. Vesanto. Som-based data visualization methods. Intelligent Data Analysis, 3:111–126, 1999.

[217] T. Villmann, R. Der, and T. Martinetz. A new quantitative measure of topology preservation in kohonen's feature maps. In Proc. IEEE International Conference on Neural Networks IEEE World Congress on Computational Intelligence, volume 2, pages 645–648, 27 June–2 July 1994.

[218] T. Villmann and E. Merenyi. Extensions and modifications of the kohonen-som and applications in remote sensing image analysis. In U. Seiffert and L. Jain, editors, Self-Organizing Maps: Recent Advances and Applications, pages 121–145. Springer-Verlag, 2001.

[219] G. Wallis and E. T. Rolls. A model of invariant object recognition in the visual system. Prog. Neurobiol, 51:167–194, 1996.

[220] D. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. Nature, 393:440–442, 1998.

[221] M. Weber, M. Welling, and P. Perona. Unsupervised learning of models for recognition. In Proc. 1st European Conf. on Computer Vision, volume 1042 of LNCS, pages 18–32. Springer Verlag, 2000.

[222] S. Weigenda, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to foreasting. In Advances in Neural Information Processing Systems 3, pages 875–882, 1991.

[223] P. J. Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560, Oct. 1990.

[224] D. Whitley. The GENITOR algorithm and selective pressure: Why rank-Based allocation of reproductive trials is best. International Conference on Genetic Algorithms '89, pages 116–121, 1989.

[225] D. Whitley, S. Dominic, R. Das, and C. W. Anderson. Genetic reinforcement learning for neurocontrol problems. Mach. Learn., 13(2-3):259–284, 1993.

[226] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Computing, 14(3):347–361, August 1990.

[227] L. D. Whitley, F. Gruau, and L. D. Pyeatt. Cellular encoding applied to neurocontrol. In L. J. Eshelman, editor, ICGA, pages 460–469. Morgan Kaufmann, 1995.

[228] J. M. Wolfe and S. C. Bennett. Preattentive object files: shapeless bundles of basic features. Vision Res, 37(1):25–43, Jan 1997.

[229] Y. Xue, L. Yang, and S. Haykin. Decoupled echo state networks with lateral inhibition. Neural Netw, 20(3):365–376, Apr 2007.

[230] X. Yao. Evolving artificial neural networks. Proceedings of the IEEE, 87(9):1423–1447, Sept. 1999.

[231] T. Yu, L. Davis, C. Baydar, and R. Roy, editors. Evolutionary Computation in Practice. Number 88 in Studies in Computational Intelligence. Springer Verlag, 2008.

[232] K. Ziemelis, editor. Complex systems, volume 410, No 6825. Nature Insight, Nature Insight, 8 March 2001.