UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CADRE AUTOMATIQUE DE DIMENSIONNEMENT DES COMPOSANTS DE CIRCUITS MICROÉLECTRONIQUES ANALOGIQUES PILOTÉ PAR RÉSEAUX DE NEURONES

MÉMOIRE PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
ÉTIENNE DUMESNIL

MARS 2019

TABLES DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

ANN	Artificial neural network
C-SNN	Cascaded shallow neural network
CMOS	Complementary metal oxide semiconductor
DNN	Deep neural network
EDA	Electronic design automation
LC-SNN	Limited cascaded shallow neural network
MLP	Multi-layer perceptron
LNA	Low-noise amplifier
RNA	Réseau de de neurones artificiels
RF	Radiofréquences / Radio-frequency
SNN	Shallow neural network
UC-SNN	Unlimited cascaded shallow neural network
VCO	Voltage-controlled oscillator

RÉSUMÉ

Dans ce mémoire, quatre méthodes sont proposées, chacune plus efficace que la précédente, afin de réaliser le dimensionnement automatique des composants d'un circuit microélectronique analogique en hautes fréquences, par réseaux de neurones artificiels. Dans les quatre méthodes, un algorithme génétique a été utilisé afin de fixer les hyperparamètres des réseaux de neurones artificiels. La première méthode inclus un unique réseau de neurones profond à plusieurs sorties. Cette méthode n'a permis de prédire correctement aucune dimension de composant, en raison du trop petit nombre d'exemplaires disponibles pour l'entraînement par rapport au très grand nombre de paramètres à optimiser. La deuxième méthode consiste en un ensemble de réseaux de neurones indépendants l'un de l'autre, où chaque réseau cherche à prédire la dimension d'un seul composant. Cette méthode a permis de prédire les dimensions de quelques composants, mais pour aucun des circuits microélectroniques utilisés elle n'a permis de prédire toutes les tailles de composants. La troisième méthode constitue une cascade de réseaux de neurones contraignant progressivement le problème de dimensionnement des composants. Cette méthode n'ajoute un réseau de neurones à la cascade que s'il prédit correctement la taille d'un composant du circuit microélectronique. De cette manière, on est parvenu à prédire correctement les dix valeurs de composants d'un amplificateur à faible bruit. Toutefois, cette méthode n'a pas permis de prédire correctement toutes les tailles de composants pour un oscillateur contrôlé par une tension, ni pour un mélangeur. Enfin, la quatrième méthode propose encore une fois une cascade de réseaux de neurones, mais, un nouveau réseau est maintenant ajouté à la cascade dès qu'il améliore la meilleure prédiction précédemment obtenue pour un même composant. Ceci fait en sorte que chacun des composants à prédire peut bénéficier de contraintes provenant de tous les autres composants. Cette méthode a permis de prédire adéquatement les valeurs des composants de trois types de circuits dans deux technologies différentes : amplificateur à faible bruit, oscillateur contrôlé par une tension et mélangeur; CMOS 180 nm et CMOS 130 nm. Tous les résultats obtenus sont présentés dans un article formant le chapitre 3 du présent mémoire.

Mots clés : dimensionnement, synthèse, réseau de neurones artificiels, algorithme génétique, circuit analogique, microélectronique, amplificateur à faible bruit, oscillateur contrôlé par une tension, mélangeur

ABSTRACT

In this thesis, four methods are proposed, each more efficient than the previous one, in order to realize the automatic sizing of the components of an analog microelectronic circuit at high frequencies, by artificial neural networks. In all four methods, a genetic algorithm was used to optimize the hyperparameters of the artificial neural networks. The first method includes a single deep neural network with multiple outputs. This method could not correctly predict the size of any components. It is hypothesized that the problem comes from the very small number of exemplars available to train the deep neural network compared to the very large number of parameters that need to be optimized. The second method consists of a set of shallow neural networks that are independent of each other, where each network seeks to predict the size of a single component. This method made it possible to predict the size of some components. However, it could not predict correctly the sizes of all the components of any given microelectronic circuits. The third method consists in a cascade of neural networks which gradually constrain the problem of component sizing. Indeed, the prediction of a neural network within the cascade is used as an input to all subsequent neural networks of the same cascade. This method only adds a neural network to the cascade if it correctly predicts the size of a microelectronic circuit component. This method allowed to correctly predict the sizes of all ten components of a low noise amplifier. However, it did not correctly predict all component sizes of a voltage-controlled oscillator, nor of a mixer. Finally, the fourth method also proposes a cascade of neural networks, however, a new network is now added to the cascade as soon as it improves the best prediction previously obtained for the same component. This ensures that each of the components to be predicted can benefit from constraints from all other components. This method adequately predicted the component values of three types of circuits in two different technologies: low noise amplifier, voltage-controlled oscillator, and mixer; 180 nm CMOS and 130 nm CMOS. All the results obtained are presented in an article forming chapter 3 of the present thesis.

Keywords : sizing, synthesis, artificial neural network, genetic algorithm, analog circuit, microelectronic, low-noise amplifier, voltage-controlled oscillator, mixer.

INTRODUCTION

Le dimensionnement des composants d'un circuits microélectroniques analogiques requiert une expertise importante et constitue souvent davantage un art qu'une science exacte. Or, l'une des caractéristiques d'une expertise est sa nature implicite : il est très difficile pour un expert de verbaliser ses connaissances et ses méthodes de résolution de problèmes. Néanmoins, ses connaissances s'actualisent bel et bien lorsque vient le temps d'exercer son savoir-faire.

L'outil principal assistant le concepteur de circuits microélectroniques analogiques est un outil d'analyse : un programme renvoyant le comportement d'un circuit, partant de ses caractéristiques physiques. Ainsi, si l'on fournit à un outil d'analyse la topologie d'un circuit, ainsi que les détails de ses composants, tels que leurs capacitances, inductances, résistances, les tailles des canaux des transistors, les valeurs des sources de tension et de courant, etc., alors le programme fournira en retour les détails du comportement du système, tels que les tensions, courants, réponses en fréquences, gains, etc., aux différents nœuds du circuit. Le concepteur pourra alors ajuster les valeurs des composants du circuits ou la topologie du circuit afin de s'approcher du comportement de circuit souhaité. Il s'agit ainsi d'un processus itératif, où le programme analyse le circuit réalisé par le concepteur et lui donne une rétroaction.

Dans cette situation, alors que le programme fait face à un problème d'analyse, le concepteur fait plutôt face à un problème de synthèse. Dans l'analyse d'un circuit, l'entrée est un ensemble de composants structurés de manière à former un tout et la sortie correspond aux tensions et courants aux différents nœuds. Dans la synthèse d'un circuit, le problème est inversé : l'entrée correspond aux tensions et courants souhaités

à différents nœuds et la sortie est un ensemble de composants structurés de manière à réaliser le circuit permettant d'obtenir ces tensions et courants. Ainsi, le problème de synthèse nécessite de renverser le processus de causalité, l'objectif étant de trouver une topologie de circuit et un ensemble de valeurs de composants répondant à un ensemble de performances souhaitées.

Formellement, deux problèmes sont réversibles l'un par rapport à l'autre si la formulation de l'un implique tout ou partie de la solution de l'autre (Keller, 1976). Le problème le mieux compris est appelé le problème direct, tandis que l'autre est appelé le problème inverse. Dans le contexte des circuits analogiques, le problème direct est de trouver le comportement du circuit compte tenu de la topologie du circuit et des valeurs des composants, alors que la recherche de la topologie et des valeurs des composants compte tenu du comportement du circuit est le problème inverse. Dans le mémoire présenté ici, le problème inverse est simplifié en considérant la topologie connue, limitant ainsi le problème de synthèse à un problème de dimensionnement des composants.

La résolution du problème de dimensionnement des composants est particulièrement difficile pour les circuits en radiofréquences, l'approche typique consistant en un processus itératif fastidieux de simulations suivi d'ajustements, en raison de la difficulté de modéliser ces circuits par des équations linéaires régulières. Par exemple, la conception actuelle des amplificateurs à faible bruit en radiofréquences nécessite généralement plusieurs heures de réglage par un expert. Une explication à cela est que la plupart des équations de circuit deviennent non linéaires aux fréquences gigahertz (GHz) et que les modèles-groupés de composants (« *lumped-element model* ») ne sont plus représentatifs du comportement du circuit. En effet, à de telles fréquences, chaque composant possède des champs électriques et magnétiques associés, ainsi qu'une perte dissipative (Bahl, 2003), ce qui conduit à des équations de circuit très complexes. Enfin, des effets secondaires tels que ceux des capacités parasites apparaissent. Ainsi, un

solveur capable de dimensionner automatiquement et efficacement les composants, qui apprendrait de lui-même sans nécessiter la programmation explicite des relations inverses « performances – valeurs des composants du circuits », s'avérerait très utile. Aussi, le présent mémoire propose une méthode afin de réaliser cette tâche, soit par la construction progressive d'une séquence de réseaux de neurones artificiels (RNA) où chaque RNA spécifie la valeur d'un composant du circuit et contribue à faciliter la construction des RNA subséquents.

Voici l'organisation du présent mémoire : le chapitre 1 présente le cadre théorique pertinent à la problématique de dimensionnement des composants dans la synthèse d'un circuit microélectronique, le chapitre 2 expose la méthodologie utilisée, ainsi que les quatre itérations ayant menées au développement de l'architecture neuronale proposée. Le chapitre 3 constitue un article présentant les résultats obtenus pour l'architecture neuronale proposée dans le dimensionnement automatique de trois types de circuits microélectroniques. Enfin, une conclusion générale présente une discussion intégrée de l'ensemble des résultats, ainsi que des perspectives sur des recherches futures.

CHAPITRE I

CADRE THÉORIQUE

L'automatisation de la solution des problèmes inverses fait l'objet de recherches pluridisciplinaires depuis près d'un siècle, car ces problèmes sont omniprésents dans de nombreux domaines scientifiques et technologiques (Anantrasirichai et al., 2017, Marques et al., 2003, Starck, 2016). En effet, chaque fois qu'un système physique doit être inféré à l'aide de mesures, un problème inverse apparaît (Tarantola, 2005) et l'on doit alors établir si ce problème est « bien posé » afin de déterminer si une solution existe. Formellement, un problème est « bien posé » s'il répond aux trois critères de Hadamard (Hadamard, 1923) : (1) le problème a une solution; (2) la solution est unique; (3) le comportement de la solution change continuellement avec la modification des données et des paramètres. Un problème qui ne respecte pas ces trois critères est dit mal posé. Pour les systèmes déterministes, le problème direct a toujours une solution unique (qui peut être dynamique), mais le problème inverse est souvent indéterminé, avec un nombre infini de solutions possibles. De plus, il peut ne pas répondre au troisième critère de Hadamard quant à la stabilité de la solution, de sorte que de petites modifications de paramètres ou de données entraînent de grandes variations dans la précision de la solution. Pour ces raisons, les problèmes inverses sont souvent « mal posés ».

De nombreuses méthodes de dimensionnement des composants et de synthèse d'un circuit ont été proposées au cours des dernières décennies, notamment les méthodes basées sur le savoir (Sheu et al., 1990, Harjani, 1989), le raisonnement basé sur les cas

(Kashef et al., 2008, Sadughi, 2002), le recuit simulé (Kirkpatrick, 1983, Ochotta et al., 1996), les algorithmes génétiques (Holland, 1992, Barros et al. 2010) et la programmation génétique (Koza et al. 1997, Lohn et Colombano, 1998). Cependant, aucune de ces approches ne s'est imposée comme méthode générale systématique pour dimensionnement des composants d'un circuits analogiques. En fait, seules les méthodes basées sur le savoir et celles basées sur les cas ont tenté de résoudre le problème inverse grâce à une correspondance générale entre les critères de performance souhaités et les valeurs des composants du circuit. En effet, les autres méthodes ont seulement essayé de trouver une solution idiosyncratique pour un ensemble donné de critères de performance. La difficulté à trouver une correspondance générale réside dans le caractère « mal posé » du problème inverse, qui rend la solution instable. Une famille de méthodes couramment utilisée pour résoudre le problème de la stabilité est la régularisation. Comme introduit dans (Tikhonov, 1963), la régularisation fait référence à l'ajout d'informations préalables au système inféré pendant le processus d'optimisation, ce qui permet un lissage de la fonction en approchant la solution. Or, les réseaux de neurones artificiels utilisent généralement la régularisation au cours de leur processus d'apprentissage, ce qui en fait une méthode de choix potentielle pour le dimensionnement des composants d'un circuits. Pourtant, malgré les succès récents obtenus par les réseaux de neurones profonds cette méthode n'a pas été proposée jusqu'ici dans la documentation dans le contexte de la conception de circuits microélectroniques. Les raisons potentielles de cette omission seront soulevées dans le présent chapitre et une manière de pallier ces raisons seront proposées dans le chapitre suivant.

Dans le présent chapitre, un survol de la documentation consacrée au problème du dimensionnement des composants d'un circuit de circuits analogiques est présenté.

1.1 Méthode basée sur le savoir

Les méthodes basées sur le savoir sont parmi les plus anciennes méthodes développées. Cette approche est basée sur l'expérience de concepteurs experts, qui élaborent généralement un plan pour définir les valeurs des paramètres de conception (Sheu et al. 1990). Le plan consiste en une série d'étapes prédéfinies comprenant des équations reliant les critères de performance aux paramètres de conception finaux. L'approche était efficace pour des conceptions relativement petites telles que des amplificateurs opérationnels (Harjani, 1989). Cependant, les circuits de plus grandes tailles augmentent rapidement en complexité, comportant des relations non linéaires à prendre en considération entre les composants et le comportement du circuit, ce qui rend difficile l'élaboration d'un plan de conception efficace (Sasikumar et al., 2016). De plus, les méthodes basées sur le savoir étant essentiellement des systèmes experts, elles souffrent de la difficulté fondamentale de ce type de système, soit la difficulté à extraire la connaissance de l'expert : rendre explicite la connaissance procédurale implicite de l'expert (Kidd, 1987). Ce problème d'acquisition de connaissances a conduit de nombreux chercheurs à rechercher d'autres méthodes capables de trouver une solution de manière autonome, au lieu de compter sur la capacité d'un expert à transférer ses connaissances.

En résumé, dans le contexte du dimensionnement des composantes d'un circuit microélectronique, l'avantage principal des méthodes basées sur le savoir est que la solution proposée est généralisable à une grande plage de circuits. Le désavantage principal de ces méthodes est qu'elle n'a pu être appliquée jusqu'ici qu'à des circuits très simples, comme des amplificateurs opérationnels.

1.2 Raisonnement basé sur des cas

La méthode de raisonnement basé sur les cas consiste en un processus de quatre étapes qui peut se résumer comme suit: récupérer, réutiliser, réviser et conserver (Aamodt et Plaza, 1994, Kolodner, 1992). Tout d'abord, une recherche est effectuée dans un espace de solution, c'est-à-dire via une bibliothèque de cas précédemment utilisés et mémorisés. Cette idée de stocker et de réutiliser les problèmes et solutions précédemment rencontrés a été présentée par Schank à travers le concept de mémoire dynamique (Schank, 1982). Dans le contexte du dimensionnement des composants d'un circuits, la recherche explore l'espace des ensembles de critères de performance préalablement mémorisés. Un cas est sélectionné et contient l'ensemble des valeurs des paramètres du composant. Si le comportement du circuit récupéré est identique à celui requis, les valeurs de ses composants sont simplement réutilisées. Dans le cas contraire, les paramètres du composant doivent être révisés, auquel cas une méthode basée sur le savoir est généralement utilisée (Al-Kashef et al., 2008). Ainsi, le raisonnement basé sur les cas permet de réduire le problème d'acquisition des connaissances, mais ne le résout pas complètement. Par conséquent, il souffre de limites similaires à celles des méthodes basées sur le savoir, en particulier lorsqu'il s'agit de réviser les solutions de circuits complexes. De plus, le raisonnement basé sur les cas nécessite une grande mémoire de stockage, compte tenu de la bibliothèque d'exemplaires de circuits mémorisés (Smiti et Elouedi, 2011). En outre, comme il sera discuté plus loin, il est très difficile de rassembler une quantité suffisamment importante d'exemplaires de circuits microélectroniques pour une topologie donnée dans une technologie donnée. Par conséquent, dans le contexte du dimensionnement des composants d'un circuits, l'espace de recherche contenant les ensembles de critères de performance mémorisés demeure clairsemé et son utilité est donc limitée.

1.3 Recuit simulé

Le recuit simulé considère le problème de dimensionnement des composants comme un problème d'optimisation et le résout en explorant l'espace de conception (Aarts, 2005). Au début, l'algorithme l'explore d'un point de vue élevé, ce qui réduit ses chances de rester coincé dans des minima locaux. Ensuite, le processus d'optimisation progresse avec une granularité de plus en plus fine jusqu'à ce que la solution finale soit atteinte. À chaque étape du processus, la solution actuelle est comparée à une solution voisine. Si le voisin donne de meilleurs résultats, il remplace la meilleure approximation précédente, mais si le voisin donne de moins bons résultats, il a seulement une certaine probabilité de la remplacer, laquelle diminue avec le temps.

Cette approche a deux limites importantes. Tout d'abord, dans le cadre du problème de dimensionnement des composants d'un circuits, pour chaque comparaison de solutions voisines, les circuits correspondants doivent être simulés (via un logiciel d'analyse de circuit) pour trouver l'erreur entre le comportement de chacun et celui souhaité. Par conséquent, l'algorithme peut prendre beaucoup de temps pour des problèmes complexes. Deuxièmement, la solution finale ne s'applique qu'au comportement du circuit actuellement conçu. Il n'est pas généralisable aux comportements d'autres circuits pouvant être obtenus avec la même topologie. En effet, la méthode n'apprend pas une solution générale, se limitant plutôt à trouver une solution idiosyncratique pour le comportement du circuit requis. Ainsi, l'algorithme doit être redémarré pour chaque nouveau comportement souhaité. Cela peut être une limitation sérieuse en termes de temps de conception. Par exemple, la méthode de recuit simulé proposée dans (Weber et al., 2013) nécessitait entre une heure et trois heures sur une unité centrale de 2,3 GHz pour compléter le dimensionnement des composants et ce, pour chaque ensemble de critères de performances d'un amplificateur à faible bruit. Une autre approche de recuit simulé proposée dans (Yang et al., 2018) nécessitait environ une heure pour synthétiser un nouvel oscillateur contrôlé par une tension sur un processeur de 2,4 GHz.

1.4 Algorithme génétique

La recherche effectuée par un algorithme génétique est similaire à celle réalisée par un recuit simulé, dans la mesure où les solutions potentielles sont en compétition. L'algorithme est toutefois différent. Au lieu de deux voisins en compétition à chaque itération, comme c'est le cas dans un recuit simulé, l'algorithme génétique comporte toute une population de solutions potentielles. Chaque solution potentielle est codée dans un "chromosome", par analogie avec la biologie. Dans le contexte du dimensionnement de composants de circuits, ce chromosome définirait typiquement les valeurs des composants. La population de chromosomes passe ensuite par une série de cycles de sélection-reproduction-mutation jusqu'à ce que l'un des chromosomes converge vers un ensemble acceptable de valeurs de composants (Whitley et Sutton, 2012). Pendant la partie de sélection d'un cycle, tous les chromosomes doivent être comparés, ce qui signifie que toutes les conceptions de circuit correspondantes doivent être simulées avec un outil d'analyse de circuit pour déterminer quels chromosomes conduisent aux plus petites erreurs. Par conséquent, de la même manière que pour un recuit simulé, l'algorithme génétique nécessite beaucoup de temps de simulation. De plus, encore comme pour un recuit simulé, la solution trouvée par l'algorithme génétique n'est pas généralisable à d'autres comportements potentiels souhaités pour une même topologie de circuit. Par exemple, un algorithme génétique implémenté dans (Barros et al., 2010) prenait constamment plus de 30 minutes pour dimensionner les composants d'un amplificateur à faible bruit. Or, dans les deux cas, le processus devait être répété pour tout nouvel ensemble de critères de performance souhaités.

1.5 Programmation génétique

La programmation génétique est désormais l'une des tendances majeures, non seulement dans le dimensionnement automatique des composants de circuits, mais plus

largement dans la synthèse automatique de circuits (par synthèse, on entend la sélection de la topologie en plus du dimensionnement des composants). En effet, il a été démontré que la programmation génétique fonctionne bien non seulement pour trouver un ensemble adéquat de valeurs pour les composants, mais aussi pour trouver une topologie adéquate, incluant la sélection des composants et des interconnexions du circuit. Tel que proposé dans (Koza, 1997), un arbre est alors utilisé comme représentation du circuit microélectronique analogique. Tout d'abord, un circuit embryonnaire est généré, qui est la base sur laquelle le circuit final sera développé. Les branches de l'arbre correspondent à des fonctions utilisées pour construire le circuit. Il existe quatre types de fonctions: (1) des fonctions de modification de connexion, (2) des fonctions de création de composants, (3) des fonctions arithmétiques pour spécifier les valeurs des composants et (4) des fonctions définies automatiquement permettant de réutiliser certaines sous-structures. Le processus d'évolution de l'arbre lui-même est similaire au processus d'un algorithme génétique, où l'évolution des branches correspond à l'évolution des fonctions et donc du programme de construction du circuit (voir également (Lohn et Colombano, 1998) pour un schéma de codage différent). Le principal avantage de la programmation génétique sur l'algorithme génétique et le recuit simulé est son efficacité dans la synthèse de l'ensemble du circuit (à savoir, trouver une topologie adéquate et un ensemble de valeurs de composants), plutôt que de n'avoir qu'une efficacité démontrée pour le dimensionnement de ses composants.

Plus récemment, de nombreuses variantes de techniques de programmation génétique ont été proposées pour le dimensionnement de circuits (e.g. Liao et al., 2017), permettant d'obtenir des solutions très précises et robustes. Cependant, ces variantes présentent toutes la même limite : la solution trouvée est toujours limitée au seul ensemble de critères de performance utilisé dans le processus d'optimisation. De plus, cette approche prend beaucoup de temps car chaque solution potentielle doit être testée via une simulation d'analyse du circuit. Or, la méthode qui sera proposée dans le présent travail présente les avantages et inconvénients inverses (c.-à-d. solution moins robuste,

mais plus générale). Ainsi, la méthode proposée ici pourrait être utilisée dans le futur pour complémenter les algorithmes de programmation génétique en leur fournissant des conditions initiales et ainsi permettre une méthode plus efficace de synthèse complète de circuits.

1.6 Réseaux de neurones profonds

Un réseau de neurones profonds est un réseau de neurones artificiels composé de plusieurs couches. Chaque couche est composée de nombreux neurones artificiels. Un neurone artificiel est une unité de traitement qui applique une fonction linéaire ou non linéaire à la somme de ses entrées. Dans un réseau de neurones profond typique, les sorties neuronales de chaque couche se connectent uniquement aux entrées neuronales de la couche suivante. L'apprentissage se fait par un processus de minimisation de l'erreur à la sortie de la couche finale. Dans l'algorithme d'apprentissage le plus populaire, soit celui de rétropropagation de l'erreur, l'erreur est propagée à travers le réseau en utilisant une chaîne de dérivées partielles allant de la sortie vers l'entrée du réseau. Les poids de connexion entre les neurones sont alors ajustés en fonction de leur contribution à l'erreur calculée en sortie. Dans le contexte des problèmes inverses, les réseaux de neurones profonds ont été particulièrement performants en imagerie médicale, où ils sont utilisés notamment pour construire des images de structures physiologiques internes à partir de mesures effectuées avec différentes techniques d'imagerie : par exemple, tomodensitométrie (Commandeur et al., 2018), résonance magnétique imagerie (IRM) (Liu et al., 2018), imagerie par résonance magnétique fonctionnelle (IRMf) (Kasabov, 2017) ou signaux électroencéphalographiques (EEG) (Hosseini, 2017). Lorsqu'un réseau de neurones profond est utilisé dans le traitement d'images, les réseaux neuronaux convolutifs sont plus précisément la méthode de choix (Fukushima, 1982, LeCun, 1989, Liu, 2017). Dans un réseau de neurones convolutif, certaines couches contiennent ce que l'on appelle des cartes d'attributs, qui tentent d'extraire des formes spécifiques de la couche précédente. Les premières couches

extraient des formes simples, telles que des formes horizontales, verticales ou diagonales, tandis que les couches finales extraient des formes plus complexes, telles que des visages, des voitures ou des animaux. Ce type de réseau neuronal s'inspire des découvertes sur l'organisation du cortex visuel chez les mammifères (Hubel et Wiesel, 1959). Un autre domaine dans lequel les réseaux de neurones profonds sont très utiles est le langage, par exemple pour la traduction (Sutskever et al., 2014). Dans ce cas, les réseaux neuronaux récurrents sont plus précisément la méthode de choix. Dans un réseau de neurones récurent, la sortie d'un neurone est fonction de son entrée actuelle et de ses états passés, ce qui permet au réseau d'apprendre des informations séquentielles. Les types de réseaux de neurones récurents les plus utilisés sont le réseau de longue mémoire à court terme (Hochreiter, 1997) et le réseau d'unités récurrentes à portes (Cho et al. 2014).

Dans le cadre du dimensionnement des composants d'un circuits, les réseaux de neurones profonds offrent la promesse de résoudre les principaux problèmes des méthodes décrites dans les sections précédentes. Premièrement, ils n'ont pas le problème d'acquisition des connaissances des approches basées sur le savoir ou basées sur des cas, car ils apprennent de manière autonome à partir d'exemples donnés. De plus, ils ne souffrent pas des deux limites principales des approches par recuit simulé, par algorithme génétique et par programmation génétique. D'abord, les données d'entraînement étant issues de circuits déjà dimensionnés, il n'est pas nécessaire d'analyser les solutions potentielles durant le processus d'apprentissage, ce qui le rend beaucoup plus rapide. Ensuite, la solution générée par un réseau de neurones profond constitue une correspondance générale entre les variables correspondant aux critères de performance et les variables correspondant aux valeurs des composants du circuit. Il ne s'agit donc pas d'une solution idiosyncratique. En effet, au terme de la période d'apprentissage, alors que les méthodes par recuit simulé, par algorithme génétique et par programmation génétique cherchent à converger vers une solution propre à un unique circuit, le réseau de neurones profond cherche plutôt à converger vers une

correspondance entre n'importe quel ensemble d'entrées et l'ensemble de sorties approprié. Par conséquent, à long terme, le temps investi par un concepteur humain dans le dimensionnement des composants d'une banque de circuits destinés à entraîner un réseau de neurones profond a le potentiel d'être largement compensé par la réutilisation du même réseau pour les dimensionnements futurs d'une même topologie de circuit.

La question suivante s'impose alors : quel est le nombre de circuits déjà dimensionnés requis pour qu'un réseau de neurones profond puisse trouver une telle solution? Or, il s'avère que la réponse à cette question constitue un obstacle majeur à l'adoption de ces réseaux en tant que technique standard pour le dimensionnement des composants des circuits analogiques. En effet, le nombre de paramètres qu'un réseau de neurones profond doit optimiser durant son apprentissage afin de converger vers une solution générale augmente très rapidement avec le nombre de paramètres à optimiser. Par conséquent, pour un réseau de neurones à plusieurs couches, afin que le processus d'apprentissage soit efficace, une très grande quantité d'exemplaires de circuits d'entraînement est généralement requise. Malheureusement, s'il est relativement facile de rassembler de grandes quantités de données correspondant à des images d'une même catégorie (ex. des chiens) ou de données correspondant à une même langue, entre autres grâce aux internets, il est beaucoup plus difficile de rassembler de grandes quantités de données correspondant à des variantes d'un même circuit microélectronique. Il y a plusieurs raisons à cela. Premièrement, il faut beaucoup plus de temps pour faire le dimensionnement des composants d'un circuit que pour prendre une photo ou écrire un message. Deuxièmement, la communauté des concepteurs de circuits microélectroniques analogiques est beaucoup moins importante que le nombre de personnes qui publient des images ou des messages sur les internets. Troisièmement, la technologie dans laquelle les circuits analogiques sont conçus évolue très rapidement. Ce faisant, alors que des images ou des messages d'il y a quelques années peuvent être utilisés dans les mêmes ensembles de données que les images ou les messages publiés

aujourd'hui, il ne serait pas possible de combiner dans un même ensemble d'entraînement des circuits dans une technologie CMOS 180 nm et des circuits dans une technologie CMOS 130 nm, car leurs comportements seront différents. Par conséquent, l'évolution rapide de la technologie rend difficile la collecte d'un ensemble de données suffisamment important pour permettre l'entraînement efficace d'un réseau de neurones profond.

Les réseaux de neurones profonds semblent ainsi constituer une méthode présentant un potentiel intéressant pour le dimensionnement des composants d'un circuit. Cependant, le nombre relativement faible de circuits disponibles pour former l'ensemble d'entraînement limite le nombre de paramètres qui peuvent être optimisés efficacement par un tel réseau. Ainsi, il serait il ne serait pas possible d'entraîner un réseau réellement profond, c'est-à-dire comportant un grand nombre de couches de neurones. Or, le présent mémoire vise à pallier cette limite en proposant un réseau de neurone profond modifié, correspondant en fait à une séquence de réseaux de neurones superficiels (i.e. peu profonds). Comme chaque réseau de neurones superficiel comporte peu de paramètres à optimiser comparativement à un réseau de neurones profond, il est alors possible de l'entraîner avec un plus petit nombre d'exemplaires de circuits. De plus, comme ces réseaux superficiels sont entraînés séquentiellement, la solution trouvée par chacun d'entre eux peut être utilisée afin de contraindre l'espace de recherche des réseaux superficiels subséquents, ce qui contribue à faciliter la convergence de la séquence complète de réseaux superficiels vers une solution globale. La méthodologie utilisée afin de développer et tester cette nouvelle technique dans le cadre du dimensionnement automatique des composants de circuits microélectroniques analogiques en hautes fréquences est présentée au chapitre 2.

CHAPITRE II

MÉTHODOLOGIE

La recherche présentée ici a pour objectif de concevoir une architecture neuronale artificielle qui permettrait de réaliser le dimensionnement automatique des composants de circuits microélectroniques analogiques en hautes fréquences. Pour ce faire, on a procédé par itération, testant au passage quatre architectures distinctes. Dans chacun des cas, on a testé l'architecture avec trois types distincts de circuits microélectroniques analogiques en hautes fréquences : un amplificateur à faible bruit, un oscillateur contrôlé par une tension et un mélangeur. Chacune des itérations réalisées a permis d'améliorer les résultats obtenus, jusqu'à permettre de prédire l'ensemble des dimensions des composants recherchées avec une précision de 5 %. Quatre itérations ont été nécessaires. La première itération a testé un réseau de neurones profond (en anglais, *deep neural network* ; DNN), plus précisément un réseau de neurones profond avec connexions résiduelles, lequel cherchait à donner en sortie l'ensemble de tailles des composants d'un circuit, en fonction de l'ensemble des critères de performances requis fournis en entrée. La deuxième itération a testé l'utilisation parallèle de dix perceptrons multicouches, soit un pour chaque dimension de composant à prédire. La troisième itération a testé une nouvelle architecture, soit une cascade de réseaux de neurones superficiels (i.e. peu profonds), où une dimension de composant prédite par un réseau de la cascade était utilisée en entrée des réseaux de neurones subséquents afin de les contraindre. Dans cette architecture, nommée « cascade limitée de réseaux de neurones superficiels » (en anglais, *limited cascaded shallow neural networks* ; LC-SNN), chaque dimension de composant est prédite par

un seul réseau de neurone. La quatrième itération consiste encore une fois en une cascade de réseaux de neurones superficiels, mais cette fois la cascade peut contenir plusieurs réseaux de neurones prédisant la même dimension de composant. Dans le cas du LC-SNN un réseau de neurones n'était ajouté que s'il prédisant une dimension correctement avec une précision de 5 %, Or, on permet maintenant l'ajout d'un réseau neurones à la séquence dès qu'il améliore la meilleure prédiction obtenue jusqu'ici pour cette même dimension par un autre réseau de neurones de la séquence. Comme il n'y a donc pas de limite prédéterminée quant au nombre de réseaux de neurones que contiendra alors la séquence finale, cette architecture est nommée « cascade illimitée de réseaux de neurones superficiels » (en anglais, *unlimited cascaded shallow neural networks* ; UC-SNN)

Dans le présent chapitre, la section 2.1 présente les circuits microélectroniques analogiques utilisés pour tester les architectures neuronales. La section 2.2 présente la méthode d'optimisation des paramètres utilisée pour entraîner l'ensemble des réseaux de neurones testés. La section 2.3 présente l'algorithme génétique qui a été utilisé afin d'optimiser les hyperparamètres de l'ensemble des réseaux de neurones testés. Les sections 2.4 à 2.7 présentent respectivement chacune des quatre itérations d'architectures neuronales utilisées. Les résultats obtenus sont présentés dans le cadre d'un article soumis (Dumesnil et Boukadoum, 2019) par l'auteur, lequel constitue le chapitre 3 du présent mémoire.

2.1 Circuits microélectroniques analogiques en hautes fréquences utilisés pour tester les architectures neuronales

Afin de tester chaque itération de l'architecture neuronale, 200 exemplaires de trois types de circuits microélectroniques analogiques différents ont été générés : un amplificateur à faible bruit, un oscillateur contrôlé par une tension et un mélangeur. Ces trois types de circuits sont décrits dans la présente section.

2.1.1 Amplificateur à faible bruit

La première topologie de circuit microélectronique utilisée correspond à un amplificateur à faible bruit, tel que présenté à la Figure 2.1. Cette topologie consiste en un étage de type cascode à source commune avec une dégénérescence et une charge inductive. La configuration cascode a été sélectionnée pour sa simplicité de conception (appariement plus facile, stabilité, etc.) et son utilisation généralisée. Toutes ses inductances, y compris celles des réseaux d'appariement, ont été supposées non idéales avec un facteur de qualité de 10, ce qui est cohérent avec les processus de fabrication CMOS actuels. Pour simplifier davantage les efforts de conception et réduire le nombre de paramètres, seul le réseau d'adaptation en forme de L représenté à la Figure 2.1b est pris en compte pour l'adaptation de l'impédance de la source et de la charge à 50 Ω.

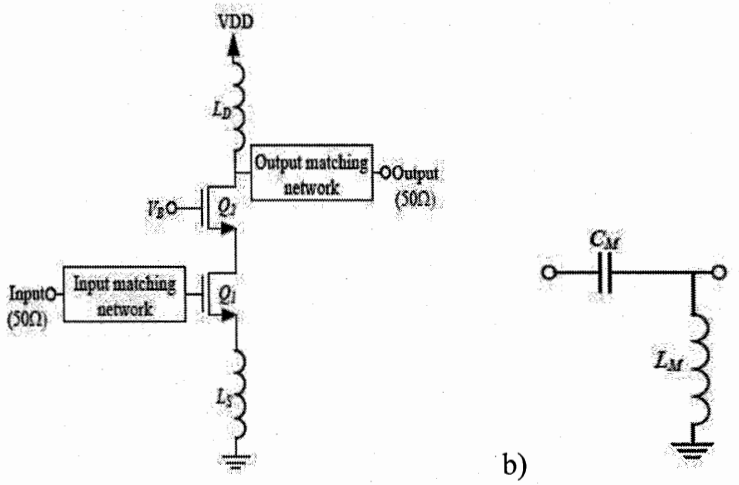

Figure 2.1. Topologie de (a) l'amplificateur à faible bruit utilisé et (b) du circuit d'appariement.

Pour tester les architectures neuronales, 200 circuits d'amplificateur à faible bruit valides ont été spécifiés aléatoirement dans la technologie CMOS 180 nm et simulés avec le simulateur de circuit *Cadence Virtuoso SpectreRF*, ce qui a requis de

nombreuses itérations pour chaque circuit. La méthodologie de conception a été effectuée en sélectionnant d'abord la valeur de l'inductance L_D pour obtenir une certaine fréquence de fonctionnement. Ensuite, la taille et la polarisation du transistor Q_1 ont été choisies de manière à générer différents exemplaires du circuit. La taille du transistor Q_2 a également été variée pour chaque circuit généré. Enfin, l'inductance L_S a été modifiée pour atteindre différentes spécifications de linéarité. Pour chaque conception, les entrées et les sorties ont été mises en correspondance en utilisant la topologie de réseau illustrée à la Figure 2.1b. Les plages de valeurs des composants utilisées dans la réalisation de ces 200 circuits d'amplificateurs à faible bruit sont détaillées au chapitre 3 (Tableau 3.2).

2.1.2 Oscillateur contrôlé par une tension

Le deuxième topologie de circuit microélectronique utilisée correspond à un oscillateur contrôlé par une tension. Celle-ci est illustrée à la Figure 2.2. Il s'agit d'une configuration commune qui permet une oscillation à la sortie rail à rail. La paire PMOS-NMOS à couplage croisé contribue à réduire le bruit 1/f (Razavi, 2011). 200 exemplaires de cet oscillateur contrôlé par une tension ont été conçus. La liste des variables de conception et de performance considérées est détaillée au chapitre 3 (Tableau 3.4). La source de courant et l'inductance ont été fixées avant de concevoir chaque exemplaire et ont été données en entrée pour le processus de dimensionnement des composants. Tous les exemplaires ont été conçus avec *Cadence Virtuoso SpecterRF* utilisant la technologie CMOS 130 nm.

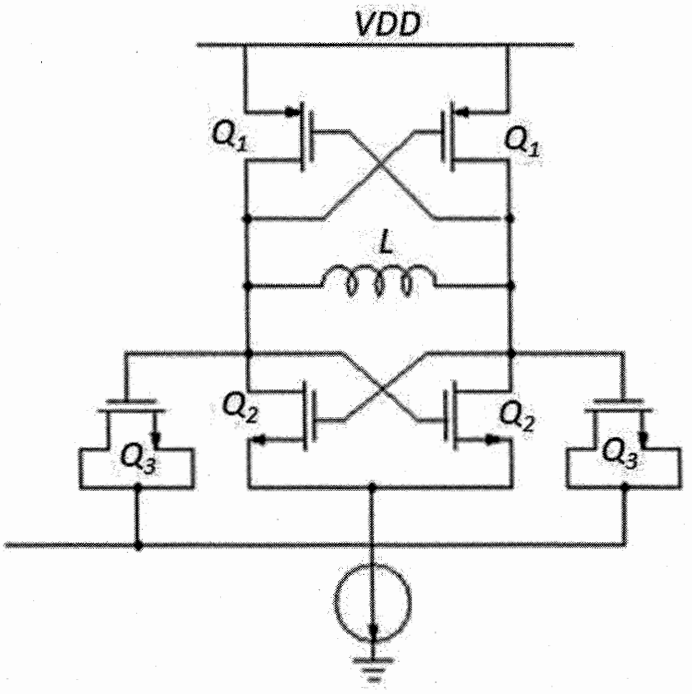

Figure 2.2. Topologie de l'oscillateur contrôlé par une tension utilisé.

2.1.3 Mélangeur

Enfin, la troisième topologie de circuit microélectronique utilisée correspond à un mélangeur et est illustrée à la Figure 2.3. Il s'agit d'une cellule de Gilbert à couplage croisé. Elle a été sélectionnée en raison de son utilisation courante, offrant un bon gain de conversion et une bonne réjection aux ports d'entrée (Razavi, 2011). De plus, la version utilisée ici comprend un résonateur qui permet, par son inductance parallèle L et sa capacité C, d'ajuster le mélangeur pour fournir une réponse maximale à la fréquence requise (Hu, 2017). 200 mélangeurs ont été conçus. La liste des variables de conception et de performance considérées est détaillée au chapitre 3 (Tableau 3.5). La résistance R, ainsi que la taille et la tension de grille du transistor Q_3, qui fait partie d'un miroir de courant, ont été données en entrée du processus de dimensionnement des

composants. Le mélangeur a été conçu avec *Cadence Virtuoso Spectre RF* en utilisant la technologie CMOS 130 nm. Les longueurs de tous les transistors ont été fixées à 130 nm.

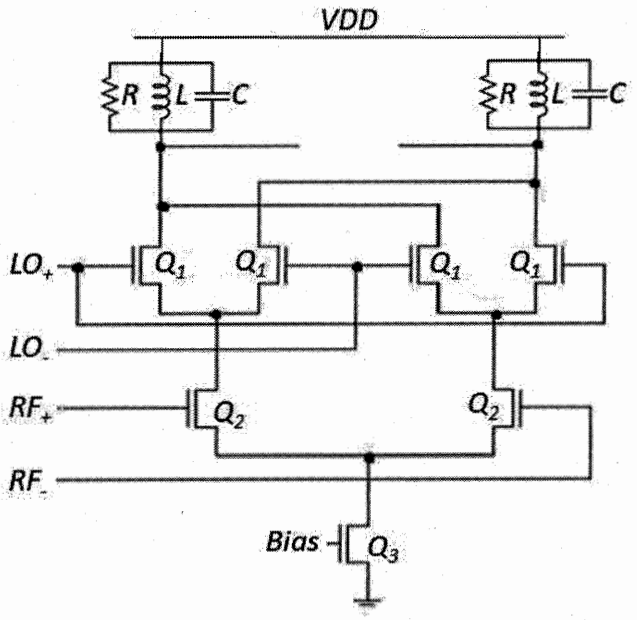

Figure 2.3. Topologie d'un mélangeur de type Cellule de Gilbert à couplage croisé.

2.2 Optimisation des paramètres des réseaux de neurones artificiels

Chaque réseau de neurone utilisé dans les différentes architectures neuronales testées a été entraîné à l'aide d'un algorithme de rétropropagation de l'erreur. Cet algorithme tente de minimiser l'erreur quadratique moyenne entre la sortie souhaitée et celle prédite, à l'aide d'une chaîne de dérivées partielles se propageant de la sortie vers l'entrée du SNN et ajustant au passage les poids des connexions en fonction de leur

contribution à l'erreur calculée en sortie (Schmidhuber, 2015). Afin d'entraîner un réseau de neurones avec ce type d'algorithme, les données empiriques utilisées consistent en des paires « entrées–cible ». Ces données sont divisées en trois groupes correspondant chacun à une phase du processus d'optimisation du modèle : (i) phase d'entraînement, (ii) phase de validation et (iii) phase de test. Une série de cycles « entraînement-validation » sont d'abord réalisés. La rétropropagation de l'erreur permettant l'optimisation des paramètres du réseau de neurone est alors effectuée seulement durant la phase d'entraînement. Les données de la phase de validation ne sont donc pas utilisées afin d'optimiser le modèle, ce qui permet de les utiliser afin de vérifier sa capacité à généraliser ses apprentissages. On réalise le cycle « entraînement-validation » itérativement jusqu'à ce que la performance durant la validation cesse de s'améliorer. Toutefois, comme à chaque cycle les mêmes données servent à la validation, cette procédure peut occasionner un biais. Ainsi, afin d'obtenir une mesure de performance de généralisation non biaisée, on préserve un groupe de données qui n'ont jamais été vues par le modèle afin de tester sa capacité à généraliser une fois que l'on a complété les cycles « entraînement-validation ». Ces données sont celles de la phase de test. Enfin, c'est la variante de rétropropagation de l'erreur nommée Adam (Kingma et Ba, 2014) qui a été utilisée, celle-ci étant présentement considérée comme l'une des plus efficaces.

2.3 Optimisation des hyperparamètres des réseaux de neurones artificiels

On a indiqué à la sous-section précédente que les poids des connexions d'un réseau de neurone sont optimisés automatiquement durant les cycles « entraînement-validation ». On réfère à ces poids de connexions en tant que « paramètres » du réseau de neurones. Or, un réseau de neurones possède également d'autres caractéristiques qui doivent être fixées, par exemples, le nombre de couches cachées, le nombre de neurones dans chaque couche cachée, le type de fonction de sortie des neurones, etc. Ces caractéristiques sont appelées « hyperparamètres ». L'optimisation de ces

hyperparamètres est l'une des principales difficultés rencontrées lors de la conception d'un réseau de neurones artificiels efficace (Bergstra et al., 2011) et exige ainsi beaucoup de temps d'ajustements au concepteur. Afin de pallier cette difficulté, un algorithme génétique a été utilisé ici afin d'optimiser certains de ces hyperparamètres.

D'abord, une population de chromosomes est générée aléatoirement. Ici, une population de 64 chromosomes a été utilisée (la taille de la population a été déterminée par essais et erreurs). À chaque chromosome correspond un réseau de neurones. Aussi, chaque chromosome est constitué d'une séquence de bits permettant de fixer les valeurs d'un ensemble d'hyperparamètres pour son réseau de neurones correspondant. Le processus de création et d'évolution des chromosomes va comme suit. Premièrement, une structure de chromosome binaire à k bits est définie, où chaque bit contribue à fixer la valeur d'un hyperparamètre d'un réseau de neurones. Ensuite, N instances de la structure chromosomique sont créées aléatoirement, formant la population de *génération 0*. La moitié de la population est sélectionnée pour la reproduction, où la probabilité qu'un chromosome soit sélectionné est proportionnelle à la performance du réseau de neurones correspondant. Dans cette moitié de population, les deux chromosomes ayant le mieux performé ont alors deux enfants ensemble, puis les deux chromosomes suivants ont également deux enfants ensemble et ainsi de suite. Le chromosome représentant chaque enfant est le résultat d'un croisement entre les chromosomes de ses parents. Deux points de croisement sont sélectionnés de manière aléatoire (par exemple, $11^{ème}$ et $22^{ème}$ bits). Ensuite, pour le premier enfant, les bits précédant le premier point de croisement et ceux suivant le second point de croisement sont copiés du premier parent, alors que les bits restants sont copiés du deuxième parent. Le chromosome du deuxième enfant provenant des mêmes parents correspond à l'inverse du chromosome du premier enfant. Enfin, chacun des bits du chromosome de chaque enfant est susceptible de subir une mutation avec une certaine probabilité, ce qui signifie que chaque bit peut être inversé. La population de N exemplaires ayant atteint ce stade est appelée *génération 1*. Ensuite, le cycle de sélection, de reproduction

et de mutation se répète, chaque nouveau cycle entraînant une nouvelle génération. L'algorithme génétique s'arrête lorsque le taux de réussite du meilleur réseau de neurones d'une génération atteint un certain seuil (une précision de 5 % durant la phase de validation). Une présentation plus détaillée de l'algorithme génétique utilisé ici est présentée au chapitre 3 (section 3.6.3).

2.4 Quatre itérations de l'architecture neuronale

De manière à permettre une comparaison aussi juste que possible entre les différentes architectures neuronales proposées, elles utilisent toutes le perceptron multicouche (en anglais, *multilayer perceptron* ; MLP) comme base fondamentale. On a choisi le MLP comme il s'agit de l'une des structures neuronales les plus connues et utilisées et que les quatre itérations comparées ici le sont pour la première fois. Toutefois, des études futures pourraient remplacer le MLP par une autre structure neuronale de base dans chacune des itérations proposées afin d'étendre les conclusion qui seront obtenues au terme de la présente étude.

Chaque MLP possède un certain nombre de couches cachées, en plus de la couche d'entrée et de la couche de sortie. Dans la couche d'entrée, chaque neurone contient simplement la valeur reçue. Dans chaque couche cachée, la sortie d'un neurone est donnée par

$$z_j = f_\theta\left(\sum_{i=1}^{M} \omega_i x_i\right)$$

où M est le nombre de neurones de la couche d'entrée, x_i est la sortie du neurone de la couche d'entrée, ω_i est un poids à déterminer et $f_\theta(\,)$ est une fonction de sortie. Ici, cette fonction de sortie était soit la fonction sigmoïde de tangente hyperbolique

$$f_{tansig} = \frac{2}{1 + e^{-2x}} - 1$$

ou la fonction linéaire rectifiée :

$$f_{rectifiée} = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}.$$

Dans la couche de sortie, la sortie d'un neurone est la somme de ses entrées et est donc donnée par

$$y_k = \sum_{j=1}^{N} \omega_j z_j$$

où N est le nombre de neurones dans la couche cachée, z_j est la sortie du $j^{ème}$ neurone de la couche cachée et ω_j est un poids de connexion à optimiser.

Les quatre sous-sections qui suivent présentent les quatre itérations d'architectures neuronales basées sur le MLP qui cherchent à prédire avec une précision de 5 % les valeurs normalisées des dimensions des composants de trois différents types de circuits microélectroniques analogiques en hautes fréquences, en fonction des critères de performances requis.

2.4.1 Première itération : MLP profond à plusieurs sorties

Tel qu'indiqué au terme du chapitre présentant le cadre théorique, on souhaiterait utiliser un réseau de neurones profond possédant un grand nombre de couches cachées et présentant ainsi un important pouvoir de représentation. Toutefois, tel qu'expliqué alors, le nombre de paramètres à optimiser serait alors potentiellement trop élevé étant donné le faible nombre d'exemplaires de circuits disponibles, soit 200 pour chacun des trois types de circuits et ce, pour réaliser les étapes d'entraînement, de validation et de

test de l'architecture neuronale. Néanmoins, la première itération a utilisé un tel réseau profond sous la forme d'un MLP, afin de tester cette hypothèse. Le réseau de neurones profond utilisé a entre 20 et 50 couches cachées (en fonction de la valeur sélectionnée par l'algorithme génétique). Or, l'entraînement d'un tel réseau de neurones profond présente le risque de souffrir du problème de disparition du gradient de l'erreur. En effet, le long chemin de rétropropagation de l'erreur traverse alors de nombreuses fonctions non linéaires et multiplications qui risquent de faire disparaître les valeurs propagées en cours de route. Afin de pallier ce problème, certaines techniques d'entraînement en apprentissage profond ont été utilisées. D'abord, l'algorithme génétique utilisé pour optimiser les hyperparamètres du réseau de neurones avait la possibilité de sélectionner une fonction de rectification linéaire comme fonction de sortie des neurones des couches cachées. Ce type de fonction, au contraire de certaines fonctions sigmoïdes fréquemment utilisées, ne présentent pas de transformation vers des valeurs extrêmement faibles et réduit ainsi le risque de disparition du gradient. Ensuite, on utilise des connexions dites résiduelles (He et al., 2016), lesquelles évitent de rencontrer des fonctions non linéaires sur leur chemin. Ceci permet encore une fois de réduire le problème de diminution de gradient. Néanmoins, comme les réseaux de neurones profonds entraînés pour une première fois nécessitent généralement une quantité d'exemplaires de l'ordre des centaines de milliers, voire des millions, on ne s'attendant pas ici à obtenir des résultats positifs, mais on cherche plutôt à confirmer que cette méthode n'est pas adaptée à un ensemble d'entraînement comportant moins de 200 exemplaires.

Cette architecture neuronale n'a pas permis d'atteindre le critère de validation utilisé, soit une précision de 5 % des valeurs normalisées de l'ensemble des dimensions de composants. Plus précisément, il n'a pas été possible d'entraîner cette architecture neuronale à prédire correctement quelque composant que ce soit. Ceci supporte l'hypothèse selon laquelle une architecture neuronale profonde ne permet pas le

dimensionnement automatique de circuit pour un nombre d'exemplaires ne dépassant pas 200.

2.4.2 Deuxième itération : MLPs indépendants

Le MLP profond n'ayant pas été en mesure d'atteindre le critère de validation, l'hypothèse selon laquelle le nombre de paramètres à optimiser est trop grand pour un si faible nombre d'exemplaires (i.e. 200) est supportée. Cherchant à pallier ce problème, la deuxième itération de l'architecture neuronale est constituée d'un ensemble de MLPs superficiels (i.e. peu profonds) distincts ne comportant qu'un ou deux couches cachées, où chaque MLP ne cherche qu'à prédire la dimension d'un seul composant du circuit microélectronique. La fonction d'appariement allant des critères de performances du circuit vers la dimension d'un unique composant est beaucoup moins complexe à représenter par un réseau de neurones que l'ensemble des fonctions d'appariements allant vers l'ensemble des dimensions de composants que devait représenter le MLP profond. Ainsi, les 200 exemplaires de circuits microélectroniques sont donc ici utilisés afin d'optimiser un nombre beaucoup plus faible de paramètres et seront réutilisés ainsi pour chaque dimension à prédire. Les hyperparamètres de l'ensemble des MLPs utilisés dans cette architectures neuronale optimisés par l'algorithme génétique sont présentés au chapitre 3 (Tableau 3.1).

Cette architecture neuronale n'a pas permis d'atteindre le critère de validation utilisé, soit une précision de 5 % de la valeur normalisée des dimensions de tous les composants durant la phase test pour l'ensemble des circuits. Les détails des résultats obtenus sont présentés au chapitre 3 (Tableau 3.5).

2.4.3 Troisième itération : LC-SNN

Bien que l'architecture neuronale correspondant à un ensemble de MLPs parallèles n'ait pas été en mesure d'atteindre le critère de validation utilisé ici, ses performances

se sont avérées supérieures à celles du MLP profond. Ainsi, l'hypothèse selon laquelle le nombre de paramètres à optimiser par le MLP profond était trop élevé pour le nombre d'exemplaires de circuits microélectroniques disponibles est encore une fois supportée. De plus, on pose maintenant l'hypothèse selon laquelle l'interdépendance entre certaines dimensions de composants peut présenter un problème majeur ayant empêché l'architecture composée de MLPs superficiels indépendants de prédire adéquatement les dimensions de certains composants. En effet, advenant le cas où il existe un ensemble de composants où la dimension de chacun d'entre eux ne peut être approximée seulement à partir des critères de performances fournis, des MLPs indépendants se trouveront dans une impasse. Par exemple, dans un cas où la variation de la longueur du canal d'un premier transistor peut être compensée par la variation de la longueur du canal d'un deuxième transistor, il est difficile de prédire la taille de l'un pour un exemplaire de circuit donné sans connaître la taille de l'autre. C'est pour faire face à cette éventualité que les MLPs parallèles ont été remplacés ici par des MLPs séquentiels. On nomme ici cette méthode « cascade de réseaux de neurones superficiels » (en anglais, *cascaded shallow neural networks* ; C-SNN). Plus précisément, cette troisième itération de l'architecture neuronale correspond à une variante « limitée » du C-SNN (i.e. LC-SNN), laquelle est décrite dans ce qui suit. Aussi, chaque SNN correspond ici à un MLP superficiel ne possédant qu'une ou deux couches cachées.

Afin de construire le LC-SNN, on génère d'abord un premier MLP qui prend en entrée les critères de performances requis pour un circuit donné, puis apprend à générer la valeur d'un unique composant du circuit microélectronique en sortie. Ensuite, un deuxième MLP est généré, lequel prend en entrée les critères de performances et la sortie du premier MLP. Ce deuxième MLP produit alors en sortie la valeur d'un second composant du circuit microélectronique. Maintenant, les critères de performances requis, ainsi que les sorties du premier et du deuxième MLP forment l'entrée d'un troisième MLP, lequel produit en sortie la valeur d'un troisième composant du circuit.

Le processus se poursuit ainsi avec de nouveaux MLPs jusqu'à ce que toutes les valeurs des composants du circuit mircroélectronique aient été prédites correctement. Le réseau complet constitue ainsi une cascade de réseaux de neurones superficiels, tel qu'illustré à la Figure 2.4. Les hyperparamètres de l'ensemble des MLPs utilisés dans cette architectures neuronale optimisés par l'algorithme génétique sont présentés au chapitre 3 (Tableau 3.1).

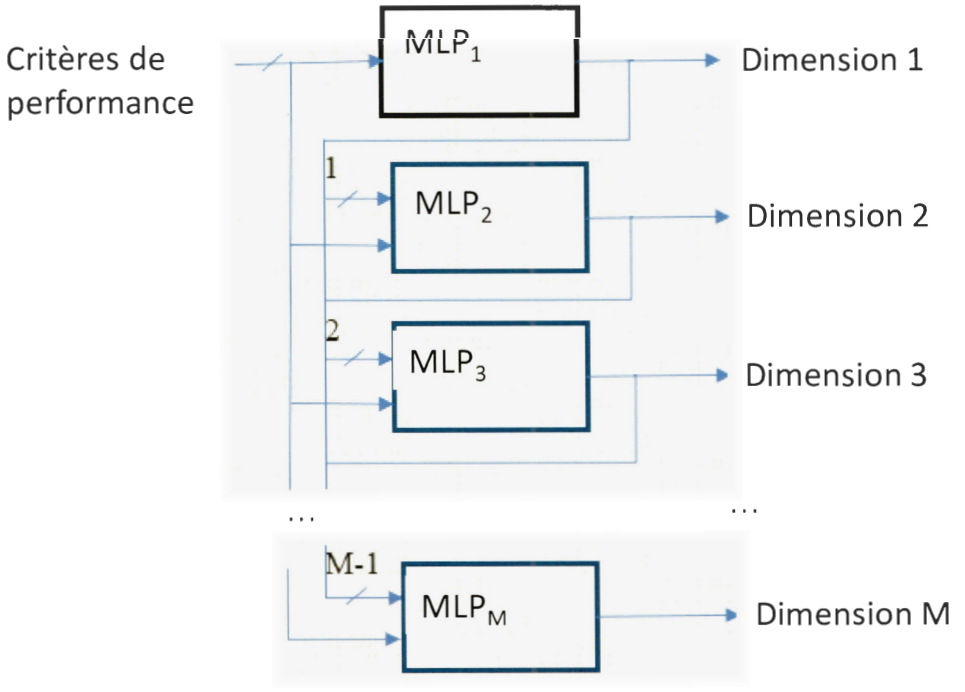

Figure 2.4. Diagramme de blocs du LC-SNN.

Cette architecture neuronale n'a pas permis d'atteindre le critère de validation utilisé, soit une précision de 5 % des valeurs normalisées des dimensions des composants durant la phase test pour l'ensemble des circuits. Néanmoins, le résultats se sont avérés supérieurs à ceux de l'architecture précédente correspondant à un ensemble de MLPs indépendants. Les détails des résultats obtenus sont présentés au chapitre 3 (Tableau 3.5).

2.4.4 Quatrième itération : UC-SNN

Dans la version limitée de la cascade de réseaux de neurones superficiels (LC-SNN), on retrouve dans l'architecture finale un seul SNN par dimension de composant recherchée. Ainsi, un premier algorithme génétique cherche à optimiser un SNN qui sera en mesure de prédire correctement la dimension d'un composant. S'il réussit, ce SNN sera ajouté à la cascade finale et la dimension de composant qu'il prédit ne pourra plus être sélectionnée par les algorithmes génétiques des niveaux subséquents. Ainsi, si les dimensions de dix composants doivent être déterminées, on aura une cascade d'exactement dix SNNs. Le LC-SNN a été proposé de manière à permettre de trouver une solution dans le cas où une dimension de composant ne peut pas être prédite uniquement sur la base des critères de performances. En effet, s'il y a interdépendance entre les dimensions de deux composants, il est nécessaire d'en connaître une pour fixer l'autre. Or, dans les cas où au moment de la conception des circuits servant d'ensemble d'entraînement on fixe systématiquement la valeur d'une dimension avant d'ajuster l'autre, la méthode LC-SNN devrait suffire. Toutefois, si l'ordre d'ajustement des deux valeurs est aléatoire au moment de la conception des circuits, la méthode LC-SNN ne suffira pas. Pour pallier ce problème, la quatrième itération de l'architecture neuronale propose une version illimitée de la cascade de réseaux de neurones superficiels (en anglais, *unlimited cascaded shallow neural networks* ; UC-SNN). Dans la version UC-SNN, plusieurs SNNs peuvent contribuer à prédire la dimension d'un même composant. En effet, un SNN est maintenant ajouté à la séquence globale dès qu'il présente une erreur de prédiction plus faible que le meilleur SNN de la séquence cherchant à prédire la même valeur de composant. Ainsi, alors que le premier SNN cherchant à prédire la dimension d'un premier composant ne peut compter que sur les critères de performances du circuit en entrée, le $n^{ième}$ SNN cherchant à prédire la dimension de ce même premier composant pourra compter en entrée non seulement sur l'ensemble des critères de performances, mais en plus sur l'ensemble des prédictions approximatives obtenues jusqu'ici, y compris celles ciblant ce même composant. L'architecture UC-

SNN permet ainsi de contraindre le C-SNN davantage que le LC-SNN en procédant par approximations itératives. Cette méthode est illustrée à la Figure 2.5. Les hyperparamètres de l'ensemble des MLPs utilisés dans cette architectures neuronale optimisés par l'algorithme génétique sont présentés au chapitre 3 (Tableau 3.1).

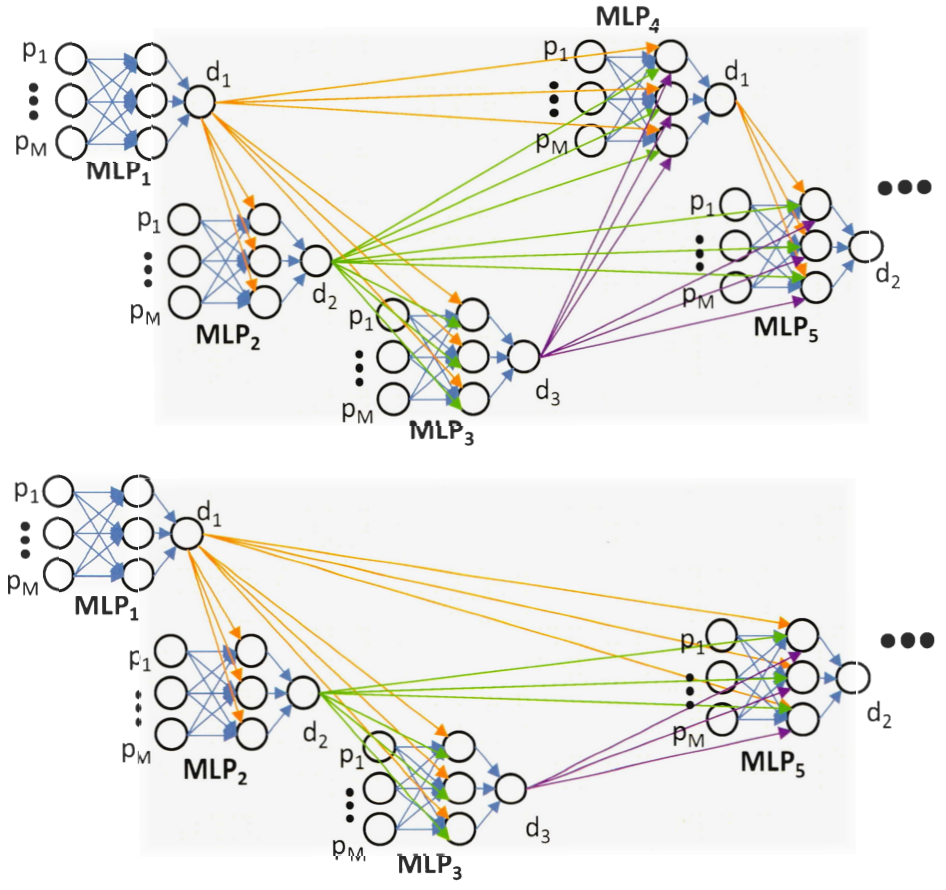

Figure 2.5. Dans une architecture UC-SNN, exemple de l'ajout d'un nouveau MLP à la séquence avec trois composants d'un circuit à prédire : a) Cas où MLP_4 fait une meilleure prédiction pour d_1 (d_1 : composant #1 du design à prédire) que MLP_1, ce qui entraîne l'ajout de MLP_4 à la séquence ; b) cas où MLP_4 fait une moins bonne prédiction que MLP_1, ce qui entraîne le non-ajout de MLP_4 à la séquence.

Cette architecture neuronale a permis d'atteindre le critère de validation utilisé, soit une précision de 5 % des valeurs normalisés des dimensions des composants durant la phase test et ce, pour l'ensemble des circuits. Ceci a donc permis de mettre fin aux itérations sur les architectures neuronales. Les détails des résultats obtenus sont présentés au chapitre 3 (Tableau 3.5).

CHAPITRE III

AUTOMATIC COMPONENTS SIZING OF ANALOG RF CIRCUITS BY CASCADED SHALLOW NEURAL NETWORKS

3.1 L'article, dans le contexte du présent mémoire

Ce chapitre consiste en un article (Dumesnil et Boukadoum, 2019) présentant une méthodologie générale de dimensionnement automatique des composants d'un circuit microélectronique analogique en haute fréquence. Cette méthodologie englobe les troisième et quatrième itérations de l'architecture neuronale présentée au chapitre précédent, soit les versions limitée et illimitée de la cascade de réseaux de neurones superficiels. Aussi, on compare les résultats obtenus pour les trois itérations ayant permis de prédire adéquatement au minimum une dimension de composant, soit les itérations deux, trois et quatre (i.e. l'ensemble de MLPs indépendants et les deux versions de la cascade de MLPs superficiels, LC-SNN et UC-SNN).

3.2 Abstract

A neural network-based method for sizing of analog circuit components is proposed, with a focus on radio frequency (RF) applications. Given a circuit topology and desired performance characteristics, it automatically determines the component values needed to achieve those characteristics. The method relies on a cascade of shallow neural networks (C-SNN), each one individually configured by a genetic algorithm to predict a single component size. Each SNN of the cascade thus constrains the subsequent SNNs

through its predicted output. Two variants are presented: a limited cascade (LC-SNN) and an unlimited cascade (UC-SNN). The LC-SNN adds a SNN to the cascade only if it successfully predicts one component size : the final cascade only comprises one SNN for each component size to predict. The UC-SNN adds one SNN each time it improves the best prediction yet for a given component size : there is no predefined limit to the number of SNNs appearing in the final cascade. The proposed cascade approach restricts learning to a few parameters at a time and thus offers a novel valuable solution to the typically small training sets available in the circuit design field, in addition to considerably speeding the circuit design. The method was successfully validated with three different types of RF microcircuits and in two different technologies: a low-noise amplifier (LNA), a voltage-controlled oscillator (VCO) and a mixer, using CMOS 180 nm and CMOS 130 nm.

3.3 Index Terms

Shallow neural networks, sizing, synthesis, artificial neural network, deep neural network, genetic algorithm, multilayer perceptron, microelectronic, analog circuit, radio frequency, low noise amplifier, voltage-controlled oscillator, mixer.

3.4 Introduction

Currently, the typical design flow of analog circuits requires substantial human intervention (Scheible et Lienig, 2015), since the designer experience is necessary to compensate for nonlinear effects and implementation factors not accounted for by the design tools, because of their complexity and/or the absence of formal models to rely on. That typically results in a time-consuming iterative process that must often be repeated for each new design, and the situation is likely to worsen with the increasing complexity of electronic circuits and systems, along with their shortening useful life spans, which in turn call for shorter design times as well, hence making electronic

design automation (EDA) tools essential to improve designer productivity. Indeed, EDA has been a research subject for many years and many works have proposed circuit design workflows based on EDA tools, particularly for digital integrated circuits and systems. However, the design and synthesis/sizing of analog circuits, particularly for RF applications, are still challenging due to nonlinear components, bias requirements, and real-world and implementation effects such as stray impedances, physical circuit layout and component coupling.

Given the topology of a radio-frequency (RF) analog circuit, the present paper addresses the problem of finding a suitable set of component values that match specified performance characteristics, referring to a desired circuit behavior in the time and/or frequency domains (i.e. the sizing problem). When designing an analog circuit, designers commonly use analysis tools for help: a software that takes the circuit topology and component values as inputs and outputs the resulting circuit behavior. However, the real design problem addresses the reversed causality of that process: find a feasible circuit topology and component values that match the given performances. Formally, two problems are reversible with respect to each other if the formulation of one involves all or part of the solution of the other (Keller, 1976). The problem that is better understood is dubbed the direct problem, while the other one is dubbed the inverse problem. In the context of analog circuits, finding the behavior of a circuit given its topology and component values is a direct problem of *analysis*, while finding the topology and component values given the behavior of a circuit is an inverse problem of *synthesis*. In this work, the inverse problem is simplified by considering the topology to be given, hence limiting the design problem to component *sizing, i.e.,* finding the component values given the topology and behavior of the circuit.

Solving the sizing problem is particularly challenging for RF circuits due to the difficulty to model those circuits by regular linear equations. Therefore, the typical approach consists in a time-consuming iterative process of circuit simulations followed

by sizing adjustments. For instance, the current design of radio frequency low noise amplifiers (RF-LNA) routinely requires hours if not days of tuning by an expert. One explanation for this is that most circuit equations become non-linear at GHz frequencies and lumped-element models are no longer representative of circuit behavior. In effect, at such frequencies, each component has associated electric and magnetic fields, along with a dissipative loss (Bahl, 2003), which lead to very complex circuit equations. Finally, secondary effects such as those of stray capacitances and layout effects arise. Hence, an efficient automatic sizing solver would be of great assistance for RF circuit design.

Automating the solution of inverse problems has been the focus of multidisciplinary research for almost a century, as those problems are pervasive in many fields of science and technology (e.g. Anantrasirichai, 2017, Marques, 2003, Starck, 2016). Indeed, whenever a physical system needs to be inferred through measurements, an inverse problem arises (Tarantola, 2005) and its well-posedness must be established for a straightforward solution to exist. Formally, a problem is well-posed if it meets the three so-called Hadamard criteria (Hadamard, 1923): (1) it can be solved; (2) the solution is unique; (3) the solution is continuous with respect to changing the data and parameters. A problem that does not respect those three criteria is ill-posed. Unfortunately, that is often the case for inverse problems as they are typically undetermined, thus violating the second Hadamard criterion with the multiple solutions to choose from, and they may not meet the third Hadamard criterion either, as making small data or parameter changes in them lead to large variations in solution accuracy. Focusing on the inverse problem that constitutes automatic sizing of RF microelectronic circuits, the works reported in the literature are relatively sparse and not generic in scope. Two main approaches have been followed, knowledge-based and metaheuristic-based, with artificial neural networks being a relatively recent third alternative.

Knowledge-based methods are among the earliest ones attempted, with the aim to reproduce the experience of expert designers (Shi, 2018). The approach may be effective for relatively small designs, but larger circuits make it difficult to build an efficient design plan (Sasikumar et Muthaiah, 2016). Case-based reasoning (Aamodt et Plaza, 1994) attempts to circumvent the problem by shifting the focus on the result of expert thinking rather than the thinking itself. The case-based reasoning process can be summarized as the search and retrieval of a "close" solved case and its revision for adaptation to the current case, with storage of the modified solution to enrich the case base. However, this requires a large number of stored designs to be efficient (Smiti et Elouedi, 2011), which makes its usefulness limited for circuit design, synthesis or sizing.

Metaheuristic-based methods view the addressed problem as one of multi-objective optimization and solve it by exploring the design space with a metaheuristic (Fakhfakh et al., 2015, Sasikumar et Muthaiah, 2016, Tripathi et al., 2013), as the problem to optimize is typically NP-hard. Several approaches have been tried, mainly using evolutionary optimization methods. For instance, simulated annealing explores the solution space with increasingly finer granularity until the final solution is reached (e.g. see Yang et al., 2018), the genetic algorithm uses a chromosome metaphor (e.g. see Kalentyev et al., 2014), genetic programming attempts to solve both the synthesis and sizing problems using a dynamically bred program (Koza et al., 2000). In all of those approaches, a design in the loop approach is used, in which the circuit of interest is built or simulated for testing during the algorithm iteration process. That can result in very long convergence times (Liu et al., 2014), unless approximation techniques such as less complex surrogate circuits are used for simulation and final tuning is done par an expert (Liu et al., 2014).

Neural networks view the synthesis/sizing process as a classification problem: given a set of successful example circuit designs, they use their generalization ability to specify

the component values of similar new circuits, even in the present of fragmentary information. Those properties make them more generic than the previous methods which do not learn "how to," but rather find idiosyncratic solutions; as a result, they must be started over for each new problem, even if the same circuit topology is considered. ANNs have been used with relative success in the modeling and design of microwave passive circuit (Kabir et al., 2010), testing RF circuit designs (Hasani et al., 2016) and designing simple analog integrated circuits (Jafari et al., 2010). However, their requirement of a large set of circuit exemplars to train efficiently and their black-box operation make the improvement of their prediction performance difficult to achieve.

In the present paper, a new ANN-based methodology is proposed to address the circuit sizing problem while reducing the need for a large dataset of circuit examples. It consists in the generation of a cascade of shallow neural networks (C-SNN), each one individually specified by a genetic algorithm to predict one circuit component size value. A shallow neural network is an ANN constituted of a small number of hidden neural layers. Here, each SNN corresponds to a multilayer perceptron (MLP) and the genetic algorithm is used to optimize its hyperparameters. Each trained MLP adds its output to the specified performances in order to constrain the learning of the future MLPs. Then, Additional MLPs are added as needed to compensate for potential component coupling. In this work, analog RF microcircuits were chosen for validation, but the proposed methodology is applicable to any analog circuit; the RF microcircuits were chosen mainly because of the difficulty to design them by conventional means, hence emphasizing the value of the proposed technique as an automatic sizing tool. It should be noted that the predicted values by the ANN are those that would be obtained by iterative simulation with a conventional tool such as Rf-spectre from Cadence. As such, they do not account for real-world implementation effects and should only be viewed as an efficient way to initialize a subsequent optimization tool that would account for those effects (Liao et al., 2017, Lourenco et al., 2016, Passos et al., 2017).

This paper is organized as follows. Section 3.5 reviews related work on existing sizing techniques; Section 3.6 lays the background for the new approach proposed here; Section 3.7 presents the C-SNN architecture generation process, along with the GA used to optimize its hyper-parameters; Section 3.8 describes the microelectronic circuits designed to validate the proposed method and provides the obtained results; finally, Section 3.9 offers concluding remarks.

3.5 Related work

Below is a brief survey of previous work devoted to the analog circuit sizing problem, with a justification of this work at the end.

3.5.1 Knowledge-Based

Knowledge-based methods are among the earliest ones attempted. The sizing process is derived from the experience of expert designers, who usually develop a plan to set the values of the design parameters (Sheu, 1990). The plan consists of a series of predefined steps that include equations relating the performance criteria to the final design parameters. The approach was effective for relatively small designs such as operational amplifiers (Harjani, 1989). However, larger circuits rapidly increase in complexity, with nonlinear relations to account for between the circuit components and the corresponding circuit behavior, making it difficult to build an efficient design plan (Sasikumar et Muthaiah, 2016, Shi, 2018). Moreover, as knowledge-based methods are essentially expert systems, they suffer from the fundamental difficulty of those systems to extract the human expert knowledge, i.e., render explicit the implicit procedural knowledge of the expert (Kidd, 1987). This knowledge acquisition problem has led many researchers to look for other methods capable of finding a solution autonomously, instead of relying on the ability of an expert to transfer her knowledge.

3.5.2 Case-Based Reasoning

Case-based reasoning (Aamodt et Plaza, 1994) attempts to circumvent the problem of expert knowledge acquisition by shifting the focus on the result of expert thinking rather than the thinking itself. It is a four-step process that can be summarized as retrieve, reuse, revise and retain (Kolodner, 1992). First, a search is made through a solution space, *i.e.*, through a library of previously used and memorized cases. This idea of storing and reusing previously encountered problems and solutions was presented by Schank through the concept of dynamic memory (Schank, 1982). In the context of circuit sizing, the search explores the space of previously memorized sets of performance criteria. One is selected and along comes the corresponding set of component parameter values. If the retrieved circuit behavior is the same as the required one, the values of its components are simply reused. If not, the component parameters need to be revised, in which case knowledge-based methods are usually used (Al-Kashef, 2008). Thus, case-based reasoning allows to reduce the knowledge acquisition problem but does not solve it completely. Therefore, it suffers from limits similar to those of knowledge-based methods, especially when it comes to revising the solutions of complex circuits. Moreover, case-based reasoning requires an increased need in storage memory, given the library of memorized designs (Smiti et Elouedi, 2011). Also, as will be discussed shortly, it is very difficult to gather an important quantity of microelectronic designs for a given topology in a given technology. Hence, in the context of circuit sizing, the search space containing the memorized sets of performance criteria remains sparse and its usefulness is thus limited. Finally, when the solution found through case-based reasoning is not directly generalizable to other sets of desired performance criteria, new cases are added to the library and increase the prior knowledge to be used for future sizing problems.

3.5.3 Simulated Annealing

Simulated annealing methods view the sizing problem as one of optimization and solve it by exploring the design space (Aarts et al., 2005). At first, the algorithm explores it from a high perspective, which reduces its chances of getting stuck in local minima. Then, the optimization process progresses with increasingly finer granularity until the final solution is reached. At each step of the process, the current solution is compared to a neighboring solution. If the neighbor performs better, it replaces the previous best approximation, but if does worse, it only has some probability of replacing it, which decays with time.

The approach has two important limits. First, in the context of circuit sizing, for each comparison of neighboring solutions, the corresponding circuits must be simulated (via a circuit analysis software) to find the error between the behavior of each one and the desired one. Hence, the algorithm can be time consuming for complex problems. Second, and more importantly, the final sizing solution is only applicable to the currently desired circuit behavior. It is not generalizable to other circuit behaviors that could be obtained with the same topology. Indeed, the method does not learn "how to" size, but only finds an idiosyncratic solution for a single set of required circuit behavior. Thus, the optimization algorithm must be restarted anew for each new desired behavior. This can be a serious limitation in terms of sizing time. For example, the simulated annealing method proposed in (Weber et al., 2013) needed between one and three hours on a 2.3 GHz CPU to complete the sizing of each low-noise amplifier (LNA) performance criteria it was given. Another simulated annealing approach proposed in (Yang et al., 2018) required approximately 25 seconds to size a new operational amplifier and one hour to size a new voltage-controlled oscillator (VCO) on a 2.4 GHz CPU.

3.5.4 Genetic Algorithm

Genetic algorithm search is similar to simulated annealing, in that potential solutions are competing in the quest to find a solution. However, the algorithm is different. Instead of two neighbors competing on each iteration, there is a whole population of potential solutions doing so. Each potential solution is coded within a "chromosome", by analogy with biology. In the context of circuit sizing, that chromosome would typically define the component values of the microelectronic circuit. The chromosome population then goes through a series of selection-reproduction-mutation cycles until one of the chromosomes converges towards an acceptable set of component parameters (Whitley et Sutton, 2012). During the selection portion of a cycle, all the chromosomes need to be compared, which means that all the corresponding circuit designs need to be simulated with a circuit analysis tool to determine which chromosomes lead to the smallest errors. Hence, similar to simulated annealing, genetic algorithm techniques are also time consuming, and the solution found by genetic algorithm also does not generalize to other potential behaviors. Therefore, the returned set of component values only applies to a single set of performance criteria. For example, a genetic algorithm implemented in (Barros et al., 2010) took approximately 54 seconds on a 2.8 GHz CPU to complete the sizing of an operational amplifier, while another genetic algorithm approach implemented in (Kalentyev et al., 2014) took consistently more than 30 minutes to size a LNA. In both cases, the process had to be repeated for any new set of performance criteria presented.

3.5.5 Genetic Programming

Genetic programming is now one of the leading trends, not only in automated circuit sizing, but also in automated circuit synthesis. Indeed, it has been shown to perform well not only to find an adequate set of values for components, but also to find an adequate topology, including selecting the actual components and interconnections of the circuit. As proposed in (Koza, 1997), a tree is used as a representation of the analog

microelectronic circuit. First, an embryonic circuit is generated, which is the basis upon which the final circuit will be evolved. The evolution of the circuit reflects the evolution of the functions that constitute the branches of the circuit-constructing program tree (see also (Lohn et Colombano, 1998) for a different coding scheme). The main advantage of genetic programming over genetic algorithm and simulated annealing is its efficiency at synthesizing the whole circuit (i.e. finding an adequate topology and a set of component values), instead of being limited to circuit sizing.

More recently, many variations of genetic programming algorithms have been proposed for circuit sizing (e.g. (Liao et al., 2017)), reaching very efficient sizing optimizations. However, in the context of the present paper, they all present the same limit: the solution found is always limited to the single set of performance criteria used for the optimization process. Moreover, this approach is very time consuming as each potential solution must be tested through an analysis simulation.

As will be discussed in the Conclusion section, the method proposed in the present work, which allows for a general mapping from performance criteria to circuit sizing, could be applied to complement such very precise but idiosyncratic genetic programming algorithms to constrain their initial conditions and therefore increase their time efficiency.

3.5.6 Deep neural network

One last sizing method will now be presented which presents the potential to (1) solve the sizing problem autonomously and (2) optimize a general mapping from performance criteria to circuit sizing instead of an idiosyncratic solution. This method corresponds to ANNs, or more precisely the deep neural network (DNN).

DNNs (Heaton et al., 2017) constitute an automated optimization method for solving inverse problems which has attracted much interest in recent years. The typical DNN

is an artificial neural network that has many intermediate layers (in principle more than two, by typically much more than that) between its input and output. Each layer is composed of artificial neurons that individually apply a linear or nonlinear function to their inputs. In a feedforward DNN, the neural outputs of each layer only connect to the neural inputs of the next one. Learning occurs via a process of minimizing the error at the output of the final layer. In the most popular learning algorithm, that error is backpropagated through the network using the chain rule for derivatives and the connection weights between neurons are adjusted as a function of their contribution to the error. In the context of inverse problems, DNNs have been particularly successful in medical imaging (e.g. Commandeur et al., 2018, Liu et al., 2018, Kasabov et al., 2017, Hosseini et al., 2017), particularly the convolutional neural network (CNN) variety (e.g. Fukushima et Miyake, 1982, LeCun et al., 1989, Liu et al., 2017), which is inspired from the organization of the visual cortex in mammals (Hubel et Wiesel, 1959). Another field in which DNNs are efficient concerns contextual problems, for example language translation (Sutskever et al., 2014). Unfolded recurrent neural networks (RNN) are the method of choice to solve those problems, with architectures based on long short-term memory (LSTM) (Hochreiter et Schmidhuber, 1997) and gated recurrent units (GRU) (Cho et al., 2014) used in the front-runner models. In a RNN, the output of a neuron is a function of its current input and its past states, which allows the network to learn sequential information.

In the context of circuit sizing, DNNs present the potential for overcoming the main limits of the methods described in the previous section. First, they do not have the knowledge acquisition problem of knowledge-based approaches, as they learn autonomously from the example data. In addition, they do not suffer from the previously described limits of simulated annealing, genetic algorithm and genetic programming, since DNN training data come from already sized circuits and the optimized solution corresponds to an actual mapping from performance criteria to circuit sizing, instead of an idiosyncratic solution. Hence, once trained, new solutions

given by a DNN to new sets of performance criteria do not require analyzing a series of potential sizing solutions through time-consuming simulations. Thus, in the long run, the time invested in sizing a set of circuits for training a DNN is largely repaid for by reuse.

The question then is how many already sized circuits are required for the DNN to find a solution? Learning, in a DNN, corresponds to optimizing the values of the connection weights modulating the signals that propagate through the network so that the outputs of the DNN converge toward the desired values. Those connection weights are called the parameters of the DNN. The number of parameters a DNN needs to optimize thus increases rapidly with the number of neurons forming the network. Now, a DNN corresponds to an ANN possessing a large amount of neuron layers. Hence, for the learning process to be efficient, a huge number of parameters usually need to be optimized. For this to be possible, a huge amount of data must be presented to the DNN. Unfortunately, while it is relatively easy to gather big datasets of images or comments, for example from the internet, it is considerably more challenging to do so for microelectronic circuits. There are many reasons for that: first, it takes substantially longer to synthesize a circuit than to take a picture or write a comment. Second, the community of circuit designers is much smaller than the number of people posting pictures or comments on the internet. Third, while pictures or comments from a few years ago may be used in the same datasets as pictures or comments posted today, the technology in which analog circuits are designed evolves very quickly and it would not be possible to include a circuit designed in a 180 nm CMOS technology in the same dataset as one designed in a 130 nm CMOS technology, as their behavior will be different. Therefore, the rapidly changing technology makes it difficult to gather a dataset large enough for learning to occur within a DNN.

This lack of large available datasets of sized microelectronic analogical circuits is the main motivation for the work presented here. Indeed, the method proposed in section

3.6 reduces the amount of training data required to train a neural network for circuit sizing.

3.6 Methodology

This section describes the circuit sizing method that is proposed in the present paper. First, the two variants of the cascaded shallow neural networks (C-SNN) approach are presented, that is the limited (LC-SNN) and unlimited (UC-SNN) versions. Then, the genetic algorithm to optimize the hyper-parameters of each generated SNN is detailed. Finally, the validation method is described.

3.6.1 Limited cascade of shallow neural networks (LC-SNN)

As argued above, DNNs are an attractive method for circuit sizing. However, the relatively small size of the training set that is typical of such problems limits the number of parameters that can be efficiently optimized by a conventional feedforward DNN, and the problem worsens as the number of hidden layers increases, as it adds more parameters to set. One approach to circumvent that is to learn the parameters in sequence (Dumesnil et Boukadoum, 2015). The process operates as follows : instead of a static DNN architecture that attempts to predict the sizes of all the components of the circuit at once, the architecture is generated in steps, one section at a time. Each section is made of a shallow multilayer perceptron (MLP), specified by a genetic algorithm that optimizes its hyperparameters, such as the number of layers, the number of neurons per layer, the output function of the neurons and the training algorithm.

The MLPs constituting the LC-SNN are said to be shallow, because they are made of an input layer, an output layer, and only one or two hidden layers. The outputs of the input layer neurons simply hold the received values, while those of the hidden layer neurons are given by

$$z_j = f_\theta\left(\sum_{i=1}^{M} \omega_i x_i\right)$$

where M is the number of neurons in the previous layer, x_i is the output of the i^{th} hidden layer neuron, ω_i is a weight to be determined and $f_\theta(\,)$ is the output function. Here, this function was either the hyperbolic tangent sigmoid

$$f_{tansig} = \frac{2}{1 + e^{-2x}} - 1$$

or the rectified linear unit (RLU)

$$f_{rectify} = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}.$$

Within the same MLP, all the hidden layer neurons use the same transfer function and have the same number of neurons. In the output layer, the neural output is a weighted summation of its inputs and is thus given by

$$y_k = \sum_{j=1}^{N} \omega_j z_j$$

where N is the number of neurons in the hidden layer, z_j is the output of the j^{th} hidden layer neuron and ω_j is another weight to be determined. All of the MLP hyper-parameters are determined by a genetic algorithm, which will be detailed in a subsection further below.

The LC-SNN method starts by generating a first MLP that takes as input the required performance criteria and outputs the value of a single component size of the target circuit. Once this MLP reaches a prediction error inferior to 0.05 (i.e., 5% component tolerance), it defines the first member of the LC-SNN sequence. Then, both the required performance criteria and the output of that first MLP form the input of a second MLP that forms the second section of the LC-SNN, and whose purpose is to output the value

of a second component size of the circuit. Then, altogether, the required performance criteria, the output of the first MLP and the output of the second MLP form the input of a third MLP. This process continues with new MLPs created until the values of all the component sizes of the circuit have been correctly predicted, making for the complete LC-SNN. The final LC-SNN architecture is presented in Figure 3.1.

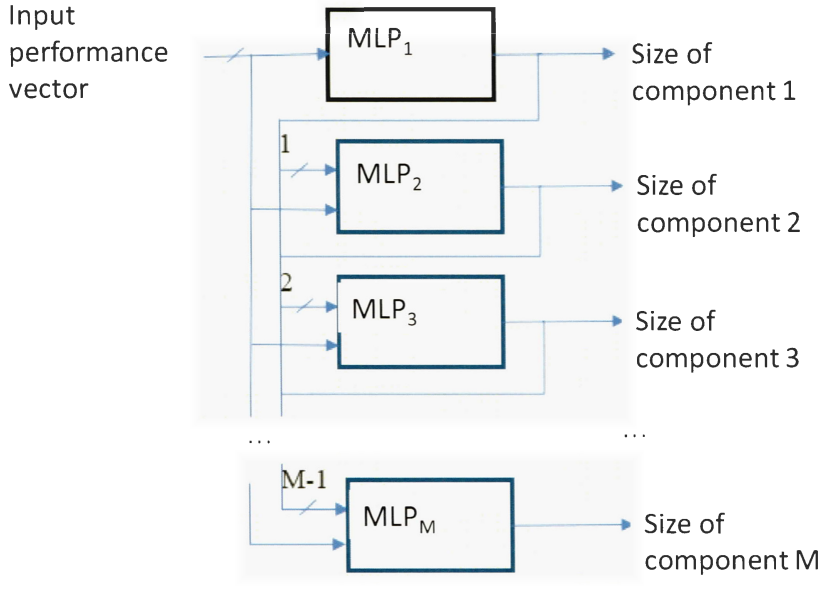

Figure 3.1. Block diagram of the C-SNN (Dumesnil et Boukadoum, 2015).

This variant of the proposed C-SNN method is called *limited* because each component size is predicted by a single MLP and thus, the final LC-SNN comprises exactly as many MLPs as there are component sizes to predict.

3.6.2 Unlimited cascade of shallow neural networks (UC-SNN)

In the limited version of the cascade of superficial neural networks (LC-SNN), we find in the final architecture a single SNN per component size sought. Indeed, a first genetic algorithm seeks to optimize an SNN that will be able to correctly predict the size of a

component. If successful, this SNN will be added to the final cascade and the component size it predicts will no longer be selectable by the genetic algorithms of the subsequent levels. Thus, if the dimensions of ten components have to be determined, we will have a cascade of exactly ten SNNs. However, given this architecture, if there is interdependence between the dimensions of two components, it is necessary that at least one of these components can be predicted without knowledge of the other one's size value. Thus, if the order of adjustment of the two values is random at the time of circuit design, the LC-SNN method will get into to an impasse. To overcome this problem, a new variation of the cascade architecture is proposed here, called unlimited cascade of shallow neural networks (UC-SNN). In the UC-SNN version, several SNNs can help predict the size of the same component. Indeed, an SNN is now added to the global sequence as soon as it has a lower prediction error than the best SNN that is already in the sequence and that tries to predict the same component size value. Thus, while the first SNN seeking to predict the size of a first component can only rely on the performance criteria of the input circuit, the n^{th} SNN seeking to predict the size of the same component can count as input not only on all the performance criteria, but in addition on all the approximate size predictions obtained so far.

The UC-SNN method starts by generating a sequence of N MLPs, in a manner similar to that of the LC-SNN, but with the exception that the best MLP for each component size is retained within the sequence regardless of if it reaches an error inferior to 5 %. Within this sequence, MLP_i is trying to predict the correct design size of the i^{th} component (d_i) and is getting an error (ε_i), which corresponds to the difference between the predicted and the desired component sizes. Eventually, a new MLP, MLP_{i+N}, tries to learn d_i again, using all the performance criteria and predicted values as inputs. If MLP_{i+N} predicts d_1 better than MLP_i, it is added to the MLP cascade ; otherwise, it is ignored. This is illustrated in Figure 3.2 for $N=3$. The whole process is repeated over and over until all the prediction errors are smaller than a set threshold (5 % component tolerance here). The UC-SNN thus makes it possible to constrain the final solution

more than the LC-SNN, proceeding by iterative approximations. The UC-SNN method is illustrated in Figure 3.2.

3.6.3 Genetic Algorithm for MLP Configuration

A separate genetic algorithm is used to optimize the hyperparameters of each MLP of the C-SNN. Each genetic algorithm starts by generating a population of chromosomes. Each chromosome encodes a set of hyperparameters through a binary sequence. The genetic algorithm starts with a population of 64 chromosomes that are randomly initialized. Then, a three-step process consisting of a selection phase, a reproduction phase and a mutation phase is repeatedly applied until reaching a stopping criterion.

3.6.3.1 Selection Phase

This phase selects the chromosome pairs for reproduction. The deterministic tournament method is used. Using tournament has been shown to be more efficient and less time-consuming than other selection methods such as ranking and roulette (Shukla, 2015), while deterministic selection has the potential to outperform probabilistic selection (Mashohor, 2005). Moreover, deterministic tournament lends itself more easily to hardware implementation in a field programmable gate array (FPGA) which is a future objective of this work.

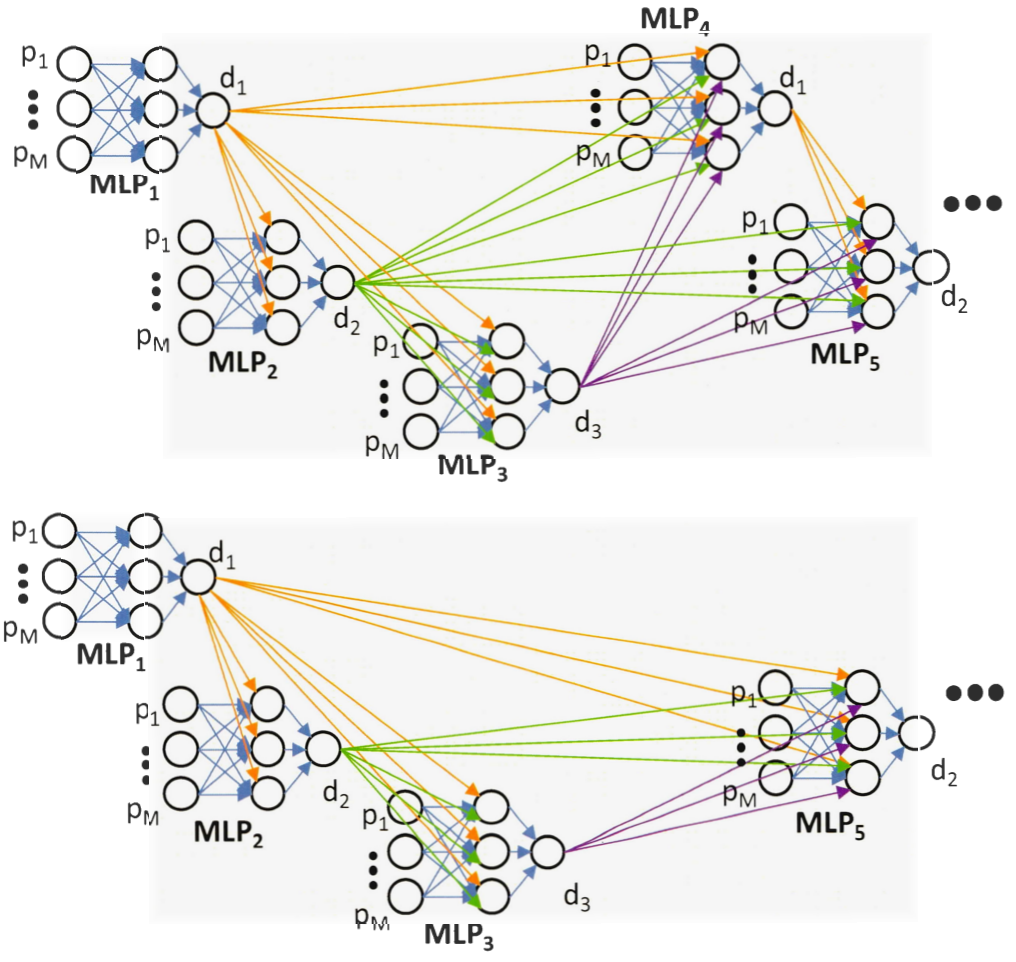

Figure 3.2. Example of MLP selection for the final cascade of a UC-SNN architecture with three component sizes to predict : a) Case where MLP_4 makes a better prediction of d_1 than MLP_1, hence leading to the addition of MLP_4 to the MLP sequence. b) Case where MLP_4's prediction is worse, leading to not adding MLP_4 to the ANN sequence.

Following the deterministic tournament bracket presented in Figure 3.3, the future parent chromosomes are selected through a series of pairwise matchups, using the fitness function of a chromosome defined as the mean-squared error between the target

and predicted outputs of the associated MLP. For each matchup, the chromosome possessing the best fitness function proceeds to the next stage of the tournament. In the first stage, pairwise fitness comparisons are made between chromosomes #1 and #64, chromosomes #2 and #63, etc., all the way to chromosomes #32 and #33. The 32 winning chromosomes progress to a second stage of matchups, while the other 32 are eliminated. In each of the following stages of the tournament, new pairwise matchups take place between the winners of previous stages until one single best chromosome remains. Through this tournament process, the chromosomes losing during stage 2 will be seeded #17 to #32 for the next generation, those losing during stage 3 will be seeded #9 to #16, those losing during stage 4 will be seeded #5 to #8, those losing during stage 5 will be seeded #3 and #4 and finally the tournament winner will be seeded #1 and the runner-up #2.

3.6.3.2 Reproduction Phase

At the end of the selection phase, 32 of the 64 chromosomes have been deleted, while the remaining 32 have been re-ranked. Hence, for the population to be full again, another 32 chromosomes are generated by a reproduction process, where each parent pair generates two children. The parents corresponding to chromosomes #1 and #2 generate children that will be seeded #33 and #34 for the tournament in the following generation (see Figure 3.3), parents #3 and #4 generate children #35 and #36, etc., all the way to parents #31 and #32 which generate children #63 and #64. Each child is created by using two randomly selected crossover points, as is commonly used in the literature (Eshelman, 1999). The chromosome corresponding to the first child of a pair of parents is identical to the chromosome of his first parent for the parts before the first crossover point and after the second crossover point, and to the chromosome of his second parent in between. The inverse is true for the second child of the same pair of parents. That crossover process is illustrated in Figure 3.4.

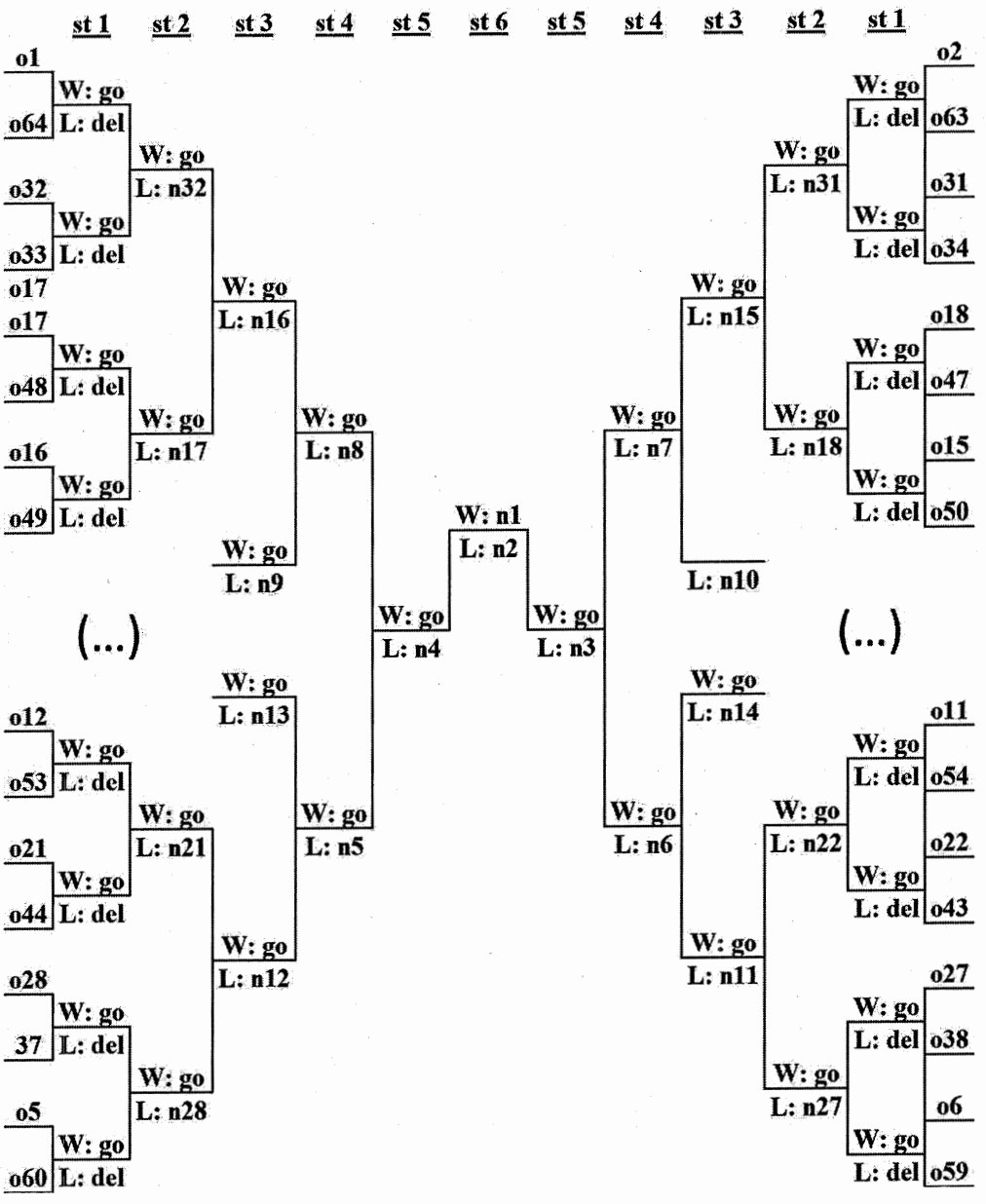

Figure 3.3. Deterministic tournament bracket used by a genetic algorithm, where *st 1* means "stage 1", *o1* means "old chromosome #1", *n1* means "new chromosome #1 for next generation", *W* means "winning chromosome", *L* means "losing chromosome", *del* means "deleted chromosome" and *go* means that the chromosome moves to the next stage.

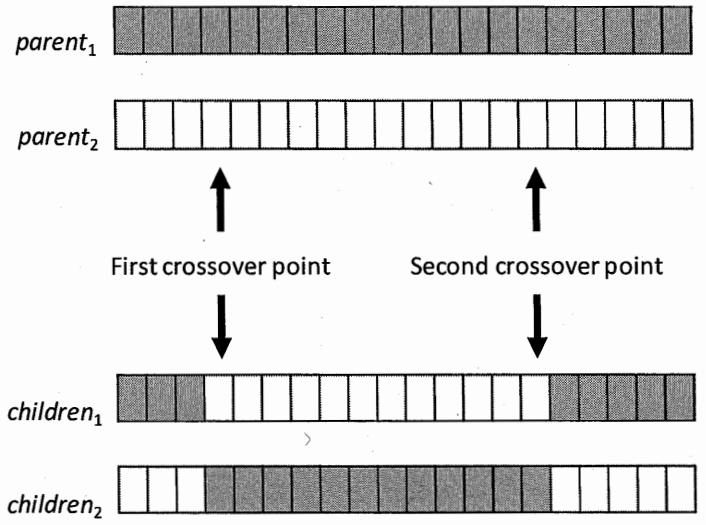

Figure 3.4. Two-point crossover reproduction method.

3.6.3.3 Mutation Phase

The mutation phase inverts each chromosome bit of the population with probability:

$$\frac{1}{bit\ quantity\ within\ one\ chromosome}.$$

This probability was suggested in (Deb et Kalyanmoy, 2001), and it was tested here along with various other values in pilot experiments; none consistently provided better results.

3.6.3.4 Application to the UC-SNN

Algorithm 3.1 presents the full pseudo-algorithm of the UC-SNN generation process.

Algorithm 1. Genetic algorithm applied to the UC-SNN.

```
1   Import dataset;
2   Normalize dataset (min-max normalization);
3   Shuffle dataset order;
4   Separate dataset into a training set, a validation set and a test set;
5   Generate N populations of M chromosomes
    (where N = number of design components to predict);
6   while max(best valid_error for each component to predict)
    > 0.05 do
7       for i = 0, i < M do
8           for j = 0, j < N do
9               Fix the hyperparameters of new_MLP_ji for chromosome
                m of population n;
10              while valid_error improves do
11                  Load training and validation set using MLP sequence
12                  Shuffle training set order.
13                  Pretrain new_MLP_ji;
14                  Train new_MLP_ji;
15                  Validate new_MLP_ji;
16              if valid_ error of new_MLP_ji <
                valid_ error of old_MLP_ji do
17                  Append MLP_ji to MLP sequence
18      Selection phase on each population;
19      Reproduction phase on each population;
20      Mutation phase on each population;
21  Test MLP sequence
```

3.6.4 Validation

The performance and generalization capacity of the UC-SNN architecture was tested with three different RF analog microelectronic circuits: a low-noise amplifier (LNA), a voltage-controlled oscillator (VCO) and a mixer. Moreover, in order to get some perspective on the obtained results, they were compared to two alternative neural network architectures: the LC-SNN architecture and a set of independent MLPs (I-MLP) each predicting one component size. This makes it possible to evaluate the impact of increasing the communication between MLPs on the sizing performance from a first architecture with no communication (I-MLP), to a second architecture with only successful MLPs communicating (LC-SNN), to a third architecture with all MLPs communicating (UC-SNN). The hyperparameters of all neural architectures were optimized with a genetic algorithm. Table 3.1 presents the associations *bits-hyperparameters* and their range of values. Meanwhile, all the neural networks parameters were optimized through the Adam algorithm (Kingma er Ba, 2014), which

Table 3.1. Description of chromosome bits for the genetic algorithm.

Bit #	Role in MLP	Plage des valeurs
1 – 3	Nb. of hidden layer neurons	10 to 45 (steps of 5)
4	Nb. of hidden layers	1 to 2
5	Hidden layer Output function	tanh or RLU
6 – 8	Size of minibatch	1 to 8
9	L1 regularization	yes/no
10	L2 regularization	yes/no
11 – 13	$\beta 1$ of Adam	0.9 to 0.99999
14 – 16	$\beta 2$ of Adam	0.9 to 0.99999
17 – 19	ε of Adam	10^{-10} to 10^{-6}

In the next subsections, the microelectronic circuits used for validation are presented.

3.6.4.1 Low-Noise Amplifier

The circuit topology of the LNA is the same as in (Boukadoum et al., 2012) and is presented in Figure 3.5. It consists of a cascode common-source stage with source degeneration and inductive load. The cascode configuration was selected for its design simplicity (easier matching, stability, etc.) and widespread use. All its inductances, including those in the matching networks, were assumed to be non-ideal with Q-factors of 10, which is consistent with current CMOS fabrication processes. To simplify the design effort further and reduce the number of parameters, only the L-shaped matching network shown in Figure 3.5b is considered for 50 Ω source and load impedance matching. 200 LNA designs were used as training exemplars during the DNN generation. The list of considered LNA design (D) and performance (P) variables is presented in Table 3.2, which also provides the ranges of values that were used during the validation experiments. Notice that the drain inductance (L_D) is not mentioned, as it was fixed at 0.578 nH for all designs. The LNAs were designed with Cadence Virtuoso Spectre RF using CMOS 180 nm technology.

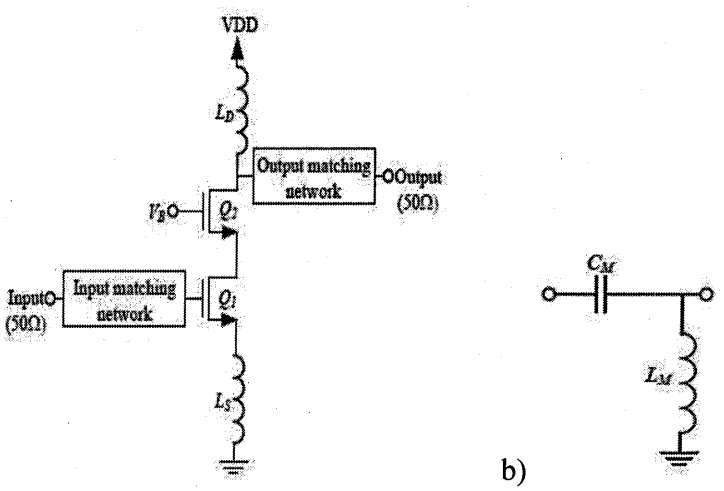

Figure 3.5. LNA topology used for validation.

Table 3.2. Performance and design variables for the LNA.

#	Variable description	Perf. (P) or Design (D)	Min. value	Max. value
1	Bandwidth (MHz)	P	387.0	882.3
2	1-dB compression Point (dB)	P	-21.04	-13.03
3	Center Frequency (GHz)	P	2.07	4.76
4	IIP3 (dB)	P	-11.58	2.64
5	Noise Figure (dB)	P	1.386	3.33
6	S21 (dB)	P	9.11	15.29
7	Input capacitance (fF)	D	238.2	1528.2
8	Input Inductance (nH)	D	1.811	10.610
9	Output capacitance (fF)	D	260.0	2902.6
10	Output Inductance (nH)	D	4.913	14.123
11	Transistor Q_1 length (nm)	D	160	510
12	Transistor Q_1 width (µm)	D	60	560
13	Transistor Q_2 length (nm)	D	200	610
14	Transistor Q_2 width (µm)	D	90	500
15	Source Inductance – L_S (nH)	D	1.707	8.671
16	Bias voltage - V_B (mV)	D	300	600

3.6.4.2 Voltage-Controlled Oscillator

The second circuit topology realized is the symmetrical cross-coupled VCO illustrated in Figure 3.6. It is a common VCO configuration that allows almost rail-to-rail output swing, and the cross-coupled PMOS-NMOS pair helps reduce 1/f noise (Razavi et al., 2011). Here also, 200 VCOs were designed. The list of considered design (D) and performance (P) variables as shown in Table 3.3, which also provides the ranges of values that were generated during the validation experiments. Note that the current source and inductance were fixed before designing each VCO and were given as input for the sizing process. Also, V_{tune} was varied within each design to get the tuning range. The VCO was designed with Cadence Virtuoso Spectre RF using CMOS 130 nm technology.

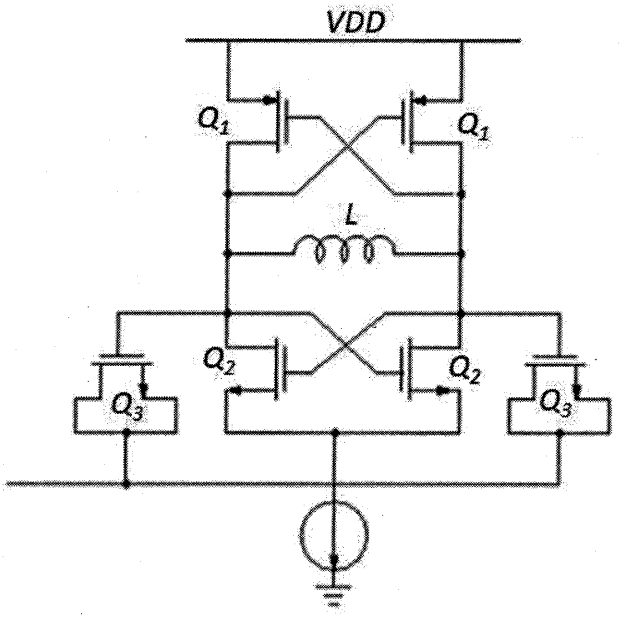

Figure 3.6. VCO topology used for validation.

Table 3.3. Performance and design variables for the VCO.

#	Variable description	Perf. (P) or Design (D)	Min. value	Max. value
1	Oscillation frequency (GHz)	P	2.022	9.924
2	Tuning range (MHz)	P	52.4490	723.40
4	Phase noise (dB/Hz)	P	-11.58	2.64
5	Power consumption (mW)	P	1.386	3.33
6	Transistor Q_1 length (μm)	D	0.5	4.1
7	Transistor Q_1 width (μm)	D	2.5	55
8	Transistor Q_2 length (μm)	D	0.5	4.4
9	Transistor Q_2 width (μm)	D	3	55
10	Transistor Q_3 length (μm)	D	0.8	4.2
11	Transistor Q_3 width (μm)	D	15	98

3.6.4.3 Mixer

The circuit topology realizing the mixer is the double-balanced Gilbert cell presented in Figure 3.7. It was selected because of its common use, since it provides good conversion gain and rejection at the input ports (Razavi, 2011). Moreover, the version used here includes a resonator that allows, through its parallel inductance L and capacitance C, to adjust the mixer to provide maximum response at the required frequency (Hu, 2017).

200 mixers were designed, with the list of considered design (D) and performance (P) variables as presented in Table 3.4, which also indicates the ranges of values that were generated during the validation experiments. Note that the resistance R, which was fixed before the mixer sizing, and the size and gate voltage of transistor Q_3, which was part of a current mirror, were given as input for the sizing process. The mixer was designed with Cadence Virtuoso Spectre RF using CMOS 130 nm technology. The lengths of all transistors were fixed to 130 nm.

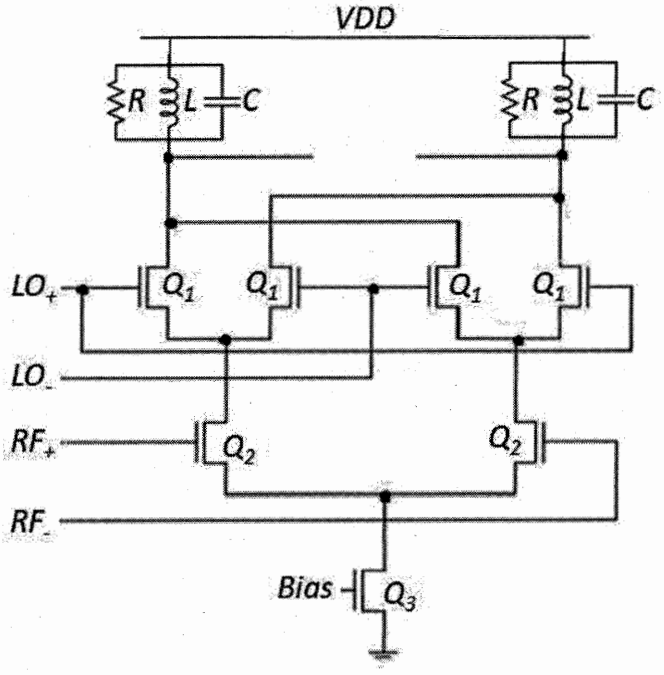

Figure 3.7. Cross-coupled Gilbert cell mixer topology used for validation.

Table 3.4. Performance and design variables for the mixer.

#	Variable description	Perf. (P) or Design (D)	Min. value	Max. value
1	RF frequency (GHz)	P	1.5	2.5
2	IF frequency (MHz)	P	50	500
3	Bandwidth (MHz)	P	48.68	409.1
4	1 db compression point(dBm)	P	-18.69	-8.613
5	IIP3 (dB)	P	-9.185	2.626
6	Voltage gain (dB)	P	4.894	15.97
7	SSB noise figure (dB)	P	4.464	5.637
8	LO to IF isolation (dB)	P	-424.2	-169.3
9	LO to RF isolation (dB)	P	-377.4	-278.4
10	RF to IF isolation (dB	P	-235.8	-137
11	Power consumption (mW)	P	8.560	47.712
12	Capacitance (pF)	D	0.965	129.899
13	Inductance (nH)	D	210	990
14	Transistor Q_1 width (µm)	D	200	430
16	Transistor Q_2 width (µm)	D	200	430
18	Voltage at the gate of Q_1 (V)	D	0.7	1.1
19	Voltage at the gate of Q_2 (V)	D	0.65	0.9

3.7 Results

The proposed LC-SNN and UC-SNN architectures, along with an architecture corresponding to a set of independent MLPs (I-MLPs), were all tested with the LNA (CMOS 180 nm), the VCO (CMOS 130 nm) and the mixer (CMOS 130 nm) circuits. For all three, 180 exemplars were used for training, 10 for validation and 10 for testing. The neuronal architectures were coded in Python using the Theano (Theano Development Team, 2016) and the Lasagne (Dieleman et al., 2015) libraries. The simulation was run on a laptop computer with a processor intel(R) Core(TM) i7-7700HQ @ 2.8 GHz, with 8 Go RAM and no dedicated graphics card.

Table 3.5 presents the results for the three microlectronics circuits. All three neuronal architectures tested are compared on three scores. The first score is the number of component sizes correctly predicted (i.e. with a prediction error below 5 % of the normalized values of the component sizes during the test phase). The second score is the mean prediction error during the testing phase. The third score is the number of generations needed to reach the testing phase (i.e. to get a prediction error below 5 % for all component sizes during the validation phase). Finally, the genetic algorithms were set to run a maximum of 10 generations.

Table 3.5. Simulation results.

Circuit	Neural network architecture	Nb of generations	Nb of components	Mean test error
LNA	UC-SNN	1	10/10	0.0391
	LC-SNN	1	10/10	0.0402
	I-MLP	10	8/10	0.0690
VCO	UC-SNN	2	6/6	0.0429
	LC-SNN	10	4/6	0.0731
	I-MLP	10	3/6	0.0924
Mixer	UC-SNN	3	6/6	0.0487
	LC-SNN	10	4/6	0.0708
	I-MLP	10	3/6	0.0861

As Table 3.5 shows, only the UC-SNN was successful in predicting all the component size values for all three types of circuits, with a final mean error at test time smaller than 5 % of the normalized values of all component sizes. Learning time for the UC-SNN was 4 minutes and 37 seconds for the LNA, 36 minutes and 21 seconds for the VCO and 47 minutes and 56 seconds for the mixer. While those learning times may be considered to be good results to find the component parameter values for a single set of performance criteria (e.g. comparing with Kalentyev et al., 2014, Weber et al., 2013, Yang et al. 2018), the biggest advantage of the method presented here is its capacity to

generalize the solution to other circuit performance criteria, as what it learns is an actual mapping from these to the component parameter values. As such, the main achievement here of the proposed method is the time needed to synthesize all ten test circuits, which was under 5 seconds for each of the three circuit topologies.

It should be noted that a single deep MLP was also tested, which tried to predict the sizes of all the components of a given circuit, but the results are not reported here since it failed to predict correctly the size of any circuit component. This deep neural network was allowed between 20 and 50 hidden layers by the genetic algorithm. Moreover, the vanishing gradient problem that often occurs in deep neural networks was attenuated by using residual connexions (He et al., 2016) and thus, it is hypothesized that its failure is due to the very small number of training exemplars (i.e. 200). This stresses the importance that an architecture such as a UC-SNN could have in contexts where a very small number of exemplars are available.

3.8 Conclusion

A neural network architecture consisting in a cascade of separately trainable shallow MLPs was proposed as a first-step solution to the sizing problem in analog RF microelectronic circuit design. This architecture was successful in predicting the correct value of each design component within a five percent error margin of its target normalized value.

Since it is difficult to gather a large enough number of microelectronic circuit designs for training a deep neural network, as opposed to other domains such as image classification or language translation, only two hundred exemplars were included in each dataset used to learn the final MLP sequences. Hence, to be able to tune all connection weights, all MLPs had only one or two hidden layers. That allowed an iterative constrained prediction process to take place and the sizing problem to be

solved with a relatively small training set. The proposed method also distinguishes itself from related work on circuit sizing, including simulated annealing, genetic algorithms and genetic programming, by going beyond finding idiosyncratic component values, as it creates a general mapping from the desired performance criteria to the component values.

While other sizing optimization methods such as genetic programming have been shown to present more precise solutions, those methods only find an idiosyncratic solution which is not generalizable to other sets of performance criteria for the same circuit topology. Moreover, those methods are very time-consuming. Hence, the new approach proposed here could be used to generate a DNN which, once trained, could quickly provide initial sizing conditions to a more precise idiosyncratic optimization method for a large range of performance criteria. This could greatly decrease the optimization time required for those methods to converge toward a solution. This application may constitute an interesting avenue for future research.

Finally, the obtained results show that the proposed method can be applied to many different types of microelectronic circuits in different technologies and, thus, it generalizes well. In particular, it was successful here in predicting the design component values of an LNA, a VCO and a mixer, where the first was designed using CMOS 180 nm technology, and the other two using CMOS 130 nm technology.

CONCLUSION

Le dimensionnement des composants d'un circuit microélectronique analogique en radiofréquences constitue un problème inverse, généralement mal posé, exigeant beaucoup de temps même pour les experts du domaine. Une nouvelle approche générant une cascade de réseaux de neurones superficiels a été proposée dans ce mémoire avec pour objectif de permettre le dimensionnement automatique des composants de tels circuits à partir de petits ensembles d'exemplaires de circuits.

Parmi les méthodes de dimensionnement automatique des composants proposées dans la documentation, les réseaux de neurones artificiels semblent présenter un très grand potentiel. D'abord, si on les compare aux méthodes à base de savoir et aux raisonnements basés sur des cas, ils partagent avec eux l'objectif de trouver une solution générale, c'est-à-dire une correspondance générale entre différents ensembles de critères de performance et des ensembles de valeurs de composants. Ceci présente un avantage énorme par rapport aux méthodes convergeant vers une solution idiosyncratique, applicable à un unique circuit, comme c'est le cas pour les autres méthodes ayant été proposées dans la documentation, tels que les recuits simulés, les algorithmes génétiques ou la programmation génétique. En effet, ces dernières méthodes doivent être reconduites pour chaque nouveau circuit, ce qui est peu efficace en termes de temps requis.

En plus, les réseaux de neurones artificiels partagent la grande force des méthodes de recuit simulé, d'algorithme génétique et de programmation génétique, soit la capacité de converger vers une solution sans qu'il ne soit nécessaire d'injecter au système du savoir explicite. La solution est simplement obtenue à travers l'interaction du réseau

de neurones artificiels avec des exemplaires qui lui sont fournis. Au contraire, les méthodes basées sur le savoir et celles basées sur les cas souffrent de la difficulté qui existe à extraire l'expertise d'un expert. Dans le cas du dimensionnement des composants d'un circuit microélectronique analogique en radiofréquences, cette expertise s'avère d'ailleurs particulièrement complexe étant donné le caractère non-linéaire des équations requises et les effets secondaires qui apparaissent, tels que ceux des capacités parasites. Les réseaux de neurones artificiels combine ainsi les forces principales propres à chacune de ces différentes méthodes, tout en étant robustes face à leurs faiblesses respectives.

Dans les dernières années, le type de réseau de neurones artificiels ayant présenté les résultats les plus impressionnants est le réseau de neurones profond. Toutefois, une limite importante des réseau de neurones profonds consiste en la très grande quantité d'exemplaires requise afin d'optimiser la très grande quantité de paramètres présents. Or, dans le domaine du dimensionnement des composants de circuits microélectroniques, chaque exemplaire nécessite beaucoup de temps de conception et il est rare d'avoir accès à une grande banque d'exemplaires, contrairement aux domaines d'analyse d'images ou de traduction de textes, ce qui rend l'utilisation des réseaux de neurones profonds traditionnels peu réaliste pour le dimensionnement des composants de circuits.

La méthode proposée dans ce mémoire a été développée à travers quatre itérations cherchant à exploiter les forces des réseaux de neurones artificiels, tout en palliant la limite du réseau de neurones profond quant au nombre d'exemplaires requis. La première itération constitue le niveau de base, où un unique réseau de neurones profond cherche à solutionner à lui seul le problème de dimensionnement de tous les composants d'un circuit microélectronique. Ce réseau, dont les hyperparamètres sont spécifiés par un algorithme génétique, n'a pas permis de dimensionner adéquatement

tous les composants du circuit, mais est tout de même parvenu à prédire correctement un sous-ensemble de composants.

La deuxième itération a cherché à exploiter les forces des réseaux de neurones artificiels, tout en palliant la limite du réseau de neurones profond quant au nombre d'exemplaires requis. Pour ce faire, plutôt que d'entraîner un réseau de neurones profond statique, le problème de dimensionnement des composants du circuit complet a été divisé en un ensemble de sous-problèmes correspondant chacun à un seul composant du circuit. Chaque composant s'est alors vu attribuer son propre réseau de neurones artificiels. Aussi, chacun de ces réseaux de neurones artificiels ne possédant qu'une ou deux couches cachées, on les appelle réseaux de neurones superficiels, par opposition aux réseaux de neurones profonds. Cette approche a permis de prédire correctement un plus grand nombre de dimensions de composants de circuits microélectronique, mais il a été systématiquement impossible de prédire certaines d'entre elles. L'hypothèse émise pour expliquer cette limite est liée au caractère sous-déterminé des problèmes inverses. En effet, il est possible que certaines dimensions de composants soient interdépendantes et que la connaissance d'un sous-ensemble soit nécessaire afin de prédire les autres. La troisième itération a cherché à pallier ce problème.

La troisième itération a proposé une méthode visant à construire une cascade de réseaux de neurones superficiels, à travers la création progressive d'une séquence de plusieurs réseaux de neurones peu profonds interconnectés. Cette méthode permet de contraindre graduellement le problème à caractère sous-déterminé que représente le dimensionnement des composants d'un circuit. Chaque réseau de neurones superficiel a alors une seule sortie et n'est inclus dans la séquence finale que s'il parvient à prédire un composant avec une erreur de validation inférieure à 5 % de sa valeur normalisée. Une fois inclus dans cette séquence, le réseau de neurones superficiel voit sa sortie utilisée en entrée par les réseaux de neurones superficiels tentant de prédire les

dimensions de composants subséquents. Cette méthode a permis de prédire correctement l'ensemble des composants du circuit microélectronique d'amplificateur à faible bruit. Toutefois, il n'a pas été possible de généraliser ce succès au dimensionnement des composants d'autres circuits, tels qu'un oscillateur contrôlé par une tension ou un mélangeur. La limite principale de cette méthode est son inefficacité lorsque deux dimensions de composants sont interdépendantes et qu'aucune des deux ne peut être prédite sans la connaissance de l'autre. Dans le cas de l'amplificateur à faible bruit, la méthode de conception des exemplaires servant à l'entraînement a fixé les valeurs des dimensions selon un ordre systématique. Ainsi, il était toujours possible de progresser dans la cascade de réseaux de neurones, comme on ne pouvait rencontrer deux dimensions de composants où chacune nécessitait absolument la connaissance de l'autre. Dans le cas des deux autres types de circuits microélectroniques, l'ordre dans lequel les dimensions des composants ont été fixées s'est avérée moins rigide. L'hypothèse a alors été posé que c'est ce qui a réduit l'efficacité de cette architecture. La quatrième itération a cherché à diminuer l'impact d'une telle situation.

La quatrième itération a ainsi cherché à raffiner la méthode de génération d'une cascade de réseaux de neurones superficiels. Cette version est similaire à la précédente, mais maintenant, tout réseau de neurones superficiel améliorant la meilleure prédiction enregistrée par un autre réseau pour la valeur d'un même composant est ajouté à la cascade de réseaux finale. Ceci permet de faire en sorte que la prédiction de chaque composant puisse bénéficier des prédictions approximées de tous les autres composants du circuit. Ceci a pour avantage de contraindre encore plus fortement la recherche d'une solution et ainsi d'augmenter la robustesse de la méthode face au caractère sous-déterminé du problème et aux potentielles interdépendance entre les valeurs des composants. La cascade finale des réseaux de neurones superficiels peut ainsi s'avérer très profonde, mais chaque réseau utilisé comporte néanmoins moins de dix couches cachées. Cette méthode a permis de résoudre le problème de dimensionnement des composants pour les trois circuits testés et dans deux technologies différentes :

amplificateur à faible bruit (CMOS 180 nm), oscillateur contrôlé par une tension (CMOS 130 nm) et mélangeur (CMOS 130 nm). Ceci a permis de valider la généralité de l'approche proposée.

Bien que d'autres méthodes d'optimisation de dimensionnement proposées dans la documentation, tel que la programmation génétique, aient présenté des solutions plus précises, ces méthodes paient cet amélioration en sacrifiant la généralité de leur solution. En effet, ces méthodes ne trouvent qu'une solution idiosyncratique qui ne peut être généralisée à d'autres ensembles de critères de performance pour la même topologie de circuit. De plus, le processus d'optimisation de ces méthodes prend généralement beaucoup de temps. Par conséquent, la nouvelle approche proposée ici pourrait être utilisée pour générer une cascade de réseaux de neurones superficiels qui, une fois entraînée, pourrait rapidement fournir des conditions de dimensionnement initiales à une méthode d'optimisation idiosyncratique plus précise et ce, pour un large éventail de critères de performance. Cela pourrait réduire considérablement le temps d'optimisation nécessaire pour que ces méthodes convergent vers une solution. Cette application peut constituer une piste intéressante pour des recherches futures.

Une limite de la méthode proposée ici est la nécessité d'obtenir un certain nombre d'exemplaires pour chacun des circuits dont on veut automatiser le dimensionnement des composants. Ici, par exemple, 200 exemplaires de chacun de ces types de circuits ont dû être réalisés au préalable, ce qui représente beaucoup de temps et d'investissement, en plus de faire reposer le succès de la méthode en partie sur la qualité des exemplaires réalisés. Aussi, des travaux futurs seront consacrés à complémenter l'approche proposée par un algorithme d'apprentissage par renforcements, qui communiquera d'un côté avec l'algorithme de dimensionnement des composants et de l'autre, avec un outil d'analyse de circuits, tel que *Cadence Virtuoso SpectreRF*. Ce faisant, le système sera en mesure de tester directement les prédictions de l'algorithme de dimensionnement des composants et d'apprendre à apporter les correctifs

nécessaires. Dans un premier temps, comme ce processus sera automatisé, cela permettra d'augmenter le nombre de designs pour lesquels la correspondance entrée-sortie est connue, ce qui permettra d'améliorer la performance de l'algorithme de dimensionnement des composants présenté ici. Dans un deuxième temps, les prédictions de l'algorithme de dimensionnement des composants devenant plus précises, les corrections requises par l'algorithme d'apprentissage par renforcement deviendront ainsi plus petites, plus rapides et plus efficaces. Ceci pourrait donc permettre d'améliorer encore davantage le dimensionnement automatique des composants des circuits microélectroniques et d'en augmenter la puissance, la robustesse et la précision.

RÉFÉRENCES

Aamodt A et Plaza E (1994) Case-based reasoning: foundational issues, methodological variations, and system approaches," *Artificial Intelligence Communications*, *7*(1), 39-59.

Aarts E., Korst J. et Michiels W. (2005). Simulated annealing. Dans Burke E. K et Kendall G. (dir.) , *Search Methodologies* (p. 187–210). Boston, MA : Springer.

Al-Kashef A., Zaky M. M., Dessouky M. I. et El-Ghitani H. (2008). A case-based reasoning approach for the automatic generation of VHDL-AMS models. Dans *BMAS 2008 - Proceedings of the 2008 IEEE International Behavioral Modeling and Simulation Workshop* (p. 100–105).

Anantrasirichai N., Hayes W., Allinovi M., Bull D. et Achim A. (2017). Line detection as an inverse problem: application to lung ultrasound imaging. *IEEE Transactions on Medical Imaging*, *36*(10), 2045–2056.

Bahl, I. J. (2003). *Lumped elements for RF and microwave circuits*. Boston : Artech House.

Barros M., Guilherme J., et Horta N. (2010). Analog circuits optimization based on evolutionary computation techniques. *Integration. VLSI Journal.*, *43*(1), 136–155.

Bengio Y., Lamblin P., Popovici D. et Larochelle H. (2007). Greedy layer-wise training of deep networks," Dans Schölkopf B., Platt J. C. et Hoffman T. (dir.). *Advances in Neural Information Processing Systems 19*, (p. 153–160). Cambridge, MA, USA : MIT Press.

Bergstra J., Bardenet R., Bengio Y. et Kégl B. (2011). Algorithms for hyper-parameter optimization. Dans *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*, (p.2546–2554).

Boukadoum M., Nabki F. et Ajib W. (2012). Towards the neural network-based design of nadiofrequency now-noise amplifiers. Dans *IEEE International Symposium on Circuits and Systems (ISCAS)*, (p. 2741–2744).

Buhman M. D. (2003). *Radial basis functions: theory and implementations*. Cambridge, UK: Cambridge University Press.

Chartier S. et Boukadoum M. (2006). A bidirectional heteroassociative memory for binary and grey-level patterns. *IEEE Transactions on Neural Networks*, *17*(2), 385-396.

Cho K., van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H. et Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. Dans *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (p.1724–1734).

Commandeur F., Goeller, M., Betancur, J., Cadet, S., Doris M. K., Chen, X., Berman D. S., Slomka, P. J., Tamarappoo B. et Dey, D. (2018). Deep learning for quantification of epicardial and thoracic adipose tissue from non-contrast CT. *IEEE Transactions on Medical Imaging*, *37*(8), 1835–1846.

Deb K. et Kalyanmoy D. (2001). *Multi-objective optimization using evolutionary algorithms*. New York, NY, USA : John Wiley & Sons, Inc.

Dieleman S., Schlüter, J., Raffel, C., Olson, E. et al. (2015). Lasagne: First release. DOI: http://dx.doi.org/10.5281/zenodo.27878.

Dumesnil E., Nabki F. et Boukadoum M. (2015). RF-LNA circuit synthesis using an array of artificial neural networks with constrained inputs. Dans *IEEE International Symposium on Circuits and Systems (ISCAS)*, (p. 573–576).

Dumesnil E. et Boukadoum M. (2019). *Automatic components sizing of analog RF circuits by cascaded shallow neural networks*. Prépublication, soumis à *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, no. de soumission: TCAD-2019-0127, mars 2019.

Eshelman L. J. (1999). Genetic algorithms. Dans Back T., Fogel D. B. et Michalewicz Z. (dir.). *Basic algorithms and operators* (p. 64-80). Bristol, UK, IOP Publ. Ltd.

Fakhfakh M., Tlelo, E et Siarry P. (2015). *Computational Intelligence in Analog and Mixed-Signal (AMS) and Radio-Frequency (RF) Circuit Design*. Springer.

Fukushima K. et Miyake S. (1982). Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, *15*(6), 455–469.

Gallagher K. et Sambridge M. (1994). Genetic algorithms: a powerful tool for large-scale nonliner optimization problems. *Computers & Geosciences*, *20*(7), 1229-1236.

Hadamard J. (1923). *Lectures on Cauchy's problem in linear partial differential equations*. New Haven : Yale University Press.

Harjani R. (1989). OASYS: a framework for analog circuit synthesis. Dans *Proceedings., Second Annual IEEE ASIC Seminar and Exhibit*, (p. 1247-1266).

Hasani R. M., Haerle D. et Grosu, R. (2016), Efficient modeling of complex analog integrated circuits using neural networks. *12th PRIME, Lisbon (Portugal)*, (p. 1-4).

Haykin S. (1999). *Neural networks: a comprehensive foundation* (2e édition). Upper Saddle River, NJ: Prentice Hall.

He K., Zhang X., Ren S. et Sun, J. (2016). Deep residual learning for image recognition. Dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 770-778).

Heaton J., Goodfellow I., Bengio Y. et Courville C. (2017). Deep learning. *Genet Programming and Evolvable Machines*, 19(1-2), 305-307.

Hochreiter S. et Schmidhuber J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Holland J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*. Cambridge, MA, USA : MIT Press.

Hosseini M. P., Pompili D., Elisevich K. et Soltanian-Zadeh H. (2017). Optimized deep learning for EEG big data and seizure prediction BCI via internet of things," *IEEE Transactions on Big Data*, *3*(4), 392–404.

Hu X., (2017). *RF CMOS tunable gilbert mixer with wide tuning frequency and controllable bandwidth: design sythesis and verification*. (Mémoire de maîtrise). Wright State University.

Hubel D. H. et Wiesel T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *Journal of Physiology*, *148*(3), 574–591.

Jafari A., Sadri S. et Zekri M. (2010). Design optimization of analog integrated circuits by using artificial neural networks. *2nd SoCPaR, Paris (France)*, (p. 385-388).

Kabir H., Zhang L., Yu M., Aaen P. H., Wood J. et Zhan Q.-J. (2010). Smart modeling of microwave devices. *IEEE Microwave Magazine*, *11*(3), 105-118.

Kalentyev A. A., Babak L. I. et Garays D. V. (2014). Genetic-algorithm-based sythesis of low-noise amplifiers with automatic selection of active elements and dc biases. Dans *9th European Microwave Integrated Circuit Conference*, (p. 520–523).

Kasabov N., Zhou L., Doborjeh M. G., Doborjeh Z. G. et Yang J. (2017). New algorithms for encoding, learning and classification of fMRI data in a spiking neural network architecture: a case on modeling and understanding of dynamic cognitive processes," *IEEE Transactions on Cognitive and Developmental Systems*, *9*(4), 293–303.

Keller, J.B. (1976). Inverse problems. *The American Mathematical Monthly*, *83*(2), 107–118.

Kidd A. L.(1987). *Knowledge acquisition for expert systems: a practical handbook.* New York, NY, USA : Plenum Press.

Kingma D. P. et Ba J. (2015). Adam: a method for stochastic optimization. Dans *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Kirkpatrick S., Gelatt C. D., et Vecchi M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kolodner J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, *6*(1), 3–34.

Korovkin M. V., Chechurin V. L. et Hayakawa M. (2007). *Inverse problems in electric circuits and electromagnetics*. US : Springer.

Kosko B. (1988). Bidirectional associative memories. *IEEE Transactions on Systems, Man and Cybernetics*, *18*(1), 49-60.

Koza J R, Bennett F. H., Andre D. et Keane M. A. (2000), "Synthesis of topology and sizing of analog electrical circuits by means of genetic programming," *Comp. Methods in Applied Mechanics and Engineering*, 186(2–4), 459–482.

Koza J. R., Bennett F. H., Andre D., Keane M. A. et Dunlap F. (1997). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, *1*(2), 109–128.

LeCun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W. et Jackel L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Liao T. et Zhang L. (2017). Parasitic-aware GP-based many-objective sizing methodology for analog and RF integrated circuits. *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. (p. 475-480).

Liu J., Pan, Y., Li, M., Chen Z., Tang L., Lu C. et Wang J. (2018). Applications of deep learning to MRI images: a survey. *Big Data Mining and Analytics*, *1*(1), 1–18.

Liu L., Shen C. et van den Hengel A. (2017). Cross-convolutional-layer pooling for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(11), 2305–2313.

Liu B., Zhang, Q. et Gielen G. (2014), A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Trans. Evolutionary Computation*, *18*(2), 180-192.

Liu B., Zhongkun M., Vandenbosch, G. A. E., Gielen G. et Excell P. (2014), An efficient method for antenna design optimization based on evolutionary computation and machine learning. *IEEE Trans. Antennas & Propagation*, *62*(1), 7- 18.

Lohn J. D. et Colombano S. P. (1998). Automated analog circuit synthesis using a linear representation. Dans *Evolvable Systems: From Biology to Hardware*, (p. 125–133).

Lourenço N., Martins R., Canelas A., Póvoa R. et Horta N. (2016). AIDA: layout-aware analog circuit-level sizing with in-loop layout generation, *Integration, the VLSI Journal*, *55*, 316-329.

Marques O., Drummond T. and Vasco D. (2003). A computational strategy for the solution of large linear inverse problems in geophysics. Dans *Proceedings of International Parallel and Distributed Processing Symposium*, (p.8-15).

Mashohor S., Evans J. R. et Arslan T. (2005). Elitist selection schemes for genetic algorithm based printed circuit board inspection system. Dans *IEEE Congress on Evolutionary Computation*, *2*, (p. 974–978).

Mitchell M. (1998). *An introduction to genetic algorithms*. Cambridge, Ma, USA : MIT Press.

Mohamad-Saleh J. et Hoyle B. S. (2008). Improved neural network performance using principal component analysis on matlab. *International Journal of The Computer, the Internet and Management*. *16*(2) 1–8 .

Ochotta E. S., Rutenbar R. A., et Carley L. R. (1996). Synthesis of high-performance analog circuits in ASTRX/OBLX," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *15*(3), 273–294.

Pandit C., Patnaik A. et Sinha S. N. (2007). Neural network-based CAD models for analysis and design of fin-lines for mm-wave applications. Dans *Applied Electromagnetics Conference*, (p.1–4), December 2007.

Passos F., Martins R., Lourenço N., Roca E., Póvoa R., Canelas A., Castro-López R., Horta N. et Fernández F.V. (2017). Enhanced systematic design of a voltage-controlled oscillator using a two-step optimization methodology, *Integration*, *63*, 351-361.

Rahnama M., Gilmanek Y. M. et Kordalivand A. M. (2010). Ultra wide-band LNA using RFCMOS technology and tunability band with neural network. Dans *Control and System Graduate Research Colloquium (ICSGRC)*, (p.75–79).

Razavi B., (2011). *RF Microelectronic* (2nd edition) Upper Saddle River, NJ, USA: Prentice Hall Press, 2011.

Sadughi S., (2002). "A hybrid reasoning-based tool for analog circuit synthesis. *Scientia Iranica*, *9*(4), 419–224.

Sasikumar A. et Muthaiah R. (2016). Towards analog design automation using evolutionary algorithm: a review. *Indian Journal of Science and Technology*, *9*(39), 1–12.

Schank R.C. (1982). *Dynamic memory: a theory of reminding and learning in computers and people*. Cambridge, UK: Cambridge University Press.

Scheible J. et Lienig J. (2015). Automation of analog IC layout: challenges and solutions. *Int. Symp. Physical Design (ISPD '15)*. (p. 33-40).

Schmidhuber J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, *61*, 85-117.

Sheu B. J., Lee J. C., et Fung A. H. (1990). Flexible architecture approach to knowledge-based analogue IC design. Dans *IEEE Proceedings G - Circuits, Devices and Systems*, *137*(4), 266–274.

Shi G. (2018). Toward automated reasoning for analog IC design by symbolic computation – a survey. *Integration, the VLSI journal*, *60*, 117–131.

Shukla A., Pandey H. M. et Mehrotra D. (2015). Comparative review of selection techniques in genetic algorithm. Dans *International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, (p. 515–519).

Smiti A. et Elouedi Z. (2011). Overview of maintenance for case based reasoning systems. *International Journal of Computer Applications*, *32*(2).

Srivastava N., Hinton G., Krizhevsky A., Sutskever I. et Salakhutdinov R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(1), 1929–1958.

Starck J.-L. (2016). Sparsity and inverse problems in astrophysics. *Journal of Physics: Conference Series, 699*(1), 1–10.

Sutskever I., Vinyals O. et Le Q. V. (2014). Sequence to sequence learning with neural networks, Dans *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, (p. 3104–3112).

Tarantola A. (2005). *Inverse problem theory and methods for model parameter estimation*. Society for Industrial and Applied Mathematics.

Theano Development Team (2016). Theano: a python framework for fast computation of mathematical expressions. DOI:http://arxiv.org/abs/1605.02688.

Tikhonov A. N. (1963). On the regularization of ill-posed problems. *Dokl. Akad. Nauk SSSR, 153*, 49–52.

Tripathi J. N., Mukherjee J. et Apte P. R. (2013). Design automation, modeling, optimization, and testing of analog/RF circuits and systems by PSO. In Fornarelli G. et Mescia L. (dir.), *Swarm intelligence for electric and electronic engineering*, (p. 57-70), IGI Global.

Weber T. O., Chaparro S. et W. A. M. Van Noije (2013). Synthesis of a narrow-band low noise amplifier in a 180 nm CMOS technology using simulatedaAnnealing with crossover operator. Dans *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, 1–5.

Whitley D. et Sutton A. M. (2012). Genetic algorithms – a survey of models and methods. Dans Rozenberg G., Bäck T. et Kok J. N. (dir.). *Handbook of Natural Computing* (p. 637–671), Berlin, Heidelberg : Springer.

Yang Y., Zhu H., Bi Z., Yan C., Zhou D., Su Y. et Zeng X. *(2018).* Smart-MSP: a self-adaptive multiple starting point optimization approach for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37*(3), 531–544.