Étude de l'Expressivité dans les Réseaux de Neurones selon une Perspective Constructiviste : Application à la Génération de Texte

Lucas Foulon
Encadré par Laëtitia Matignon & Fréderic Armetta

Université Claude Bernard, Lyon 1, France
Équipe SMA, Laboratoire LIRIS

Résumé Les réseaux de neurones sont de plus en plus utilisés dans le domaine du *Deep Learning*. Ils ont révélé de nombreux résultats satisfaisants, dans la traduction de textes, et la reconnaissance d'images. Des avancées scientifiques ont été menées sur les réseaux de neurones récurrents, en améliorant leurs capacités d'apprentissage et de généralisation des concepts appris. Cependant, la puissance de calcul disponible joue un rôle prépondérant dans l'efficacité de ces réseaux. La motivation de ce travail a ainsi été de proposer des améliorations de l'expressivité des réseaux, vers une modération de la puissance de calcul nécessaire à leur fonctionnement, en s'inspirant du cadre d'analyse proposé par les neurosciences et les approches développementales. Nous nous sommes ainsi intéressés à réduire la taille du réseau et à la définition dynamique de la couche d'entrée, ainsi qu'à la coordination de plusieurs réseaux exploitant des horizons perceptifs différents, pour le problème applicatif de la génération réaliste de textes.

Mots clés : Réseaux de neurones récurrents, apprentissage constructiviste, régularité de long terme, algorithme du gradient

Abstract. Neural networks are increasingly used in the field of Deep Learning. They revealed many satisfying results, in texts translation, and image recognition. Scientific advances have been conducted on recurrent neural networks improving mostly their learning abilities and learned concepts generalization. However, the available computing power has a major role in the effectiveness of these networks. The motivation of this work has been to suggest improvements to networks expressiveness towards a moderation of the computing power needed for their operation taking inspiration from neuroscience and developmental approaches studies. So we are interested to reduce the network size and dynamically setting at input layer, as well as the coordination of multiple networks, exploiting different perceptual horizons for the application problem of the realistic texts generation.

Keywords: Recurrent neural networks, constructivism learning, long-term dependancies, gradient descent

1 Introduction

L'apprentissage développemental est un domaine émergent situé entre les sciences cognitives et les sciences développementales, en particulier la psychologie développementale et les neurosciences développementales[15]. On peut citer deux objectifs de ce domaine :

- construire des systèmes pouvant apprendre et s'adapter à leur environnement pour obtenir une certaine autonomie partielle ou totale,
- tester et valider les modèles cognitifs provenant de la biologie et de la psychologie que l'on peut qualifier de *Méthodologie Synthétique* souvent mis en expérimentation grâce aux techniques de la robotique[27].

La question du développement cognitif est au centre du constructivisme introduit par Piaget[31]. L'interprétation de la réalité, supposée être la conséquence entre les interactions et l'expérience, dépend-t-elle du développement cognitif ? Comment être sûr que notre apprentissage se construit grâce à la structure de notre esprit ? Pour cela, tel que le proposent les biologistes et plus particulièrement les neuroscientifiques, s'inspirer des comportements observés et étudier leur mécanique chez des espèces dotées d'intelligence, est une voie prometteuse pour améliorer l'efficacité des systèmes apprenants.

C'est selon cette perspective que les réseaux de neurones artificiels ont été conçus en tentant d'imiter les neurones biologiques. Le neurone formel a été conçu par Mc Culloch et Pitts [18], pour réaliser des fonctions logiques, arithmétiques et symboliques. Bien sûr, on peut y voir dans le nom une certaine métaphore, un paradigme, et non pas la copie conforme du modèle biologique.

Depuis, de nombreux progrès ont été réalisés et différents modèles ont été proposés. Beaucoup d'entre eux se consacrent à la réalisation de représentations efficaces des données en entrée, d'autres modifient l'architecture du réseau pour permettre d'améliorer l'apprentissage. Nous développons ces travaux, de manière distincte, dans les parties 3.3 et 3.5.

La question encore ouverte à laquelle nous nous sommes intéressés au cours de ce stage concerne le compromis entre la qualité des prédictions réalisées par les réseaux de neurones par rapport à leur taille et le volume de données disponibles pour leur paramétrage. Nous proposons ainsi dans la partie 4 de faire évoluer la couche d'entrée afin de réduire la taille du jeu de paramètre à apprendre.

Dans la partie 5 nous discutons de la possibilité de coordonner différents réseaux dont le périmètre des perceptions est différent, toujours afin d'améliorer leur efficacité de prédiction et de modérer les données nécessaires à leur paramétrage.

2 Contexte

2.1 Les Réseaux de Neurones Récurrents

Les Réseaux de Neurones Artificiels. Inspiré du modèle biologique, ils furent créés à partir du modèle du *Perceptron* crée en 1957 par Frank Rosenblatt

[25]. Ce modèle est vu comme le réseau de neurone le plus simple, c'est un classifieur linéaire de la forme :

$$f(x) = \begin{cases} 1 & \text{si } w * x + b > 0 \\ 0 & \text{sinon} \end{cases}$$

où x est le vecteur d'entrée du neurone, $f(x)$ sa sortie, et w le vecteur de poids. Rosenblatt s'inspire de la loi de Hebb, dédiée à l'apprentissage non supervisé, pour ajuster la valeurs des poids et permettre d'obtenir les valeurs de sortie voulues. Cependant, il rajoute un paramètre à la loi de Hebb d'origine qui permet de prendre en compte l'erreur observée en sortie, en apprentissage supervisé.

Cependant, Minsky et Papert démontrèrent dans leur article, datant de 1969, que ce modèle était limité car il n'était pas capable de résoudre les problèmes non-linéaire. Quelques années s'écoulèrent avant que Werbos et Rumelhart[26], respectivement proposent et mettent au point la rétropropagation dans un réseau de neurones multicouches. C'est à partir de cette méthode que l'on peut parler de "*backpropagation* de l'erreur", ou encore "la rétropropagation du gradient".

De nombreux termes existent, l'idée étant que cet algorithme d'apprentissage est capable de retourner l'erreur en sortie sur tous les poids d'un réseau possèdant plusieurs couches. Cette méthode est encore très convoitée, et nous l'utiliserons par la suite dans les différents modèles que nous proposerons.

La Récurrence dans les Réseaux de Neurones. Les réseaux de neurones récurrents (RNN) sont une inspiration des réseaux de neurones classiques fonctionnant en *feedforward* (c'est-à-dire prédiction à partir de l'input), développés par Hopfield, en 1982, durant les années sombres du réseau de neurones qui suivirent l'article de Minsky et Papert. La différence qui réside dans les RNNs, c'est qu'ils peuvent prédire la sortie suivant l'entrée et suivant l'état précédant appelé état de récurrence (dans la couche cachée). En résumé, la prédiction à l'instant t se fera à partir de l'input x_t et de l'état "caché" h_t.

Bien entendu, l'état caché au temps $t = 0$ est nul. Dans tous les autres cas, pour tout $t > 0$, l'état caché à l'instant t est calculé à partir de l'entrée courante x_t et de l'état caché précédent (à l'instant $t - 1$).

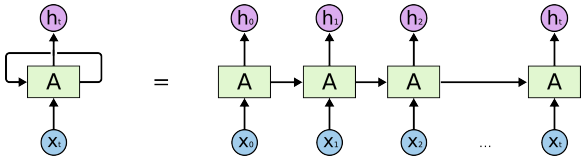

Figure 1. Transmission du contexte dans un réseau de neurones récurrents au cours du temps[22]

Cette étape de récurrence le différencie grandement des NN (*Neural Network*) classiques, car cela ajoute la notion de temporalité dans sa prédiction. Dans un

réseau classique, la sortie ne dépend que de son entrée, alors que le RNN dépend des états précédents. Sachant que h_t est le résultat de l'opération de $input_t$ et de h_{t-1}, on peut rapidement en déduire que h_t dépendra aussi, à plus petite échelle, de $input_{t-1}$ et h_{t-2}. La formule suivante explique le calcul du contexte de la couche cachée : $h_t = g(W_{xh} * input_t + W_{hh} * h_{t-1})$, avec W_n les matrices des poids entre les différentes couches, et g une fonction non linéaire, comme la fonction *Sigmoid*. Comme $input_{t-2}$ est inclus dans h_{t-1}, il est facile d'en conclure que $input_t$ influencera d'avantage le calcul de h_t que tous les autres $input_{n<t}$. Il y a une propagation de l'information en entrée au cours du temps, mais qui s'atténue petit à petit.

Les RNN permettent de donner de très bons résultats de prédictions, notamment dans la génération de texte. L'exemple de la figure 2 montre comment le réseau arrive à prédire le mot "hello". Sur cet exemple, le réseau réccurent enregistre consécutivement les différents caractères du texte sur la couche d'entrée (bit à 1 à chaque pas de temps), et prédit le caractère suivant sur la couche de sortie.

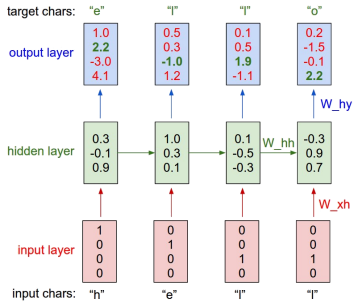

Figure 2. Prédiction du mot "hello" dans un RNN[13]

Avec un ensemble de données basé sur Wikipedia contenant 10 millions de caractères, le RNN de l'article [29] est capable de générer un texte en initialisant le réseau par la phrase "*The meaning of life is*" :

"*The meaning of life is the tradition of the ancient human reproduction : it is less favorable to the good boy for when to remove her bigger. In the show's agreement unanimously resurfaced.* [...] *To adapt in most parts of North America, the dynamic fairy Dan please believes, the free speech are much related to the*"

Les phrases ainsi générées échappent au bon sens, mais obtiennent une forme correcte. L'orthographe des mots et la grammaire sont plutôt bien respectés. Il y a un apprentissage plus profond que la simple recomposition de phonème qui s'est créé. Dénué de sens, le réseau a pu tout de même générer un texte respectant certaines règles. Tout en sachant que le réseau apprend caractère par caractère, comme dans la figure 2, on peut conclure qu'il s'est construit d'autres connaissances de plus haut niveau, même si elles ne sont pas encore totalement cohérentes.

On peut ainsi constater que la qualité de l'apprentissage est très aboutie par certains aspects, mais reste très perfectible pour d'autres. Au cours de ce travail, on discute et propose différents modèles visant à améliorer la capacité d'apprentissage.

2.2 L'Apprentissage Constructiviste

Il y a de nombreuses divergences quant au processus d'apprentissage chez les êtres vivants. Et lorsque l'on parle d'apprentissage, on peut parfois entendre développement cognitif pour certains. Chez les neuroscientifiques, l'apprentissage est la réaction cérébrale à un stimulus. Après avoir perçu l'information, elle est traitée puis intégrée. Pour les psychologues, il est le résultat du vécu, le changement de comportement d'un individu, dû à l'introduction de nouvelles connaissances.

Si l'on se place dans le domaine des neurosciences, on peut constater que l'intégration des connaissances est alors perçue physiquement comme une empreinte laissée dans la structure du cerveau. Mais qu'est-ce qui incite et favorise la création de ces microscopiques changements lorsque les êtres vivants sont en interaction avec leur environnement ?

Les Premières Théories. Différents courants existent. Les plus anciennes doctrines autour de l'apprentissage ont été fondées par les philosophes de l'Antiquité. On peut citer deux pensées qui divergent sur le savoir *a priori*, l'empirisme et le rationalisme. L'empirisme définit l'expérience sensorielle comme étant au centre de la construction de la connaissance. Tandis que le rationalisme donne plus de place à l'inné pour fonder la connaissance.

Pour certains scientifiques, ces modifications sont causées par l'environnement. Le moteur de l'apprentissage serait externe à l'individu. C'est le monde extérieur qui orchestrerait le développement de l'apprentissage. Cette vision est celle des *béhavioristes*, établie à l'origine par John B. Watson [30] en 1913. Ils défendent l'idée selon laquelle une personne change de comportement grâce aux interactions auxquelles elle est confrontée, appartenant à un monde en perpétuel changement. En d'autres termes, le béhaviorisme ne s'intéresse plus à la conception interne de la personne, vu alors comme une *boîte noire*, mais aux stimuli, et aux réactions qui sont engendrés. Seules ces deux entités et leurs associations, observables et mesurables, sont alors prises en compte.

Le cognitivisme, quant à lui, conceptualise la représentation interne chez l'apprenant. Le processus mental est étudié, de la réponse à la stratégie élaborée, en passant par la planification. Les cognitivistes tentent d'expliquer comment l'information est reçue, organisée, stockée et récupérée pour former la connaissance. Mais tout comme le béhaviorisme, le cognitivisme donne une grande importance au rôle que joue l'environnement dans l'apprentissage.

Le Constructivisme. Comme expliqué dans l'introduction, pour les constructivistes, l'expérience joue un grand rôle dans le développement de l'intelligence,

alors que l'inné jouerait un rôle mineur. De plus, l'apprentissage constructiviste tente de se détacher du cognitivisme en n'expliquant plus seulement le cerveau comme étant un système effectuant des tâches de traitement de l'information[17].

L'ensemble des théories constructivistes a permis d'étoffer les connaissances dans le domaine du développement cognitif. En particulier celle de Jean Piaget, qui définit plusieurs stades dans le développement cognitif chez l'enfant[23]. Pour Piaget, les connaissances ne relèvent pas de l'inné, mais des constructions faites par les interactions avec l'environnement. Il définit l'apprentissage comme étant un équilibre entre *assimilation* et *accommodation*. L'*assimilation* est la capacité à généraliser les nouvelles connaissances. Ces dernières sont des variations de celles déjà acquises, qui pourront être de plus en plus complexes à des étapes différentes du développement. Tandis que l'*accommodation* permet de spécialiser l'information, et donc de permettre au système cognitif de s'adapter aux *schèmes*[1] qui ne sont pas en capacité d'être assimilés avec les connaissances déjà acquises. Ces deux processus nécéssitent tout de même la capacité innée de cerner les similarités, et de se souvenir.

Parmis les différentes théories présentées, le constructivisme s'intéresse aux mêmes problématiques que celles rencontrées dans le domaine de l'*Intelligence Artificielle*, et peut ainsi servir, en se basant sur l'intelligence humaine, comme un cadre de référence.

Le Modèle Constructiviste. Le modèle constructiviste se base alors sur les interactions entre le monde réel et le système. Le système contient une "image du monde" qu'il s'est construit, et ses capacitées sensorimotrices. D'un côté, le système perçoit et agit sur sa propre vision de l'environnement, construite à partir de ses refléxions cognitives, et de l'autre, il interagit avec le "réel". Pour résumer, le système créé sa représentation du monde à partir de ses interactions, et agit sur ce qu'il pense être la réalité. Bien sûr, le monde réel sera aussi impacté par ces actions. Ce modèle dynamique permet de définir une frontière entre la réalité et le système.

La vision constructiviste se base alors sur le fait même que le sujet n'a pas accès à cette réalité. Son système cognitif acquiert des connaissances pour ensuite représenter le réel à sa façon. Si l'organisation de ces connaissances est mise à l'épreuve, il finira par la modifier pour obtenir une toute autre représentation du monde qui n'en sera pas plus objective, car la réalité ne deviendra pas moins *inaccessible*.

Apprentissage du langage. Pour Piaget, le langage est une capacité qui se développe chez l'enfant autour de l'âge de 2 ans, et qui est quelque chose d'enraciné dans l'intelligence sensorimotrice. Il construit son langage grâce à la forme des objets qu'il apprend, qu'elle soit réelle ou fictive. Noam Chomsky est assez proche de cette idée[4]. Pour lui, la *Grammaire Universelle* est une capacité

[1] "Un *schème* est la structure ou l'organisation des actions telles qu'elles se transfèrent ou se généralisent lors de la répétition de cette action en des circonstances semblables ou analogues"[24]

innée chez l'être humain, ce qui nous différencierait des autres êtres vivants. Pour Chomsky, la construction du langage se ferait à partir d'une série d'expériences grammaticales jusqu'à l'obtention de solides compétences à l'âge adulte. Il faut tout de même noter dans sa théorie que cela ne peut se faire sans la connaissance de certaines compétences spécifiques telles que les règles de dépendances structurelles.

Cela dit, il est encore difficile de savoir ce qui est appris, comment cela est appris, et ce qui est *inné*. C'est pour cela qu'il semble judicieux d'optimiser les réseaux de neurones de manière constructiviste pour cerner les élements nécéssaires à la construction et à la généralisation des connaissances.

Dans la prochaine partie, nous allons présenter les problématiques et les travaux liés aux réseaux de neurones, selon une perspective constructiviste.

3 Parcours des Approches Connexionistes, Étude de leurs Expressivités et de leurs Capacités Prédictives

3.1 Introduction

Comme on a pu le voir précédement, les RNN produisent de très bons résultats. Le réseau est capable d'apprendre pour prédire des suites de caractères, ce qui lui permet de générer du texte comme vu dans la partie 2.1. Mais aussi, on a pu constater que certaines règles lui échappaient, notamment lorsqu'il est question de donner un sens au texte généré.

Le réseau de neurone ne parvient donc pas à monter en abstraction sur les concepts de plus haut niveau, ce qui est un des principal verrou dans l'*Intelligence Artificielle*. Il se pourrait que le problème vienne de la taille du jeu d'apprentissage, qui empêcherait le réseau d'acquérir assez d'expérience. Mais le problème pourrait notamment venir de l'architecture du neurone ou encore du réseau. On peut se poser la question car, par exemple, chez l'être humain, les neurones sont beaucoup plus connectés que dans les modèles de réseaux de neurones artificiels que l'on peut rencontrer. Cet exemple est un détail physique qui pourrait être une explication du problème, mais il y a encore beaucoup d'autres différences. Comme expliqué par Sébastien Mazac[17], la représentation symbolique est un des points forts de l'humain, ce qui lui permet d'obtenir une bonne perception de la réalité. Alors que dans le cas de la recherche en *Intelligence Artificielle*, la difficulté, pour un système dit "intelligent", de monter en abstraction ses connaissances semble être la contrainte qui l'empêche d'accéder à une intelligence émergente. Une des limitations du réseau récurrent présentée provient de la perception imposée caractère par caractère comme dans l'exemple de la figure 2. On peut ainsi s'interroger sur l'impact de ce choix imposé, sachant que la perception serait le résultat de l'usage que l'on a, tel que proposé dans la sous partie 3.3. L'idée étant alors que le réseau n'apprenne pas une *suite de caractères* par cœur, mais puisse faire un apprentissage de plus haut niveau, comme par exemple autour des phonèmes du texte. D'ailleurs, une des premières connaissances acquises par le réseau lui permet de reécrire des phonèmes, voire des

non-mots, comme le montre l'exemple de Karpathy sur son blog [13] avec un apprentissage de 700 itérations : "*Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.*".

Mais si son apprentissage est trop long, difficile de dire si le concept de phonème existe toujours car il recopie les mots avec beaucoup trop d'exactitude. On rentre dans le cadre du "sur-apprentissage". Et le réseau perd sa capacité de généralisation[7]. Alors quels genres de régularités sont réellement apprises ? Comment faire pour que le réseau apprenne les régularités nécéssaires pour lui permettre de créer un texte cohérent ? Nous avons fait face à deux problématiques différentes.

Comme nous allons présenter et expérimenter ce travail avec des textes comme donnée en entrée, nous souhaitons apprendre des régularités, pas seulement sur les caractères, mais aussi sur les mots, car répertorier tous les phonèmes serait plus long et coûteux en entrée. Pour obtenir une perception supplémentaire dans le réseau, nous allons mettre en place un second niveau. Notre modèle hybride sera un NN qui apprendra des régularités de long terme sur les mots dans le second niveau, tout en gardant un réseau de premier niveau sur les caractères[2]. La représentation passée en entrée de ce second réseau ne pourra donc pas être basée sur les caractères comme proposé sur la figure 2. Le problème de la combinatoire est cependant toujours très présent. En effet, sachant qu'un texte peut contenir plusieurs milliers de mots différents, ce type de codage de vecteur d'entrée va grandement ralentir l'apprentissage du réseau. Cette problématique sera abordée dans la partie 3.2 suivante, et nous proposerons différents codages visant à réduire de façon construite la taille de la couche d'entrée tel que présenté dans la partie 4.

Pour cela, nous souhaitons apprendre des régularités à différents niveaux temporels, tels que les mots, ou même encore les phrases. Dans la continuité de la théorie de Piaget, il serait intéressant que le réseau puisse constater qu'un mot peut être remplacé par un autre, ou encore, que deux phrases signifient la même chose, mais sur un ton différent, car le contexte n'est pas le même. C'est pourquoi, nous étudierons l'apprentissage des dépendances de long terme dans la partie 3.4, et nous proposerons un modèle hybride à deux niveaux dans la partie 5.

3.2 Problématique de la Représentation

La représentation des concepts en entrée et en sortie doit permettre au réseau de reconnaître certaines similarités dans les mots, voire même dans le texte. Cependant, les caractères étant codés de manière discrète et unique, comment reconnaître le concept même de mot sachant qu'il est impossible au réseau de différencier un espace d'un caractère de l'alphabet ?

Comme on élabore l'apprentissage sur deux niveaux différents, les régularités que nous souhaîtons apprendre sont différentes de celles qui sont apprises par un

[2] Chaque niveau correspond à un réseau de neurones.

réseau de premier niveau. Les motifs sensorimoteurs doivent être différents. C'est pour cela que nous nous intéressons au mot. L'idéal aurait été peut-être d'abord de passer par la représentation des phonèmes, mais le concept de phonème est plus difficile à connaître *à priori*, contrairement aux mots qui sont simplement placés entre des caractères ne faisant pas partie de l'alphabet.

Dans l'exemple de la figure 2, le NN apprend sur les caractères, avec une représentation sous forme de vecteur de bit, avec un bit pour chaque caractère possible, et le caractère passé en entrée active le bit correspondant. Le nombre de caractères définit la taille de l'entrée du réseau de premier niveau . Il est alors difficile d'imaginer une représentation qui permettrait de distinguer les différents caractères, pour monter en abstraction sur des mots. Pour le réseau, la prédiction d'un espace est de la même nature que la prédiction de toute autre lettre, à la différence qu'ils ne suivent pas toujours les mêmes caractères.

Dans les NN, la taille de l'entrée, tout comme celle de la sortie, joue un rôle important dans l'apprentissage et dans la prédiction. Comme expliqué dans l'article[21], si la taille de la sortie est trop petite, la classification et la prédiction seront mauvaises. Néanmoins, plus le nombre de neurones en entrée et en sortie augmente, et plus le nombre de connexions entre l'entrée et la couche cachée, tout comme entre la couche cachée et la sortie, augmentent aussi. Le nombre de calcul et le temps d'apprentissage vont augmenter car il y aura plus de poids à mettre à jour à chaque pas d'apprentissage.

Sachant qu'un texte peut contenir plusieurs dizaines de milliers de mots, il faut donc trouver le nombre de neurones suffisants pour obtenir une représentation adéquate. Si l'on réduit la taille du vecteur d'entrée, et l'on discrétise la représentation des mots, on devrait pouvoir représenter un nombre plus important de mots[16], sans faire exploser le temps de calcul du réseau.

L'idée étant de pouvoir vaincre le problème de dimensionnalité[2] dans un réseau de neurones supplémentaires pour obtenir une *réalité* différente que celle du réseau de premier niveau. Beaucoup d'études et de travaux sont consacrés au problème de la dimensionnalité. Nous allons énumérer ici certaines des solutions envisageables. De plus, nous parlerons de la représentation des mots en entrée dans un réseau de neurones grâce à la technique du *Word Embedding*.

3.3 État de l'Art 1 : Représentation des Données dans les Réseaux de Neurones

Dans cette optique, comme le temps de calcul dépend de la taille du vecteur d'entrée, le choix de réduire sa taille parait être inévitable pour que le second niveau puisse apprendre en un temps relativement correct. L'état de l'art présenté ci-après s'intéresse donc à la manière de représenter les mots dans divers modèles, mais aussi dans la façon de réduire l'entrée.

La ligne directrice des travaux scientifiques répertoriés est principalement concentrée sur la compréhension du langage dans les réseaux de neurones.

La Notion de Syntaxe et de Grammaticalité. La syntaxe est la manière de mettre des mots les uns à la suite des autres pour obtenir bien plus que le sens

même de ces mots, faire emerger un énoncé. Un langage, dit "humain", puise sa force dans les régularités et les redondances qui le régisse. On peut constater que certains mots font partie de la même classe, car l'un peut remplacer l'autre sans fausser le sens de la phrase. Par exemple, dans la suite de terme "Le sandwich est", on peut remplacer "sandwich" par "casse-croûte". Cependant, cette règle de substitution est régie par la *grammaticalité* de ces termes. Une grammaire regroupe l'ensemble des règles qui permettent de dire si les mots, ou unités, sont correctement placés dans une phrase. Dans son livre[5], Noam Chomsky résume très simplement l'objectif de la grammaire comme dans la figure 3.

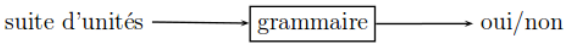

Figure 3. rôle d'une grammaire selon Chomsky

Pour ne pas confondre *interprétabilité* et *grammaticalité*, Chomsky donne un exemple souvent cité : "d'incolores idées vertes dorment furieusement". Cette phrase est grammaticalement correcte mais il est tout de même difficile d'y donner un sens. À l'inverse : "vous faire moi rigoler", est interprétable mais non grammaticale. Il serait donc intéressant que le réseau puisse obtenir une génération de texte grammaticalement correct et interprétable.

Word Embedding. Comme on a pu le voir précédemment, des travaux sont consacrés à l'analyse syntaxique et sémantique des mots dans une phrase donnée. Le *Word Embedding* fait son apparition dans le domaine du *Deep Learning*, en tant que méthode d'apprentissage automatique spécialisée dans l'apprentissage de mots. Cette méthode est utilisée dans de nombreux domaines de recherche tels que la traduction, la reconnaissance vocale, et même dans l'analyse des sentiments. La méthode du *Word Embedding* permet de donner une représentation différente et adaptée à ce travail, en codant les mots du dictionnaire de manière vectorielle.

Le vrai atout de cette conversion est d'obtenir un espace beaucoup plus petit. Deux mots, ayant des points communs, peuvent être codés de façon très rapprochés, mais jamais de manière parfaitement identique. La distance entre deux mots similaires sera donc réduite. On constate très vite que cette méthode répond très bien au problème de dimensionalité. On peut rencontrer ce genre de résultats :

$$W(\text{"chat"}) = (0.2, -0.4, 0.7, ...) \tag{1}$$
$$W(\text{"tapis"}) = (0.0, -0.6, 0.1, ...) \tag{2}$$

Comme on a pu le voir précédemment avec Chomsky, l'analyse sémantique dépend du contexte et du sens des mots. En plus de répondre au problème

de dimensionnalité, le simple fait de pouvoir coder de manière presque similaire deux mots très proches pouvant apparaître dans un même contexte, réduit considérablement la complexité de calcul pour savoir si une phrase est grammaticalement correcte. On peut observer un exemple de résultats des travaux sur le *Word Embedding* en annexe dans la section A.

Cartes Auto-Organisatrices. Les cartes auto organisatrices ont été développées par le statisticien Teuvo Kohonen dans son article *The self-organizing map*[14]. Leur principal atout est de pouvoir projeter l'espace des données dans un espace de plus faible dimension et de manière discrète.

La méthode utilisée provient de l'auto-organisation des régions du système nerveux. Ce modèle est donc bio-inspiré. En effet, une région du cerveau aura tendance à être excité par des stimuli similaires. Cette compétence ne provenant pas de l'inné, mais de l'acquis, Kohoben propose un modèle de carte topologique auto-adaptative. L'espace de données, appelé *carte*, est alors un espace discret, souvent de dimension inférieure à 3 Chaque neurone est représenté par un *vecteur réfèrent*. Un schéma détaillé est disponible dans l'annexe B.

La notion de voisinage est respectée. Chacun des neurones représente une zone de l'ensemble des entrées partageant certaines caractéristiques en commun. Les neurones conservent la forme topologique de l'ensemble de départ.

Comme la représentation des données se discrétise en un temps assez court, grâce au déplacement de l'ensemble des vecteurs de manière synchrone, cela permet d'obtenir un espace plus petit et plus facilement manipulable des données en entrée. Le résultat de la carte peut être interprété comme étant une montée en abstraction des connaissances *a priori*, et pourrait permettre, par exemple, de généraliser les vecteurs de mots obtenus dans le *Word Embedding*. De plus, ce modèle est dynamique et permet donc de s'adapter s'il acquiert de nouvelles connaissances. Mais encore faut-il avoir une représentation ayant du sens dans l'espace de départ, comme dans le *Word Embedding*, car les notions de distance et de voisinage sont seulement respectées, et non émergentes.

Restricted Boltzmann Machine et Autoencoder. Une Machine de Boltzmann Restreinte (RBM) est un réseau de neurones artificiels capable d'apprendre une distribution des probabilités sur l'ensemble de ses entrées. Crée par Paul Smolensky[28] en 1986, elle permet de réduire la dimension des données en entrée, et de les classifier. Elle peut être utilisée de manière supervisée ou non supervisée. Ce modèle est une variante de la Boltzmann machine, dont l'algorithme d'apprentissage a été créé par Ackley et al. en 1985[1], avec la différence que les neurones d'une même couche ne peuvent pas avoir de connexions entre elles.

Une fois les valeurs calculées en sortie, la sortie devient l'entrée. Les valeurs sont réinjectées pour être calculées dans l'autre sens, avec les mêmes valeurs de poids, seuls les biais dans la nouvelle sortie changent. L'image de la figure 4(a) montre très simplement cette deuxième étape.

Comme dans un réseau classique, les poids sont initialisés aléatoirement ce qui fait que la reconstruction de l'entrée va être très différente avec l'entrée

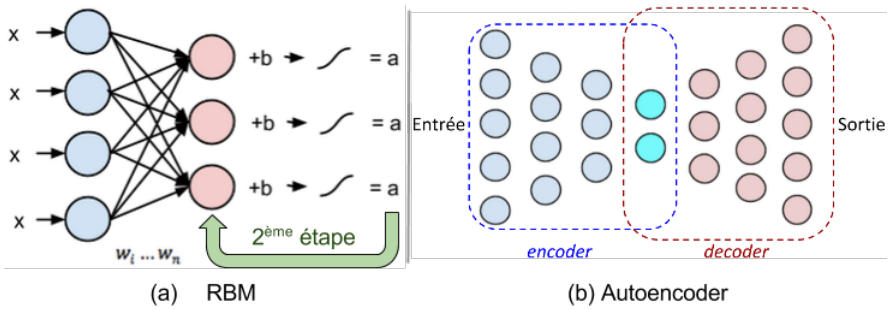

Figure 4. (a) Les valeurs de biais b changent d'une étape à l'autre, seuls les poids $w_i ... w_n$ restent inchangés. (b) L'architecture de l'*Autoencoder*, la partie centrale (en bleu clair) représente le vecteur compressé.

initiale. L'erreur va donc être rétropropagée dans le réseau et les deux étapes précédentes vont être ainsi répétées jusqu'à minimisation de l'erreur.

L'idée étant alors que la sortie dans la couche cachée va permettre de donner une représentation différente pour chaque entrée mais de taille réduite. Si l'on imagine que l'on insère en entrée une image, où chaque neurone de la couche visible représente un pixel de l'image, la sortie sera une représentation de tous ses pixels, mais de taille réduite. On obtiendra une représentation réduite en sortie qui sera le moyen le plus efficace de représenter l'image suivant les différences et les similitudes avec les autres images. Suivant les valeurs de sortie de la couche cachée, on peut en conclure que certains neurones seront les représentants d'un état particulier de l'image, et la sortie de dimension réduite pourra permettre une classification des différentes images.

Un réseau de neurones *Autoencoder* est un algorithme d'apprentissage non supervisé, utilisant la *backpropagation* pour que la sortie soit égale à l'entrée. Ce réseau de neurones non récurrents est très proche d'un réseau multicouche classique, utilisant environ une dizaine de couches. Il possède une couche d'entrée, une couche de sortie, et un certain nombre de couches cachées, comme on peut le voir dans la figure 4(b). La seule différence réside dans le fait que l'*Autoencoder* cherche à obtenir les mêmes valeurs en sortie qu'en entrée.

Un peu à la manière du RBM, les couches cachées se réduisent progressivement pour obtenir un codage de dimension plus petite que l'entrée d'origine. Néanmoins, cette réduction se fait sur plusieurs couches, et la réduction est obtenue progressivement (comme les RBM multi-couche). Le *decoder*, quant à lui, part de la sortie de l'*encoder* et possède une architecture symétrique à l'*encoder*. Idéalement, la sortie du *decoder* devrait être l'entrée de l'*encoder*. C'est pour cela qu'une backpropagation est ensuite réalisée pour minimiser l'erreur.

Ces deux modèles est très intéressante pour résoudre le problème de dimensionnalité. L'*Autoencoder* permet d'apporter une reconstruction assez efficace. Il encode mais aussi de classifie si l'on observe le vecteur dans la plus petite couche

cachée. Cependant, le RBM est d'autant plus efficace qu'il peut ajouter à cela une meilleure classification à partir de sa couche cachée.

Ces deux modèles sont d'ailleurs utilisés dans l'article de Hinton et al.[11] expliqué plus en détail en annexe, partie C.

Boostrapping dans l'Apprentissage Constructiviste. Le *bootstrap* fait référence au chargement à partir de rien, inspiré des aventures du Baron de Munchahausen, qui s'extirpa des marécages en tirant sur les languettes de ses chaussures. Cette métaphore explique donc que le démarrage doit se faire de façon autonome. Si le succès du démarrage est perçu par les sensorimoteurs, cela entraînera d'autres réactions. Il pourra faire le lien entre une action et une conséquence. Ces réactions vont pouvoir inciter le système à réaliser d'autres succès, ce qui lui permettra de continuer à agir pour apprendre.

Mais construire les liens entre action et conséquence n'est pas aussi évident. En effet, les conséquences dans un environnement ne sont pas toujours simples à cerner lorsque l'environnement est fortement dynamique.

Drescher propose une technique d'apprentissage constructiviste appelé le *schema learning*[6]. Le terme provient de celui de *schème* (comme expliqué dans la note 1)[24]. Piaget introduit ce terme pour l'analyse de l'apprentissage dans les premiers stades du développement. Son modèle est structuré de la manière suivante : *contexte* → *action* → *résultat*. Le contexte et le résultat sont les représentations de l'environnement obtenues par les interactions sensorimoteurs du système. Cela permet au système de ne pas seulement agir suivant ce qu'il perçoit, mais de comprendre les conséquences de ses actes en observant continuellement l'environnement.

Comme Mazac l'exprime[17], les difficultés pour amorcer l'apprentissage sans connaissances peuvent venir de différentes causes. Si une action est effectuée à différents moments, où l'environnement n'est jamais totalement identique, la prédiction des conséquences peut s'avérer difficile. De plus, une même conséquence peut être engendrée par différentes causes. Enfin, après une action, les conséquences seront peut-être parmi un ensemble d'évènements temporellement proches mais sans aucun lien avec l'action. C'est pour cela que la construction des liens entre actions et conséquences, même pour les résultats pertinents, sera très difficile à réaliser si le système ne possède pas de connaissances *a priori* portées sur le contexte.

Aussi, il est des cas où certaines conséquences peuvent être identifiées sans que le système n'ait pas conscience de ses origines. C'est là que l'on fait face au *bootstrapping problem* de Drescher[6].

Le *marginal attribution*, solution proposée par Drescher, peut potentiellement résoudre ce problème. Chaque schéma détecté sera évalué selon deux définitions : sa *pertinence* et sa *fiabilité*. La *pertinence* est mesurée sans tenir compte de la fréquence d'apparition lorsque l'action est effectuée, mais seulement par rapport au nombre d'apparition du résultat lorsque l'action a été déclenchée et lorsque l'action ne l'a pas été. Si un résultat est pertinent, sa fiabilité sera mesurée, en

relevant les contextes au moment où l'action est un succès, et ceux lorsque c'est un échec.

Dans notre cas, on cherche à ce que la représentation des entrées soit le plus conforme possible selon leurs appartenances aux différentes classes grammaticales. On comprend alors que le système doit agir de manière co-évolutive. D'une part, la représentation qui est apprise s'adapte au cours de l'apprentissage, et d'autre part, l'apprentissage s'adapte à la représentation. De cela, on pourra obtenir des similarités dans le codage entre les mots.

Résume de l'État de l'Art 1. Ces différentes approches répondent totalement à notre problématique. Le *Word Embedding* permet tout d'abord de créer des liens logiques entre les mots. La distance entre les mots dans l'espace permet de les classifier, de créer des corrélations cohérentes, et permet notamment de mesurer si la phrase est grammaticalement correcte. Les trois autres approches complètent ce travail en permettant de réduire davantage, si nécessaire, la dimension des données et apporter une classification supplémentaire de plus haut niveau. Néanmoins, ces approches restent statiques, et sont incapables de s'adapter correctement si de nouvelles entrées surviennent après un premier apprentissage, tout comme pour les réseaux de neurones. Cette problématique est d'ailleurs abordée plus en détail dans la partie 3.4 suivante.

La montée en abstraction sur le sens des mots, de leur catégorisation, peut se faire à travers l'utilisation d'une représentation vectorielle cohérente des mots puis d'une compression sur plusieurs niveaux des données. Cependant, obtenir un codage sensé des mots pour obtenir des corrélations spatiales, comme dans le *Word Embedding*, n'est pas aussi simple, car les mots sont souvent mal modélisés, comme expliqué dans l'article[16], et appuyé par Hawkins dans [10].

On s'inspire de ces travaux pour proposer dans la partie 4 une représentation dynamique pour coder les mots sur le réseau de second niveau.

3.4 Problématique de l'Apprentissage des Régularités à Long Terme

Comme expliqué dans cet article[3], l'apprentissage par un algorithme de gradient devient de plus en plus difficile lorsque les dépendances à apprendre s'étendent sur le long terme. C'est le problème du *Gradient Evanescent*. Il existe certes d'autres méthodes mais qui peuvent par exemple s'avérer coûteuses en temps de calcul. Sur le blog de Karpathy[13], on constate que lorsque le réseau apprend sur du code source C, il apprend très bien à indenter, à ouvrir et fermer entre crochets les fonctions qu'il génère, tout comme il peut le faire avec des guillemets dans un texte, exemple également illustré sur l'article de son blog. On peut remarquer que le réseau n'est pas capable de déclarer correctement une variable, ou encore, qu'il retourne un argument du mauvais type à la fin d'une fonction.

Cela est toujours dû au fait que le réseau n'est pas capable d'apprendre les dépendances de long terme. Un exemple simple sur la génération de phrase dans un contexte inconnu est disponible en annexe à la section D.

Un autre des grands problèmes rencontrés dans le *Machine Learning* est le *Catastrophic Forgetting*[7]. Un réseau qui apprend une nouvelle tâche aura tendance à oublier la première tâche apprise. Il se passe la même chose dans les Machine à vecteurs de support. Il est assez simple de mesurer le *Catastrophic Forgetting*, par exemple, avec la formule présentée dans [20]. Si l'erreur calculée, distance entre la sortie à l'instant t par le réseau et la sortie attendue au même instant, est minime, cela est dû aux similitudes très fortes entre les deux tâches apprises l'une après l'autre.

Par exemple, si l'on introduit un nouveau texte, avec un style tout à fait différent, par exemple en *switchant* d'une pièce de théâtre à une fable, on peut très vite faire le constat que le réseau saura apprendre le nouveau texte seulement en effaçant un grand nombre de concepts appris dans le texte précédent.

La question de la généralisation est au centre de notre problématique. Si le réseau vient à lire un nouveau texte, il n'aura rien retenu d'assez intéressant qui lui permettrait de ne pas avoir à tout réapprendre, surtout si le vocabulaire est trop différent. C'est le problème de l'ancrage des symboles[17]. Ce qui limite le réseau à ne pas pouvoir apprendre de nouveaux concepts, et lui permettre de généraliser ses connaissances en corrélant ses nouvelles connaissances avec celles déjà apprises.

Du point vu biologique, il semble normal de dire que certains facteurs sont innés chez l'être humain pour l'aider dans son développement cognitif. De cela, on pourrait dire, par exemple, que deux sons différents vont être interprétés différemment, ce qui est déjà une forme d'intelligence *a priori*[23]. Cela va permettre à l'individu, sachant faire la différence entre deux sons, une explosion ou un rire, de distinguer par exemple le danger et une émotion positive. Cet amorçage va permettre par la suite d'apprendre à corréler des sons similaires, car il est tout de même très rare qu'une explosion ait toujours la même fréquence et la même amplitude.

Le problème que l'on rencontre vient de l'incapacité de notre réseau à apprendre des concepts de plus haut niveau, comme généraliser le son d'une explosion, qu'elle soit "sourde" ou "bruyante". L'être humain, à un âge avancé, sait faire la distinction entre ces deux sons mais est aussi capable de les ressentir de la même façon, de telle sorte que ces deux bruits évoqueront dans son esprit qu'il y a possiblement un danger.

Dans la partie 3.5 qui suit, nous allons présenter les principaux travaux qui tentent de remédier à la problématique du *Catastrophic Forgetting* et du *Gradient Evanescent*.

3.5 Etat de l'Art 2 : l'Apprentissage des Dépendances de Long Terme

Les techniques présentées dans cet état de l'art s'intéressent à résoudre, de différentes manières, la problématique du *Gradent Evanescent*, et du *Catastrophic Forgetting*. L'objectif est de permettre aux réseaux de neurones de converger plus rapidement, et de manière plus intelligente, vers un état d'apprentissage plus efficace, et d'apprendre d'avantage sans avoir besoin de cumuler plusieurs

parallèlisations de machine, autrement dit, sans avoir recours à du matériel toujours plus puissant. Cet état de l'art aborde différentes architectures permettant de minimiser le risque de sur-apprentissage, d'effacement du gradient, et d'apprendre d'avantages sur les expériences du système.

LSTM. Le Long Short-Term Memory est un type de réseau de neurones récurrents, mais avec une architecture neuronale un peu plus complexe, proposé par Hochreiter et Schmidhuber[12] en 1997. Le LSTM a été conçu particulièrement pour répondre au problème des dépendances de long terme.

C'est donc la structure même de la cellule qui permet de surpasser ce problème. Sans rentrer trop dans les détails, on peut observer, sur la figure 5, que les cellules n'ont pas seulement une information récursive de l'état précédent, mais deux. Ces deux informations seront alors complémentaires pour la prédiction de la sortie.

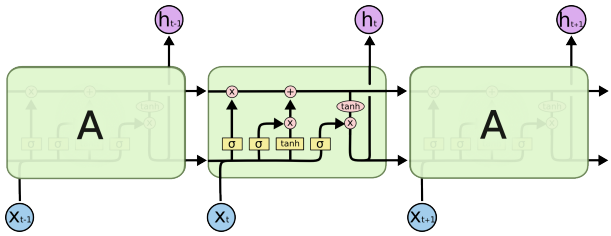

Figure 5. Le LSTM avec un des trois neurones vu en coupe[22]

Les résultats montrent qu'il est capable de surpasser le RNN classique dans de nombreuses tâches, d'autant plus qu'il apprend plus vite. Cette nouvelle architecture de réseau de neurone récurrent utilise un algorithme d'apprentissage approprié, mais toujours basé sur la descente du gradient.

La clef de voûte du bon fonctionnement du réseau provient de l'ajout d'une information récurrente qui permet d'appuyer la prédiction. Cet état récurrent, appelé *état cellule*, permet d'obtenir une information sur le plus long terme. Les modifications qui lui sont apportées au cours du temps sont très faibles, et linéaires.

Dans le cas d'une prédiction de mot dans un texte, cette information voyageant à travers le temps pourrait, par exemple, transporter le type du sujet (sexe, pluriel, ...) et être modifiée par l'entrée seulement lorsque le sujet de la phrase change. Elle peut aussi transporter de nouvelles informations au besoin.

Il y a donc une information contenant le contexte de court terme et une information contenant celui du long terme utilisé dans les neurones de manières récursives.

Pour résoudre le manque d'information sur les dépendances de long terme, au lieu de modifier l'état de récurrence classique du RNN, les auteurs ont préféré

rajouter un état supplémentaire de récurrence. Au cours du temps, cet état subira que de très légères modificationss, et plus rarement, des modifications plus importantes si le contexte change fortement.

Neural Turing Machine. Crée en 2014 par Google DeepMind à Londres[8], le Neural Turing Machine est un réseau de neurones possédant une mémoire. Tout comme un NN classique, il possède une couche d'entrée et une couche de sortie connectées avec l'extérieur.

Les vraies prouesses d'un tel réseau résident dans la capacité à apprendre et surtout à mémoriser ce qu'il a appris. Il est en interaction, via son contrôleur, avec sa mémoire, ce qui lui permettra d'obtenir d'avantage d'informations quant au contexte courant. NTM et LSTM peuvent aussi être complémentaires, car ils ne regroupent pas les informations de la même façon, schéma disponible à l'annexe E.

Contrairement au LSTM qui possède une représentation distribuée de l'information, et effectue des changements globaux sur l'ensemble de la cellule à chaque étape. Le NTM encourage les changements locaux dans la mémoire, et lui permettra d'obtenir des combinaisons de vecteur pour créer correctement le contexte approprié. Cette mémoire n'est qu'une matrice, mais qui permet de transporter des informations sur le plus long terme.

Cette mémoire est donc un moyen inné de mieux préserver les connaissances acquises par la machine. Il semblerait, vu son utilisation, que l'accès en lecture/écriture de la mémoire serait en quelque sorte similaire a celle d'une table de hâchage, car les vecteurs sont stockés dans des emplacements de mémoire contrôlés par priorité et ensuite lus par ordre de priorité pour donner une liste triée.

Hierarchical Temporal Memory. L'architecture du modèle de Hawkins est très différente de celle des réseaux de neurones classiques. Elle cherche principalement à imiter le néo-cortex[10]. La hiérarchisation de sa structure est à la fois dans l'espace et dans le temps.

Elle se dessine d'abord en colonne activé/activable ou non. La hiérarchie peut être composée de plusieurs régions. Chaque région comprend plusieurs colonnes. Chaque colonne comprend un nombre identique de neurones, et les colonnes peuvent être situées sur une ou plusieurs dimensions. Ce dernier point n'est qu'un détail pour la suite. Une image d'un modèle HTM est disponible à l'annexe F

Grâce à la hiérarchisation du modèle, et à la représentation partagée, ce modèle a pour objectif de généraliser les données et de rendre le stockage efficace. Pour cela, les causes de niveaux inférieurs sont partagées avec les niveaux supérieurs. Cela devrait permettre de réduire le temps d'apprentissage des nouvelles causes et faciliter HTM pour généraliser les comportements déjà appris, en les corrélant ou en les dissociant si les nouvelles causes sont très différentes.

Dans le papier[9], Numenta explique que le dans le cas de la vision, au plus bas de la hiérarchie, notre cerveau stocke les informations de bas niveau tel que les coins ou les arêtes d'une table. Et c'est en montant plus haut dans la

hiérarchie que les concepts fusionnent pour donner une vision plus complexe de l'objet observé. L'avantage est qu'il n'est pas nécéssaire pour notre cerveau d'avoir à réapprendre les concepts de bas niveaux pour apprendre de nouveaux objets dans les niveaux supérieurs de la hiérarchie.

La hiérarchie de HTM permet donc de réduire le temps d'apprentissage, la taille de la mémoire, et surtout, de généraliser les concepts qui ont certains points communs sur les niveaux plus bas, car les patterns appris à chaque niveau de la hiérarchie sont réutilisés lorsqu'ils sont combinés à de plus haut niveau.

Résumé de l'État de l'Art 2. La plupart de ces modèles proposent des solutions différentes pour pouvoir stocker efficacement les dépendances de long terme, et réduire le temps d'apprentissage. LSTM modifie l'architecture du neurone, NTM ajoute une mémoire accessible via un contrôleur, et HTM propose de hiérarchiser de façon bio-inspirée différents réseaux de neurones dont la structure est spécifique à HTM.

Ces solutions permettent de conclure que les dépendances doivent être traîtées de manières différentes suivant leur origines et leurs impacts. Elles peuvent provenir de la même source, mais leurs implications après traitement ne seront pas semblables et pourront être complémentaires.

4 Première Contribution : Représentation Dynamique

4.1 Introduction

Nous allons présenter trois modèles différents pour tenter de répondre à la problématique de la dimensionnalité en entrée du réseau abordé dans la section 3.2.

L'objectif est de faire évoluer la représentation en entrée du NN de sorte à ce que cette représentation soit adaptée à l'apprentissage de corrélation entre les mots. C'est pourquoi nous allons réduire la taille de la couche d'entrée, en proposant un codage dynamique et adaptatif pour la représentation des mots.

Premièrement, nous expliquons dans la partie 4.2 le fonctionnement du réseau de neurones récurrents que nous utilisons, sans se soucier du codage de l'entrée. Ensuite dans la partie 4.3, nous allons proposer trois codages différents. Le premier sera représenté par l'anagramme des mots, ce qui limitera la taille de l'entrée du second niveau à celle du premier niveau. Le second possèdera les coordonnées des mots, qui seront placés dans une matrice à deux dimensions, ce qui réduira le nombre de bits à 1 en entrée. Et le dernier codage n'aura qu'un seul bit à 1, qui représente la case d'une table de hachage dans lequel se trouve le mot, sachant que les mots seront classifiés dans la table au cours de l'apprentissage selon leurs similitudes. Ces trois modèles ont été conçus et testés suivant cet ordre chronologique. Des résultats de classifications seront présentés dans la partie 5.3.

4.2 Fonctionnement du Réseau Classique

L'architecture du réseau de neurones récurrents que nous utilisons est simple. Il possède une couche en entrée, une couche en sortie, et une couche cachée. Tous les neurones de la couche d'entrée sont reliés à tous les neurones de la couche cachée, qui eux-même sont reliés à tous les neurones de la couche de sortie. Le réseau de neurones récurrents fonctionne en deux temps, une phase d'apprentissage et une phase de génération.

Tout d'abord, il passe par une phase d'apprentissage supervisée qui va lui permettre d'ajuster la valeurs de ses poids, suivant la sortie attendue. Chaque entrée est un mot codé sous la forme d'un vecteur de bits par le réseau. À partir d'une entrée, représentant un mot du texte, le réseau prédit un vecteur de sortie qui sera comparé au vecteur représentant le mot suivant dans le texte. Le réseau va alors calculer la distance entre les bits du vecteur représentant la prédiction faite par le réseau et les bits du vecteurs représentant la prédiction attendue. La distance entre chaque bit est considérée comme étant l'erreur en sortie du réseau. La technique de *backpropagation* va utiliser l'erreur calculée pour mettre à jour la valeur des poids. Puis la sortie attendue est réinjectée dans l'entrée du réseau, pour ensuite effectuer la même démarche que pour l'entrée précédente.

Dans un second temps, après la phase d'apprentissage, on peut demander au réseau de générer un texte en lui appliquant en entrée un ou plusieurs mots. Après cela, la sortie prédite à l'instant t sera alors l'entrée à l'instant $t + 1$.

Dans la figure 2 de la partie 2.1, la sortie est codée avec des nombres réels. En réalité, notre RNN fait la même chose, sauf qu'il applique la formule du *Softmax* avant de choisir le mot qui sera généré. On obtient : $\begin{bmatrix} y_1 & y_2 & ... & y_i \end{bmatrix} \rightarrow$ *Softmax* $\rightarrow \begin{pmatrix} p_1 & p_2 & ... & p_i \end{pmatrix}$. Le *Softmax* de l'indice$_i$ du vecteur de sortie est calculé de cette façon : $p_{indice_i} = e^{y_i} / \sum_{j=0}^{n} e^{y_j}$, avec n la taille du vecteur de sortie. Par exemple, la sortie $\begin{bmatrix} 3.0 & 3.0 & 3.0 & 3.0 \end{bmatrix}$ calculera comme valeur de prédiction $\begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$, et tous les indices auront la même probabilité d'être tiré au sort. Alors que si la sortie génère $\begin{bmatrix} 0.3 & 1.0 & 0.3 & 0.3 \end{bmatrix} \rightarrow$ *Softmax* $\rightarrow \begin{pmatrix} 0.2 & 0.4 & 0.2 & 0.2 \end{pmatrix}$, l'indice à la deuxième position aura deux fois plus de chance d'être tiré au sort. On vient d'aborder le cas le plus général, lorsque l'on cherche la prédiction pour chaque indice du vecteur de sortie, et que chaque indice représente un mot distinct. Bien entendu, les modèles que nous allons proposer ne sont pas codés de cette manière. Néanmoins, dans tous les modèles que nous allons proposer, le codage du vecteur de sortie attendu sera le même que celui de l'entrée.

Le *Softmax* est utilisé pour l'apprentissage supervisé, lors du calcul de l'erreur, et pour la génération de texte, pour choisir aléatoirement le mot à générer. L'aléatoire permet au réseau de varier la génération des mots.

Dans ce mémoire, la notation mathématique est différente pour les valeurs précédant le *Softmax*, et les valeurs de prédictions qui suivent. Elles seront respectivement notées $\begin{bmatrix} y_1 & y_2 & ... & y_i \end{bmatrix}$ et $\begin{pmatrix} p_1 & p_2 & ... & p_i \end{pmatrix}$. Cette notation permettra de comprendre dans les exemples qui suivront, de quel type de valeurs nous parlerons.

4.3 Modèle

Nous présentons ici différents modèles pour la couche d'entrée du réseau chargé d'apprendre des régularités sur les mots.

Entrée Anagramme. Comme le codage en entrée doit être plus petit, nous proposons de coder l'entrée avec les anagrammes des mots, en l'accompagnant d'un *dictionnaire* qui lui permettra de retrouver les mots correspondant à l'anagramme si le réseau de second niveau vient à prédire seul. Dans le modèle hybride, le dictionnaire n'est pas nécéssaire.

La taille du vecteur d'entrée/sortie sera alors du nombre de caractères dans le texte compris dans l'alphabet, car il ne s'occupe pas des caractères spéciaux[3].

Ce qui est intéressant, dans cette approche, c'est que l'apprentissage se fait sur l'ensemble des lettres de l'alphabet, et non pas des mots. C'est-à-dire, en entrée, nous aurons plusieurs bits à 1 (si la longueur du mot est strictement supérieur à 1). Par exemple, si nous avons six lettres dans notre vocabulaire tel que $input = \big(a\ e\ l\ m\ t\big)$, le pronom "elle" sera codé $input_{elle} = \big(0.0\ 1.0\ 1.0\ 0.0\ 0.0\big)$. Comme on peut le voir, les lettres du codage ne sont pas organisées de manière à pouvoir retrouver le mot, tout ce que le codage apporte, c'est qu'il y a au moins une fois la lettre e et au moins une fois la lettre l. Le nombre d'occurrence des lettres n'apparaît donc pas.

Cependant, la taille de l'entrée est encore plus petite que celle du premier niveau car on ignore les caractères spéciaux. De plus, si le second niveau doit prédire le pronom "elle", avec une prédiction parfaite, il générera en sortie $output = \big(0\ 0.5\ 0.5\ 0\ 0\big)$ et ce résultat pourra être envoyé directement au premier niveau pour lui dire d'appuyer fortement sa prédiction sur ces deux caractères. Le premier niveau saurait que le second niveau lui conseille d'utiliser ces deux caractères dans ses prochaines générations de caractères.

Dans cet optique, le second niveau n'a pas besoin d'apprendre les mots, mais juste les éléments qui le compose, dans notre cas les caractères. Cela facilite les échanges entre les deux niveaux car ils font tous les deux des prédictions sur les caractères.

Néanmoins, on constate que la prédiction de plusieurs bits est difficile. En effet, la méthode de calcul du *Softmax* va tenter de départager le plus possible tous les bits. Ensuite, nous obtenons rarement une égalité parfaite entre les bits qui aurait dû être prédit. C'est pourquoi nous nous sommes orientés vers une matrice à deux dimensions pour coder les mots.

Nous proposons alors une nouvelle solution, qui permettra d'activer moins de bits dans le vecteur d'entrée et de sortie.

Entrée Matrice 2D. Pour permettre au réseau d'avoir moins de bit à prédire au même instant à la sortie du réseau, on élabore une matrice de deux dimensions

[3] Ce que nous définissons comme caractères spéciaux dans un texte sont les espaces, les retours à la ligne, la ponctuation, les guillemets, les apostrophes, etc... Tout les caractères ne faisant pas partis de l'alphabet.

qui stockera les mots. Un mot sera alors composé de 2 bits actif, un représentant la ligne de la matrice, et l'autre la colonne.

En somme, si le texte contient, par exemple 2000 mots uniques, la matrice sera de taille $45 * 45$. Le vecteur d'entrée/sortie pour coder la ligne et la colonne correspondant sera de taille $45 + 45 = 90$. La taille de la matrice dépend de la racine carrée du nombre de mots, arrondit à l'entier supérieur. Le codage est alors $input_{mot_p} = \big(Ligne_1\ Ligne_2\ ...\ Ligne_n\ Col_1\ Col_2\ ...\ Col_n\big)$. On peut imaginer, dans le cas où l'on aurait seulement quatre mots, une organisation des mots dans la matrice telle que :

$$\begin{array}{c} \\ Ligne_1 \\ Ligne_2 \end{array} \begin{array}{c} \begin{array}{cc} Col_1 & Col_2 \end{array} \\ \begin{pmatrix} ciel & le \\ bleu & est \end{pmatrix} \end{array} \tag{3}$$

Pour retrouver un mot de la table, on effectue alors deux *Softmax*, sur chaque moitié du vecteur de sortie. La première opération est effectuée sur les bits représentant les lignes, et la deuxième sur les bits représentant les colonnes. Cela permet d'obtenir une valeur majorant chacune des dimensions de la matrice. Ces deux majorants sont les coordonnées du mot à prédire.

Par exemple, si en sortie du réseau on obtient $output = \big[3; -2; -1; -4\big]$, L'application du *Softmax* sur les deux premières valeurs donnera une prédiction pour la ligne de $\big(0.98\ 0.02\big)$, et pour les deux dernières valeurs, une prédiction pour la colonne de $\big(0.95\ 0.05\big)$. Le mot prédit dans la figure 3 est "ciel".

Malheureusement, la prédiction est souvent mauvaise, car si la mauvaise colonne est prédite, ou la mauvaise ligne, cela amène le réseau à prédire un tout autre mot, qui n'a aucune corrélation avec son voisin (voisin car ils ont tout de même un bit correct en commun). Le contexte changera fortement par l'influence de la dernière prédiction, ce qui pourra momentanément perturber le réseau dans les prochaines prédictions.

C'est pour cela que nous envisageons dans le modèle qui suit, de mettre en place un classifieur. Cela permettra de regrouper les mots possèdant des corrélations.

Entrée Table de Hachage avec Classifieur. Comme la prédiction a été constatée assez mauvaise avec la matrice 2D, nous avons préféré concevoir une table de hachage. Le principe est quasiment le même que pour la matrice 2D, à la différence que l'on code seulement la ligne, que l'on appellera case. Le nombre de cases se calcule comme le nombre de ligne dans le modèle précédent. Chaque case de la table pourra contenir plusieurs mots. Le réseau va prédire en sortie une case, donc un ensemble possible de mots contenus dans cette case. La couche d'entrée/sortie est de la taille du nombre de cases de la table. Pour activer un mot, on active l'entrée correspondant à la case contenant ce mot. Donc un seul bit à prédire en sortie du réseau.

En effet, comme le second niveau est là pour assister le premier niveau, et non pour faire des prédictions seul, il est envisageable d'envoyer une liste de mots, qui pourrait appuyer les prédictions du premier niveau.

De plus, le fait que la matrice 2D précédente pouvait faire de fausses prédictions à cause de la non-existence de corrélation entre les mots voisins, il semblait judicieux de pouvoir regrouper dans les cases les mots appartenant à la même classe. À l'initialisation, les cases acceptent un nombre limité de mots, égal au nombre de cases précédemment calculé, et les mots sont répartis de manière uniforme dans les cases. Chaque case est remplie l'une après l'autre, la dernière case aura alors potentiellement moins de mots que les autres. Le classifieur va alors, durant l'apprentissage, déplacer les mots selon les taux de prédictions qui auront été faits, pour que les mots pssèdant des corrélations se retrouvent dans la même case. Si les mots sont correctement classés, chaque case possèdera des mots qui pouront être placés dans le même contexte. Ce sera au premier niveau de les départager.

La classification va s'effectuer tout au long de l'apprentissage. Dans un premier temps, lorsque le réseau n'a pas encore appris, les cases auront, à peu de choses près, le même taux de prédictions.

Chaque mot possède un vecteur, que l'on nommera *vecteur de classe*, qui contient une valeur pour chaque case. Les valeurs des vecteurs sont toute initialisées à zéro, sauf l'indice de la case où se trouve le mot. Par exemple, en partant du principe que la première valeur représente la case 1, si un mot se trouve dans la case 2, son *vecteur de classe* aura cette apparence : $\{0.0\ 1.0\ 0.0\ ...\ 0.0\}$

Les mots vont pouvoir se déplacer de case en case suivant les prédictions attribuées durant l'apprentissage. Si le mot attendu en sortie se trouve dans la case 2, c'est l'indice 2 du vecteur de sortie qui doit être à 1. Si la prédiction maximum se trouve bien sur le 2ème bit du vecteur prédit en sortie, on ne fait rien. Par contre, si c'est l'indice 3 qui a été prédit, alors la valeur de prédiction sera ajoutée à l'indice 3 de son *vecteur de classe*. Au bout d'un certain nombre d'itérations, on déplace les mots suivant les valeurs contenues dans leur *vecteur de classe*. L'indice le plus élevé dans chaque *vecteur de classe* indiquera la prochaine case occupée par chacun des mots.

Par exemple, si nous avons trois cases, que le mot "test" est initialement placé dans la case 1, si la prédiction est parfaite, elle devrait donner $output = (1\ 0\ 0)$. On peut remarquer que sur les premières itérations, le réseau n'a aucune connaissance du texte, et génère de faibles prédictions, par exemple $output = (0.33\ 0.34\ 0.32)$.

À partir de ce résultat, le classifieur va encourager les cases où le mot n'est pas présent, dans notre exemple case 2 et 3. Il peut aussi pénaliser une case si la prédiction est trop faible, encourager d'avantage les cases qui ont peu d'éléments, et inversement pour les cases qui en possèdent beaucoup. L'algorithme suivant explique comment s'effectue les déplacements pour un mot donné, avec $prediction$ un vecteur de taille $nombre_{case}$ contenant, pour chacune des cases, la somme des prédictions qui ont été faites pour ce mot durant le dernier

apprentissage (si la case 1 n'a jamais été prédite, l'élément 1, dans le vecteur *prediction*, sera égal à zéro) :

Algorithme 1 MODIFICATION DES VALEURS DE CLASSIFICATION D'UN MOT

Entrée(s) $taux_{seuil}$ un paramètre avec $taux_{seuil} < 1.0$,
le vecteur *prediction*,
$occurencesMots_{case_i}$, le nombre d'occurrences des mots contenu dans la case i,
$nombreMots_{dictionnaire}$, le nombre de mots total dans le texte
pour tout i, indice des éléments du vecteur *prediction* **faire**
$pred_i = prediction_i \div \sum_{j=0}^{n} prediction_j$ avec n le nombre de cases
fin du pour
pour tout $indice_n$, élément du *vecteur de classe* à l'indice n **faire**
si le mot ne se trouve pas dans la case n **alors**
$seuil = occurencesMots_{case_i} * taux_{seuil} \div nombreMots_{dictionnaire}$
si $pred_i > seuil$ **alors**
$indice_i += prediction * ratio$
sinon
$indice_i -= (1.0 - prediction) * ratio$
fin du si
fin du si
fin du pour

Le *seuil* dépend de la somme du nombre d'occurrences des mots compris dans une case. Sinon les cases contenant des mots qui apparaissent souvent risque d'influencer les autres mots des cases qui ont trop peu d'occurrences dans le texte.

4.4 Résultats

Nous allons présenter quelques résultats sur la classification du dernier modèle décrit. Ces résultats montreront l'efficacité du classifieur, sur un exemple simple. Cet exemple démarre avec des cases parfaitement équilibrées, c'est-à-dire que toutes les cases possèdent la même proportion de chaque classe de mot. Cela permet de ne pas avantager dès le départ le classifieur. Pour mieux comprendre, voici la table de départ :

$$Case_1 \ (11\ 12\ 13\ 21\ 22\ 23) \tag{4}$$
$$Case_2 \ (31\ 32\ 33\ 41\ 42\ 43) \tag{5}$$
$$Case_3 \ (51\ 52\ 53\ 61\ 62\ 63) \tag{6}$$
$$Case_4 \ (71\ 72\ 73\ 81\ 82\ 83) \tag{7}$$
$$Case_5 \quad (91\ 92\ 93) \tag{8}$$

Nous utiliserons des chiffres plutôt que des mots, et nous utiliserons la notation suivante : en regardant le dernier chiffre, on connaît la classe du "mot". Par exemple, 11 et 93 appartiennent respectivement à la classe 1 et 3.

Variété et Généralisation. Les textes sont générés via des règles d'expressions régulières. Pour mieux comprendre, nous présentons les expressions régulières suivantes :

$$[1-9]1 \Rightarrow [1-9]2 \tag{9}$$
$$[1-9]2 \Rightarrow [1-9]3 \tag{10}$$
$$[1-9]3 \Rightarrow [1-9]1 \tag{11}$$

La règle (9) exprime qu'après un terme de deux chiffres terminant par 1, on peut placer n'importe quel terme de deux chiffres terminant par 2. Par exemple "31 22" respecte cette règle. L'en-tête $[1-9]$ présente dans les trois règles (9, 10, 11), signifie que le terme peut commencer par n'importe quel chiffre entre 1 et 9.

Cependant, un texte, que ce soit dans la langue française ou autre, ne respecte pas seulement une seule règle ! Bien au contraire, les règles de syntaxe et de grammaire sont nombreuses. De plus, elles entraînent bien souvent des irrégularités, que l'on nommera ici *variété*.

Dans notre cas, on peut constater que nos trois règles font preuve de généralités. En effet, dans la règle (10), les termes terminant par la lettre 2 sont toujours suivis par un terme terminant par la lettre 3. Ces règles permettent alors au classifieur d'apprendre à classer les mots dans les cases de façon à les regrouper suivant leur terminaison.

Ceci permet, par la suite, de faire appel à une case regroupant une classe de termes (par exemple, les termes terminant par 1). Les cases ne sont plus seulement présentes pour permettre d'alléger la taille de l'entrée, elles permettent aussi de choisir la bonne catégorie de termes suivant un contexte donné.

En apportant, ce que l'on appelle de la *variété*, on va pouvoir tester la robustesse de notre classifieur. En effet, si l'on rajoute la règle :

$$[1-9]1 \text{ ET } [1-2]2 \Rightarrow [1-2]2 \tag{12}$$

cela va permettre des générer des phrases du type 11 12 12 13 qui n'appartiendront plus à la généralité des trois premières règles. La question que l'on se pose n'est pas seulement de savoir à quel point le classifieur risque d'être perturbé par cette nouvelle règle, mais plutôt à quelle fréquence d'apparition cette nouvelle règle (12) va passer de la *varîété* à la *généralité*.

En modulant la probabilité de tirer au sort la règle (12) plutôt que la règle (10), on va pouvoir constater comment sera réalisée la classification des éléments 12 et 22.

On décide de commencer par une probabilité de 1/10 et d'augmenter progressivement sur différents réseaux, en analysant les types d'éléments contenus avec les éléments 12 et 22 au cours de la phase d'apprentissage. Le graphique de

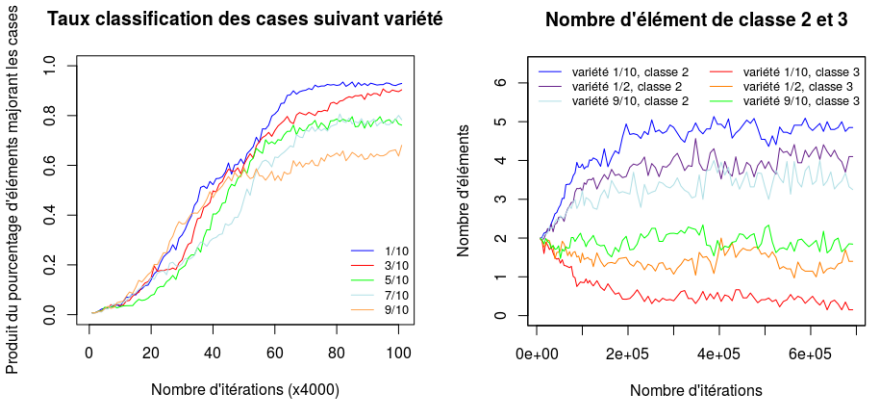

Figure 6. À gauche : taux de classification moyen des cases. À droite : moyenne du nombre d'éléments de la classe 2 et de la classe 3 contenus dans les cases où se trouvent les éléments 12 et 22 suivant la valeur de variété.

droite de la figure 6 nous montre l'évolution du nombre d'élements de la classe 2 et 3 présent dans les cases contenant les éléments 12 et 22. Ces moyennes sont calculées sur différents apprentissages.

Dans la figure 6 dans le graphique de gauche, on calcule le taux de présence en moyenne de la classe majorante, calculé sur différents apprentissages. La classe majorante est la classe qui possède le plus d'élements dans une case donnée, on notera l'occurence de ses éléments $occ_{majorant}$. Le calcul du taux de classification des cases dans la figure 6 à un instant t se fait de cette manière :

$$\prod_{i=0}^{n} occ_{majorant} \div nombreElements_{case_i}, \text{ avec } n \text{ le nombre de cases.}$$

On constate lorsque la variété augmente, la classification ralentie. En effet, avec un taux inférieur de 1/2 de variétés, les cases obtiennent un taux de classification supérieur à 80%, contrairement aux taux de variétés strictements inférieurs à 1/2. Cela entraîne certaines cases à contenir deux classes différentes. Comme dans notre exemple, les éléments 12 et 22 auront plus de mal à se détacher des élements de la classe 3.

5 Contribution 2 : Modèle hybride

5.1 Positionnement

Comme les connaissances sur les données sont parfois temporellement différentes, leur stockage et leur lecture doit être fait différemment. C'est pourquoi nous proposons dans cette partie un réseau qui soit capable d'apprendre avec deux temporalité différentes, pour lui permettre d'avoir deux visions différentes. Cela dit,

ses connaissances devront être complémentaires pour apporter une plus grande cohérence dans la prédiction en sortie.

Notre objectif est de pouvoir se consacrer aux mêmes données (par exemple, même texte) mais de façon asynchrone. Pour cela, il sera nécéssaire d'implémenter un deuxième réseau. On ne s'intéresse plus à l'organisation des caractères, mais plutôt à l'organisation des mots.

Tout comme HTM[10], on souhaite utiliser le concept de hiérarchisation des connaissances pour améliorer l'efficacité du réseau.

5.2 Représentation du Modèle sous forme d'*Ellipse*

Le premier niveau apprenant les régularités de court terme, le second niveau doit être capable d'apprendre des régularités de plus long terme. L'objectif étant de faire collaborer les deux niveaux. En ne connaissant seulement que le concept de mot, le réseau va apprendre à générer des phrases, à partir des mots qu'il a appris.

Comme le premier réseau sait plutôt bien marquer la ponctuation (avec un temps d'apprentissage suffisant), l'idée est que le second réseau ne réapprenne pas ce qui a déjà été appris, et la connaisssance "structure de mot" sera considérée comme une connaissance *innée* dans le second réseau.

Notre idée est de permettre aux deux niveaux de s'entraider pour obtenir de meilleures prédictions. Les contextes d'un réseau peuvent être représentés par, ce que l'on nommera par la suite, des *ellipses*. Lorsqu'un caractère est prédit, notre réseau se déplace dans une nouvelle *ellipse*. Le réseau possède donc un graphe (non-orienté) où les noeuds du graphe sont illustrés ici par des *ellipses*, et chaque *ellipse* est reliée à autant d'*ellipses* qu'il y a de prédictions possibles dans un contexte. Autrement dit, le premier niveau fait correspondre à chaque caractère une *ellipse* et se déplace de caractère en caractère suivant le contexte précedent et le caractère courant. On peut visualiser cette métaphore dans la figure 7(a), où "souri" est le contexte du réseau, c'est-à-dire ses dernière prédictions.

Pour le second niveau, l'espace des mots peut être illustré de la même manière, comme dans figure 7(b). Cela dit, un caractère et un contexte vont être amené à prédire un caractère qui n'appartiendra pas forcément au mot prédit au second niveau. Il faut donc trouver un moyen de faire collaborer leurs connaissances acquises.

La prédiction du prochain caractère va alors obliger la vision du réseau à changer d'*ellipse*. Lorsque l'on se trouve dans une *ellipse*, tout les caractères sont envisageables, mais pas avec la même probabilité. Imaginons que l'on souhaite écrire le mot "sourire". Nous avons déjà pu écrire correctement "souri", c'est notre contexte. La figure 7(a) donne un graphe représentatif de ce que pourrait être l'*ellipse*, en imaginant que le vocabulaire est limité à six caractères.

On observe que la prédiction la plus forte est la lettre r qui pourra ensuite, dans la nouvelle *ellipse*, préduire fortement la lettre e pour compléter le mot. Cependant, imaginons que dans le contexte, la lettre s soit aussi prédite avec une forte probabilité. Comment savoir si le prochain mot cohérent doit être "sourire" ou "souris" ?

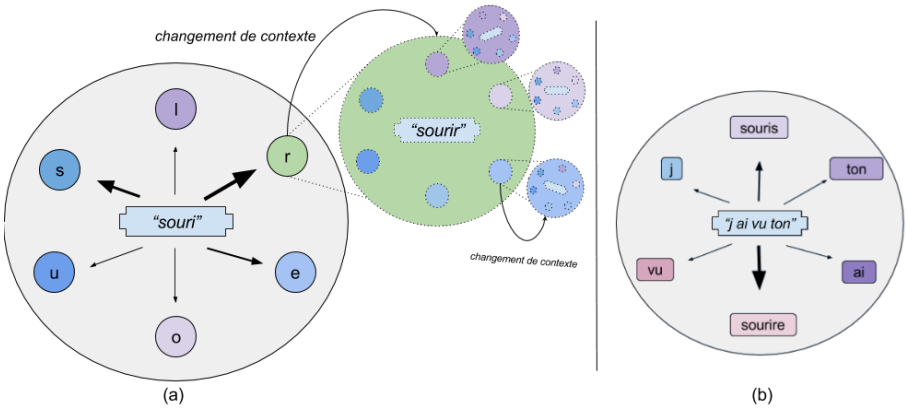

Figure 7. (a) Chaque lettre mène vers un nouveau contexte, mais le dessin ne montre que celui de la lettre *r*. L'épaisseur de la flèche représente la sortie du réseau : plus la flèche est épaisse, plus la probabilité de prédiction associée au caractère sera élevée. (b) Chaque mot possède aussi un sous-contexte (non représenté dans cette figure).

C'est là que peut intervenir le réseau du second niveau. En imaginant qu'il puisse représenter les mots dans des *ellipses*, comme le premier niveau représente les caractères, on peut imaginer qu'il ressemblera à la figure 7(b).

Dans la figure 7(b), le mot "sourire" aura une forte probabilité car le RNN de niveau 2 se souvient que le dernier mot était "ton", qui ne peut être suivi de "souris". En effet, le second niveau n'analyse que les mots et cette dépendance est plus facile à retenir pour le second niveau plutôt que pour le premier niveau car ces deux mots sont séparés par seulement une itération dans le second niveau. Le second niveau, qui possède une prédiction dominante, pourra alors encourager le premier niveau à choisir la bonne lettre. Le second niveau doit, bien entendu, connaître la longueur du mot en cours d'écriture par le premier niveau, pour retrouver le caractère adéquat et l'encourager à l'utiliser.

De plus, le premier niveau devrait aussi pouvoir aider le second niveau dans sa prédiction. Comme dans notre exemple, le mot est déjà à moitié complété, ce qui devrait forcer le second niveau à ignorer les mots ne commençant pas par ces premières lettres.

C'est sur cette repésentation que se base notre modèle. Nous souhaitons renforcer les prédictions du premier niveau grâce aux connaissances du second niveau. La prédiction finale se fera alors caractère par caractère. Nous allons voir plus en détail comme ce modèle fonctionne.

Détails du Modèle. On propose alors un modèle hybride à deux niveaux pour tenter d'avoir de meilleures prédictions. Dans un premier temps, on ne s'intéresse pas au type de codage en entrée au second niveau, mais seulement à l'aspect général du réseau.

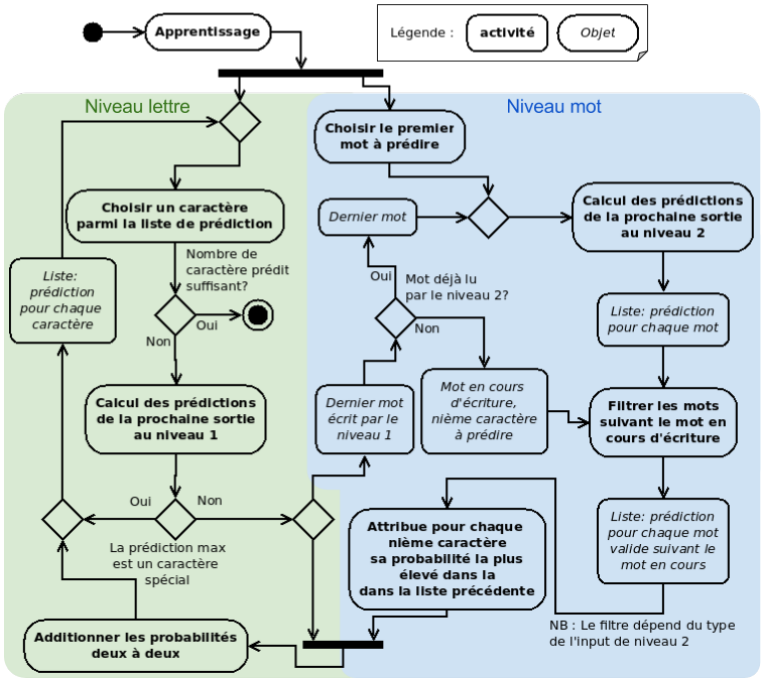

Figure 8. Processus de décision à deux niveaux de granularité (caractère et lettre)

On peut voir sur la figure 8 le diagramme d'activité lors d'une prédiction hybride. Ce diagramme représente le cas général, ne spécifie pas de quel type est l'entrée du second niveau.

À la première itération, le premier niveau choisit un caractère alétoirement, et le second niveau un mot. Par la suite, le premier niveau commence sa prédiction. Si le caractère ayant la plus forte prédiction n'est pas un caractère spécial (définit dans la note 3), il interroge le second niveau. Sinon, il choisit un caractère suivant les prédictions qu'il a effectuées. L'arrêt se fait lorsque le réseau de premier niveau à prédit le nombre de caractères voulu.

Comme dit juste avant, le second niveau est sollicité par le premier niveau si sa prédiction maximum n'est pas un caractère spécial. Le second niveau a besoin de connaître le dernier mot écrit et le mot en cours d'écriture. Par exemple, si la phrase en cours d'écriture est "Apprentissage co", le dernier mot est "Apprentissage" et le mot en cours d'écriture est "co". Le dernier mot est toujours suivi par un caractère spécial, sinon c'est potentiellement un mot en cours d'écriture.

5.3 Résultats

Nous avons experimenté notre modèle sur un texte généré par des expressions régulières comme dans la partie 4.4. Dans cette partie, nous garderons des chiffres

au lieu des mots dans le texte, cependant, les chiffres seront de taille plus variable, tout comme la longueur des phrases. De plus, nous avons des règles de court-terme, et des règles de long terme.

Mesures entre le Modèle à un Niveau et le Modèle Hybride. Les règles d'écriture du texte sont peu nombreuses. Ainsi, les résultats obtenus sur la durée du stage n'ont pu être significatifs. Nous avons utilisé pour le réseau de niveau 2 le codage sous forme de table de hachage pour la couche d'entrée.

Sur la figure 9, les deux réseaux possèdent 10 neurones dans la couche cachée. Le texte a été généré par des règles de court, comme dans la partie 9 (séparées par seulement un espace) et de long terme (séparées par plusieurs caractères, en moyenne entre 16 et 39 caractères). Toute les 100 itérations (une itération parcourt respectivement 12 caractères et 12 mots), une génération de texte d'une longueur de 10000 caractères est faite par le réseau de premier niveau seul, puis le modèle hybride avec deux réseaux connectés (cf. figure 9). On mesure le nombre de règles à respecter dans le texte généré, en différenciant les règles de court terme de celle de long terme. Puis on divise le nombre de règles réellement respectées, par les chiffres précédemments obtenus.

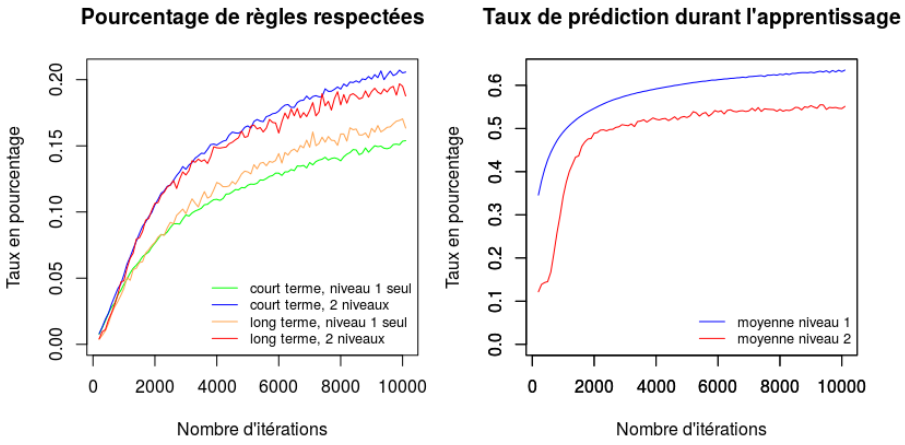

Figure 9. Comparaison du respect des règles, entre le premier niveau seul, et le modèle hybride

On constate que le modèle hybride est plus performant en terme de respect des règles, qu'elles soient de court terme ou de long terme. Le pourcentage de règles respecté converge lentement, mais continue d'augmenter. Il est possible alors, qu'avec un nombre supérieur d'itérations, on obtiennent de meilleurs résultats.

Pour ce qui est du taux de prédiction, on peut remarquer que le premier niveau converge plus rapidement que le second niveau. Mais son palier de conver-

gence n'atteint pas 70%, tout comme pour le second niveau. Il faut noter que les caractères à apprendre au premier niveau ne sont pas très nombreux, c'est pour cela que dans la figure 9, la prédiction du premier niveau augmente très vite. En effet, il a seulement 12 caractères, contre 162 mots. Cela donne 13 cases pour le second niveau, mais la prédiction reste difficile lorsque le réseau n'a pas eu le temps de classifier correctement. Il serait intéressant d'observer comment évolue la classification des mots pendant la phase d'apprentissage.

Dans un cas plus complexe, tel qu'un texte court d'une centaine de mots en français, on peut trouver, en comptant les caractères contenant des accents, la ponctuation, jusqu'à 80 caractères en entrée dans le premier niveau. Par contre, le second niveau ne sera pas impacté par cela.

Ce test n'est pas significatif, mais cela reste un élément positif pour la suite. Le résultats montrent qu'un second niveau apportant une vision différente permet d'atténuer le problème du *Vanishing Gradient*. En modifiant certains paramètres, tel que l'influence du second niveau sur le premier niveau, le nombre de neurones dans les couches cachées de chaque niveau, il est probablement possible d'obtenir de meilleurs résultats. Il serait intéressant d'augmenter le nombre de règles et d'avoir des règles de plus long terme. De plus, il serait intéressant de tester la résistance du réseau au problème de *Catastrophic Forgetting*, en lui proposant un nouveau texte, avec un faible nombre de concepts en commun. La classification devrait permettre d'ordonnancer les mots, appartenant à une même classe, ensemble.

6 Conclusion

Les concepts, et les techniques que nous avons cités dans ce rapport, nous ont inspirés pour proposer un modèle qui tentent de répondre au mieux à la problématique du *Catastrophic Forgetting* et du *Vanishing Gradient* en permettant d'apprendre sur un plus haut niveau, de nouvelles connaissances ne pouvant être apprises par un premier niveau seul.

Dans la première partie du modèle proposé, nous avons fait évoluer la couche d'entrée afin d'en réduire sa taille tout en conservant une capacité de perception pertinente pour le réseau. Les résultats de la première partie 5.3 sont convaincants. Les trois modèles proposés répondent à la problématique de dimentionnalité. Les différentes modèles utilisés sont ainsi satisfaisants, même s'ils ont, à leur tour, soulevés d'autres problématiques telles que l'activation de plusieurs bits en sortie dans le modèle de l'anagramme, ou encore l'absence de corrélation entre les mots voisins dans le modèle de la matrice 2D. La réduction de l'entrée sur le second niveau permet au réseau d'apprendre plus vite.

Les premières expérimentations montrent que la classification réalisée sur la seconde couche est cohérente car elle est exploitée par le réseau de façon à améliorer la prédiction. Les classes de mots permettent au réseau de généraliser les informations nouvelles, et de restituer au premier niveau une prédiction pertinente. Comme ce premier travail est axé autour des cohérences spatiales entre

les mots dans un texte, on constate que l'on peut utiliser l'étape d'apprentissage pour regrouper les mots appartenant à une même classe.

Dans la deuxième partie du modèle proposé, nous nous sommes intéressés à la coordination de plusieurs réseaux exploitant des horizons perceptifs différents. Nous observons sur les premiers résultats permettent de conclure que le complément apporté par le second niveau engendre de meilleurs prédictions quant au nombre de règles respectées, de court comme de long terme, tout en modérant la taille du réseau et ainsi réduire le temps d'apprentissage.

7 Perspectives

Ce travail a permis d'apporter de premiers résultats, et permet d'identifier de nouvelles problématiques telles que l'impossibilité d'arreter le classifieur sans savoir *a priori* les différentes classes de mots, le nombre de neurones nécéssaire dans la couche cachée, ou encore l'influence du second réseau sur le premier lors de la génération de texte.

Afin d'améliorer la capacité du classifieur, nous pouvons envisager de diminuer les chances pour un mot de changer de case lorsque l'on obtient de bonnes prédiction, voire stopper le classifieur. Il est aussi envisageable de modifier le nombre de cases suivant le nombre de classe que le réseau comptabilisera. Pour cela, le classifieur devra être capable de différencier les classes durant l'apprentissage, par exemple en commenant avec un très grand nombre de cases pour ensuite supprimer celle qui sont vides. Par ailleurs, concernant la taille de la couche cachée sur les deux réseaux, tout comme le taux d'apprentissage dans l'algorithme du gradient, et l'influence du second niveau sur le premier, dans la continuité des dernières expérimentations, l'idéal serait de mettre en place un algorithme génétique pour trouver les paramètres adéquats pour obtenir de meilleurs résultats.

D'autres perspectives sont aussi à étudier, comme permettre d'apprendre de manière synchrone sur les deux niveaux. Ceci permettrait de ne pas avoir à réapprendre des concepts acquis. Le concept de *mot* devrait aussi émerger entre le premier niveau et le second niveau. L'utilisation de certains concepts d'HTM pourrait être d'une grande utilité, comme par exemple la notion de *sparse distributed* et celle de la spacialité des données en sortie. La première idée serait d'utiliser aussi un classifieur sur premier niveau, sans pour autant modifier l'entrée/sortie classique du premier niveau car le problème de dimentionnalité n'apparait pas. Mais seulement pour permettre de classer les caractères spéciaux entre eux. Ainsi, le second niveau pourrait analyser cette classification pour faire émerger le concept de "mot".

De plus, un niveau supplémentaire pourrait aussi aider d'avantages ces deux niveaux, en analysant une combinaison de mot, ou une phrase complète. Cela dit, pour poursuivre cet objectif, il faudrait obtenir une meilleure stabilité de prédiction dans le modèle actuel.

Finalement, il serait aussi intéressant de tester le modèle sur d'autres types de données, comme par exemple des images ou même du son.

Remerciements

Je tiens à remercier tout particulièrement Laëtitia et Frédéric qui m'ont accompagné tout au long de mon stage, pour me permettre de mener à bien ce projet. Je remercie également l'ensemble des membres de l'équipe SMA pour leur accueil, et leurs conseils.

Références

1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J. : A learning algorithm for boltzmann machines. Cognitive science 9(1), 147–169 (1985)
2. Bengio, Y., Schwenk, H., Senécal, J.S., Morin, F., Gauvain, J.L. : Neural probabilistic language models. In : Innovations in Machine Learning, pp. 137–186. Springer (2006)
3. Bengio, Y., Simard, P., Frasconi, P. : Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on 5(2), 157–166 (1994)
4. Chomsky, N. : Structures syntaxiques, vol. 98. Editions du SEUIL (1969)
5. Chomsky, N. : Syntactic structures. Walter de Gruyter (2002)
6. Drescher, G.L., Minds, M.U. : A constructivist approach to artificial intelligence (1991)
7. Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y. : An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv :1312.6211 (2013)
8. Graves, A., Wayne, G., Danihelka, I. : Neural turing machines. arXiv preprint arXiv :1410.5401 (2014)
9. Hawkins, J., Ahmad, S., Dubinsky, D. : Hierarchical temporal memory (htm) whitepaper. Tech. rep., Technical report, Numenta (2011)
10. Hawkins, J., George, D. : Hierarchical temporal memory : Concepts, theory and terminology. Tech. rep., Technical report, Numenta (2006)
11. Hinton, G.E., Salakhutdinov, R.R. : Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006)
12. Hochreiter, S., Schmidhuber, J. : Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
13. Karpathy, A. : The unreasonable effectiveness of recurrent neural networks (2015), `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
14. Kohonen, T. : The self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (1990)
15. Lungarella, M., Metta, G., Pfeifer, R., Sandini, G. : Developmental robotics : a survey. Connection Science 15(4), 151–190 (2003)
16. Luong, T., Socher, R., Manning, C.D. : Better word representations with recursive neural networks for morphology. In : CoNLL. pp. 104–113. Citeseer (2013)
17. Mazac, S. : Approche décentralisée de lapprentissage constructiviste et modélisation multi-agent du problème damorçage de lapprentissage sensorimoteur en environnement continu. Application à lintelligence ambiante. Ph.D. thesis, Université de Lyon (2015)
18. McCulloch, W.S., Pitts, W. : A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5(4), 115–133 (1943)

19. Mikolov, T., Chen, K., Corrado, G., Dean, J. : Efficient estimation of word representations in vector space. arXiv preprint arXiv :1301.3781 (2013)
20. Moe-Helgesen, O.M., Stranden, H. : Catastophic forgetting in neural networks. Dept. Comput. & Information Sci., Norwegian Univ. Science & Technology (NTNU), Trondheim, Norway, Tech. Rep 1, 22 (2005)
21. Nielsen, M.A. : Neural networks and deep learning. URL : http ://neuralnetworksanddeeplearning.com/.(visited : 01.11.2014) (2015)
22. Olah, C. : Understanding lstm networks (2015), `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`
23. Piaget, J. : La naissance de l'intelligence chez l'enfant (1948)
24. Piaget, J., Inhelder, B. : psychologie de l'enfant [la] (1975)
25. Rosemblat, F. : The perceptron : A perceiving and recognizing automation. Cornell Aeronautical Laboratory Report (1957)
26. Rumelhart, D.E., Hinton, G.E., Williams, R.J. : Learning internal representations by error propagation. Tech. rep., DTIC Document (1985)
27. Sandini, G., Metta, G., Konczak, J. : Human sensori-motor development and artificial systems. In : Proc. of the Int. Symp. on Artificial Intelligence, Robotics, and Intellectual Human Activity Support for Applications. pp. 303–314 (1997)
28. Smolensky, P. : Information processing in dynamical systems : Foundations of harmony theory. Tech. rep., DTIC Document (1986)
29. Sutskever, I., Martens, J., Hinton, G.E. : Generating text with recurrent neural networks. In : Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1017–1024 (2011)
30. Watson, J.B. : Psychology as the behaviorist views it. Psychological review 20(2), 158 (1913)
31. Wikipedia : Constructivisme (psychologie), wikipedia, the free encyclopedia (2016), `https://fr.wikipedia.org/wiki/Constructivisme_%28psychologie%29`

A Résultats de *Word Embedding*

Grâce aux deux modèles d'architectures proposés par Mikolov et al.[19], il est possible de créer des relations entre des mots appartenant à une même classe sur la base de test *Skip-gram* contenant 783M mots de dimension 300. Des relations sont obtenues en faisant la soustraction entre deux mots, et en y additionnant un mot proche du diminuteur (dans notre exemple *France*) dans l'espace de cette façon *Paris - France + Italy = Rome*.

À cela, on peut en conclure que si les relations sont parfaitement construites, comme dans l'exemple suivant de la figure 10 fournit en annexe, on pourrait alors avoir des relations et des classes pouvant permettre à un réseau d'avoir des connaissances plus solides quant à l'utilisation des mots d'une même catégorie.

B Cartes auto-organisatrices : schéma

L'application de l'algorithme permet de discrétiser l'espace V de la figure 11. Les zones de cette dernière sont divisées et représentées par un neurone dans la grille A de la figure 11, et chacun des neurones, nombre fini et fixe a priori, possède un vecteur dit *vecteur référent*.

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Figure 10. exemple de relation obtenu entre des mots dans l'article de Mikolov et al.[19]

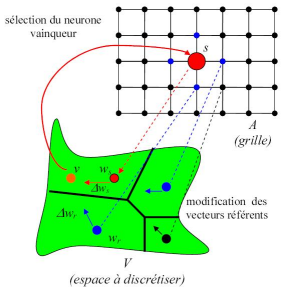

Figure 11. Algorithme SOM de Kohonen. Chaque neurone a un vecteur *référent* qui le représente dans l'espace d'entrée. Le neurone vainqueur s, le plus proche dans l'espace d'entrée. Le vecteur *référent* du vainqueur w_s est rapproché de v. Les vecteurs référents des autres neurones sont aussi déplacés vers v, mais avec une amplitude moins importante.

C *Autoencoders* couplé avec RBM

Ces deux modèles sont d'ailleurs utilisés dans l'article de Hinton et al.[11], utilisant le RBM en pré-apprentissage sur plusieurs niveaux de réseaux. Ensuite, ils utilisent un *Autoencoder* pour l'apprentissage. Il y a le même nombre de réseaux RBM qu'il y a de couches dans l'*encoder*. Cela donne des résultats très intéressants comme sur l'image de la figure 12.

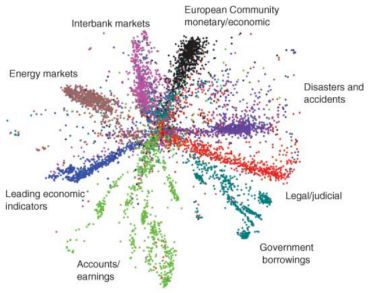

Figure 12. Résultat produit par un *Autoencoder* 2000-500-250-125-2 dans l'article de Hinton et Salakhutdinov[11]

Le couplage d'un RBM et d'un *Autoencoder* permet alors d'obtenir une classification assez précise et un codage très optimisé. À partir du codage, il est donc possible de retrouver la donnée d'origine, et de calculer la distance entre les différentes données. La figure 12 montre à quel point un codage en deux dimensions peut être assez proche de la réalité.

D Exemple sur la problématique du sur-apprentissage

.

Autre exemple, sur un texte court tel que *"Je chante la macarena. Nous mangeons une tarte."*, avec 10 neurones dans la couche cachée et en parcourant environ 3000 fois le texte, on obtient un très bon taux de prédiction, très proche de 100%, mais si l'on introduit un nouveau contexte tel que *"Nous chantons"*, le réseau va avoir tendance à prédire n'importe quoi comme *"Nous chantons une tarte."* ou encore *"Nous chantons une enes mangeons une tarte."*. Le réseau a, en effet très bien apprit le texte, mais n'a pas su apprendre les concepts de plus haut niveau. Bien sur, cet exemple n'est que très minimaliste, et il faudrait avoir une base de test beaucoup plus grande pour permettre justement au réseau de généraliser ces concepts.

E NTM et LSTM : schéma

Un NTM avec des neurones LSTM en guise de contrôleur par Graves et al.[8] sur la figure 13.

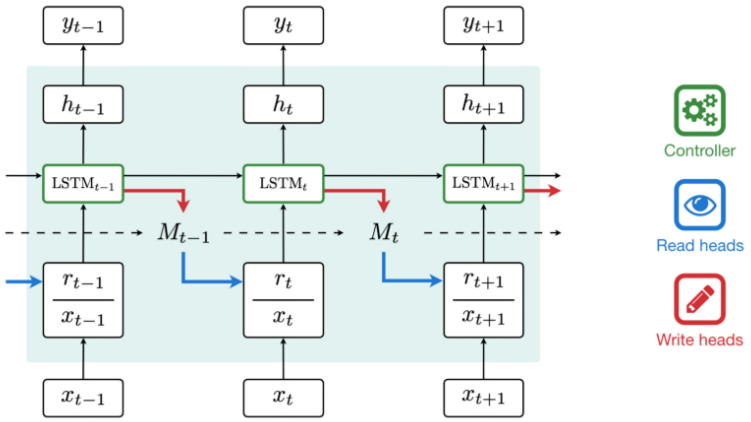

Figure 13. Un NTM avec des neurones LSTM en guise de contrôleur par Graves et al.[8]

F Hierachical Temporal Memory : schéma

La structure des régions est faite à partir des colonnes. Les régions sont dites *Sparse* car il y a un petit pourcentage de neurone actif en même temps, et *Distributed* car il y a plusieurs neurones actifs requis pour représenter quelque chose.

Dans l'apprentissage, on retrouve deux *pools*. La *pool* spatiale examine les combinaisons de bits en entrée qui se produisent souvent ensemble. Et la *pool* temporelle regarde comment ces *spatials patterns* apparaissent séquentiellement dans le temps.

L'organisation hiérarchique permet de réduire le temps d'apprentissage et l'utilisation de la mémoire car les patterns appris à chaque niveau de la hiérarchie sont réutilisés lorsqu'ils sont combinés à de plus haut niveau. La hiérarchie permet aussi aux nouveaux concepts d'hériter des propriétés connues des sous-composants.

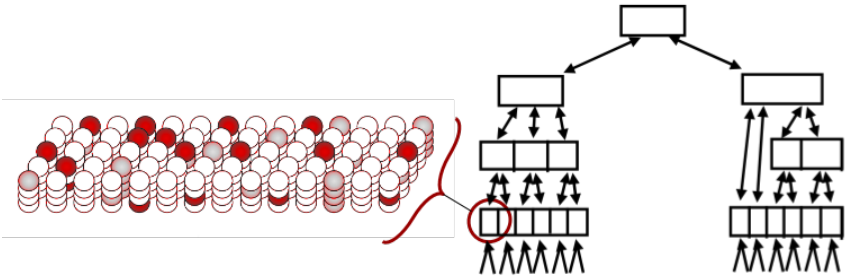

Figure 14. Sur la figure de gauche, en rouge, une région HTM avec plusieurs colonnes, chacunes possèdant quatres neurones. À droite, un exemple de comment les régions pourraient être hiérarchisées.