



MASTER RESEARCH INTERNSHIP



BIBLIOGRAPHIC REPORT

Deep Learning and Time Series

Domain: (examples) Data Structures and Algorithms - Logic in Computer Science

Author:
Arthur LE GUENNEC

Supervisor:
Romain TAVENARD of your first supervisor
Simon MALINOWSKI of your second supervisor
OBELIX

Abstract: write your abstract here

Table des matières

1	Introduction	1
2	État de l'art	1
2.1	Ce qui ce fait actuellement	1
2.1.1	Déformation dynamique temporelle	1
2.1.2	Bag-of-Words	2
2.2	Réseau de neurones	2
2.2.1	Un réseau de neurone en général	3
2.2.2	Réseau de neurones profond	4
2.2.3	Réseau de neurones convolutionnels	5
2.3	Le manque de donnée	6
2.3.1	L'augmentation de données	6
2.3.2	Dropout	8
3	Conclusion	8

1 Introduction

Il y a beaucoup d'articles sur la classification des séries temporelles, et est présente dans beaucoup de domaine comme la médecine, la biologie, la reconnaissance audio, ... Il existe beaucoup de méthodes différentes pour y parvenir. Par exemple, l'article [1] utilise les bag-of-words, une technique basée sur la description des documents par des mots. [9] utilise plutôt les réseaux de neurones convolutionnels. Dans ce document, nous allons utiliser les réseaux de neurones profonds. Dans ce type de modèle d'apprentissage, les réseaux convolutionnels (ou Convolutional Neural Networks) ont de très bons résultats dans le domaine de la classification d'image ([2]). Comme dans le domaine des séries temporelles les nombres de données annotées est faible, et que les architectures qui vont seront utilisé demandent beaucoup de donnée afin de fonctionner correctement. Nous allons donc explorer les méthodes permettant de diminuer les conséquences de ce manque de données. Ce papier est organisé de la manière suivante. La section 2 concerne l'état de l'art dans ce domaine, et la section 3 s'interessera aux problèmes du manque de données pour les séries temporelles.

2 État de l'art

La classification de série temporelle est un sujet qui est étudié depuis maintenant plusieurs années, donc différentes techniques existent et de nombreux articles sur le sujet. Dans cette partie nous allons voir quelques unes de ces différentes techniques.

2.1 Ce qui se fait actuellement

2.1.1 Déformation dynamique temporelle

La déformation dynamique temporelle (ou DTW en anglais pour Dynamic Time Warping) est un algorithme qui permet de détecter si deux séries ont des similarité même s'il y a eu une déformation (voir figure 1).

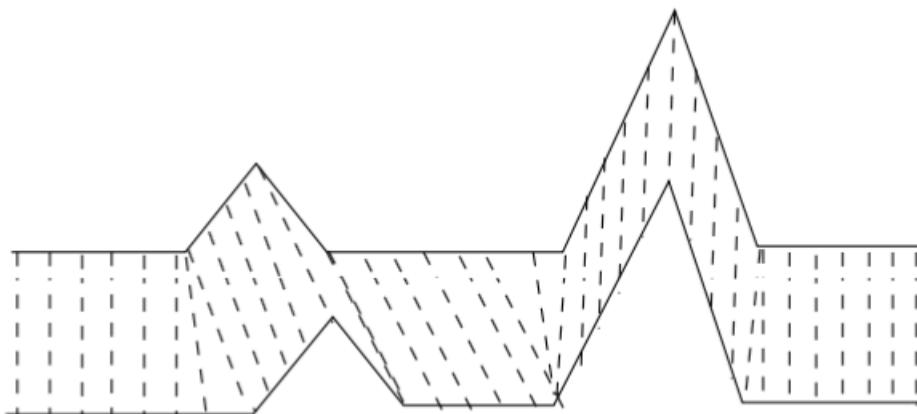


FIGURE 1: Comme on peut le voir, les deux séries temporelles ne sont pas les mêmes mais on peut reconnaître le même schéma.

2.1.2 Bag-of-Words

Le Bag-of-Words (BoW) est un modèle de description de document avec des descripteurs (des mots). Il fonctionne en attribuant des mots aux documents, chaque mot ayant un probabilité par document (voir figure 2). [1] traite justement le sujet de la classification des séries temporelles avec les Bag-of-Words. En utilisant les maximums et minimums locaux, ils obtiennent des mots qui seront ensuite utilisés dans la description des séries temporelles.

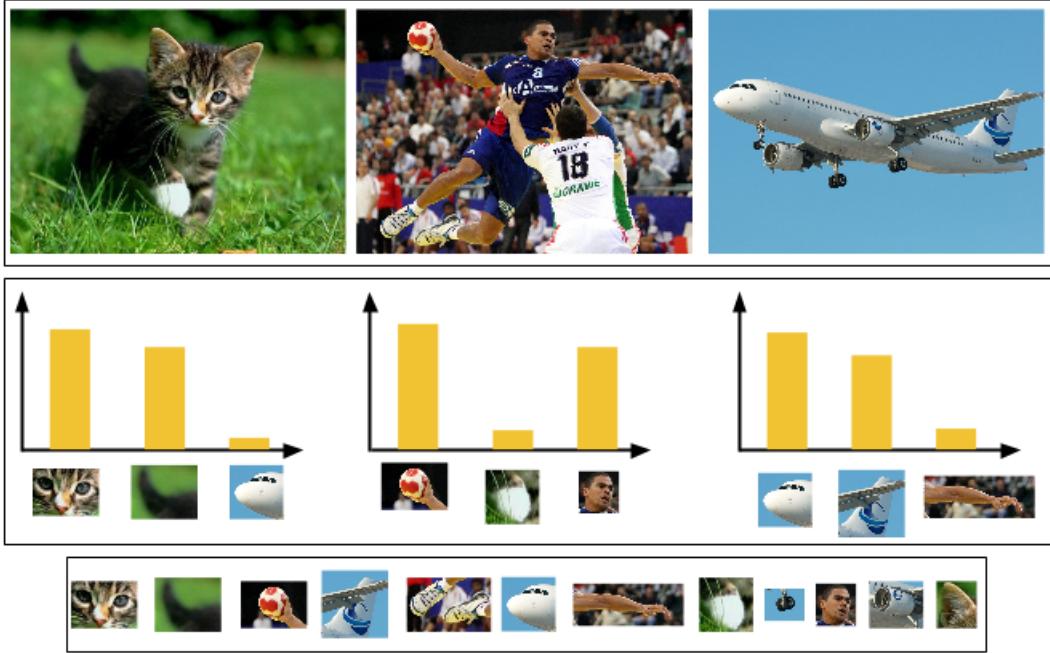


FIGURE 2: Le bag-of-words est en bas de la figure, il est composé de différents mots, ici des images, qui sont reconnus sur les documents (en haut), qui sont ici des images. Par exemple, pour la première image, on a une forte correspondance avec le premier et deuxième mot, alors que l'on a une faible correspondance avec la dernière image.

En utilisant l'algorithme de SIFT (Scale-Invariant Feature Transform) sur les données d'UCR ([3]), les taux d'erreurs obtenus dans [1] sont parfois meilleurs qu'en utilisant le DTW (Dynamic Time Warping) et la méthode des k plus proches voisins seuls.

2.2 Réseau de neurones

Les réseaux de neurones sont un système inspiré du modèle biologique du cerveau. Un réseau de neurone est composé de plusieurs neurones (voir figure 4) qui ont chacun une fonction d'activation (voir figure 3) et qui sont reliés à d'autres neurones par des poids.

La taille de la base de donnée utilisé influence aussi les résultats, Par exemple, [J'ai oublié la référence] montre bien que les performances dépendent bien de la taille des données utilisées.

2.2.1 Un réseau de neurone en général

Les neurones

Un neurone est une unité qui contient une fonction d'activation qui définit leur comportement. La figure 3 montre un neurone qui a 4 entrées et un sortir. On note X le vecteur d'entrée, W le vecteur des poids, b le biais, $f(\cdot)$ la fonction d'activation, et $h(X)$ la sortie du neurone. Notons aussi z la somme des poids et n le nombre d'entrée d'un neurone (sans compter le biais). On a donc les équations suivantes :

$$z = \sum_{k=0}^n w_k x_k + b \quad (1)$$

$$h_{W,b}(X) = f(z) \quad (2)$$

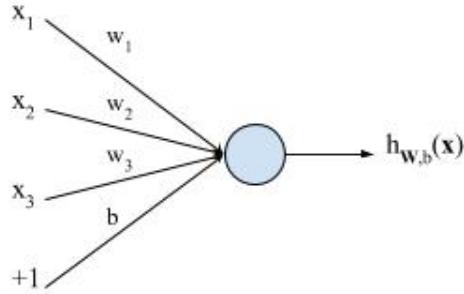


FIGURE 3: Comme on peut le voir, le neurone ci-dessus reçoit 4 valeurs, dont le biais. Dans un réseau de neurones, les entrées x sont en fait les sorties d'autres neurones.

Un neurone seul ne peut pas effectuer de tâche, c'est pourquoi ils sont utilisés à plusieurs pour effectuer diverses tâches comme de l'analyse de données ou de la classification.

L'architecture

L'efficacité d'un réseau de neurones dépend de son architecture. Il existe différentes architectures pour un réseau de neurone (voir figure 4), la plus connue et la plus répandue étant le réseau de neurone non-bouclés (ou feedforward neural network en anglais) (voir figure 4a). Selon le nombre de couche et le nombre de neurones par couche, le réseau peut être plus ou moins performants ([2][8]).

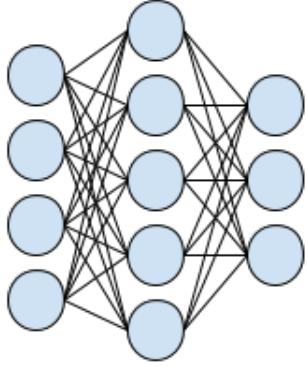
Pour faire fonctionner un réseau de neurone non-bouclés, on lui donne d'une entrée qui est de la même taille que la couche d'entrée. Ces valeurs vont ensuite être transmises aux neurones de la couche suivante jusqu'à couche de sortie de sortie.

En appliquant les formules 1 et 2, et en notant L le nombre de couche dans le réseau, a l'activation d'un neurone (qui correspond à la sortie du neurone), $n^{(l)}$ le nombre de neurones de la couche l , $w_{ij}^{(l)}$ le poids allant du neurone j de la couche $l - 1$ au neurone i de la couche l , x le vecteur d'entrée du réseau, on obtient les équations suivantes :

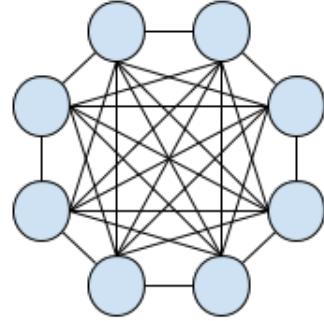
$$a_i^{(0)} = x_i \quad (3)$$

$$z_i^{(l)} = \sum_{k=0}^{n^{(l-1)}} w_{ik}^{(l)} a_k^{(l-1)} \quad \forall 1 < l \leq L \quad (4)$$

$$a_i^{(l)} = f(z_i^{(l)}) \quad \forall 1 < l \leq L \quad (5)$$



(a) Réseau de neurones à couches



(b) Réseau de neurones à connexion complète

FIGURE 4: Il existe différents types de réseaux de neurones. Par exemple, la figure 1.a est un réseau avec une architecture à couche, alors que le réseau de la figure 2.b à une architecture à connexion complète. Dans le domaine de la classification, l'architecture à couche est souvent utilisé.

L'entraînement

L'entraînement est indispensable dans un réseau de neurone. C'est pendant l'que l'on fixe les poids entre les neurones, ces poids étant souvent décidées aléatoirement lors de la création du réseau. Le méthode la plus souvent utilisé est la backpropagation. Cette méthode consiste à ajuster le poids peu à peu, le faisant converger vers une valeur. Pour cela, on commence par lancer le réseau de neurone une fois. Les valeurs alors obtenues sont alors très loin de ce que l'on veut obtenir.

2.2.2 Réseau de neurones profond

Un réseau de neurones profond est un réseau qui contient plusieurs couches cachées. Ce type de réseau a l'avantage de pouvoir apprendre un plus grand nombre de fonction avec un nombre de neurone dans les couches cachées bien inférieurs à celui d'un réseau classique (ne contenant qu'une couche cachée). En effet, à la place d'avoir un nombre exponentielle de neurones cachés (par rapport aux neurones d'entrés), nous avons un nombre polynomiale. Cependant, les réseaux de neurones profonds sont difficiles à entraîner car ils nécessitent beaucoup de données labelées. L'entraînement d'un réseau avec peu de données nous donnerait un sur-apprentissage. De plus, l'entraînement par backpropagation est peu efficace et induit souvent en erreur (du aux extrema locaux). Il faut donc les entraîner différemment. Une des méthodes existantes est greedy layerwise training qui consiste à d'abord entraîner un réseau avec une couche cachée, de garder les paramètres obtenus, puis de rajouter une seconde couche cachée, et ainsi de suite (voir figure 5).

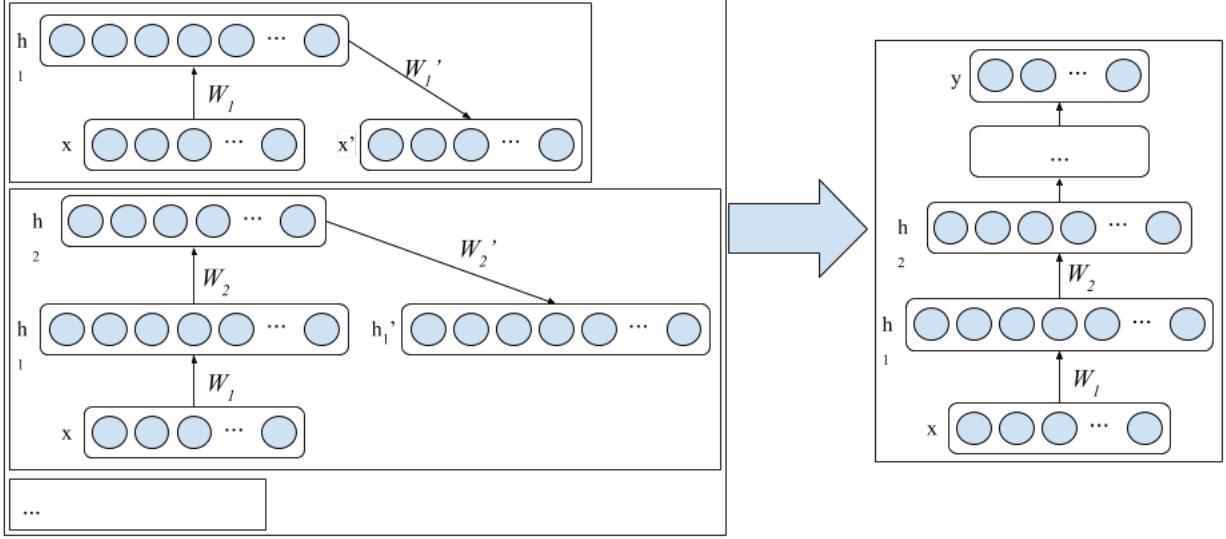


FIGURE 5: On utilise ici l'algorithme greedy layerwise training avec des stacked autoencoders. C'est à dire que l'on veut que la sortie soit le plus proche possible de la couche qui la précède.

2.2.3 Réseau de neurones convolutionnels

Les CNN (Convolutional Neural Network, ou réseaux convolutionnels) sont une structure particulière des réseaux de neurones profond car les neurones d'une couche à une autre ne sont pas tous reliés entre eux. En effet, les connections entre neurones se font localement, ce qui fait des entrées d'une couche cachée un sous-ensemble de motif de la couche précédente (voir figure 6).

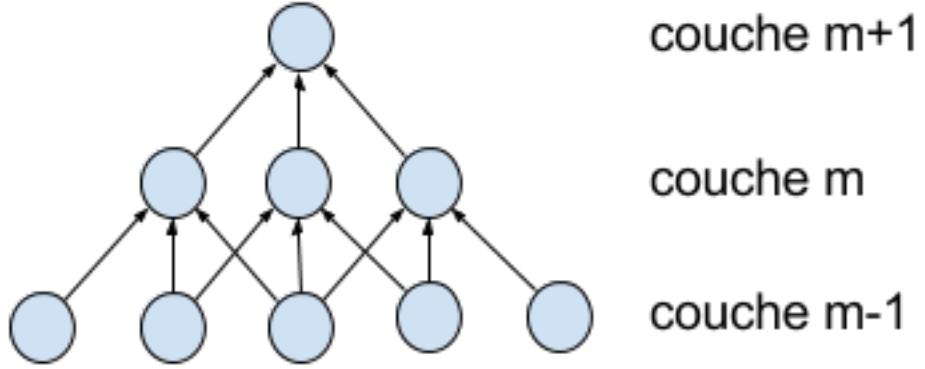


FIGURE 6: Ici, imaginons que la couche $m-1$ est une rétine où chaque neurone correspond à un pixel d'une image. Dans la couche m , on s'aperçoit que chaque neurone a 3 entrées qui correspondent chacune à un pixel. Chaque neurone de la couche m est donc un sous-ensemble de l'image de 3 pixels. De la même façon, chaque neurone de la couche $m+1$ est donc un sous-ensemble de l'image de 5 pixels.

Dans la figure 6, les neurones de la couche $m-1$ sont les filtres de cette couche. De la même façon, les neurones de la couche $m-2$ sont aussi des filtres de cette couche. Un CNN peut avoir plusieurs

couches de filtres, le nombre de filtre d'une couche à une autre décroissant. Appliqué à une image, les filtres de la première couche cachée peuvent être associés à un ensemble de pixel (voir figure 7), et ensuite de suite avec les autres couches.

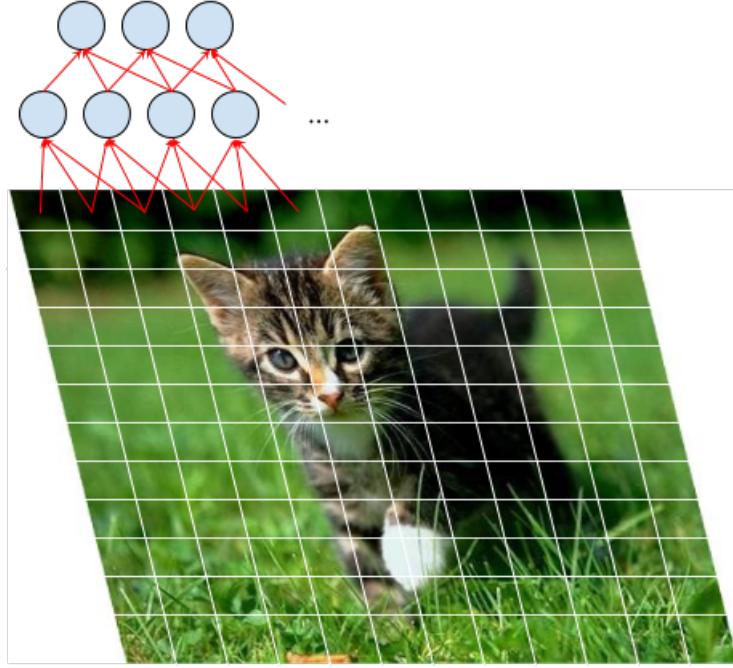


FIGURE 7: L'image d'entrée contient ici 166 composantes, donc la couche d'entrée contiendra 166 neurones. On peut voir que la première couche cachée est bien un ensemble de pixel.

Pour les séries temporelles, les CNN peuvent aussi s'appliquer. Par exemple, l'article [9] utilise les CNN pour faire de la classification de séries temporelles multivariées, c'est à dire des séries temporelles qui sont sur plusieurs dimensions.

2.3 Le manque de donnée

Que faire lorsque l'on n'a pas assez de données.

2.3.1 L'augmentation de données

Pourquoi chercher à augmenter la base de données ?

On a besoin de palier le manque de donnée car les architectures de réseau de neurones profond, comme les CNN, ont besoin de beaucoup de données pour pouvoir être entraînés efficacement. Les structures tels que MC-DCNN (voir [9]) peuvent être utilisées. De plus il y a peu, comparés aux images, de données labelées sur les séries temporelles. On aura donc besoin d'augmenter ces données de différentes façons. Par analogie aux images, on regardera comment transformer une série temporelle. Dans les données pour les images, on applique certaines transformations ([6, 4]), tel que la translation, l'inversion, le changement de couleur, de contraste, de luminosité, ... (voir figure 8). Ces transformations ne sont pas forcément applicables aux séries temporelles et on regardera donc comment faire.

Comment y parvenir ?

Pour les images, plusieurs techniques existent. Par exemple, [2] explique que pour réaliser une augmentation de données, ils perturbent une image avec des transformations qui n'invalident pas cette image, c'est à dire que si l'image d'origine montrait un chat, l'image transformée doit toujours montrer un chat. [6] explique que pour augmenter ses données, il sépare tout d'abord ses données en deux parties, l'une pour l'apprentissage, et l'autre pour les tests. Dans les données pour l'apprentissage, ils réalisent des sous-images en extractant des zones de 224x224 pixels de l'image d'origine et de sa réflexion horizontale, qui a une taille de 256x256. On obtient donc une multiplication de 2048 de la base de données d'origine. Les données de test prendront 5 sous-images et leur réflexion (donc 10 images) selon leurs prédictions. Toujours dans l'idée d'augmenter leurs données d'apprentissage, ils effectuent une ACP pour pouvoir, non pas réduire les dimensions, ajouter du bruit contrôlé sur les composants principaux. Krizhevsky et al. [5] arrivent donc à baisser le taux d'erreur de plus de 1% ainsi. Dans le papier [6], Howard utilise le même procédé mais rajoute d'encore d'autres étapes.

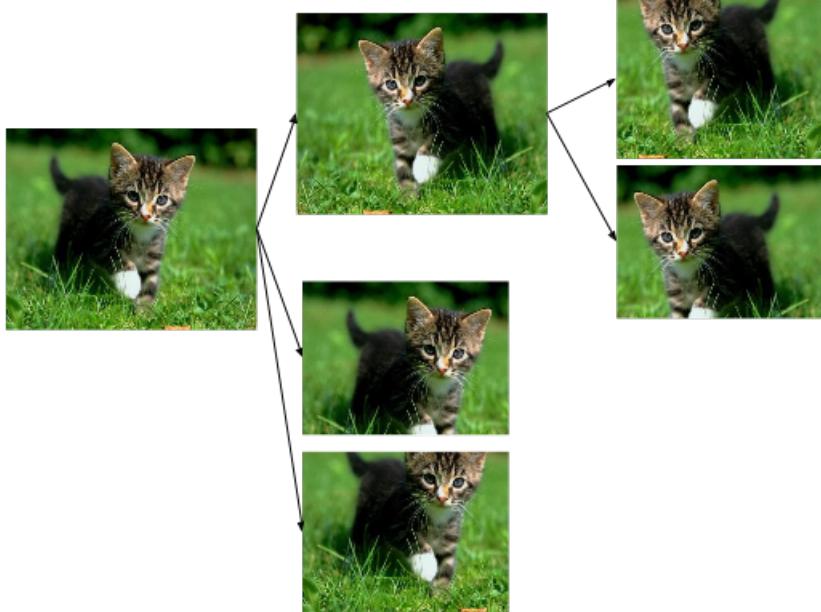


FIGURE 8: Exemple d'augmentation de donnée pour une image. Ici, on a obtenu six images à partir d'une seule. On a d'abord appliqué une inversion d'image et puis on a rogné les images obtenues de deux manières différentes. En réalité, beaucoup plus de transformations sont appliquées, permettant de multiplier la base de données des images de beaucoup (1024 dans [4]).

Beaucoup de méthodes de data-augmentation le sont pour les images, la question étant de savoir si parmi ces méthodes, on peut en réutiliser ou non. Par exemple, on ne peut pas réaliser une réflexion sur une série temporelle. Cependant, l'approche de Krizhevsky dans [6] pour les données d'entraînement avec une ACP pourrait être réalisable sur des séries temporelles. On pourrait peut-être aussi rajouter du bruit contrôlé selon différents paramètres comme dans [6] et [4].

2.3.2 Dropout

Le dropout (vient de la contraction de deux mots anglais *dropping out*) est une méthode d'apprentissage qui permet d'obtenir de bons résultats avec un nombre de données réduites. L'article [8] introduit cette notion. Le principe est de désactiver certains neurones par couches aléatoirement (voir figure 9).

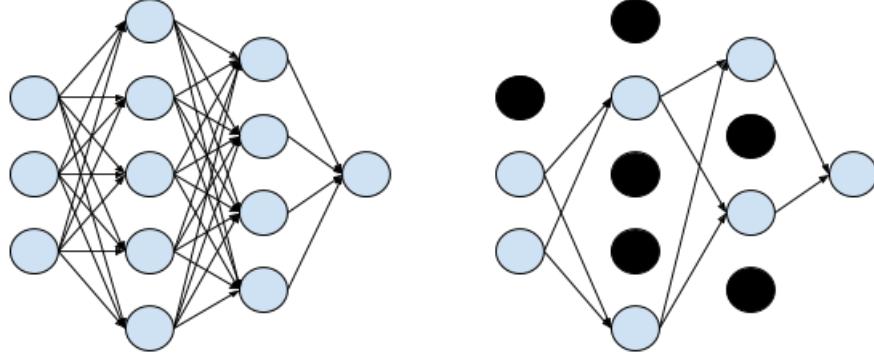


FIGURE 9: Par exemple, le réseau de neurone à droite est celui de gauche après que l'on est appliqué le dropout. Comme on peut le voir, on réduit de beaucoup de nombre de connexion entre les neurones.

On applique en fait une probabilité à chaque couche. Par exemple, une probabilité de 0.5 sur la couche l d'un réseau de neurone signifie que les neurones de cette couche ont chacun une probabilité de 0.5 d'être activé. On note p la probabilité et on a donc la formule (4) qui devient :

$$r_j^{(l)} \sim Bernoulli(p) \quad (6)$$

$$\tilde{a}^{(l)} = r^{(l)} * a^{(l)} \quad (7)$$

$$z_i^{(l)} = \sum_{k=0}^{n^{(l-1)}} w_{ik}^{(l)} \tilde{a}_k^{(l-1)} \quad \forall 1 < l \leq L \quad (8)$$

3 Conclusion

Pour conclure, il existe de nombreuses manières d'augmenter le volume des données. Cependant, ces méthodes dépendent du type de données. Avant de commencer à classifier des séries temporelles, il va donc falloir trouver comment augmenter ce volume. Nous pourrions, par exemple, utiliser l'algorithme ACP ou s'inspirer des méthodes utilisées sur les images. Pour la classification, les CNN semblent être un bon choix étant donné les bons résultats obtenus dans [9]. Pour cela, nous allons utiliser le framework Caffe (voir [5]). Une fois mis en place, nous pourrions alors observer les résultats.

Références

- [1] Adeline Bailly, Simon Malinowski, Romain Tavenard, Thomas Guyet, and Laetitia Chapel. Bag-of-temporal-sift-words for time series classification. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2015.
- [2] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details : Delving deep into convolutional nets. *arXiv preprint arXiv :1405.3531*, 2014.
- [3] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/.
- [4] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv :1312.5402*, 2013.
- [5] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe : Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.
- [6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [7] Andrew Ng, Jiquan Ngiam, Chuan Y Foo, Yifan Mai, and Caroline Suen. Ufldl tutorial, 2012.
- [8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1) :1929–1958, 2014.
- [9] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *Web-Age Information Management*, pages 298–310. Springer, 2014.