



## MASTER RECHERCHE INFORMATIQUE



## RAPPORT BIBLIOGRAPHIQUE

---

### Apprentissage profond et séries temporelle

---

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

*Auteur:*

Arthur LE GUENNEC

*Superviseur:*

Romain TAVENARD

Simon MALINOWSKI

OBELIX

**Abstract:** Beaucoup de techniques existent pour classifier des données, que ce soit des images, ou des séries temporelles. Des méthodes comme les *bag-of-words*[1], les réseaux de neurones [8][13][3] sont couramment utilisés et donnent des résultats différents. Ces derniers peuvent donner de bons résultats à la condition que le jeu de données soit suffisant. Heureusement, il existe plusieurs méthodes pour augmenter ce jeu de données ou pour diminuer l'impact d'un jeu de donnée trop faible.

**Résumé:**

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>État de l'art</b>	<b>1</b>
2.1	Différents types d'apprentissage . . . . .	1
2.2	Séries temporelle . . . . .	1
2.3	Ce qui ce fait actuellement . . . . .	2
2.3.1	Techniques basées sur la distance . . . . .	2
2.3.2	Sac à mots . . . . .	2
2.4	Réseau de neurones . . . . .	3
2.4.1	Un réseau de neurones en général . . . . .	3
2.4.2	Réseau de neurones profond . . . . .	6
2.4.3	Réseau de neurones convolutionnels . . . . .	6
2.5	Le manque de donnée . . . . .	8
2.5.1	Sur-apprentissage et sous-apprentissage . . . . .	8
2.5.2	L'augmentation de données . . . . .	9
2.5.3	Dropout . . . . .	10
<b>3</b>	<b>Conclusion</b>	<b>11</b>

# 1 Introduction

Beaucoup d'articles sur la classification des séries temporelles ont été écrits ces dernières années. Ce sujet est présent dans beaucoup de domaine comme la médecine, la biologie, la reconnaissance audio, ... Il existe des méthodes simples tels que les kNN (k Nearest Neighbor ou les k plus proche voisins en français) souvent utilisées avec les DTW (Dynamic Time Warping ou déformation temporelle dynamique en français) ou avec les ED (Euclidean Distance ou distance euclidienne en français). Il existe aussi des méthodes plus avancées pour y parvenir. Par exemple, l'article de Bailly et coll. ([1]) utilise les *bag-of-words*, une technique basée sur la description des documents par des mots. Zheng et coll. ([13]) 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*) donnent de très bons résultats dans le domaine de la classification d'image ([3]).

Comme dans le domaine des séries temporelles, le nombre de données annotées est faible, et que les architectures qui vont seront utilisées demandent beaucoup de données afin de fonctionner correctement, nous allons donc explorer les méthodes permettant de diminuer les conséquences de ce manque de données.

Cet article est organisé comme suit. La section 2.3 et 2.4 concerne l'état de l'art des outils dans ce domaine, et la section 2.5 s'intéressera aux problèmes du manque de données pour les séries temporelles. Pour finir, la section 3 conclut cet article.

## 2 État de l'art

La classification de série temporelle est un sujet qui est étudié depuis maintenant plusieurs années. Différentes techniques existent et de nombreux articles ont été écrits sur le sujet. Ces techniques peuvent être séparées en deux parties, celles basées sur les données au niveau global que le DTW ou ED, et celle basées sur les données au niveau local telle que les *bag-of-words*. Dans cette partie nous allons voir quelques-unes de ces différentes techniques.

### 2.1 Différents types d'apprentissage

Il existe différents types d'apprentissage, les trois principaux étant l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. L'apprentissage supervisé consiste à forcer le réseau à converger vers un état souhaité et nécessite donc des données labelées (figure 1). L'apprentissage non-supervisé consiste à converger le réseau vers n'importe quel état et ne nécessite pas de données non-labelées. Une méthode efficace pour entraîner un réseau de neurones sans supervision est les autoencoders décrits plus loin. L'apprentissage par renforcement consiste à apprendre au réseau à s'adapter par rapport aux différentes situations.

### 2.2 Séries temporelle

Une série temporelle est un série qui évolue dans le temps. Ces séries sont utilisées dans de nombreux domaines, tels que la médecine, la biologie, les mouvements [10], ... Les images peuvent aussi être transformées en série [10]. Les séries temporelles qui seront utilisés lors ce stage seront tirées des archives d'UCR [4].

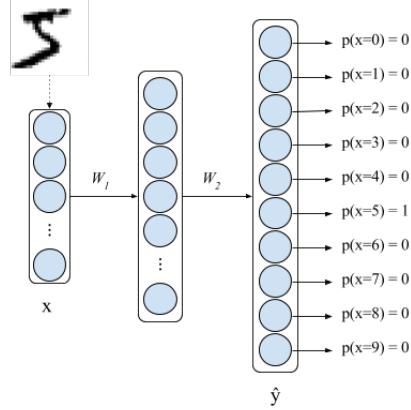


FIGURE 1: Exemple avec un réseau de neurones. L'entrée  $x$  correspond à l'image du nombre, et  $\hat{y}$  est la sortie souhaitée. Ici, le nombre est 5, donc on souhaite que la sortie nous donne une probabilité de 1 pour 5.

## 2.3 Ce qui se fait actuellement

### 2.3.1 Techniques basées sur la distance

Les techniques DTW ou ED sont des méthodes permettant de comparer point-à-point différentes séries temporelles (figure 2). Ces deux techniques présentent chacun leurs avantages et leurs inconvénients. Si l'ED est rapide à calculer, il n'est pas très efficace lorsque deux séries sont déphasées. Et si le DTW est efficace, il est plus complexe à mettre en place et demande plus ressources. Cependant, Ratanamahatana et Keogh [11] informe que le nombre de ressources mis en place pour faire fonctionner le DTW peuvent être réduites.

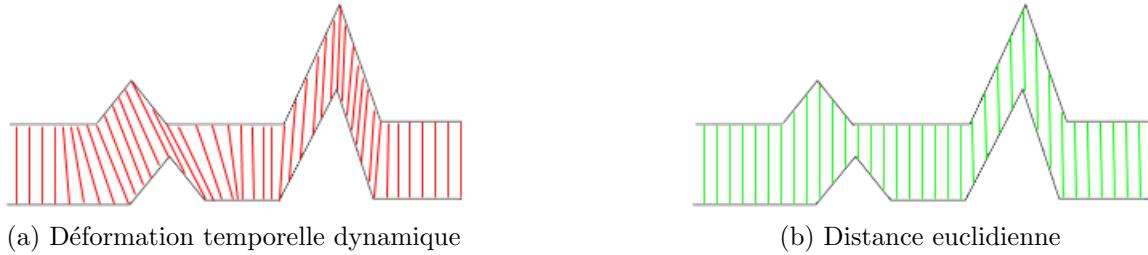


FIGURE 2: Comparaison entre le DTW (2a) et ED (2b).

### 2.3.2 Sac à mots

Les sacs à mots (BoW en anglais pour *Bag-of-Words*) sont un modèle de description de document avec des descripteurs (qui sont considérés comme des mots). Il fonctionne en attribuant des mots aux documents, chaque mot ayant une probabilité par document (figure 3). L'article de Bailly et coll. [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.

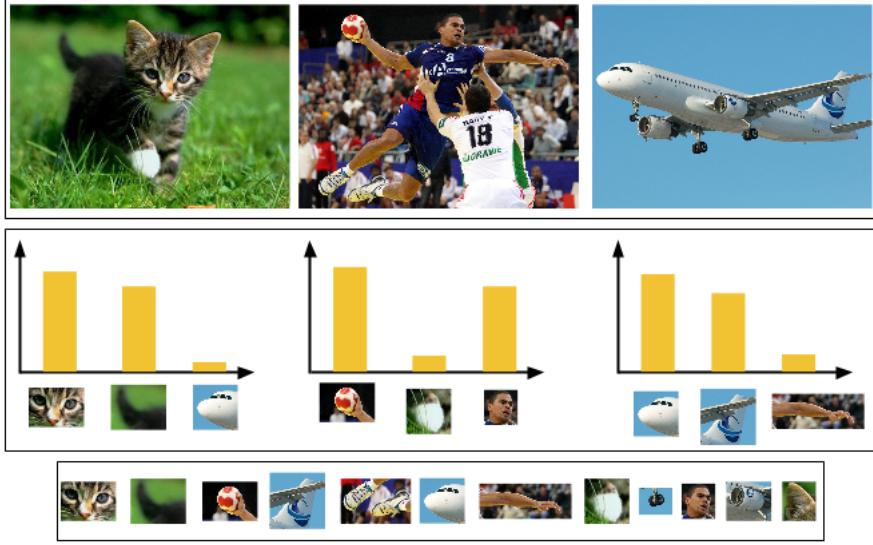


FIGURE 3: Le bag-of-words est en bas de la figure, il est composé de différents mots, ici des images, qui sont reconnus sur les documents (en haut), qui sont ici des images. Par exemple, pour la première image, on a une forte correspondance avec le premier et deuxième mot, alors que l'elle est faible correspondance avec la dernière image.

En utilisant l'algorithme de SIFT (*Scale-Invariant Feature Transform*) sur les données d'UCR ([4]), les taux d'erreurs obtenus dans [1] sont parfois meilleurs qu'en utilisant la méthode des k plus proches voisins seuls couplé avec le DTW. Pour créer des mots, plusieurs méthodes.

## 2.4 Réseau de neurones

Les réseaux de neurones sont un système inspiré du modèle biologique du cerveau. Un réseau de neurones est composé de plusieurs neurones (figure 5) qui ont chacun une fonction d'activation (figure 4) et qui sont reliés à d'autres neurones par des poids.

### 2.4.1 Un réseau de neurones en général

*Les neurones*

$$f(x) = \frac{1}{1 + \exp -x} \quad (1)$$

Un neurone est une unité qui contient une fonction d'activation qui définit son comportement. La figure 4 représente un neurone qui a 4 entrées et une sortie. On note  $X$  le vecteur d'entrée,  $W$  le vecteur des poids,  $b$  le biais,  $f(\cdot)$  la fonction d'activation (équation 1), et  $h(X)$  la sortie du neurone. Notons aussi  $z$  la somme des poids et  $n$  le nombre d'entrées d'un neurone (sans compter le biais). On a donc les équations suivantes :

$$z = \sum_{k=0}^n w_k x_k + b \quad (2)$$

$$h_{W,b}(X) = f(z) \quad (3)$$

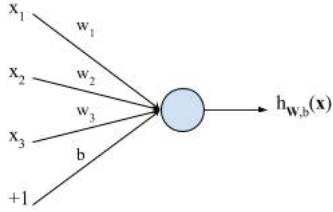
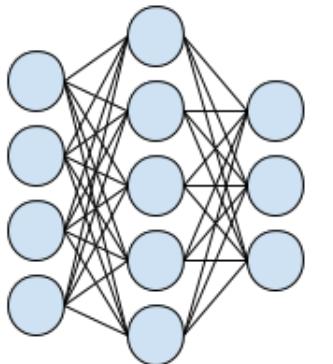


FIGURE 4: Comme on peut le voir, le neurone ci-dessus reçoit 4 valeurs, dont le biais. Dans un réseau de neurones, les entrées  $x$  sont les sorties d'autres neurones.

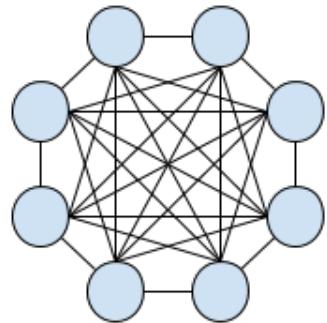
Un neurone seul ne peut pas effectuer de tâche, c'est pourquoi ils sont utilisés à plusieurs pour effectuer diverses tâches comme de l'analyse de donnée ou de la classification.

#### *L'architecture*

L'efficacité d'un réseau de neurones dépend de son architecture. Il existe différentes architectures pour un réseau de neurones (figure 5), la plus connue et la plus répandue étant le réseau de neurones non bouclés (ou feedforward neural network en anglais) (figure 5a). Selon le nombre de couche cachée et le nombre de neurones par couche, le réseau peut être plus ou moins performant ([3], [12]).



(a) Réseau de neurones à couche



(b) Réseau de neurones à connexion complète

FIGURE 5: 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 utilisée.

Pour faire fonctionner un réseau de neurones non bouclés, on lui transmet une entrée qui est de la même taille que la couche d'entrée (par exemple, si l'entrée est une image, sa taille peut correspondre au nombre de pixel). Ces valeurs sont ensuite transmises aux neurones de la couche suivante de manière répétée jusqu'à couche de sortie.

En appliquant les formules 2 et 3, 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 :

$$a_i^{(0)} = x_i \quad (4)$$

$$z_i^{(l)} = \sum_{k=0}^{n^{(l-1)}} w_{ik}^{(l)} a_k^{(l-1)} \quad \forall 1 < l \leq L \quad (5)$$

$$a_i^{(l)} = f(z_i^{(l)}) \quad \forall 1 < l \leq L \quad (6)$$

### *L'apprentissage d'un réseau de neurones*

L'objectif de l'apprentissage est de trouver les poids qui lient les neurones. Le méthode la plus souvent utilisée est la backpropagation qui consiste à diminuer l'erreur de la sortie, l'erreur étant la différence entre la valeur obtenue et la valeur souhaitée dans le cas d'un apprentissage supervisé. Cette différence est aussi appelé la fonction de coût et est noté  $J(W, b; x, y)$  avec  $W$  la matrice des poids,  $b$  le vecteur des biais,  $x$  le vecteur d'entrée, et  $y$  le vecteur de sortie espéré. Le but est donc bien de diminuer au maximum cette fonction de coût. Dans le tutoriel de Ng et coll. ([9]), cette fonction de coût est décrite par l'équation suivante :

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2 \quad (7)$$

Une fois cette fonction trouvée, on applique une descente de gradient sur les paramètres  $W$  et  $b$ . On a donc :

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) \quad (8)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b) \quad (9)$$

$\alpha$  correspond au taux d'apprentissage et doit être soigneusement choisi. Si  $\alpha$  est trop grand, on risque de louper les bonnes valeurs. Si  $\alpha$  est trop petit, l'apprentissage sera plus long, mais on risque surtout de rester bloquer dans un extremum local. Pour résumer les formules (7), (8) et (9), on part de la sortie du réseau de neurones pour revenir jusqu'à l'entrée tout en modifiant sur le chemin les poids qui relient les neurones entre chaque couche. On relance ensuite le réseau de neurones avec la même donnée. On recommence ce processus selon diverses conditions, comme un nombre fixé à l'avance, un taux d'erreur atteint, ... On relance tout ce processus autant de fois que l'on a de données d'entraînement.

Dans le cas d'un apprentissage non supervisé, une des méthodes utilisées est d'entraîner, de la même manière qu'un réseau classique, un autoencoder. Un autoencoder est un réseau de neurones où la sortie est approximativement égale à l'entrée (figure 6). On essaye donc de trouver la fonction  $h_{W,b}(x) \approx x$ . Le réseau autoencoder apprend donc à trouver les axes principaux de la donnée d'entrée [5] (de la même manière que l'ACP).

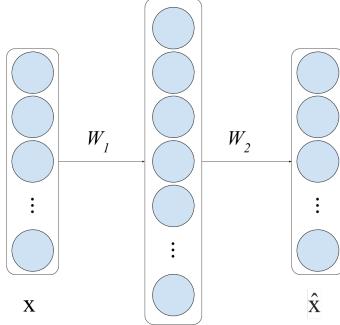


FIGURE 6:  $x$  est 'entrée, et  $\hat{x}$  est une approximation de l'entrée appliquée sur la sortie.

#### 2.4.2 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 fonctions avec un nombre de neurones bien inférieurs dans les couches cachées à celui d'un réseau classique. En effet, à la place d'avoir un nombre exponentiel de neurones cachés (par rapport aux neurones d'entrées), nous avons un nombre polynomial.

Cependant, les réseaux de neurones profonds sont plus difficiles à entraîner qu'un réseau classique, car ils nécessitent beaucoup de données labelées pour l'entraînement du réseau [8][6]. L'entraînement d'un réseau avec peu de données mènerait un sur-apprentissage (figure 11a). En plus du manque de données, l'entraînement par backpropagation vu à la section 2.4.1 est peu efficace s'il est utilisé tel quel, et mène souvent au sur-apprentissage (due aux extrema locaux). Il faut donc les entraîner différemment.

Il ne faut donc pas initialiser les poids aléatoirement, mais les initialiser à une valeur plus juste par un pré-entraînement. Pour cela, on va utiliser l'algorithme du *greedy layerwise training* sur un réseau de neurones *stacked autoencoder*. L'algorithme *greedy layer-wise* consiste d'abord entraîner un réseau avec une couche cachée par *backpropagation*, de garder les paramètres obtenus, puis de rajouter une seconde couche cachée, et ainsi de suite. Selon le type d'apprentissage (supervisé ou non), la sortie souhaitée à chacune de ces étapes est la couche cachée précédente (figure 7). Un fois ce pré-entraînement terminé, on peut commencer l'entraînement du réseau par *backpropagation*.

#### 2.4.3 Réseau de neurones convolutionnels

Les CNN (Convolutional Neural Network, ou réseaux de neurones convolutionnels) sont une structure particulière des réseaux de neurones profonds car les neurones d'une couche à une autre ne sont pas tous reliés entre eux. En effet, ses connexions entre neurones se font localement, ce qui fait des entrées d'une couche cachée un sous-ensemble de motif de la couche précédente (figure 8).

Dans la figure 8, 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 filtres décroissant d'une couche à une autre. Appliqués à une image, les filtres de la première couche cachée peuvent être associés à un ensemble de pixels (figure 9), et ensuite de suite avec les autres couches.

Pour les séries temporelles, les CNN peuvent aussi s'appliquer. Par exemple, l'article de Zheng et coll. ([13]) utilise les CNN pour faire de la classification de séries temporelles multivariées, c'est-

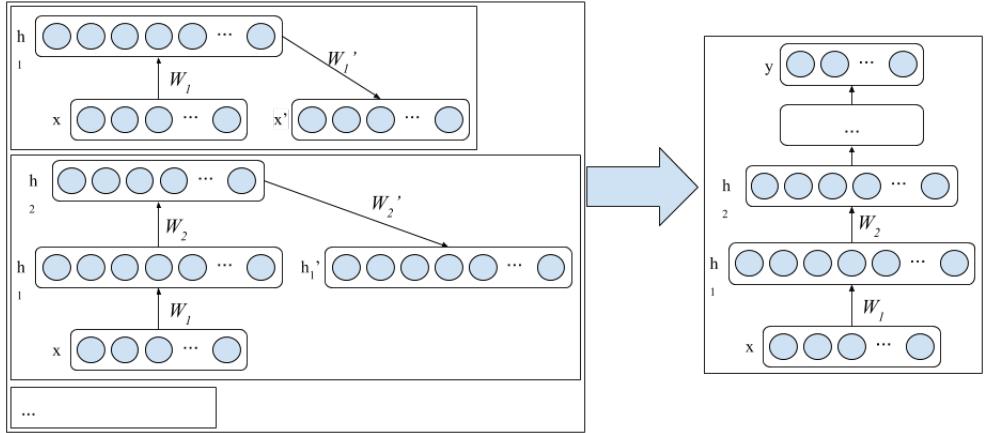


FIGURE 7: Entraînement d'un réseau de neurones profond pour un apprentissage non supervisé.

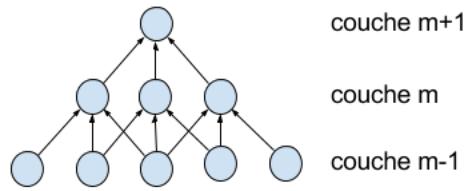


FIGURE 8: Ici, imaginons que la couche  $m-1$  est une rétine où chaque neurone correspond à un pixel d'une image. Dans la couche  $m$ , on s'aperçoit que chaque neurone a 3 entrées qui correspondent chacune à un pixel. Chaque neurone de la couche  $m$  est donc un sous-ensemble de l'image de 3 pixels. De la même façon, chaque neurone de la couche  $m+1$  est donc un sous-ensemble de l'image de 5 pixels.

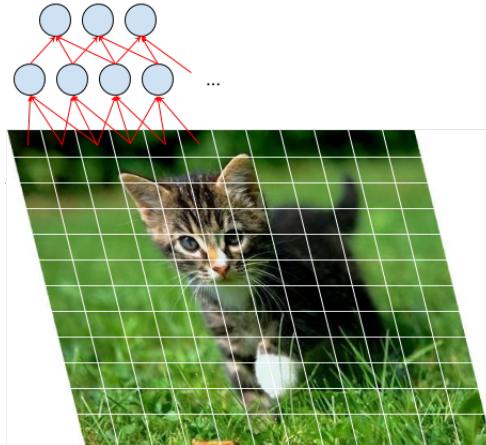


FIGURE 9: L'image d'entrée contient ici 144 composantes, donc la couche d'entrée contiendra 144 neurones. On peut voir que la première couche cachée est bien un ensemble de pixels.

à-dire des séries temporelles qui sont sur plusieurs dimensions.

### *Pooling*

On peut imaginer que la figure 9 nous montre une image avec 144 pixels, que chaque filtre prend un zone de  $1 \times 3$  pixels (comme sur la figure), et que nous voulons apprendre 400 fonctions pour chaque zone de pixels, on se retrouve avec  $(12 - 3 + 1)(13 - 1 + 1) \times 400 = 48\ 000$  fonctions à apprendre. Apprendre 48 000 fonctions n'est pas difficile pour un ordinateur. Cependant, si l'image fait  $100 \times 100$  pixels, qu'on élargit les zones de filtre à  $5 \times 5$  pixels par exemple, on obtient  $(100 - 5 + 1)(100 - 5 + 1) \times 400 = 3\ 686\ 400$  fonctions à apprendre. Apprendre une telle quantité de fonction est très lent et augmente surtout le risque de sur-apprentissage.

Pour éviter cela, on peut utiliser le *pooling*. Très utile pour utiliser les CNN, le *pooling* va réduire significativement le nombre de filtres présent dans un réseau de neurones convolutionnel. Par exemple, pour image, on va sélectionner des zones de pixels aléatoirement qui passeront à un autoencoder. Il est important de préciser qu'il ne doit y avoir aucun recouvrement. Une fois les filtres appris, on pourra utiliser les CNN avec les images entières. Ensuite, selon le *pooling* choisi, par exemple le *max-pooling*, on prendra le maximum de toutes les réponses de filtres. On réduit donc ainsi le nombre de fonctions à apprendre. Par exemple, si on garde les 400 fonctions à apprendre par filtre, la figure 10 donnera 4800 fonctions à apprendre, au lieu de 48000 pour la figure 9.

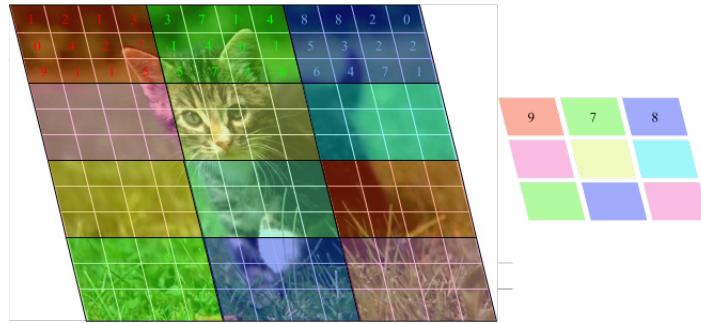


FIGURE 10: Exemple de *max-pooling* avec des zones de  $3 \times 4$  pixels.

## 2.5 Le manque de donnée

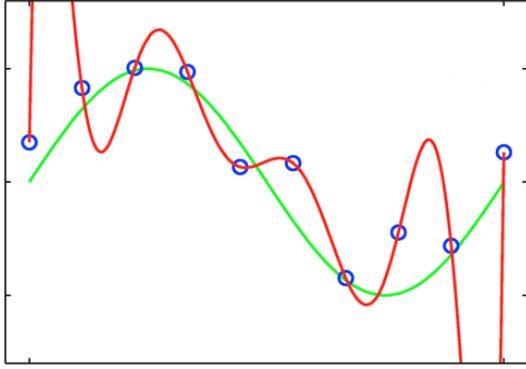
### 2.5.1 Sur-apprentissage et sous-apprentissage

#### *Sur-apprentissage*

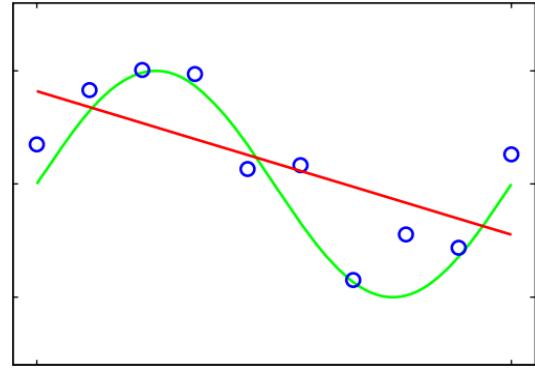
L'apprentissage connaît deux problèmes principaux, le sur-apprentissage (overfitting en anglais) et le sous-apprentissage (underfitting en anglais). Le sur-apprentissage apparaît lorsque le réseau de neurones s'est trop spécialisé. Cela arrive lorsque l'on entraîne le réseau trop de fois sur le même jeu de données. Par exemple, la figure 11a montre un exemple de sur-apprentissage, la courbe rouge étant bien capable de retrouver les données d'entraînement, mais serait incapable de retrouver une donnée se situant sur la courbe verte.

*Sous-apprentissage* Le sous-apprentissage est lui un problème de manque d'entraînement. Comme on peut le voir sur la figure 11b, la courbe rouge ne s'est pas assez spécialisée contrairement au cas du sur-apprentissage.

C'est pourquoi la taille de la base de données utilisée influence beaucoup les résultats.



(a) Sur-apprentissage



(b) Sous-apprentissage

FIGURE 11: Les points bleus sont les données à apprendre, la courbe verte est la fonction que l'on aimerait avoir, et la courbe rouge est la fonction que l'on a obtenue.

### 2.5.2 L'augmentation de données

*Pourquoi chercher à augmenter la base de données ?*

On a besoin de pallier le manque de données car les architectures de réseau de neurones profonds, comme les CNN, ont besoin de beaucoup de données pour pouvoir être entraînés efficacement. Les structures telles que MC-DCNN de Zheng et coll. ([13]) peuvent être utilisées. De plus il y a peu, comparé 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 ([8][6]), tel que la translation, l'inversion, le changement de couleur, de contraste, de luminosité, ... (figure 12). 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, Chatfield et coll. ([3]) 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. Krizhevsky et coll. ([8]) 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 coll. ([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.

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 [8] pour les données

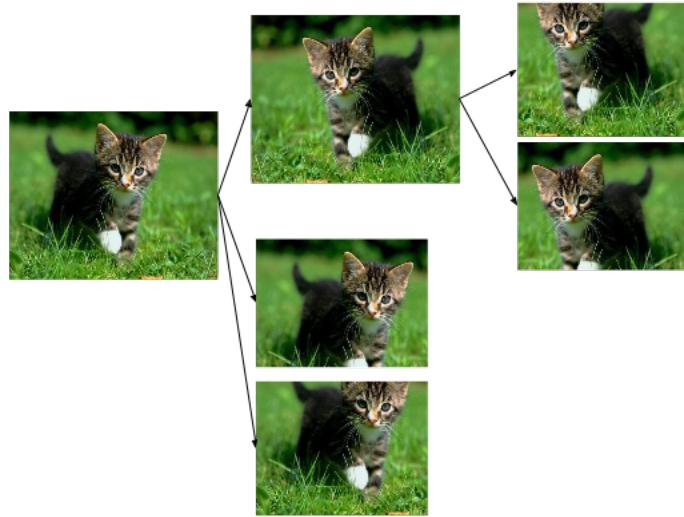


FIGURE 12: Exemple d'augmentation de données pour une image. Ici, on a obtenu six images à partir d'une seule. On a d'abord appliqué une inversion d'image et puis on a rogné les images obtenues de deux manières différentes. En réalité, beaucoup plus de transformations sont appliquées, permettant de multiplier la base de données des images de beaucoup (1024 dans [6]).

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 [8] ou [6].

### 2.5.3 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 de Srivastava et coll. ([12]) introduit cette notion. Le principe est de désactiver certains neurones par couche aléatoirement (figure 13).

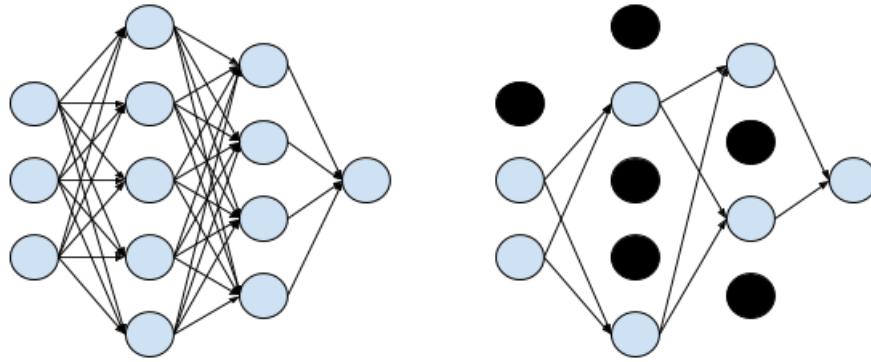


FIGURE 13: Le réseau de neurones à droite est celui de gauche après que l'on ait appliqué le dropout. Comme on peut le voir, on réduit de beaucoup le nombre de connexions entre les neurones.

On applique en fait une probabilité à chaque couche. Par exemple, une probabilité de 0.5 sur la

couche  $l$  d'un réseau de neurones signifie que les neurones de cette couche ont chacun une probabilité de 0.5 d'être activé. On note cette probabilité  $p$  et on a donc la formule (5) qui devient :

$$r_j^{(l)} \sim Bernouilli(p) \quad (10)$$

$$\tilde{a}^{(l)} = r^{(l)} * a^{(l)} \quad (11)$$

$$z_i^{(l)} = \sum_{k=0}^{n^{(l-1)}} w_{ik}^{(l)} \tilde{a}_k^{(l-1)} \quad \forall 1 < l \leq L \quad (12)$$

On comprend donc que la probabilité  $p$  est une valeur influence l'efficacité du dropout. En effet, si  $p$  est trop grand (proche de 1), on se rapproche trop d'un apprentissage classique, et ne présente donc peu d'intérêt. Au contraire, si  $p$  trop petit, le réseau aura beaucoup de difficulté à apprendre car il y a trop peu de neurone présent. Srivastava et coll. ([12]) nous informe que choisir  $p = 0.5$  donne en général de bons résultats.

On peut entraîner un réseau de neurones en combinant le dropout et la *backpropagation*, mais cette *backpropagation* n'est pas appliquée au réseau tout entier, mais à un réseau "aminci" par le dropout. Les poids sont alors partagés entre chaque réseau de neurones aminci. Étant donné que l'on doit générer plusieurs réseaux amincis, il n'est pas étonnant que cette entraînement prend plus de temps qu'un réseau de neurones classique. En effet, entraîner un réseau de neurones avec le dropout prend 2 à 3 fois plus de temps [12]. Lors du test, il faut alors multiplier tous les poids par  $p$  sur le réseau entier.

Le taux d'erreur observé est significativement plus petit en utilisant le dropout que sans [12]. De plus, le dropout permet de prévenir l'apparition de sur-apprentissage.

### 3 Conclusion

Pour conclure , il existe de nombreuses manières d'augmenter le volume des données. Cependant, ces méthodes dépendent du type de donnée. 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 par Zheng et coll. ([13]). Pour cela, nous allons utiliser le framework Caffe (voir [7]). 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] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19 :153, 2007.

- [3] 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.
- [4] 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/](http://www.cs.ucr.edu/~eamonn/time_series_data/).
- [5] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, 2006.
- [6] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv :1312.5402*, 2013.
- [7] 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.
- [8] 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.
- [9] Andrew Ng, Jiquan Ngiam, Chuan Y Foo, Yifan Mai, and Caroline Suen. Ufldl tutorial, 2012.
- [10] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004.
- [11] Chotirat Ann Ratanamahatana and Eamonn Keogh. Three myths about dynamic time warping data mining. In *Proceedings of SIAM International Conference on Data Mining (SDM05)*, pages 506–510. SIAM, 2005.
- [12] 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.
- [13] 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.