UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

MASTER 2 BIG DATA ET FOUILLE DE DONNÉES

APPRENTISSAGE AUTOMATIQUE

DATE : 03/01/2017

Travail de recherche
Deep learning
Auto-encodeurs

Réalisé par

NABIL REDHA BELKHOUS

NUMÉRO D'ÉTUDIANT : 16705491

Enseignant

JEAN-JACQUES MARIAGE

JJMARIAGE@FREE.FR

Table des matières

1 Introduction

Le **machine learning** est une méthode employée en IA afin d'automatiser des traitements ou de trouver des solutions à des problèmes qui n'ont pas de solution algorithmique.

L'une des techniques les plus utilisées sont les **réseaux de neurones**.
Dans un réseaux de neurones, plus il y a de neurones et de connexions entre eux plus l'algorithme sera efficace.

Dans cette optique-là, plusieurs recherches ont donné lieux à des **réseaux multi-couches** comme le MLP.
Le but est de multiplier les couches et construire un modèle polyvalent et puissant.
Mais on s'est vite rendu compte que plus il y a de couches plus le réseau de neurones est complexe et plus il est difficile de l'entraîner.

Si on cite l'exemple de la reconnaissance de formes, une simple image de **1200x1200** donnera **1440000 données** en entrée du réseau de neurones, ce qui implique une grande complexité pour entraîner le réseau sur toutes ses données.
Une alternative à cela, c'est qu'un spécialiste caractérise uniquement les informations pertinentes qu'on va donner en entrée au réseau.
Mais s'il y a des milliers de fichiers comment faire ?

C'est là où interviendra le **deep learning**. En lui donnant les données à l'état brute (par exemple des images et non pas leurs caractéristiques), il va par lui-même extraire les caractéristiques pertinentes des données et faire son apprentissage.
Le **deep learning** est une méthode qui est basée sur le machine learning et plus précisément elle utilise un réseau de neurones multi-couches.
Il permet à des modèles de calculs multi-couches d'apprendre les représentations des données avec plusieurs couches d'abstraction.
Plus concrètement, il permet de découvrir des structures complexes dans de grands ensembles de données. Il utilise les méthodes de rétro-propagation pour indiquer comment changer des paramètres qui serviront au calcul de la représentation d'une couche à partir de sa couche précédente. [8] [Link]

En 2012, à travers la compétition de reconnaissance d'image (**Large Scale Visual Recognition Challenge**) c'est un algorithme de Deep learning qui a remporté la compétition.
C'est l'un des éléments déclencheurs de la vague deep learning qu'on connaît actuellement (Facebook, Google, Amazon, ...) avec le projet de deep learning de google **Google Brain** qui est arrivé à reconnaître le concept de chat par lui même (non supervisé).

Le Deep learning a fait ses preuves notamment dans le domaine de la reconnaissance de voix et la reconnaissance de formes.

Il existe diférentes techniques et méthodes qui sont utilisées dans le domaine du deep learning parmis lesquelles on peut citer pour la reconnaissance de formes, les **réseaux convolutionnels (ConvNet)** qui ont donné de très bons résultats pour la détection, la segmentation et la reconnaissance d'objet et de région dans une image.

On peut aussi parler des **Deep Auto-encodeurs** qui traites différentes problématiques (reconnaissance de formes, reconnaissance de voix) avec différentes techniques. Mais l'un des aspects les plus intéressants est la capacité de **réduction de dimensions** de données pour leurs traitements.

C'est pour cela que j'ai choisi de faire un travail de recherche sur **les auto-encodeurs**.
À travers ce document je présenterai dans un premier temps les auto-encodeurs basiques, leurs caractéristiques, et certains types d'auto-encodeurs puis dans un second temps je parlerai de l'utilisation des auto-encodeur dans le cadre du deep learning.

2 Auto-encodeurs

Les auto-encodeurs sont des réseaux de neurones qui appartiennent à la famille de **l'apprentissage non supervisé**.

Ils sont de type **«feed-forward»**[2][Link] c.à.d que les connexions entre les unités des différentes couches ne forment pas de cycle. En d'autres termes l'information se déplace à partir des nœuds d'entrée à travers les nœuds des couches cachées jusqu'aux nœuds de sortie.

Les auto-encodeurs sont entraînés à **reproduire l'input en sortie du réseau** à travers une architecture à trois couches.

1. **La couche d'entrée** : qui représente les données qu'on fournit en entrée au réseau.
2. **La couche cachée** : qui est une compression de l'input où on ignore les parties qui ne sont pas nécessaires à le reproduire en sortie.
3. **La couche de sortie** : qui représente la reconstitution de l'input.

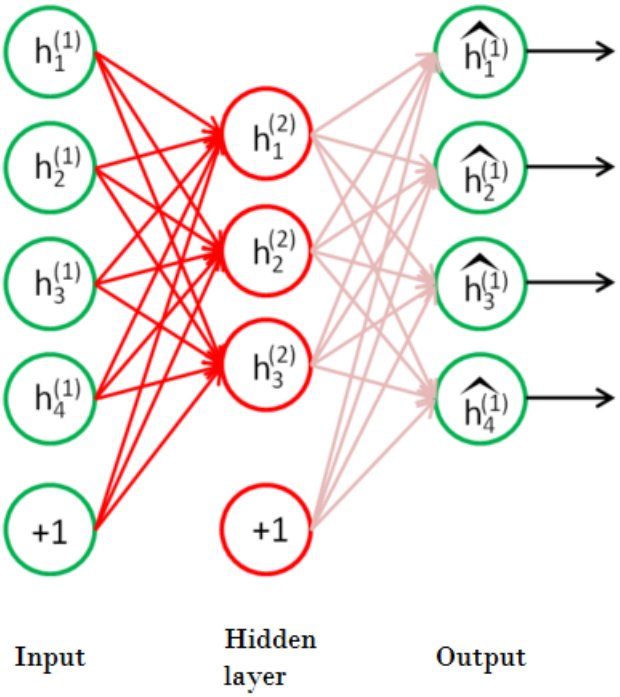

FIGURE 1 – Architecture d'un auto-encodeur

Les auto-encodeurs ont une caractéristique particulière qui est la **compression des données**.

On **compresse** les données entrées dans le réseau, on **extrait** les caractéristiques les plus intéressantes puis on **décompresse** les données.

Ceci est résumé par le schéma suivant :

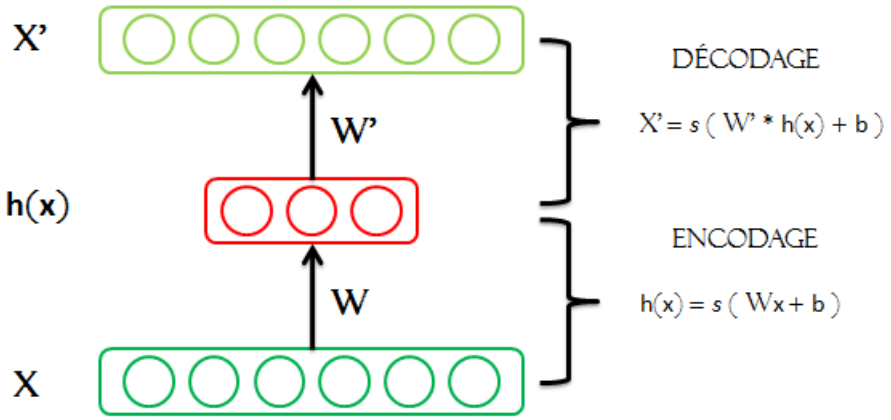

FIGURE 2 – Phase d'encodage et de décodage d'un auto-encodeur

La fonction d'**encodage** notée $\mathbf{h(x)} = \boldsymbol{s}(\ \mathbf{W}_X + \mathbf{b}\ )$ est celle qui va réaliser le mapping de l'input X avec une couche cachée[6][Link] ou **s** est une fonction d'activation généralement le sigmoïde d'une transformation linéaire , $\mathbf{W}_X$ est le poids correspondant à X dans la matrice des poids et un certain biais **b**.

La fonction de **décodage** notée $\mathbf{Y} = \boldsymbol{s}(\mathbf{W'} * \mathbf{h(x)} + \mathbf{b}\ )$ est celle qui va faire la mapping entre la couche cachée et l'output prédit. **W'** est la transposition de la connexion réalisée lors du passage à la couche précédente, **h(x)** le résultat de l'encodage et un certain biais **(Un biais est un neurone dans lequel la fonction d'activation est en permanence égale à 1)**.

Maintenant afin d'entraîner notre auto-encodeur, il faut déterminer ce qu'on appelle une **Loss Function**[6][Link] qui est une fonction qui va **comparer l'input et l'output** de notre réseau pour mesurer la qualité de la reconstruction.

Grâce à cette fonction on va **entraîner** notre réseau de neurones à minimiser ce résultat en utilisant la méthode de **rétro-propagation du gradient descendant** afin que l'output ressemble le plus à l'input donné au réseau.

Cependant, dans certains cas, les auto-encodeurs peuvent utiliser pour la phase d'entraînement des méthodes autres que la rétro propagation comme la **recisrculation**[3][Link] malgré qu'ils soient de type feed-forward.

Loss function est spécifiée selon le type des données[6][Link] :
Par exemple :
— pour les données binaires la $lossFunction$ est définie par :

$$lossFunction(Y) = -\sum_k (x_k \times log(x'_k)) + (1 - x_k) \times log(1 - x'_k)$$

— pour les données réels la $lossFunction$ est définie par :

$$lossFunction(Y) = \frac{1}{2}\sum_k (x'_k - x_k)^2$$

3 Familles d'auto-encodeurs

Dans la littérature on est habitué à voir des auto-encodeurs avec une architecture symétrique où les couches sont plus petites à chaque niveau, on parle alors de **Under-complet hidden layer**.

Cependant il existe une autre architecture où cette fois les couches sont plus grandes à chaque niveau, on parle là de **Over-complet hidden layer**.

3.1 Under-complet hidden layer [3][Link] [4][Link]

Parmis les caractèristiques d'une topologie **Under-complet hidden layer** on trouve :

— Les couches cachées sont plus petites que l'input. Elles sont de plus en plus petites d'une couche à une autre.
— Les couches cachées ont pour rôle de compresser les données.
— Le réseau va être entraîné à compresser uniquement les données qui sont présentes dans l'ensemble d'entraînement.
— Les unités de la couche cachée permettent d'extraire de bonnes caractéristiques à partir de l'ensemble d'enraiement.
Cependant cette topologie donne de mauvais résultats avec des données qui ne viennent pas de l'ensemble d'entraînement.

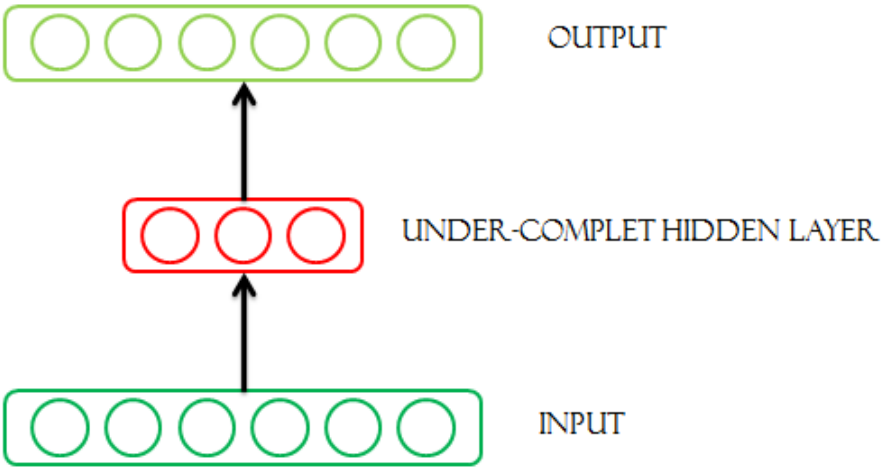

FIGURE 3 – Topologie Under-complet hidden layer

3.2 Over-complet hidden layer [3][Link] [4][Link]

— Les couches cachées sont plus grandes que l'input. Elles sont de plus en plus grandes d'une couche à une autre.
— Les couches cachées ne compressent plus les données.
— Le réseau va être entraîné à compresser uniquement les données qui sont présentes dans l'ensemble d'entraînement.
— Chaque unité des couches cachées peuvent copier des données de l'input ou ajouter des données aléatoirement.
— Cette méthode ne garantit pas que les unités des couches cachées vont extraire des structures significatives.

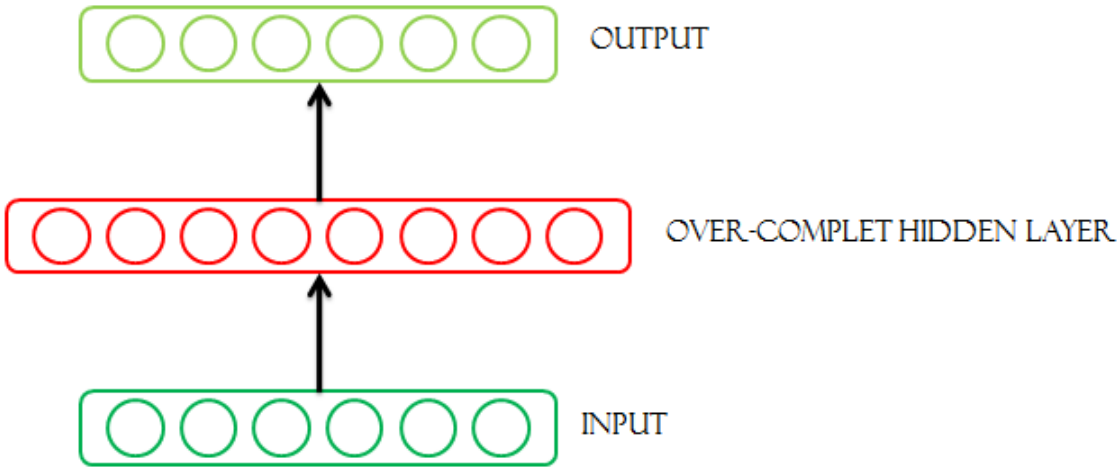

FIGURE 4 – Topologie Over-complet hidden layer

4 Type d'auto-encodeurs

Il existe plusieurs types d'auto-encodeurs, on peut citer : **auto-encodeur basique**, **denoising auto-encodeurs**, **contractive auto-encodeurs**, **transforming auto-encodeurs**, **sparsed auto-encodeurs**, **...**

Dans mes recherches je me suis consentré sur trois types d'auto-encodeurs : **Basique**, **débruité** et **contractive**.

4.1 Auto-encodeur basique

C'est le type d'auto-encodeur que j'ai décrit dans la partie 2 du document.

4.2 Denoising autoencoder [6][Link]

Le denoising auto-encodeur ou **auto-encodeur débruité** est une variante de l'auto-encodeur basique qu'on a vu précédemment sauf qu'il a une représentation **over-complete** de la couche cachée.

L'idée est de construire un input **X'** qui est une version bruitée de l'input initial **X**.

La construction de la copie corrompue peut se faire en copiant l'input original et on lui ajoutant des données aléatoirement avec la méthode d'ajout de **bruit gaussien** par exemple.

Le but est d'essayer de reproduire l'input **X** en sortie du réseau mais à partir de **X'**. En d'autres termes c'est la version bruitée de l'input qui va être transmise aux couches cachées et la **Loss function** va comparer l'output du réseau avec l'input non bruité **X**.

Les couches cachées auront pour objectif d'étudier les différentes corrélations entre les unités afin de pouvoir prédire et corriger les données bruitées, améliorer la reconnaissance des caractéristiques et reconstruire correctement l'input.

Ce type d'auto-encodeur permet grâce à son architecture **over-complet** d'extraire des informations de la structure des données et d'apprendre plus de caractéristiques.

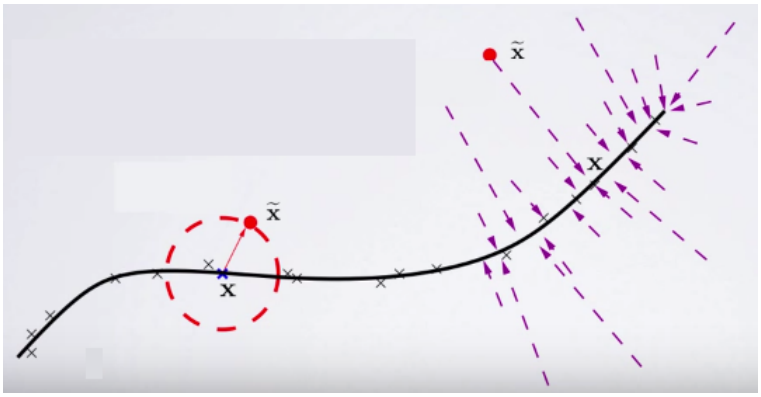

FIGURE 5 – Exemple concret d'un denoising auto-encodeur

Si on prend l'exemple de ces données qui représentent une courbe et que pour chaque input **X** on ajoute du bruit pour l'éloigner de la courbe, le denoising auto-encodeur va apprendre à projeter la donnée à son emplacement exact et cela nous permettra **d'apprendre des informations sur la structure et la forme des données**.

4.3 Contractive autoencoder[7][Link]

Le contractive auto-encodeur est aussi une variante de l'auto-encodeur basique mais avec une représentation over-complet des couches cachées.

A l'inverse du denoising auto-encodeur où on va essayer d'extraire de nouvelles caractéristiques, on va extraire uniquement les caractéristiques qui reflètent **les variations observées dans les données d'entraînement**.

Une nouvelle Loss function la $NewLossFunc$ est définie :
$NewLossFunc$ = lossFunction (Y) + $\lambda||\nabla_{x^{(t)}} h(x^{(t)})||_F^2$

elle reprend la loss function qu'on a vu pour les auto-encodeurs basiques :
lossFunction (Y) = - $\sum_k (x_k \times log(x'_k)) + (1 - x_k) \times log(1 - x'_k)$

— et la jacobienne de l'encodage **(H(x))** :

$|| \nabla_{x^{(t)}} h(x^{(t)})||_F^2 = \sum_j \sum_k (\frac{\partial h(x^{(t)})_j}{\partial x_k^{(t)}})^2$ C'est l'addition de la Loss Function et la jacobienne qui va nous permettre de garder uniquement les caractéristiques qui reflètent les variations observées dans les données.

Soit un ensemble de données qui sont les différentes variations du nombre deux.

Si on prend l'exemple de la rotation des images, les unités qui s'occupent de prendre en considération la rotation vertical (en jaune) vont être préservées car on remarque que les deux varient selon la rotation verticale. tandis que pour les unités qui s'occupent de regarder la rotation horizontale, ils ne seront pas pris en compte car nos données ne varient pas horizontalement.

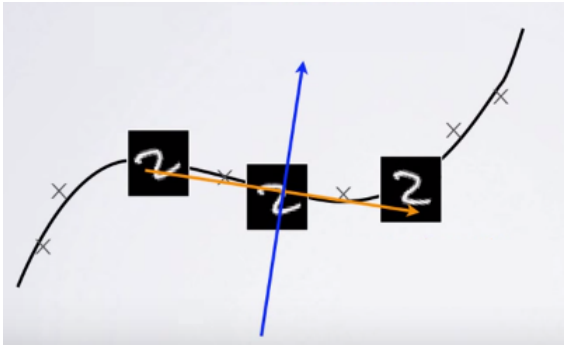

FIGURE 6 – Exemple concret d'un contractive auto-encodeur

5 Deep auto-encoders

5.1 Principe

Le deep auto-encoder network ou **réseau d'auto-encodeur profond** est un apprentissage non supervisé qui n'est qu'un auto-encodeur multi-couches.

Selon le nombre de couches cachées, si l'auto-encodeur à plus d'une couche, il est considéré comme deep (profond)[5][Link]

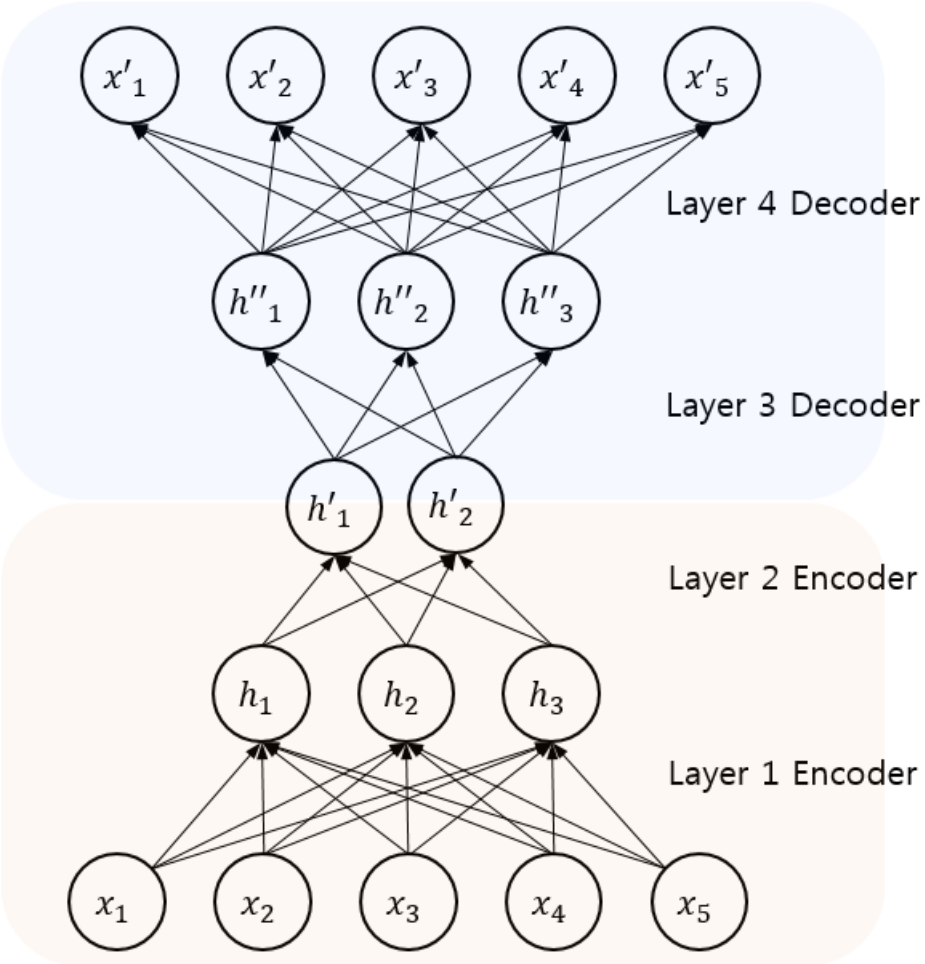

FIGURE 7 – Deep auto-encodeur

5.2 Problèmes et solutions

L'un des problèmes liés à l'utilisation des architectures profondes est la volumétrie des données.

Auparavant la méthode Principal Component Analisis PCA était un moyen de réduire la dimension des données. Elle consiste à trouver les variables liées entre elles et les transformer en une variable décorrélée afin de réduire la dimension des données [1][Link].

Cependant la compression de données effectuées dans les Deep auto-encoders donne de meilleurs résultats de compression grâce aux phases d'encodages des couches cachées.

Résultat obtenu avec les deux méthodes avec les mêmes données :

— Avec PCA :

FIGURE 8 – Résultat de clustering avec PCA

— Avec auto-encodeur :

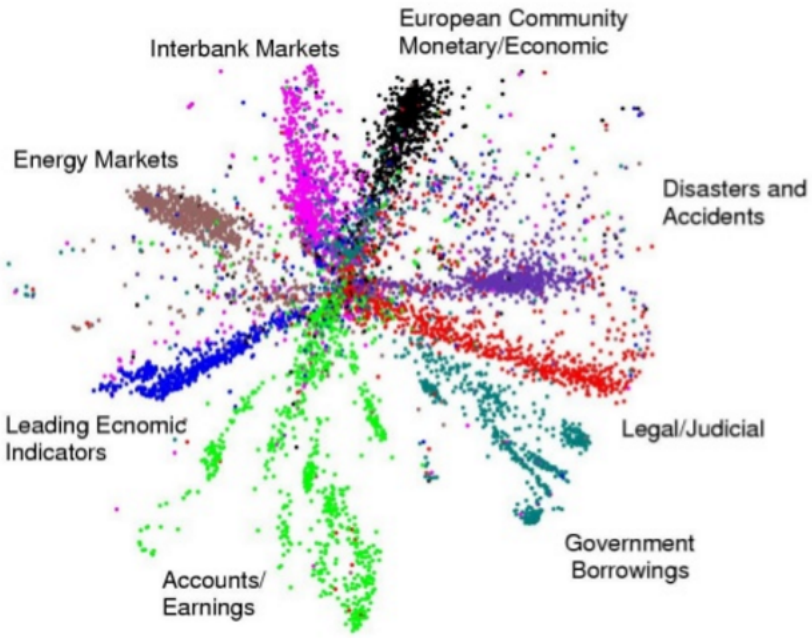

FIGURE 9 – Résultat de clustering avec les Auto-encodeurs

La réduction de dimension facilite la classification, la visualisation, la communication, et la sauvegarde de données multidimensionnelles[1][Link].

Les auto-encodeurs sont souvent entraînés en utilisant des méthodes de back-propagation comme la rétro propagation du gradient.

Malgré son efficacité, il y a malheureusement des problèmes avec l'utilisation de la rétro-propagation car elle est dépendante de la phase d'initialisation des poids (Si l'initialisation des poids se rapproche de la bonne solution, la retro-propagation permet de faire un bon apprentissage, si ce n'est pas le cas les résultats ne sont pas performants).

En effet lors de l'entraînement du réseau avec plusieurs couches cachées, les erreurs sont rétro-propagées aux premières couches et elles deviennent petites et l'entraînement est inefficace.
Même avec une rétro-propagation avancée, l'apprentissage reste lent surtout lorsqu'on restreint la quantité de données d'entraînement[1][Link].

Ce problème peut être allégé en faisant des pré-entrînements sur chaque couche comme une simple couche.

Une nouvelle méthode de pré-entraînement a été mise en place afin d'améliorer l'initialisation des poids des neurones. Elle consiste à faire un pré-entraînement sur chaque couche de l'auto-encodeur avec un apprentissage non supervisé en utilisant la **machine de bolzmann réduit RBM** pour donner une meilleure initialisation des poids sur chaque couche.

Si on prend l'exemple d'un réseau à deux couches cachées, chaque couche sera entraînée avec une **RBM** et elle servira d'input à la couche suivante.

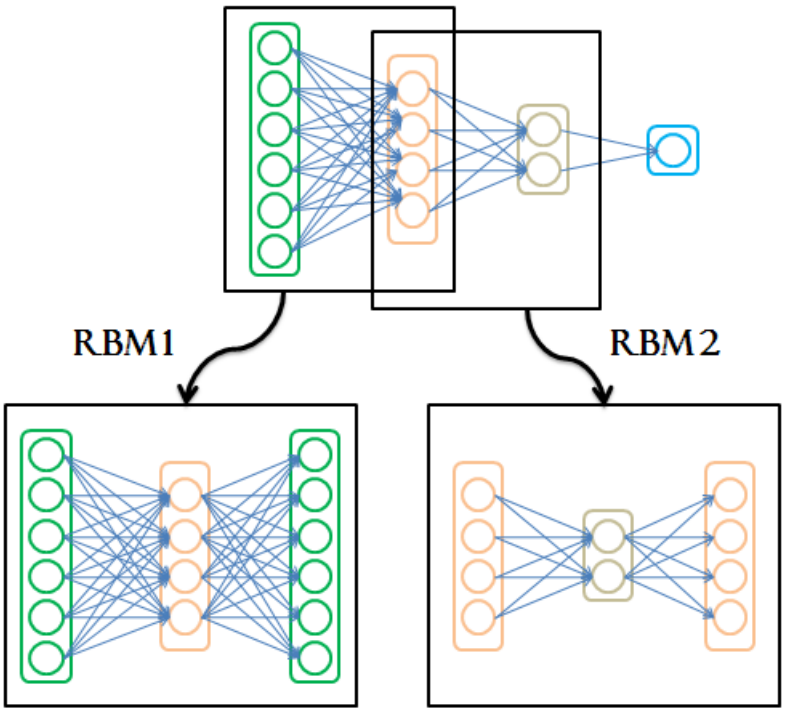

FIGURE 10 – Pré-entraînement avec la Machine de Bolzmann Réduite(RBM)

Après la phase de pré-entraînement qui va déterminer les poids idéaux au niveau des connexions entre les nœuds des différentes couches, on aura la phase d'encodage puis de décodage. On conclut avec la phase d'entraînement du réseau en utilisant la rétro-propagation du gradient descendant.

Le schéma tiré de l'article **Reducing the Dimensionality of Data with Neural Networks, G. E. Hinton and R. R. Salakhutdinov** [1][Link] résume bien les choses :

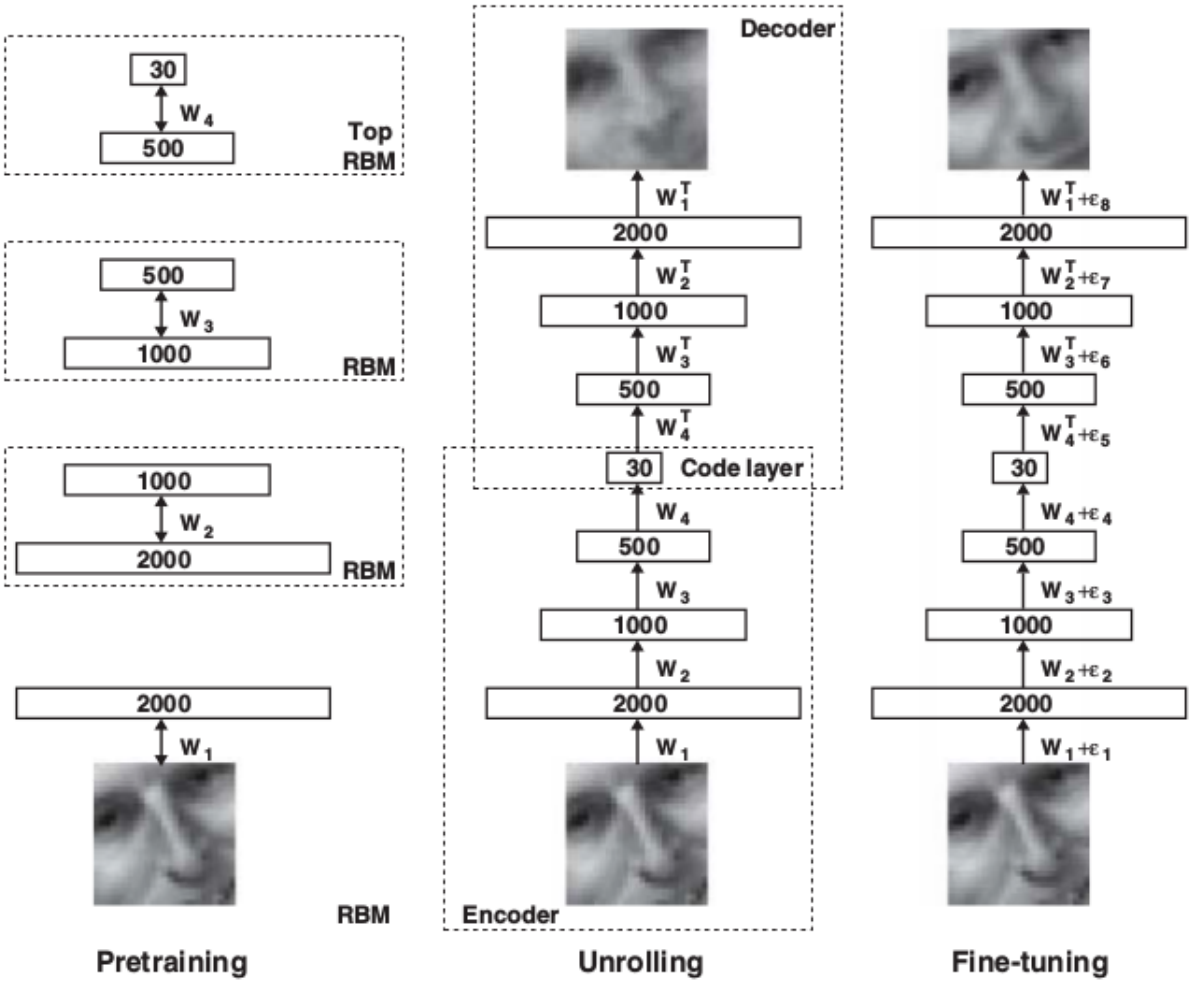

FIGURE 11 – Deep auto-encodeur avec pré-entrînement

Avec :

- La phase de pré-entraînement qui est un apprentissage non supervisé sur chaque couche du réseau.
- Unrolling : dès que toute les pré-entraînements sont faits la phase de codage et de décodages sont prêtes.
- Fine-tunning : Une fois la tâche du réseau d'ento-encodeur profond effectuée, la rétro-propagation est appliquée afin de réduire les erreurs au niveau des unités des couches cachées. Dans ce cas-là, la rétro-propagation donne de bons résultats

6 Conclusion

Le but de mon travail de recherche était de synthétiser le concept d'auto-encodeur dans le contexte du deep learning.

A travers mes nombreuses recherches j'ai pu constater que les auto-encodeurs :

— faisaient partie de **l'apprentissage automatique**.

— étaient des réseaux de type **feed-forward**.

— existaient sous différents types : **basic, denoising, contractive, sparse, transforming, ...**

— avaient deux topologies : **Under-complet hidden layer** et **Over-complet hidden layer**.

— utilisaient les méthodes de **rétro-propagation** pour entraîner le réseau.

— pouvaient recourir à des techniques de **pré-entrainement** pour pallier au problèmes qu'on peut avoir avec les méthodes de retro-propagation.

— étaient une bonne solution pour faire de l'apprentissage sur des **données multi-dimensionnelles** grâce à la compression de données réalisée par les couches cachées.

— donnaient de **meilleurs résultats que le Principal Component Analisis** pour le clustering.

Références

[1] R. Salakhutdinov G. E. Hinton. Reducing the dimensionality of data with neural networks. pages 504–507, 2006.

[2] S. D. Wang G. E. Hinton, A. Krizhevsky. Transforming Auto-encoders. *ICANN-11 : International Conference on Artificial Neural Networks, Helsinki.*, 2011.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[4] Hugo Larochelle. Neural networks [6.5] : Autoencoder - undercomplete vs. overcomplete hidden layer, 2013.

[5] Dong Yu Li Deng. *Deep Learning : Methods and Applications*. Foundations and Trends in Signal Processing : Vol. 7, 2014.

[6] I. Lajoie Y. Bengio P.-A. Manzagol P. Vincent, H. Larochelle. Stacked Denoising Autoencoders : Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research, vol. 11*, pages 3371–3408, 2010.

[7] Xavier Muller Xavier Glorot Yoshua Bengio Salah Rifai, Pascal Vincent. Contractive auto-encoders : Explicit invariance during feature extraction. 2011.

[8] Geoffrey Hinton Yann LeCun, Yoshua Bengio. Deep learning. *REVIEW INSIGHT : Annals of Mathematical Logic*, pages 436–442, 2015.