Autoencodeurs : comparaison avec d'autres méthodes de réduction de la dimension

Damien BABET, Julie DJIRIGUIAN,
Techniques avancées d'apprentissage
compte-rendu de projet

22 mai 2017

1 Qu'est-ce qu'un auto-encodeur ?

1.1 Principe et architecture d'un auto-encodeur

Les auto-encodeurs sont des réseaux de neurones constitués de deux phases. La première vise à compresser une information alors que la seconde effectue l'opération inverse, c'est-à-dire qu'elle cherche à reconstituer l'information initiale. La Figure 1 fournit un exemple de l'architecture d'un auto-encodeur simple. Les deux phases peuvent toutefois être constituées de plusieurs couches, comme sur la Figure 3.

La structure du réseau de neurones se caractérise par une égalité entre le nombre d'*inputs* et le nombre d'*outputs*. La couche cachée du milieu correspond à l'information compressée. Cette couche contient par conséquent le nombre d'unités le plus faible. Un auto-encodeur présente une architecture symétrique par rapport à cette couche centrale, c'est-à-dire que les différentes couches de l'étape de compression ont toutes une couche équivalente dans la couche de décompression.

Le passage d'une couche à une autre s'opère selon le processus schématisé sur la Figure 2. En effet, l'unité de la couche en sortie notée x_j^{n+1} est égale à la fonction d'activation de la combinaison linéaire des unités x_i^n de la couche en entrée où i est compris entre 1 et l avec l le nombre d'unités de la couche en entrée.

Chaque unité de la couche $n+1$ est donc calculée selon la formule suivante :

$$x_j^{n+1} = \varphi(\sum_{i=1}^{l} w_{ij} x_i^n)$$

Où $\varphi(x)$ est la fonction dite d'activation. Si $\varphi(x)$ est linéaire, l'auto-encodeur est lui-aussi linéaire. En revanche, si $\varphi(x)$ est non-linéaire, comme dans le cas de la fonction sigmoïde, l'auto-encodeur est non-linéaire.

Une fois que l'architecture de l'auto-encodeur est fixée, l'objectif est de déterminer l'ensemble des poids w_{ij} tels que la fonction de perte soit minimisée, par exemple :

$$\boxed{argmin_{w_{ij}} ||\hat{X}^{AE} - X||^2}$$

Où X est la donnée en entrée et $\hat{X}^{AE}$ est la donnée en sortie de l'auto-encodeur.

Dans le cas d'une fonction d'activation linéaire, le programme de minimisation peut s'écrire sous la forme matricielle suivante :

$$argmin_{w_{ij}} ||X(W'W - I)||^2$$

Nous pouvons illustrer ces explications avec l'exemple d'une image de dimension 28x28. Les *inputs* sont constitués par les 784 pixels de l'image et l'*output* comprend lui-aussi 784 unités qui doivent ainsi s'approcher le plus possible des 784 unités utilisées en entrée. En revanche, la couche cachée du milieu correspond aux données compressées et sa dimension est largement inférieure à 784. Sur le schéma de la Figure 3, notre auto-encodeur est composé de cinq couches cachées. La couche centrale, qui contient l'image compressée, comprend 32 unités.

1.2 Les différents types d'auto-encodeurs

Le terme auto-encodeur peut désigner une gamme plus large de réseaux de neurones que ceux qui viennent d'être décrits [2] [5]. Trois variations importantes sont notamment possibles : la régularisation, le débruitage et l'auto-encodeur stochastique.

Tout d'abord, la régularisation ne force pas l'auto-encodeur à parvenir à une représentation de faible dimension à l'aide d'une couche centrale avec un faible nombre d'unité (auto-encodeur « sous-complet ». Il vise à construire un auto-encodeur avec des couches larges, et même plus larges que les données (auto-encodeur « sur-complet »), mais en imposant une contrainte de régularisation pour éviter que le réseau se contente d'apprendre la fonction identité, par exemple une pénalité de sparsité. Ensuite, un auto-encodeur stochastique vise à produire en sortie une distribution de probabilité, à partir de laquelle on peut générer des échantillons, plutôt qu'un simple décodage. Le « code » correspond alors à des variables latentes qui résument la distribution (approximée) des données. Nous allons nous pencher plus particulièrement sur un dernier type, le *denoising* auto-encodeur.

De nos jours, les auto-encodeurs sont essentiellement utilisés pour la réduction de dimension afin d'améliorer la visualisation et pour le débruitage des données. Cet objectif, dit de *denoising*, vise à nettoyer l'information principale de la donnée initiale en enlevant le bruit contenu et en conservant seulement les caractéristiques principales de celle-ci. Afin de réaliser un *denoising* auto-encodeur, nous créons un réseau de neurones sur le même principe que celui décrit sur la Figure 3. Cependant, la donnée en entrée du réseau de neurones est préalablement bruitée en ajoutant, aux valeurs initiales, un terme résiduel gaussien généré aléatoirement. Le *denoising* auto-encodeur peut être considéré comme une sous-catégorie d'auto-encodeur régularisé, le bruit jouant ici un rôle de régularisation. A ce titre, il peut être complet ou sur-complet.

1.3 Méthodes d'entraînement

Comme tous les réseaux de neurone, les auto-encodeurs peuvent également admettre différentes fonctions d'activation, différentes fonctions de perte, et différents algorithmes d'optimisation pendant l'entraînement. Les réseaux profonds, avec de nombreuses couches, sont plus difficiles à entraîner et l'algorithme classique de *backpropagation* du gradient a longtemps constitué un obstacle pour construire des réseaux profonds. Les auto-encodeurs n'échappent pas à ce problème. Pour le surmonter, Hinton et Salakhutdinov [3] utilisaient en 2006 une méthode de pré-entraînement couche par couche à l'aide de *Restricted Boltzmann machines* (RBM). Nous allons présenter rapidement cette méthode, puis la technique d'optimisation que nous avons effectivement mise en œuvre.

Une RBM est un réseau de neurone probabiliste bipartite, c'est-à-dire constitué de deux couches reliées l'une à l'autre mais sans connexion au sein de chacune, une couche de données visibles et une couche de variables latentes. Pour des paramètres de poids donnés, on associe une distribution de

probabilité (difficilement calculable) sur les états des unités des deux couches. L'entraînement du réseau vise à trouver les poids qui maximisent la probabilité des données effectivement observées (inputs de la couche visible). Le succès des RBM réside dans la possibilité de calculer facilement les probabilités conditionnelles d'une couche étant donnée l'autre couche, ce qui permet de générer des tirages aléatoires selon une loi de probabilité jointe qui reste inconnue. On peut alors concevoir la couche des variables latentes comme un encodage probabiliste des données. Hinton et Salakhutdinov utilisent donc une RBM pour pré-entraîner chacune des couches de leur encodeur, rendu déterministe, puis « retourné » pour constituer le décodeur. L'apprentissage plus classique (descente de gradient et *backpropagation*) peut alors commencer à partir de ce point de départ raisonnable.

Nous n'avons pas cherché à implémenter cette technique. Nous nous sommes appuyés sur des algorithmes d'optimisation plus récents. En pratique, nous utilisons l'optimiseur *Adadelta* [4], en comptant sur le fait que les progrès des algorithmes de gradient stochastique (et des capacités de calcul) compensaient l'abandon de la méthode des RBM. L'optimiseur *Adadelta* est un algorithme de gradient stochastique par *batch* (ou « mini-lot ») : à chaque étape, un mini-lot de données est sélectionné aléatoirement et un gradient global sur le lot est calculé. Les paramètres (les poids de l'auto-encodeur) sont mis-à-jour dans la direction du gradient, proportionnellement à un taux d'apprentissage. L'intérêt d'*Adadelta* réside dans ce taux d'apprentissage. En effet, ce dernier est propre à chaque paramètre et dynamique, comprenant un numérateur d'« inertie » qui s'accroît lorsque les mises à jour récentes sont importantes (pour ce paramètre) et un dénominateur qui augmente lorsque les gradients récents sont élevés (pour ce paramètre), ce qui favorise la mise à jour des paramètres pour lesquels le gradient est rarement élevé.

2 De la factorisation matricielle aux auto-encodeurs

2.1 La factorisation matricielle

L'abondance actuelle des données de grande dimension et de nombreuses applications font naître le besoin de réduire la dimension des données exploitées. On cherche une approximation (en fonction d'une distance choisie) de plus faible dimension (ie. avec moins de variables). Cette réduction de dimension peut être un objectif (par exemple pour visualiser les données) ou un moyen pour d'autres types d'analyses (pour jouer par exemple un rôle de sélection de modèle). Le problème de la réduction de dimension ainsi posée est peu spécifié. On ajoute couramment une hypothèse de linéarité : les nouvelles variables sont des combinaisons linéaires des variables originales. Le problème devient alors un problème d'approximation de faible rang, qui équivaut à une factorisation de matrice.

Soit $\mathbf{X} \in \mathbb{R}^{n*p}$ une matrice de données. L'approximation de faible rang consiste à l'approximer par une matrice $\mathbf{X}_k$ de rang $k < min(n, p)$. Par définition du rang, $\mathbf{X}_k$ peut s'exprimer comme le produit de deux matrices de plus faibles dimensions : $\mathbf{X}_k = \mathbf{PQ}, \mathbf{P} \in \mathbb{R}^{n*k}, \mathbf{Q} \in \mathbb{R}^{k*p}$. Réciproquement, une version linéairement réduite ou encodée des données sous forme d'une matrice $\mathbf{P}$ peut être étendue en une approximation de faible rang des données originales $\mathbf{X}_k$.

Ainsi posée, l'approximation qui minimise la distance selon la norme de Frobenius et selon la norme spectrale est obtenue directement par l'analyse en composantes principales (ACP, voir ci-dessous), ou par la décomposition spectrale (SVD) équivalente et qui a de meilleures propriétés numériques.

Cependant, dans de nombreuses applications des contraintes supplémentaires sont ajoutées [1], et il n'existe plus de solution en forme close. Par exemple, en ajoutant une contrainte de positivité sur les données et les éléments des matrices facteurs, on obtient le problème NMF (*Non-negative Matrix Factorization*) qui est NP-difficile en général. De même le problème de clusterisation *k-means* peut être

1. Une longue liste en est donnée sur `https://sites.google.com/site/igorcarron2/matrixfactorizations`

formulé comme un problème de factorisation de matrice, avec la contrainte que $\mathbf{P}$ représente la partition des observations dans les clusters ($\mathbf{P}$ est une matrice de 0 ou de 1 et $\mathbf{PP}^T$ est une matrice diagonale). Ce problème est également NP-difficile. Dernier exemple, la fonction de perte peut varier selon les observations, les variables ou les deux, selon une matrice de poids. Le problème d'approximation pondérée de faible rang est, lui-aussi, NP-difficile si la matrice des poids est de rang supérieur à 1.

Ces différents problèmes n'ayant pas de solution simple, la question peut se poser de la pertinence d'un recours à l'auto-encodeur pour les résoudre. Dans le cas général de l'approximation de faible rang sans contrainte particulière, nous allons voir que l'ACP constitue souvent une méthode préférable à l'auto-encodeur.

2.2 Le principe de l'Analyse en Composantes Principales (ACP)

Soit $\mathbf{X} \in \mathbb{R}^{n*p}$ une matrice de données centrées. On cherche à l'approximer par une matrice $\mathbf{X}_k$ de rang $k < min(n, p)$. D'après le théorème d'Eckart-Young, la meilleure approximation au sens de la perte quadratique (norme de Frobenius) est celle obtenue par ACP et donnée par $\widehat{\mathbf{X_k}} = \mathbf{XWW}^T$ où $\mathbf{W} \in \mathbb{R}^{p*k}$ est la matrice des k vecteurs propres de $\mathbf{X}^T\mathbf{X}$ associés aux k plus grandes valeurs propres. $\mathbf{X}^T\mathbf{X}$ est carrée, symétrique et réelle, elle est donc diagonalisable dans une base orthonormée : les vecteurs propres de $\mathbf{W}$ constituent une base orthonormé du sous-espace du dimension k où se trouvent les lignes de $\widehat{\mathbf{X_k}}$. Cette meilleure approximation est unique si $\mathbf{W}$ est unique, donc si la k^{ieme} valeur propre de $\mathbf{X}^T\mathbf{X}$ est strictement supérieure à la $k + 1^{ieme}$.

On peut exprimer cette approximation comme le résultat d'une factorisation de matrices, selon les notations précédentes, en prenant $\mathbf{P} = \mathbf{XW} \in \mathbb{R}^{n*k}$ et $\mathbf{Q} = \mathbf{W}^T \in \mathbb{R}^{k*p}$

2.3 Un auto-encodeur « simple » est équivalent à une ACP

Si nous construisons un auto-encodeur linéaire de profondeur 1, dense, avec k unités dans la couche cachée, et sans termes de biais, un apprentissage qui minimise l'erreur quadratique de reconstruction admet la même meilleure approximation de rang k comme optimum global. Soit en effet $\mathbf{W'} \in \mathbb{R}^{p*k}$ la matrice des poids, l'encodage des données est $\mathbf{XW'}$ et la sortie de l'auto-encodeur est $\mathbf{XW'W'}^T$ (si les poids sont symétriques). $\widehat{\mathbf{X_k}}$ est donc une valeur possible (en particulier pour $\mathbf{W} = \mathbf{W'}$), minimise la perte, et est unique sous la condition ci-dessus. Si l'algorithme d'apprentissage est convergent, il converge vers la même approximation que celle obtenue par l'ACP.

En revanche, comme il n'y a pas de raison pour que $\mathbf{W'}$ tende vers $\mathbf{W}$, la factorisation $\widehat{\mathbf{X_k}} = \mathbf{XW'W'}^T$ n'admet généralement pas une solution unique en $\mathbf{W'}$ (nous pouvons par exemple en permuter les k colonnes). Autrement dit, les deux méthodes de réduction de la dimension n'identifient pas les mêmes variables latentes, mais ces variables forment le même sous-espace de dimension k. La différence essentielle est que l'ACP impose deux contraintes aux composantes qui assurent l'unicité de la décomposition : elles sont orthogonales les unes aux autres, et elles sont ordonnées par taille de la valeur propre associée. Ces contraintes n'existent pas pour l'auto-encodeur. Les variables latentes identifiées peuvent donc être différentes.

Cette preuve est étendue dans l'article [1] : elle est valide si les données ne sont pas centrées (et en admettant des termes de biais), si la couche cachée reçoit une fonction non-linéaire (mais qui peut être approximée par une fonction linéaire à proximité d'un certain point) des données d'entrée, et même si plusieurs couches linéaires ou non, précèdent la couche d'encodage. Sous ces conditions, les auto-encodeurs sont donc des équivalents moins efficaces de l'ACP, qui demandent un long apprentissage et peuvent éventuellement rester bloqués dans des optimum locaux.

En revanche si le lien entre la dernière couche et la sortie est non linéaire, l'auto-encodeur offre une gamme d'encodage plus large que l'approximation de faible rang que permet l'ACP. Alors que l'auto-encodeur linéaire poursuit la même approximation quel que soit le nombre de couches, un auto-encodeur non linéaire gagne en richesse lorsqu'il gagne en profondeur.

3 Les auto-encodeurs : une extension non linéaire de la factorisation matricielle

Si les réseaux de neurones utilisés de nos jours sont à la fois non linéaires en raison des fonctions d'activation non linéaires et à la fois constitués d'une multitude de couches afin d'apprendre et de d'identifier aussi bien que le cerveau humain, mettre en œuvre un auto-encodeur avec un architecture non linéaire complexe paraît cohérent. Nous choisissons dans nos applications, développées dans les sections suivantes, de comparer les performances d'un auto-encodeur non linéaire à celles d'un auto-encodeur linéaire ou d'une ACP. Nous optons pour un empilement de couches totalement connectées (*full connected layers*, c'est-à-dire que les neurones de la couche n sont connectées à toutes les unités activées de la couche précédente $n-1$). Nos expérimentations pourraient, par exemple, être prolongées en ajoutant des couches de convolutions, souvent utilisées pour l'analyse d'images en raison de leurs performances pour l'analyse d'images et des couches de *dropout* qui conduisent à ne connecter aléatoirement qu'une partie des unités activées à une unité de la couche suivante.

3.1 Nos expérimentations confortent l'équivalence entre l'auto-encodeur linéaire « simple » et l'ACP

Tous nos code se trouvent, sous forme de *notebooks* python, sur notre *github* à l'adresse `https://github.com/DamienBabet/Auto-encodeurs`. Nous faisons ici une rapide présentation de notre démarche et de nos principaux résultats [2]. Nous avons d'abord souhaité explorer, au-delà de l'équivalence mathématique, la convergence empirique entre un auto-encodeur (AE) linéaire et l'ACP. Nous avons testée sur les données MNIST [3] avec un encodage en 32 dimensions dans les deux cas. La comparaison visuelle (Figures 4 et 5) montre que les images reconstituées sur l'échantillon test sont pratiquement indiscernables entre AE linéaire et ACP.

Plus rigoureusement, nous avons mesuré la distance entre ces reconstruction pour chacune des périodes d'entraînement de l'auto-encodeur linéaire (Figure 6). Ces deux reconstructions ont également des performances quasiment identiques lorsqu'elles servent de données d'entrée dans un algorithme SVM de classification dans les dix catégories de *digits*.

La comparaison de ces deux reconstructions avec celle obtenue par un auto-encodeur non linéaire (fonctions d'activation *relu* et sigmoïde en sortie, 5 couches cachées) est éclairante. L'entraînement prend plus de temps. Les images sont visuellement plus satisfaisantes, notamment parce que les noirs sont plus noirs et les blancs plus blancs, tandis que la reconstruction linéaire tend vers des valeurs moyennes grises. Mais la performance de l'algorithme de classification sur cette reconstruction par auto-encodeur non linéaire est moins bonne.

3.2 L'auto-encodeur non linéaire assure un débruitage des images légèrement meilleur que l'auto-encodeur linéaire

Nous avons ensuite voulu tester l'une des principales applications des auto-encodeurs, le débruitage. Là encore, nous avons mené la comparaison entre le débruitage par ACP, le débruitage par auto-

2. Outre [3], nous nous sommes également inspiré pour l'implémentation de `https://blog.keras.io/building-autoencoders-in-keras.html`

3. `http://yann.lecun.com/exdb/mnist/`

encodeur linéaire, et le débruitage par auto-encodeur non linéaire. La supériorité visuelle de l'auto-encodeur non linéaire est encore plus affirmée, car les reconstructions linéaires souffrent manifestement du bruit (Figure 7. Mais là encore, l'auto-encodeur non linéaire montre les limites d'un entraînement plus exigeant (par exemple, l'image bruitée d'un 2 peut être transformée en 3 par l'auto-encodeur, Figure 8).

3.3 Une bonne visualisation des données nécessite de perfectionner notre auto-encodeur non linéaire

Enfin, nous avons testé l'utilisation des auto-encodeurs pour visualiser, en deux dimensions, des données qui en comptent beaucoup plus. De même que l'ACP est couramment utilisée pour projeter sur le plan un nuage de points, cette application consiste à mettre au centre de l'auto-encodeur une couche d'encodage comprenant deux unités et à utiliser les valeurs de ces unités comme coordonnées dans le plan pour représenter les données. Nous avons testé cette procédure sur les données MNIST (comme [3] ainsi que sur les données du premier tour des élections présidentielles françaises. Chaque bureau de vote était décrit par onze variables respectivement associées aux scores de onze candidats (en pourcentage des inscrits). Nous avons alors cherché à visualiser les départements sur un plan de l'espace politique obtenu grâce à l'auto-encodeur (Figures 9 et 10). Dans les deux cas, le recours à un auto-encodeur non linéaire n'améliorait en rien la visualisation obtenue par l'ACP. Pourquoi l'auto-encodeur n'a-t-il pas tenu ses promesses dans nos expérimentations ?

Dès 1988 dans l'article ([1]), H. Bourlard et Y. Kamp insiste sur l'inefficacité d'un auto-encodeur non-linéaire par rapport à un auto-encodeur linéaire, et donc a fortiori par rapport à l'ACP. Seule la non linéarité de la fonction d'activation associée à la couche de sortie dans le cadre d'une classification est considérée comme nécessaire, et même dans ce dernier cas il est fait l'hypothèse que des résultats d'équivalence avec la factorisation de matrice peuvent être généralisés.

L'article d'Hinton et Sakakhutdinov ([3]) explique toutefois que la non linéarité de l'auto-encodeur conduit notamment à des visualisations des données largement plus visibles et interprétables que celles obtenues par une ACP. Les deux auteurs ne réfutent pas les conclusions de l'article de Bourlard et Kamp mais ils précisent que les performances de l'auto-encodeur non linéaire sont largement meilleures que celles de l'ACP si celui-ci est bien initialisé. En pratique, de gros poids initiaux conduisent souvent à des optimaux locaux alors que des petits poids rendent l'entraînement du réseau de neurones quasiment impossible. En revanche, si les poids sont proches de la valeur effective, la descente de gradient est alors performante. L'initialisation joue ainsi un rôle crucial. Des pré-entraînements, comme ceux assurés par les RBM ou les *stacked* auto-encodeurs, sont alors indispensables.

Les RBM permettent d'améliorer cette initialisation dans le cas de variables d'entrée binaires. En réalité, les briques successives du RBM correspondent à une forme automatique de *feature engineering*. Au sein de chaque brique de compression, les unités de la couche cachée sont connectées aux unités de la couche visible, initialement la couche d'entrée. On parle d'énergie pour désigner la force de la connexion entre deux unités. Comme l'énergie est inversement proportionnelle à la probabilité jointe, la connexion entre une unité de la couche visible et une unité de la couche cachée est plus probable lorsque l'énergie associée à ce couple est faible. Selon leur probabilité jointe, une unité de la couche cachée est alors mise à 1 avec une probabilité fonction des paramètres de la couche visible. Réciproquement, après cette étape, nous effectuons cette opération sur les unités de la couche visible à partir des valeurs des unités de la couche cachée. Cette opération est finalement réitérée sur les unités de la couche cachée. Au terme de cette procédure d'apprentissage, les poids associés à cette phase de compression sont ainsi mis à jour. Les RBM permettent ainsi une exploration stochastique de l'espace des paramètres à partir de lois marginales qui approximent la distribution des données. Ce mécanisme d'apprentissage est réalisée pour initialiser chaque couche de compression de l'auto-encodeur et peut être réitéré autant que nécessaire afin d'aboutir à une initialisation des poids la plus fidèle possible.

4 Conclusion

Les auto-encodeurs sont un type classique de réseau de neurones qui, dans leur version la plus simple, ont pour limite de simplement reproduire les résultats d'une ACP, parce qu'ils résolvent le même problème d'optimisation. Nous avons pu constater expérimentalement cette convergence entre les deux méthodes. Dans une version plus intéressante, non linéaire, et avec plusieurs couches cachées, les auto-encodeurs explorent un espace fonctionnel beaucoup plus grand et peuvent permettre de mettre à jour des structures plus riches des données. Nous avons aperçu ce potentiel dans nos expérimentations. Les résultats n'ont cependant pas été extraordinaires, et nous pensons que cela est principalement dû aux difficultés d'entraînement de réseaux avec plusieurs couches cachées, et notamment aux problèmes d'initialisation. De telles difficultés peuvent être traitées par la force brute, par divers procédures de régularisation, ou encore par une initialisation par pré-entraînement, par exemple avec des RBM, comme nous l'avons expliqué pour finir.

Références

[1] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4) :291–294, 1988.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786) :504–507, 2006.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[5] Fjodor Van Veen. The neural network zoo, 2016.

Figures

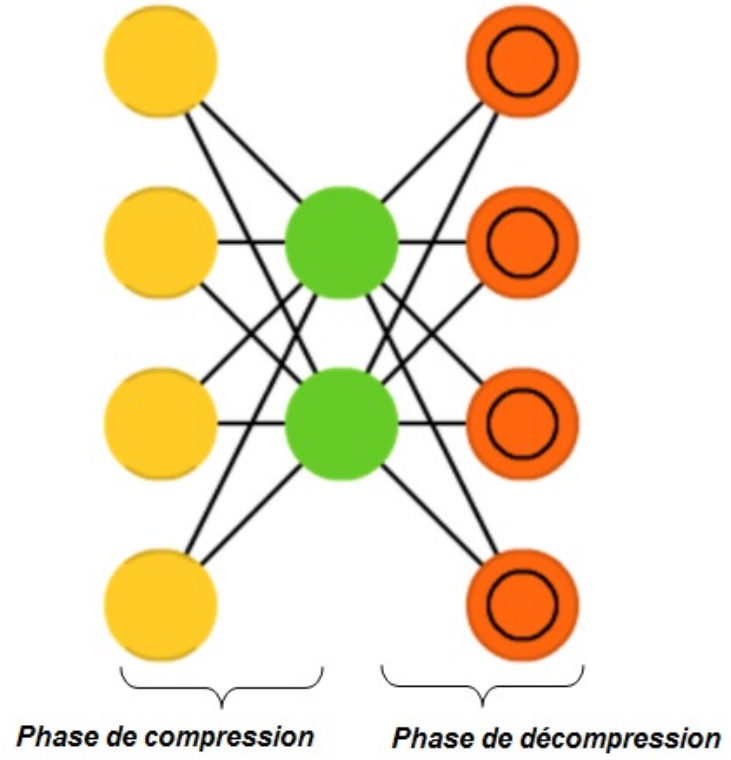

FIGURE 1 – Architecture d'un auto-encodeur simple d'après [5]

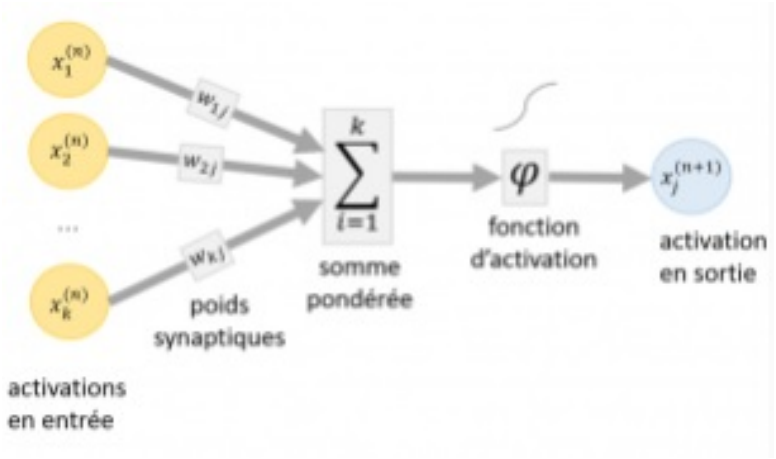

FIGURE 2 – Principe d'activation de chaque couche

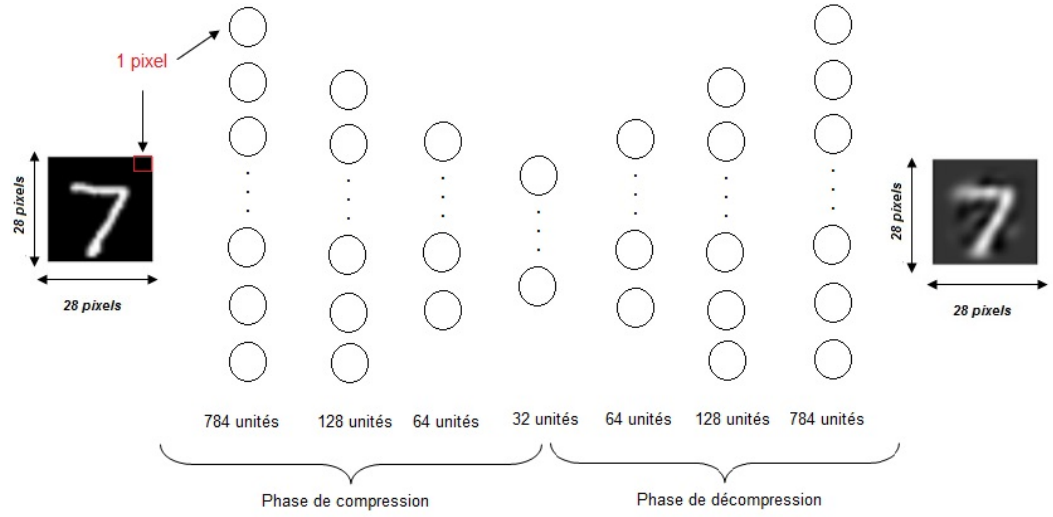

FIGURE 3 – Architecture de l'auto-encodeur avec plusieurs couches pour une image 28x28

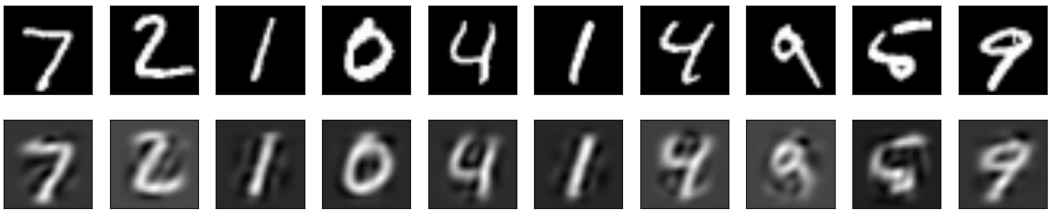

FIGURE 4 – Original et reconstruction via AE linéaire

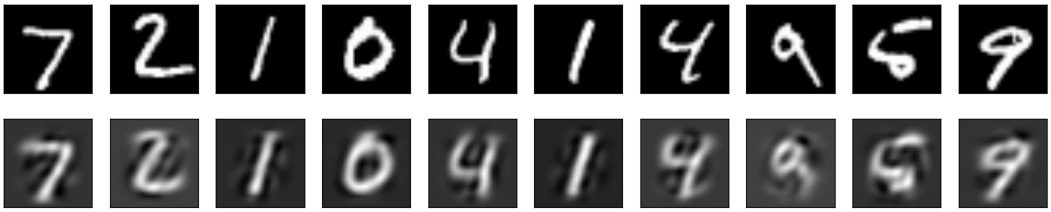

FIGURE 5 – Original et reconstruction via ACP

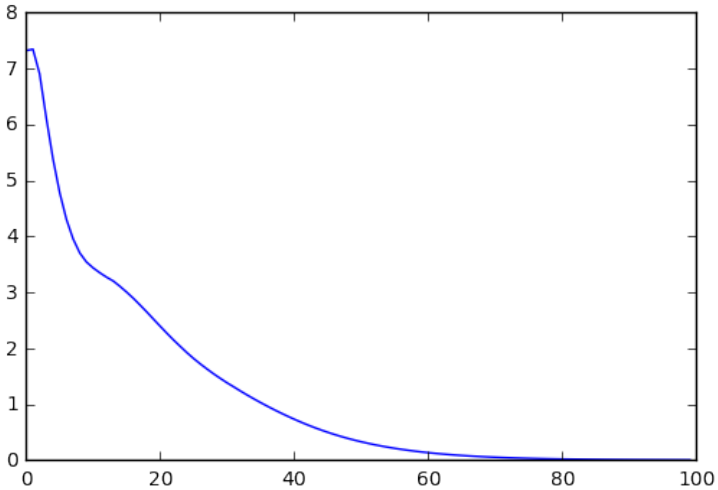

FIGURE 6 – Distance entre les deux reconstruction en fonction du nombre d'époque de l'entraînement

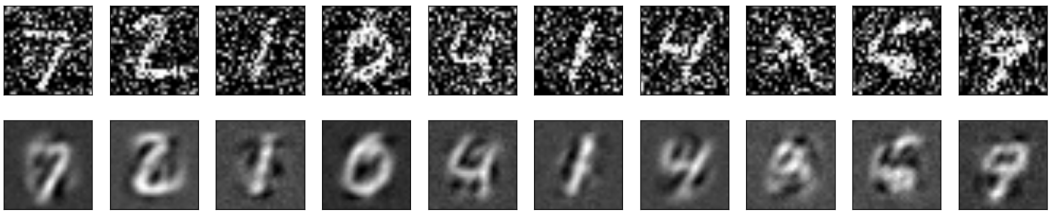

FIGURE 7 – Débruitage via AE linéaire

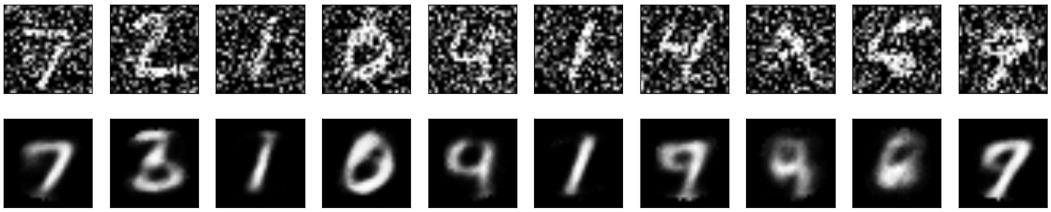

FIGURE 8 – Débruitage via AE 5 couches non linéaire

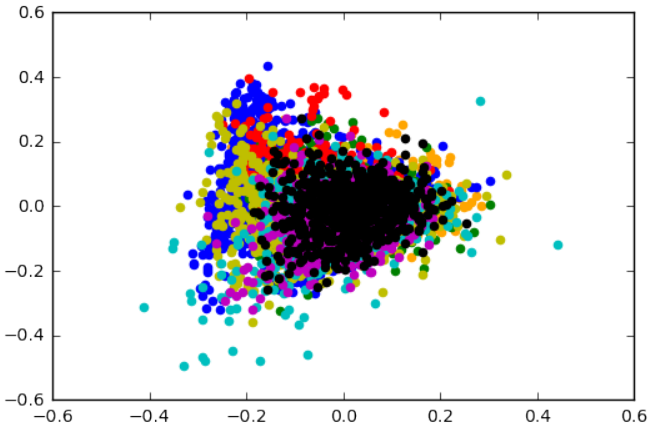

FIGURE 9 – Les bureaux de votes des 8 départements de la région Rhône-Alpes au premier tour de la présidentielle 2017, projetés par ACP

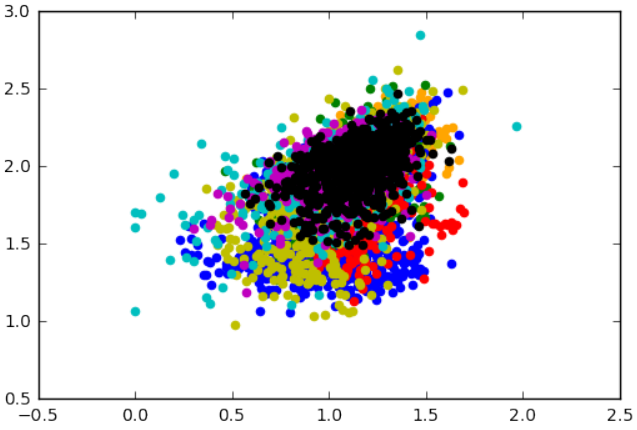

FIGURE 10 – Les mêmes bureaux et les mêmes départements, visualisés grâce à un AE non linéaire