AUTO-ENCODEURS

SOMMAIRE

1- INTRODUCTION

Un autoencodeur est algorithme entraîné de manière non supervisée à reproduire son entrée $\mathbf{x} \in \mathcal{X}$. Il peut être vu (figure 5-6) comme composé de deux parties : un encodeur `E` qui transforme $\mathbf{x}$ en un code déterministe $\mathbf{h} = f(\mathbf{x}; \mathbf{w_E}) \in \mathcal{H}$ ou une distribution $\mathbf{p}_{encodeur}(\mathbf{h}|\mathbf{x}, \mathbf{w_E})$, qui représente l'entrée ; et un décodeur `D` qui produit une reconstruction déterministe $\hat{\mathbf{x}} = g(\mathbf{h}; \mathbf{w_D})$ de $\mathbf{x}$ ou une distribution $\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})$. Les vecteurs $\mathbf{w_E}$ et $\mathbf{w_D}$ sont les paramètres de `E` et `D`. Le plus souvent, l'encodeur et le décodeur sont des réseaux de neurones (perceptrons multicouches plus ou moins profonds, réseaux convolutifs) et les paramètres sont donc les poids de ces réseaux. À ce titre, l'entraînement peut être réalisé avec les mêmes algorithmes que ceux utilisés dans les réseaux de neurones classiques.
Entraîner un autoencodeur à reconstruire $g \circ f(\mathbf{x}) = \mathbf{x}$ pour tout $\mathbf{x}$ n'est pas utile (on apprend l'identité). On

contraint donc le réseau à ne pas reproduire parfaitement l'entrée, et à ne s'intéresser qu'à certains aspects de la reconstruction, ce qui lui permet d'apprendre des propriétés utilies des données.

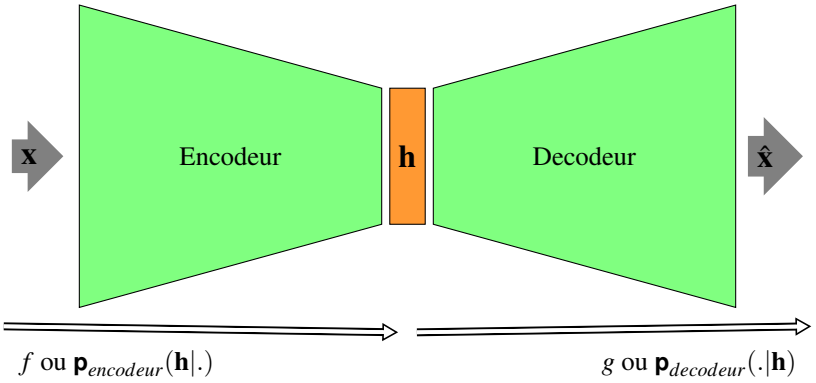


FIGURE 1-1 – Architecture générale d'un autoencodeur

2- INFLUENCE DE LA TAILLE DE L'ESPACE DE CODAGE

2.1- Le cas $|\mathcal{H}| < |\mathcal{X}|$

Lorsque la dimension du code **h** est inférieure à celle de **x**, l'encodeur E apprend à réduire la dimension. Le décodeur, une fois appris, permet de créer une donnée dans $\mathcal{X}$ à partir d'un point de $\mathcal{H}$: il agit donc comme un modèle génératif.
L'apprentissage (la recherche des valeurs de $\mathbf{w_E}$ et $\mathbf{w_D}$) s'effectue par minimisation d'une fonction de perte

$$\ell(\mathbf{x}, g\left[f(\mathbf{x}; \mathbf{w_E}); \mathbf{w_D}\right])$$

Si g est linéaire et ℓ est la fonction de perte quadratique, alors l'autoencodeur agit comme l'analyse en composantes principales. Dans le cas plus général, l'autoencodeur apprend une représentation plus complexe des données. Il faut cependant prendre garde à ce que f et g ne soient pas trop complexes, auquel cas l'autoencodeur ne saura faire que copier exactement l'entrée, sans extraire dans $\mathcal{H}$ d'information utile sur les données.
L'espace $\mathcal{H}$ peut être utilisé pour de la visualisation en dimension réduite des données, pour des tâches de classification, ou plus simplement pour un espace de représentation plus compact des données de $\mathcal{X}$ (figure 2-2).

2.2- Le cas $|\mathcal{H}| \geq |\mathcal{X}|$

Si la taille de l'espace de représentation $\mathcal{H}$ est supérieure à celle de l'espace d'entrée, on comprend assez facilement qu'il est très aisé pour l'autoencodeur d'apprendre l'identité, sans extraire d'information utile des données initiales (il suffit de propager $\mathcal{X}$ dans $\mathcal{H}$). Il est donc nécessaire de contraindre le modèle.

3- AUTOENCODEURS RÉGULARISÉS

Régulariser un autoencodeur permet d'entraîner efficacement l'algorithme, en choisissant de plus la dimension de $\mathcal{H}$ et la complexité de f et g en fonction de la complexité de la distribution à modéliser. Plutôt que

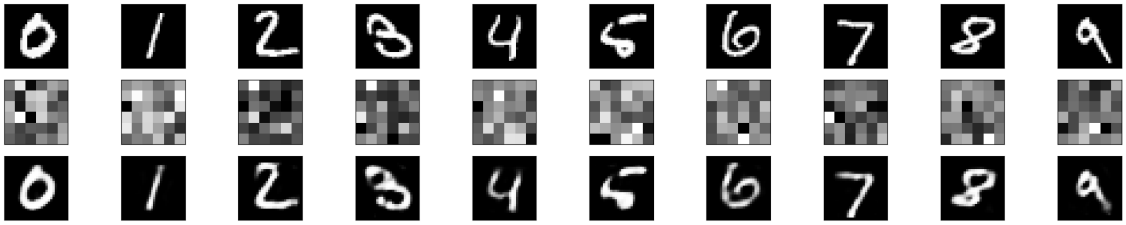


FIGURE 2-2 – Utilisation d'un autoencodeur pour la compression et la génération de chiffres manuscrits. Les images MNIST (28×28, ligne du haut) sont encodées par un simple perceptron multicouche à activation sigmoïde et une seule couche cachée de taille 36. Le code est visualisé (ligne du milieu) sous la forme d'images 6×6. Le décodeur produit des images reconstruites (ligne du bas) à partir de ce code.

de limiter la capacité du modèle (en imposant par exemple que E et D soient des réseaux multicouches à faible profondeur et/ou que $\mathcal{H}$ soit de faible dimension), la régularisation construit une fonction de perte qui encourage l'autoencodeur à avoir des propriétés supplémentaires, en plus de celle de reproduire son entrée. Un autoencodeur régularisé minimise la fonction

$$\ell(\mathbf{x}, g\left[f(\mathbf{x};\mathbf{w_E});\mathbf{w_D}\right]) + \beta\Omega(\mathbf{h})$$

où $\Omega(\mathbf{h})$ est un terme de pénalisation permettant de contraindre les paramètres du modèle et $\beta \in \mathbb{R}$ contrôle le poids du terme de pénalité dans l'optimisation.

3.1- Autoencodeurs parcimonieux

Les autoencodeurs parcimonieux (ou épars) sont typiquement utilisés pour apprendre des caractéristiques pertinentes des données d'entrée, qui sont ensuite utilisées comme entrées d'algorithmes de classification ou de régression.

Supposons que E et D soient des perceptrons multicouches. Il est alors par exemple possible d'imposer aux neurones d'être "inactifs" la plupart du temps, en définissant l'inactivité comme une valeur de sortie du neurone proche de zéro (pour une sigmoïde, ou -1 pour une tangente hyperbolique). Pour cela, on dispose de m exemples $\mathcal{S} = \{\mathbf{x_1} \cdots \mathbf{x_m}\}$. On note $y_j^{(l)}(\mathbf{x})$ l'activation du neurone caché j de la couche l lorsque l'entrée $\mathbf{x}$ est présentée au réseau. On note également

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[y_j^{(l)}(\mathbf{x_i})\right]$$

l'activation moyenne du neurone caché j sur présentation de $\mathcal{S}$. L'objectif est alors d'imposer $\hat{\rho}_j = \rho$, où ρ est une valeur proche de zéro (ainsi l'activation moyenne de chaque neurone caché doit être faible), par l'intermédiaire d'une définition adaptée de Ω. De nombreux choix sont possibles. Par exemple pour un réseau à une couche cachée :

$$\Omega(\mathbf{h}) = \sum_{j=1}^{n^{(2)}} \rho \log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j} = \sum_{j=1}^{n^{(2)}} KL(\rho||\hat{\rho}_j),$$

où $KL(\rho||\hat{\rho}_j)$ est la divergence de Kullback-Leibler (KL) entre une variable aléatoire de loi de Bernoulli de moyenne ρ et une variable aléatoire de loi de Bernoulli de moyenne $\hat{\rho}_j$.
On peut alors montrer que $KL(\rho||\hat{\rho}_j) = 0$ si $\hat{\rho}_j = \rho$, et KL croît de façon monotone lorsque $\hat{\rho}_j$ s'éloigne de ρ.

Le calcul des dérivées partielles et la descente de gradient changent peu pour l'algorithme d'optimisation. Il faut cependant connaître au préalable les $\hat{\rho}_j$ et donc faire dans un premier temps une propagation avant sur tous les exemples de $\mathcal{S}$ permettant de calculer les activations moyennes.

Dans le cas où la base d'apprentissage est suffisamment petite, elle tient entièrement en mémoire et les activations peuvent être stockées pour calculer $\hat{\rho}_j$. Les activations stockées peuvent alors être utilisées dans l'étape de rétropropagation sur l'ensemble des exemples.
Dans le cas contraire, le calcul de $\hat{\rho}_j$ peut être fait en accumulant les activations calculées exemple par exemple, mais sans sauvegarder les valeurs de ces activations. Une seconde propagation sur chaque exemple sera alors nécessaire pour permettre la rétropropagation.

Les autoencodeurs parcimonieux peuvent également être vus d'un point de vue probabiliste comme des algorithmes maximisant la vraisemblance maximale d'un modèle génératif à variables latentes $\mathbf{h}$. Supposons disposer d'une distribution jointe explicite

$$\mathbf{p}_{modele}(\mathbf{x},\mathbf{h}) = \mathbf{p}_{modele}(\mathbf{h})\mathbf{p}_{modele}(\mathbf{x}|\mathbf{h})$$

La log vraisemblance peut alors s'écrire

$$log(\mathbf{p}_{modele}(\mathbf{x})) = log\left(\sum_{\mathbf{h}} \mathbf{p}_{modele}(\mathbf{x},\mathbf{h})\right)$$

L'autoencodeur approche cette somme juste pour une une valeur de $\mathbf{h}$ fortement probable. Pour cette valeur, on maximise alors

$$log(\mathbf{p}_{modele}(\mathbf{x},\mathbf{h})) = log(\mathbf{p}_{modele}(\mathbf{h})) + log(\mathbf{p}_{modele}(\mathbf{x}|\mathbf{h}))$$

et $log(\mathbf{p}_{modele}(\mathbf{h}))$ peut être utilisée pour introduire de la parcimonie.
Par exemple si $\mathbf{p}_{modele}(h_i) = \frac{\lambda}{2}e^{-\beta|h_i|}$ (Laplace prior), alors

$$-log(\mathbf{p}_{modele}(\mathbf{h})) = \sum_{i=1}^{|\mathcal{H}|}\left(\lambda|h_i| - log\frac{\lambda}{2}\right) = \Omega(\mathbf{h}) + c$$

et l'on retrouve une régularisation ℓ_1 (méthode Lasso).

3.2- Autoencodeurs contractifs

Une autre stratégie de régularisation consiste à faire dépendre Ω du gradient du code en fonction des entrées :

$$\Omega(\mathbf{x},\mathbf{h}) = \sum_{i=1}^{|\mathcal{H}|} \|\nabla_{\mathbf{x}} h_i\|^2 = \left\|\frac{\partial f(\mathbf{x},\mathbf{w_E})}{\partial \mathbf{x}}\right\|_F^2$$

où $\|.\|_F$ est la norme de Frobenius. Le modèle apprend alors une fonction qui change peu lorsque $\mathbf{x}$ varie peu. Puisque la pénalité n'est appliquée que sur les exemples de $\mathcal{S}$, les informations capturées dans le code concernent la distribution des données d'entraînement, et plus précisément la variété sur laquelle vivent les données de $\mathcal{S}$. En ce sens, ces autoencodeurs sont à rapprocher des méthodes de *Manifold Learning*.

3.3- Autoencodeurs de débruitage

Plutôt que d'ajouter un terme à la fonction de perte, on peut directement changer cette dernière pour apprendre des caractéristiques utiles des données.
Un autoencodeur de débruitage considère la fonction de perte

$$\ell(\mathbf{x}, g\left[f(\tilde{\mathbf{x}};\mathbf{w_E});\mathbf{w_D}\right])$$

où $\tilde{\mathbf{x}}$ est une version de $\mathbf{x}$ bruitée par une distribution conditionnelle $C(\tilde{\mathbf{x}},\mathbf{x})$. L'autoencodeur apprend alors une distribution de reconstruction $\mathbf{p}_R(\mathbf{x} \mid \tilde{\mathbf{x}})$ selon l'algorithme 1 (figure 3-3).
L'apprentissage peut être vu comme une descente de gradient stochastique de

$$-\mathbb{E}_{\mathbf{x}\sim\mathbf{p}_S(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}},\mathbf{x})}\left(log\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}},\mathbf{w_E}),\mathbf{w_D})\right)$$

Algorithme 1 : Algorithme d'apprentissage d'un autoencodeur de débruitage

Données : $\mathcal{S}$, $C(\tilde{\mathbf{x}}, \mathbf{x})$, un autoencodeur (f, g)
Résultat : Un autoencodeur de débruitage
début
  tant que non stop **faire**
    Tirer un exemple $\mathbf{x}$ de $\mathcal{S}$
    Tirer $\tilde{\mathbf{x}}$ selon $C(\tilde{\mathbf{x}}, \mathbf{x})$
    Estimer $\mathbf{p}_R(\mathbf{x} \mid \tilde{\mathbf{x}}) = \mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D}) = g(\mathbf{h}, \mathbf{w_D})$ où $\mathbf{h} = f(\tilde{\mathbf{x}}, \mathbf{w_E})$

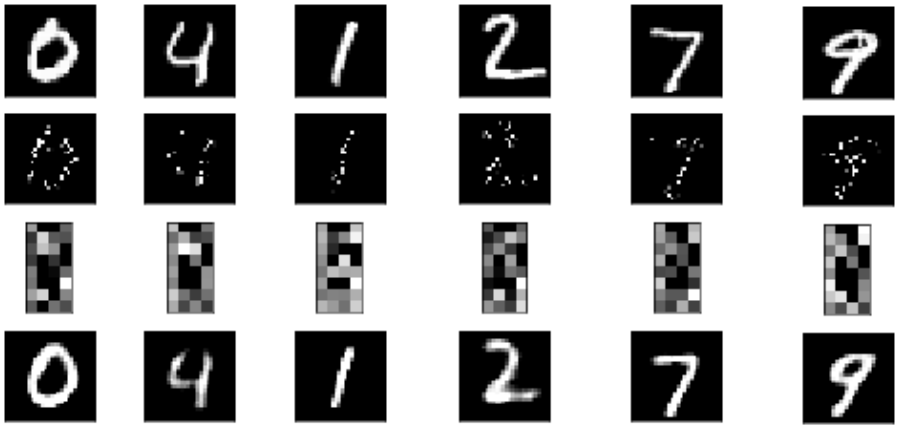


FIGURE 3-3 – Autoencodeur de débruitage sur les données MNIST. Les images $\mathbf{x}$ (ligne du haut) sont corrompues par un bruit gaussien centré de variance unité (deuxième ligne). Un autoencodeur de débruitage est ensuite entraîné. Le code $\mathbf{h}$ de taille 32 est visualisé (troisième ligne) sous la forme d'images 8×4. Le décodeur produit les images débruitées de la dernière ligne.

4- AUTOENCODEURS VARIATIONNELS

Le dernier modèle d'autoencodeurs que nous abordons fait le lien avec la section suivante sur les réseaux antagonistes générateurs.
Les autoencodeurs variationnels (VAE) [1] sont des modèles génératifs. Ce ne sont pas à proprement parler des autoencodeurs tels que nous les avons abordés dans les paragraphes précédents, ils empruntent juste une architecture similaire (figure 4-4), d'où leur nom.

Au lieu d'apprendre $f(., \mathbf{w_E})$ et $g(., \mathbf{w_D})$, un autoencodeur variationnel apprend des distributions de probabilité $\mathbf{p}_{encodeur}(\mathbf{h}|\mathbf{x}, \mathbf{w_E})$ et $\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})$.
Apprendre des distributions plutôt que des fonctions déterministes présente plusieurs avantages, et notamment :

- les données d'entrée peuvent être bruitées, et un modèle de distribution $\mathbf{p_x}$ peut être plus utile
- il est possible d'utiliser $\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})$ pour échantillonner $\mathbf{h}$ puis $\mathbf{x}$, et donc de générer des données ayant des statistiques similaires aux éléments de $\mathcal{S}$ (figure 4-5).

Abordons ces autoencodeurs sous l'angle des modèles génératifs. Supposons que nous voulions générer des points suivant la distribution $\mathbf{p_x}$. Plutôt que d'inférer directement sur cette distribution, nous pouvons utiliser des variables latentes (le code des autoencodeurs). Les modèles à variables latentes font l'hypothèse que les données $\mathbf{x}$ sont issues d'une variable non observée $\mathbf{h}$. S'il peut être difficile de modéliser directement $\mathbf{p_x}$, il peut être plus facile de choisir *a priori* une distribution $\mathbf{p_h}$ et chercher à modéliser $\mathbf{p}_{\mathbf{x}|\mathbf{h}}$.
Pour générer $\mathbf{x} \sim \mathbf{p_x}$, un autoencodeur variationnel tire donc tout d'abord $\mathbf{h} \sim \mathbf{p_h}$. $\mathbf{h}$ est ensuite passé à un

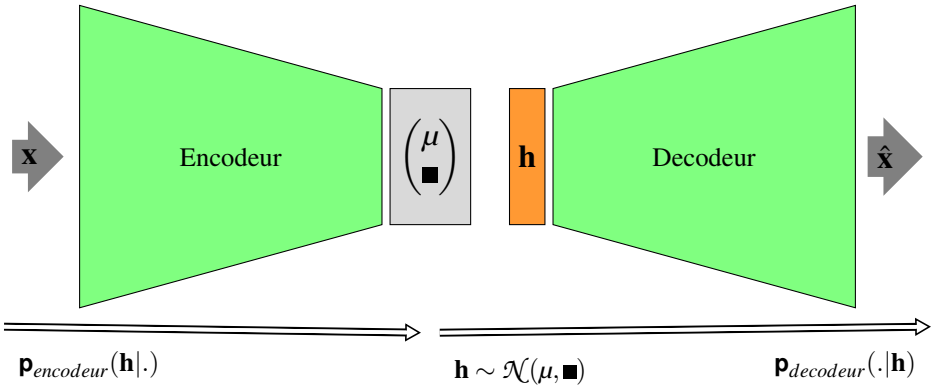


FIGURE 4-4 – Architecture générale d'un autoencodeur variationnel

réseau de neurones et $\mathbf{x}$ est finalement tiré selon $\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})$. L'entraînement est réalisé en maximisant la borne inférieure variationnelle :

$$\mathcal{L}(q) = \mathbb{E}_{\mathbf{h}\sim q(\mathbf{h}|\mathbf{x})} log\left(\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})\right) - KL\left(q(\mathbf{h}|\mathbf{x})||\mathbf{p_h}\right)$$

où KL est la divergence de Kullback Leibler déjà rencontrée dans les autoencodeurs parcimonieux. Le premier terme de $\mathcal{L}(q)$ est la log vraisemblance de la reconstruction trouvée dans les autoencodeurs classiques, tandis que le second terme tend à rapprocher la distribution *a posteriori* $q(\mathbf{h}|\mathbf{x})$ et le modèle *a priori* $\mathbf{p_h}$. Dans les techniques classiques d'inférence, q est approché par optimisation. Dans les autoencodeurs variationnels, on entraîne un encodeur paramétrique (un réseau de neurones paramétré par $\mathbf{w_E}$) qui produit les paramètres de q. Tant que $\mathbf{h}$ est continue, il est donc possible de rétropropager à travers les tirages de $\mathbf{h}$ effectués selon $q(\mathbf{h}|\mathbf{x}) = q(\mathbf{h}|f(\mathbf{x}; \mathbf{w_E}))$ pour obtenir le gradient par rapport à $\mathbf{w_E}$. L'apprentissage consiste alors simplement à maximiser $\mathcal{L}$ par rapport à $(\mathbf{w_E}, \mathbf{w_D})$.
Il est courant de choisir comme prior $\mathbf{p_h}$ une loi normale centrée réduite $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Cette simplicité apparente ne réduit pas le pouvoir d'expression du modèle si l'effort est fait sur l'optimisation de la distribution $\mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h}, \mathbf{w_D})$. L'encodeur `E` est alors un réseau de neurones générant des paramètres de distribution de q dans $\mathcal{H}$, soit un vecteur de moyenne μ et une matrice de covariance ■.
Notons enfin que la rétropropagation du gradient nécessite une astuce de calcul dans $\mathcal{H}$, dite astuce de reparamétrisation : la génération de $\mathbf{h} \sim \mathbf{p}_{encodeur}(\mathbf{h}|\mathbf{x}, \mathbf{w_E})$ se fait effectivement en tirant une variable aléatoire $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, puis en calculant $\mathbf{h}=\mu + \Sigma^{1/2}\varepsilon$. L'échantillonnage se fait alors seulement pour ε, qui n'a pas besoin d'être rétropropagé.

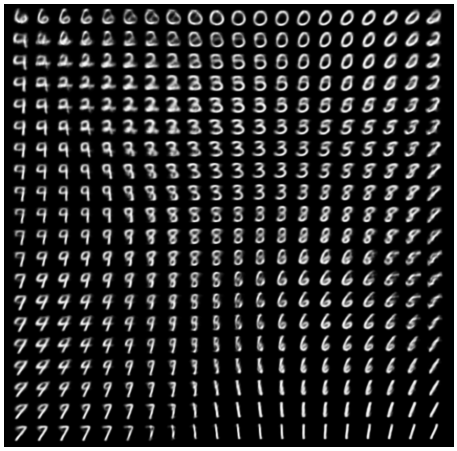


FIGURE 4-5 – Visualisation de l'espace latent $\mathcal{H} = \mathbb{R}^2$ appris par un autoencodeur variationnel sur les données MNIST. Pour chaque valeur $\mathbf{h_i}$ discrétisée sur $\mathcal{H}$ est affichée une image $\mathbf{x} \sim \mathbf{p}_{decodeur}(\mathbf{x}|\mathbf{h_i}, \mathbf{w_D})$. Les chiffres de la même classe sont groupés dans cet espace, et les axes de $\mathcal{H}$ ont une interprétation (l'axe horizontal semble souligner le caractère "penché" des chiffres)

5- PARTIE PRATIQUE

5.1- Construction d'autoencodeurs simples

Les squelettes notebook vous étant donnés, il est demandé de construire deux auto-encodeurs :

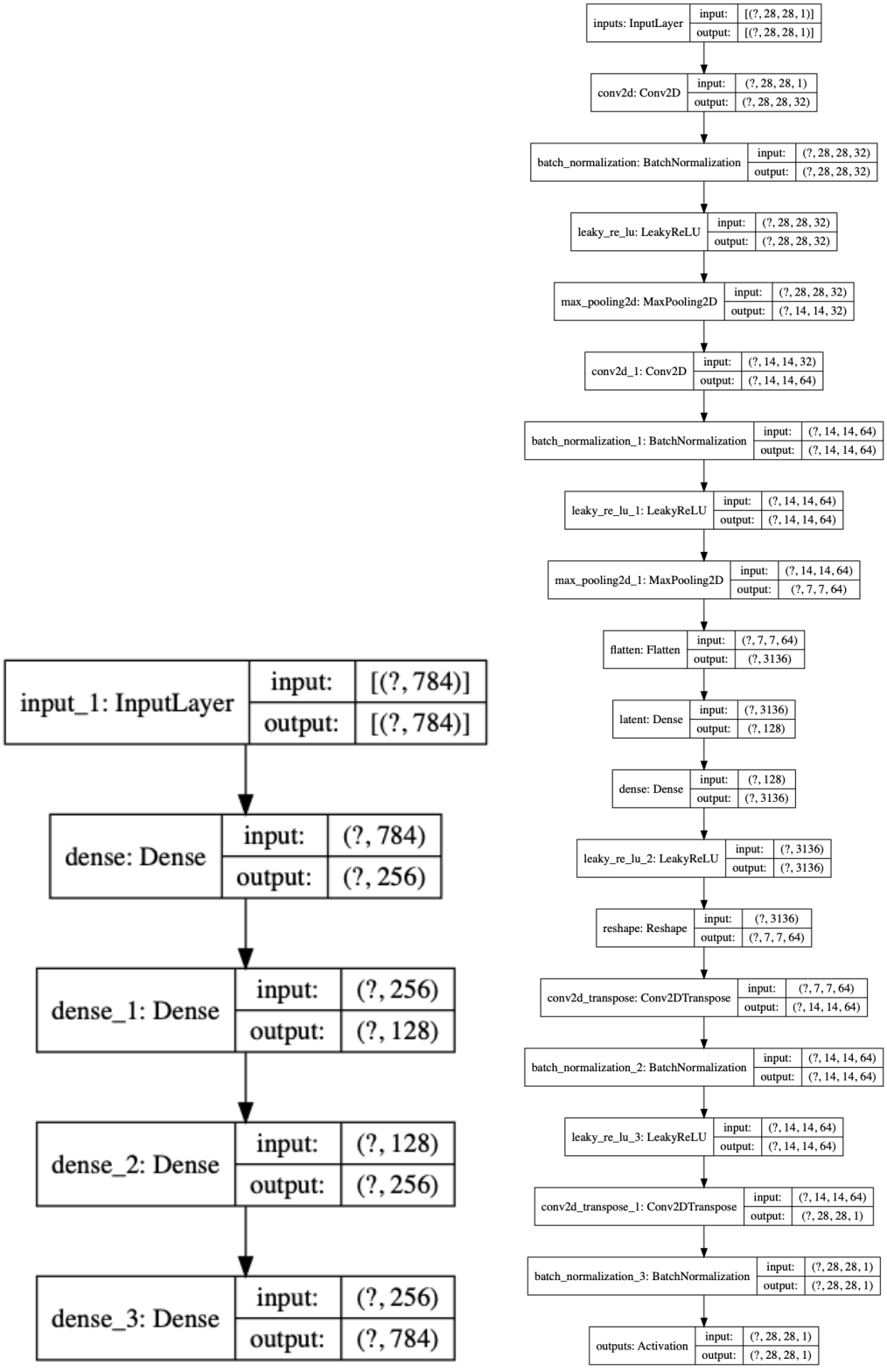
1. un premier avec un encodeur et un décodeur définis par des perceptrons multicouches (figure 5-6-(a))
2. un second dont l'encodeur et le décodeur sont composés de réseaux convolutifs (figure 5-6-(b))

Vous ferez varier les paramètres de vos autoencodeurs (nombre de neurones sur les couches cachées, learning rate, nombre d'itérations, taille des batchs...), et observerez :

— l'espace latent de l'autoencodeur ;
— les images reconstruites pour les deux autoencodeurs produits, et vous jugerez de la qualité de la reconstruction.

5.2- Autoencodeur de débruitage

Réalisez un autoencodeur permettant de réaliser du débruitage d'image. En vous appuyant sur la partie précédente, apprenez le modèle en présentant des données corrompues (changer les valeurs aléatoirement pour $p\%$ des pixels de l'image d'entrainement).



(a)Autoencodeur à base de PMC (b) Autoencodeur à base de CNN

FIGURE 5-6 – Autoencodeurs

BIBLIOGRAPHIE

[1] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. CoRR, abs/1312.6114, 2013.