Expansion en nombre de neurones du perceptron multicouche dans la structure des Auto-encodeurs Variationnels

Rapport Projet Fin d'Etude

H.N Huy NGUYEN & Quoc Tan NGUYEN

5A ACAD - GSI
INSA Centre Val de Loire
Blois
Février 2025

Table des matières

1 Introduction

Aujourd'hui, nous observons que la taille des modèles génératifs augmente chaque jour, ainsi que leur efficacité pour réaliser des tâches génératives. Il existe des modèles à l'état de l'art comportant des milliards de paramètres. Nous savons que la capacité de tels modèles a une forte relation avec leur nombre de paramètres.

Mais est-ce que c'est toujours le cas ? Pourrions-nous trouver une structure avec une taille optimale pour une tâche spécifique ? Dans ce Projet de Fin d'Études, nous allons expérimenter l'expansion en nombre de paramètres pour évaluer si elle entraîne une performance supérieure d'un modèle génératif appelé Auto-encodeur Variationnel.

Nous rappelons d'abord la structure de l'Auto-encodeur Variationnel (VAE). Nous nous référons aux articles pertinents [1, 2, 3] pour mieux comprendre cette structure de réseau de neurones en Section 2, ainsi que les fondements mathématiques qui expliquent son fonctionnement [6, 7].

Ensuite, en Section 3, nous expliquons comment nous pouvons ajouter de nouveaux neurones dans les couches cachées de notre structure, en nous basant sur le travail similaire d'une doctorante du laboratoire LIFAT. Nous définissons également les concepts de structure optimisée et de structure cible utilisée pour la comparaison. Nous appliquons alors la méthode d'augmentation sur les couches denses du VAE. Les résultats d'expérimentation sont présentés en Section 4. Nous détaillons le nombre de neurones ajoutés pour chaque augmentation et commentons l'impact de ces modifications.

Finalement, nous concluons notre travail. Nous tenons à remercier Monsieur Moncef Hidane et Monsieur Julien Mille de nous avoir encadrés pour ce projet de fin d'études et d'avoir consacré du temps pour nous aider à mieux comprendre ce sujet.

2 Auto-encodeur Variationnel

Auto-encodeur Variationnel apprend à représenter les données en modélisant une distribution des variables latentes [2, 7].
L'objectif du VAE est d'apprendre une distribution marginale de paramètres $\boldsymbol{\theta}$ qui maximise la log-vraisemblance des données $\log p_\theta(\mathbf{X})$, avec $\mathbf{X}$ l'ensemble de données observées

$$\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}.$$

La log-vraisemblance peut être décomposée en deux termes

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x^{(i)}}) \tag{2.1}$$

$$= \sum_{i=1}^{N} \mathcal{L}(\phi, \theta; \mathbf{x^{(i)}}) + \sum_{i=1}^{N} \mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x^{(i)}})\right] \tag{2.2}$$

où :

— $\mathcal{L}(\phi, \theta; \mathbf{x^{(i)}})$: la borne inférieure d'évidence (Evidence Lower Bound, ELBO) de la log-vraisemblance $\log p_\theta(\mathbf{X})$ sachant un vecteur de donnée $\mathbf{x^{(i)}}$

— $\mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x^{(i)}})\right]$: la divergence de Kullback-Leibler entre la distribution approximative de $\mathbf{z}$ conditionnée par $\mathbf{x^{(i)}}$ $q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}})$ et la distribution postérieure réelle

$p_\theta(\mathbf{z} \mid \mathbf{X})$.

Dans ce contexte, $\boldsymbol{\theta}$ représente les **paramètres du modèle générateur (décodeur)**, tandis que $\boldsymbol{\phi}$ représente les **paramètres du modèle d'inférence (encodeur)**. Ces deux ensembles de paramètres sont optimisés conjointement lors de l'entraînement du VAE.

Nous allons détailler, dans cette grande section, les différentes composantes structurelles du VAE, y compris :

Section 2.1 : La distribution postérieure du vecteur latent z Nous étudions la distribution a posteriori $p_\theta(\mathbf{z} \mid \mathbf{x})$, qui est généralement intractable. Afin d'approximer cette distribution, nous utilisons un modèle d'inférence variationnelle $q_\phi(\mathbf{z} \mid \mathbf{x})$, optimisé conjointement avec le modèle génératif.

Section 2.2 : La maximisation de la borne inférieure d'évidence L'ELBO est une borne inférieure sur la log-vraisemblance des données observées, utilisée pour optimiser le modèle par maximisation :

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta \left(\mathbf{x^{(i)}}, \mathbf{z} \right) - \log q_\phi \left(\mathbf{z} \mid \mathbf{x^{(i)}} \right) \right]. \tag{2.3}$$

L'optimisation de cette borne permet de rapprocher $q_\phi(\mathbf{z} \mid \mathbf{x})$ de la distribution postérieure réelle tout en maximisant la log-vraisemblance des données.

Section 2.3 : La technique de reparamétrisation Cette redéfinition de moyen que nous échantillonnons $\mathbf{z}$ permettre la rétro-propagation à travers l'échantillonnage stochastique de la couche latente, on utilise généralement le "reparametrization trick".

Section 2.4 : L'entraînement du VAE Nous discussions de la méthode de descente de gradient par mini-lots (Section 2.4.2) en minimisant la négative de l'estimateur de la borne inférieure d'évidence $-\tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}})$ (Section 2.4.1) :

$$\phi, \theta = \arg\min_{\phi,\theta} -\tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}}) \tag{2.4}$$

où le gradient de l'ELBO, $\nabla_{\phi,\theta} \tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}})$, est calculé par la méthode de rétro-propagation du gradient.

2.1 Inférence variationnelle

Pour une donnée d'entrée $\mathbf{x^{(i)}}$, nous supposons que la distribution approximative $q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}})$ est **une distribution gaussienne multivariée** :

$$q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) = \mathcal{N}(\mathbf{z} \mid \mu_\phi(\mathbf{x^{(i)}}), \sigma^2_\phi(\mathbf{x^{(i)}}) \cdot I) \tag{2.5}$$

où la moyenne $\mu_\phi(\mathbf{x^{(i)}})$ et l'écart-type $\sigma_\phi(\mathbf{x^{(i)}})$ sont calculés par les fonctions du réseau de neurones (en appliquant des transformations non-linéaires à la donnée $\mathbf{x^{(i)}}$).

Sachant que la distribution a priori $p(\mathbf{z})$ est une distribution gaussienne standard $\mathcal{N}(0, I)$, la divergence de Kullback-Leibler entre les deux distributions gaussiennes multivariées $q_\phi(\mathbf{z} \mid$

$\mathbf{x}^{(\mathbf{i})}$) et $p(\mathbf{z})$ est donnée par :

$$\mathrm{KL}\left[(q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})}) \parallel p(\mathbf{z})\right] = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_{j,\phi}^2(\mathbf{x}^{(\mathbf{i})})) - \mu_{j,\phi}^2(\mathbf{x}^{(\mathbf{i})}) - \sigma_{j,\phi}^2(\mathbf{x}^{(\mathbf{i})})\right) \tag{2.6}$$

où J est la dimension de l'espace latent, et $\mu_{j,\phi}(\mathbf{x}^{(\mathbf{i})})$ et $\sigma_{j,\phi}(\mathbf{x}^{(\mathbf{i})})$ sont respectivement les j-èmes composantes de $\mu_\phi(\mathbf{x}^{(\mathbf{i})})$ et $\sigma_\phi(\mathbf{x}^{(\mathbf{i})})$.

2.2 Maximisation du borne inférieure d'évidence

La borne inférieure d'évidence fournit une estimation de la probabilité marginale de $\mathbf{X}$. Considérons que :

$$\log p_\theta(\mathbf{x}^{(\mathbf{i})}) \geq \mathcal{L}(\phi, \theta; \mathbf{x}^{(\mathbf{i})}) \tag{2.7}$$

En maximisant l'ELBO pour chaque point de donnée $\mathbf{x}^{(\mathbf{i})}$, nous cherchons à approcher la log-vraisemblance $\log p_\theta(\mathbf{X})$ et ainsi à mieux estimer la distribution postérieure réelle $p_\theta(\mathbf{z} \mid \mathbf{X})$. La distribution définie par les paramètres θ représente la capacité du modèle à exprimer une loi de probabilité cohérente avec les données observées $\mathbf{X}$. Ainsi, elle reflète également l'aptitude du modèle à générer de nouvelles données similaires à celles de l'ensemble d'apprentissage.

L'ELBO peut être développée comme suit :

$$\mathcal{L}(\phi, \theta; \mathbf{x}^{(\mathbf{i})}) = \int q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})}) \log\left(\frac{p_\theta(\mathbf{x}^{(\mathbf{i})}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})})}\right) d\mathbf{z} \tag{2.8}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(\mathbf{i})})}\left[\log\left(\frac{p_\theta(\mathbf{x}^{(\mathbf{i})}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})})}\right)\right] \tag{2.9}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(\mathbf{i})})}\left[\log p_\theta\left(\mathbf{x}^{(\mathbf{i})}, \mathbf{z}\right) - \log q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})}\right)\right] \tag{2.10}$$

où $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid 0, I)$ est la distribution a priori des variables latentes $\mathbf{z}$.
Nous examinons les deux termes du côté droit de l'Équation 2.10, qui se développent en deux expressions fondamentales : la log-probabilité de vraisemblance $\log p_\theta(\mathbf{x}^{(\mathbf{i})} \mid \mathbf{z})$ et la divergence de Kullback-Leibler entre la distribution d'inférence variationnelle $q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})})$ et la distribution a priori $p(\mathbf{z})$. La fonction ELBO s'écrit alors :

$$\mathcal{L}(\phi, \theta; \mathbf{x}^{(\mathbf{i})}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(\mathbf{i})})}\left[\log p_\theta\left(\mathbf{x}^{(\mathbf{i})} \mid \mathbf{z}\right)\right] - \mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})}) \parallel p(\mathbf{z})\right]. \tag{2.11}$$

Cette équation met en évidence deux termes qui interviennent dans la définition de la fonction de perte du modèle utilisé lors de la phase d'entraînement (qui va être détaillée plus spécifiquement dans Section 2.4).
Chacun de ces termes correspond aux deux parties principales de l'architecture du VAE : l'encodeur et le décodeur.

- **(Décodage)** L'erreur de reconstruction négative du modèle génératif, représentée par $\log p_\theta(\mathbf{x}^{(\mathbf{i})} \mid \mathbf{z})$, où $\mathbf{z}^{(l,i)}$ est échantillonné à partir de la distribution approximative $q_\phi(\mathbf{z} \mid \mathbf{x}^{(\mathbf{i})})$.
- **(Encodage)** La divergence de Kullback-Leibler négative entre la distribution approximative et la distribution a priori des variables latentes, qui régularise l'espace latent en encourageant une représentation $\mathbf{z}^{(l,i)}$ conforme à la distribution $p(\mathbf{z})$.

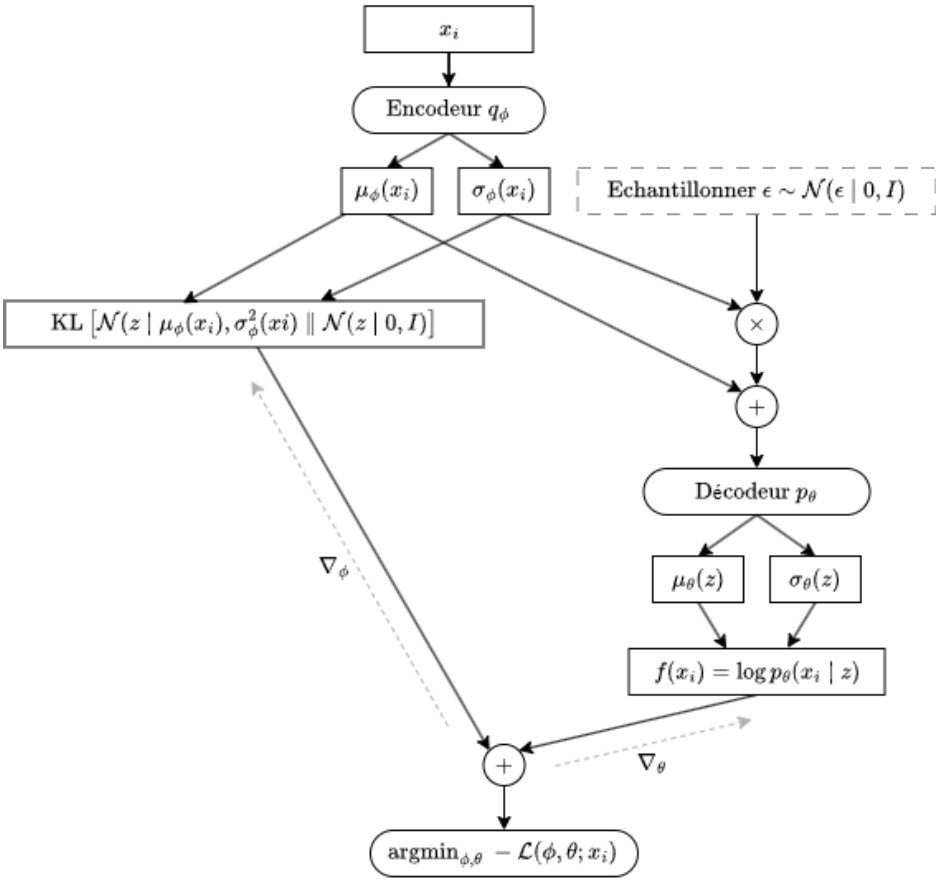

Figure 1 – Entraînement d'un VAE avec le point de donnée $\mathbf{x^{(i)}}$ et la méthode de reparamétrisation. L'échantillonnage stochastique de la couche latente $\mathbf{z}$ est remplacé par une fonction déterministe $\mathbf{z} = g(\epsilon, \mathbf{x^{(i)}})$ où ϵ est échantillonné à partir de $\mathcal{N}(0, I)$. Cette méthode permet d'utiliser la rétropropagation du gradient à travers la variable stochastique $\mathbf{z}$. Ainsi, le gradient de la fonction de perte est calculé par la méthode de descente de gradient, et les paramètres ϕ de l'encodeur et θ du décodeur sont mis à jour.

2.3 Méthode de reparamétrisation

Cette technique consiste à exprimer l'échantillonnage de la variable latente $\mathbf{z}$ comme une fonction déterministe $\mathbf{z} = g_\phi(\epsilon, \mathbf{x})$, où ϵ est un bruit gaussien $\epsilon \sim \mathcal{N}(0, I)$ et des paramètres μ_ϕ et σ_ϕ :

$$\mathbf{z} = \mu_\phi + \sigma_\phi \odot \epsilon$$

où $\odot$ représente le produit élément par élément. Cette reformulation permet de calculer les gradients nécessaires pour l'optimisation du modèle (voir Figure 6).
Grâce à cette méthode de reparamétrisation, nous pouvons calculer le gradient par rapport

au paramètre ϕ de l'expression à droite de l'Équation 2.9 :

$$\nabla_\phi \mathcal{L}(\phi, \theta; \mathbf{x^{(i)}}) = \nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x^{(i)}})} \left[\log \left(\frac{p_\theta(\mathbf{x^{(i)}}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}})} \right) \right] \tag{2.12}$$

$$= \nabla_\phi \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\log \left(\frac{p_\theta(\mathbf{x^{(i)}}, g_\phi(\epsilon, \mathbf{x^{(i)}}))}{q_\phi(g_\phi(\epsilon, \mathbf{x^{(i)}}) \mid \mathbf{x^{(i)}})} \right) \right] \tag{2.13}$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\nabla_\phi \log \left(\frac{p_\theta(\mathbf{x^{(i)}}, g_\phi(\epsilon, \mathbf{x^{(i)}}))}{q_\phi(g_\phi(\epsilon, \mathbf{x^{(i)}}) \mid \mathbf{x^{(i)}})} \right) \right] \tag{2.14}$$

où $\epsilon \sim p(\epsilon)$ sont des échantillons indépendants du bruit gaussien, et L est le nombre d'échantillons utilisés pour l'estimation. Le **"reparametrization trick"** permet de calculer le gradient de l'ELBO par rapport aux paramètres ϕ, ce qui correspond à passer de l'Équation 2.13 à l'Équation 2.14. Cette transformation permet de calculer le gradient par rapport à ϕ à l'intérieur de l'espérance sur ϵ échantillonné à partir de la distribution $p(\epsilon) = \mathcal{N}(0, I)$.

2.4 Entraînement de VAE

2.4.1 Minimisation de la fonction de perte

Premièrement, il est obligatoire de définir un estimateur de l'ELBO, ce qui permet d'établir un lien avec une fonction de perte à minimiser. Grâce à la simulation de Monte Carlo, nous pouvons estimer le premier terme à droite de l'Équation 2.11, en échantillonnant L points de l'espace latent $\mathbf{z}$ à partir de la distribution $q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}})$.

$$\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x^{(i)}})} \left[\log p_\theta(\mathbf{x^{(i)}} \mid \mathbf{z}) \right] \approx \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x^{(i)}} \mid \mathbf{z}^{(l,i)}). \tag{2.15}$$

Deuxièmement, en supposant que la distribution a priori $p(\mathbf{z})$ est une **distribution gaussienne** de moyenne nulle et d'écart-type unitaire, nous pouvons exprimer la divergence de Kullback-Leibler entre $q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}})$ et $p(\mathbf{z})$ à l'aide de l'Équation 2.6, définie dans la Section 2.1 :

$$\mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) \parallel p(\mathbf{z}) \right] = -\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log(\sigma^2_{j,\phi}(\mathbf{x^{(i)}})) - \mu^2_{j,\phi}(\mathbf{x^{(i)}}) - \sigma^2_{j,\phi}(\mathbf{x^{(i)}}) \right).$$

L'estimateur du modèle avec le point de donnée $\mathbf{x^{(i)}}$ (Équation 2.11) est alors

$$\tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}}) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x^{(i)}} \mid \mathbf{z}^{(l,i)}) + \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log(\sigma^2_{j,\phi}(\mathbf{x^{(i)}})) - \mu^2_{j,\phi}(\mathbf{x^{(i)}}) - \sigma^2_{j,\phi}(\mathbf{x^{(i)}}) \right) \tag{2.16}$$

Notre objectif est de maximiser cet estimateur de l'ELBO (comme expliqué dans la Section 2.2), ce qui revient à minimiser son opposé. Cette minimisation constitue l'étape centrale de l'entraînement du modèle. Nous définissons ainsi la fonction de perte :

$$L(\phi, \theta; \mathbf{x^{(i)}}) = -\tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}}). \tag{2.17}$$

Lors de l'apprentissage, le modèle ajuste les paramètres de l'encodeur et du décodeur, ϕ et θ, en minimisant cette fonction de perte :

$$\phi, \theta = \arg\max_{\phi,\theta} \tilde{\mathcal{L}}(\phi, \theta; \mathbf{x^{(i)}}) = \arg\min_{\phi,\theta} L(\phi, \theta; \mathbf{x^{(i)}}). \tag{2.18}$$

2.4.2 Algorithme de descente de gradient par mini-lots

Étant donné un ensemble de données d'entraînement $\mathbf{X}$ contenant N points de données, soit :

$$\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \ldots, \mathbf{x}^{(N)}\}, \tag{2.19}$$

nous pouvons estimer la borne inférieure d'évidence de la log-vraisemblance de l'ensemble de données en utilisant des mini-lots :

$$\mathcal{L}(\phi, \theta; \mathbf{X}) \approx \tilde{\mathcal{L}}^M(\phi, \theta; \mathbf{X}^M) = \frac{N}{M} \sum_{i=1}^{M} \tilde{\mathcal{L}}(\phi, \theta; \mathbf{x}^{(\mathbf{i})}). \tag{2.20}$$

Ici, $\mathbf{X}^M = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}\}$ représente un mini-lot de taille M, échantillonné aléatoirement à partir de l'ensemble de données d'entraînement $\mathbf{X}$.
L'entraînement du VAE consiste à minimiser la négative de la borne inférieure d'évidence en calculant le gradient $\nabla_{\phi,\theta}\tilde{\mathcal{L}}^M(\phi, \theta; \mathbf{X}^M)$ via la méthode de descente de gradient. Les paramètres du modèle, ϕ et θ, sont mis à jour à l'aide d'un optimiseur tel que la descente de gradient stochastique (SGD) ou Adam.

L'algorithme d'entraînement du VAE avec mini-lots est présenté dans l'Algorithme 1.

Algorithm 1: Algorithme AEVB avec mini-lots

Input: Données d'entraînement par mini-lots x_n, $n = \{1, \ldots, M\}$,
Nombre d'itérations : T, Taille du mini-lot : B, Taux d'apprentissage : η,
Le dimensionnement de $\mathbf{z}$: J, Le nombre d'échantillonnage de $\mathbf{z}$: L
Estimateur de l'ELBO par mini-lot $\tilde{\mathcal{L}}^M(\phi, \theta; \mathbf{x^{(i)}})$,
Paramètres initiaux ϕ et θ.
Output: Paramètres entraînés ϕ et θ.

```
begin
    φ, θ ← initialisés aléatoirement
    repeat
        t ← t + 1
        n ← 1
        L̃^M ← 0
        for i ∈ {n, ..., n + B − 1} do
            for j ∈ {1, ..., J} do
                L̃^M ← L̃^M + 1/2 (1 + log(σ²_{j,φ}(x^(i))) − μ²_{j,φ}(x^(i)) − σ²_{j,φ}(x^(i)))
            end
            for l ∈ {1, ..., L} do
                ε^(l,i) ∼ N(0, I)  : Echantillonner aléatoirement ε^(l,i)
                z^(l,i) ← g_φ(x^(i), ε^(l,i))  : Reparamétriser de z^(l,i) par g_φ(x^(i), ε^(l,i))
                L̃^M ← L̃^M + log p_θ(x^(i) | z^(l,i))
            end
        end
        θ ← θ + η∇_θ L̃^M
        φ ← θ + η∇_φ L̃^M
        n ← n + B
        if n ≥ N then
            n ← 1
            mélanger les données
        end
    until t = T ou convergé
    return φ et θ
end
```

3 Auto-encodeur Variationnel à Expansion

Dans cette section, nous allons développer une méthode d'expansion de la structure VAE. Notre objectif est de trouver une structure optimale correspondant au travail demandé, ici la génération de nouvelles images à partir du jeu de données MNIST [8].

Nous expliquerons ensuite pourquoi nous utilisons cette méthode pour effectuer des expérimentations sur la structure générative des VAEs et comment nous appliquons cette expansion de la structure.

3.1 Relation entre la taille du modèle et son efficacité

La taille d'un modèle s'exprime principalement par son nombre de paramètres et son nombre de couches. Nous constatons qu'un grand nombre de modèles modernes améliore leurs performances en augmentant ces deux paramètres.

En augmentant le nombre de neurones (nœuds), nous pouvons mieux comprendre la représentation des données, en particulier avec la structure d'autoencodeur [1], où le modèle apprend à représenter les données d'entrée $\mathbf{x}$ dans un vecteur latent (ou code latent) $\mathbf{z}$. Cette représentation latente montre que l'autoencodeur peut compresser les données $\mathbf{x}$ en une forme compacte $\mathbf{z}$ via l'encodeur, puis reconstruire une sortie $\hat{\mathbf{x}}$ via le décodeur.

Par le fait d'ajouter des neurones, nous améliorons la capacité de cette représentation déterministe, c'est-à-dire à trouver un vecteur $\mathbf{z}$ plus riche. Mais en considérant maintenant la structure de l'auto-encodeur variationnel (VAE), cette relation est-elle toujours valable pour ce type de modèle ?

Puisque le principe de base des VAEs ne consiste pas seulement à projeter $\mathbf{x}$ à travers la structure pour obtenir $\mathbf{z}$ et sa reconstruction $\hat{\mathbf{x}}$, le modèle nous aide également à définir une loi implicite de distribution des données d'entrée $\mathbf{X}$ ou la log-vraisemblance des données $\log p_\theta(\mathbf{X})$. Nous avons ensuite mené une expérience pour trouver la réponse à cette question, qui sera expliquée dans les sections suivantes.

3.2 Structure optimale et structure cible

Nous détaillons par nous-même la définition d'une **structure cible** (Figure 2, l'information du dimensionnement de cette structure mentionné dans Section 4.1), qui indique que le modèle possède la pleine capacité de réaliser la tâche de génération. Cela signifie que ce type de structure (avec un entraînement adéquat) a le potentiel de réaliser la tâche de génération de **manière optimale** ou, au minimum, tant qu'une structure de taille supérieure.

La structure cible est également la plus idéale que nous puissions identifier en testant différentes tailles de modèles. En revanche, la structure optimale n'a pas encore été identifiée. Il s'agit d'une taille spécifique et plus réduite des VAEs, où nous constatons de meilleures performances pour la tâche spécifique comparée à la structure cible.

Nous appliquons ensuite l'algorithme d'expansion des neurones dans les deux premières couches de la partie encodeur lors de la phase d'entraînement (voir Section 3.3) et observons à quel moment le modèle commence à afficher de meilleures performances (par exemple, une réduction de la valeur de la fonction de perte).

3.3 Expansion des couches denses

Dans cette section, nous discuterons de la manière dont une couche peut s'étendre en ajoutant des neurones de type MLP pendant la phase d'entraînement.

Nous avons développé, en nous basant sur le code de base de Yifan Wang [9], doctorante au laboratoire LIFAT, une méthode pour ajouter des neurones aux couches MLP. Alors que ses travaux portaient sur l'ajustement des deux dernières couches d'un AlexNet pour la classification, notre cas concerne l'expansion des deux couches cachées de la partie **Encodeur** d'une structure VAE pour la génération de nouvelles images MNIST (voir Figure 2).

Nous avons défini une méthode appelée *expand_layer* dans notre code, qui permet d'ajouter un certain nombre de nouveaux neurones à une couche spécifique de la structure, en fonction de leurs indices. Les nouveaux neurones sont empilés sur les anciens et leurs valeurs sont initialisées aléatoirement via l'initialisation de He [4]. Ensuite, nous ajustons les connexions entre les nouvelles couches croissantes et celles qui précèdent et suivent la couche affectée.

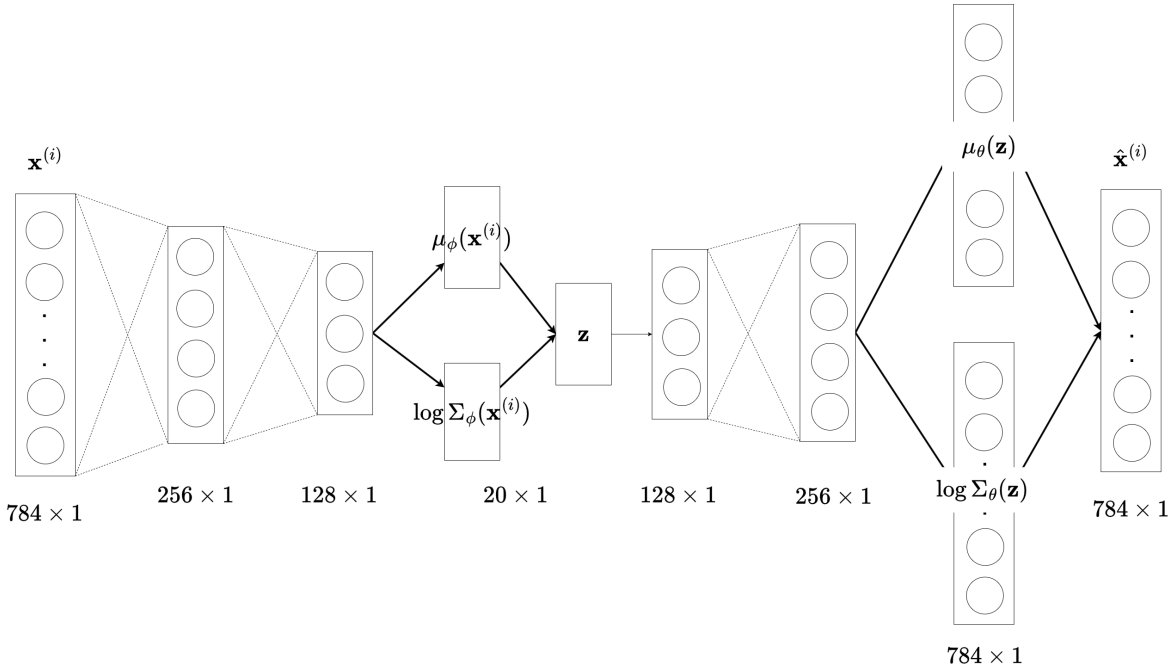

Figure 2 – Visualisation de la structure VAE que nous avons exploités, l'augmentation de neurones se fait dans les deux couches cachées font partie de l'encodeur.

3.4 Métrique pour comparer les performances des modèles

Pour établir une observation probabiliste afin de déterminer si la performance de notre modèle en expansion s'améliore, nous l'évaluons en fonction du nombre d'échantillons L du vecteur latent. Cela nous ramène à l'expression de la log-vraisemblance des données $\log p_\theta(\mathbf{X})$ (Équation 2.2) :

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \mathcal{L}(\phi, \theta; \mathbf{x^{(i)}}) + \sum_{i=1}^{N} \mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x^{(i)}})\right]$$

Nous supposons que dans le cas le modèle atteint l'état où il approxime le distribution postérieure avec l'inférence variationnelle.

$$q_\phi(\mathbf{z} \mid \mathbf{x^{(i)}}) \approx p_\theta(\mathbf{z} \mid \mathbf{x^{(i)}}) \tag{3.1}$$

Supposons que cette condition parfaite soit atteinte et que le modèle apprenne la meilleure représentation implicite des données $\mathbf{X}$. Dans ce cas, l'ELBO devient :

$$\mathcal{L}(\phi, \theta; \mathbf{x^{(i)}}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta\left(\mathbf{x^{(i)}}, \mathbf{z}\right)}{p_\theta\left(\mathbf{z} \mid \mathbf{x^{(i)}}\right)}\right] \tag{3.2}$$

$$\approx \frac{1}{L} \sum_{l=1}^{L} \log \frac{p_\theta\left(\mathbf{x^{(i)}}, \mathbf{z}^{(l,i)}\right)}{p_\theta\left(\mathbf{z}^{(l,i)} \mid \mathbf{x^{(i)}}\right)} \tag{3.3}$$

Nous comprenons ainsi que lorsque le modèle atteint son état optimal, l'ELBO devient presque équivalent à la log-vraisemblance $p_\theta(\mathbf{x^{(i)}})$, indépendante de L.

4 Expérimentation

Dans cette section, nous présentons les différents résultats obtenus lors de nos expérimentations. Nous avons d'abord considéré une petite architecture pour le VAE, puis nous avons défini un nombre fixe d'époques d'entraînement durant lesquelles nous avons progressivement augmenté la taille du modèle.

4.1 Génération des nouvelles images par VAEs

Nous avons d'abord ré-implémenté le modèle VAE en utilisant des configurations variées pour gérer différentes tailles de modèles (voir Figure 4). Après plusieurs analyses, nous avons appliqué le modèle à deux tâches principales : la reconstruction et la génération d'images. Les images de la reconstruction sont satisfaisants, puisque les images reconstruites s'approchent bien des images originales. Les résultats montrent que le modèle fonctionne efficacement avec le jeu de données MNIST lorsque la configuration suivante est utilisée :

- **Encodeur :** $\mathbb{R}^{784\times1} \to \mathbb{R}^{256\times1} \to \mathbb{R}^{128\times1}$
- **Vecteur latent :** $\mathbb{R}^{20\times1}$
- **Décodeur :** $\mathbb{R}^{128\times1} \to \mathbb{R}^{256\times1} \to \mathbb{R}^{784\times1}$

En revanche, pour ce qui est de la génération d'images, nous constatons que celles-ci, obtenues par échantillonnage stochastique du vecteur latent, apparaissent assez floues (voir Figure 3). Ce comportement a également été observé dans la littérature, notamment dans l'article de Kingma [6] à la section 2.8.2. Aucune méthode d'amélioration précise n'a été mise en œuvre dans notre projet pour cette partie. En effet, la génération stochastique peut produire des images qui ne correspondent pas toujours à des chiffres clairement définis ou qui résultent en une combinaison de différents chiffres.

Figure 3 – Exemples des images générées par inférence à partir de décodeur entraîné

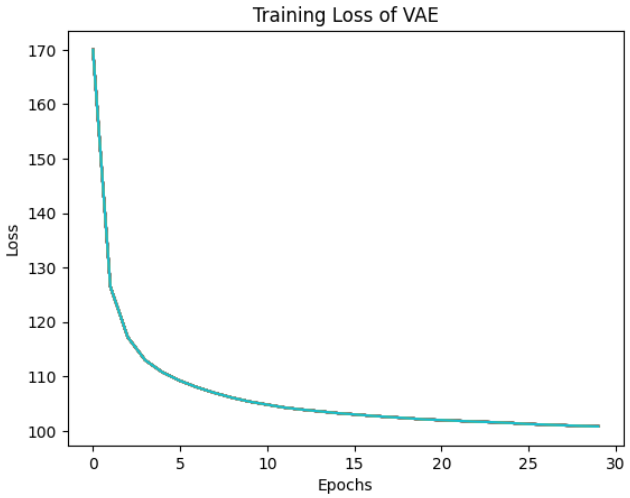

Figure 4 – Fonction de perte pendant le phase d'apprentissage de la structure avec la partie **Encodeur :** $\mathbb{R}^{256\times1} \to \mathbb{R}^{128\times1}$

Étant donné que les résultats obtenus avec cette structure sont satisfaisants (après avoir essayé plusieurs modèles de tailles différentes), elle a été choisie comme **structure cible** pour l'implémentation d'une nouvelle méthode qui consiste à démarrer à partir d'un modèle de petite taille, puis à ajouter des neurones dans les couches cachées pendant l'entraînement

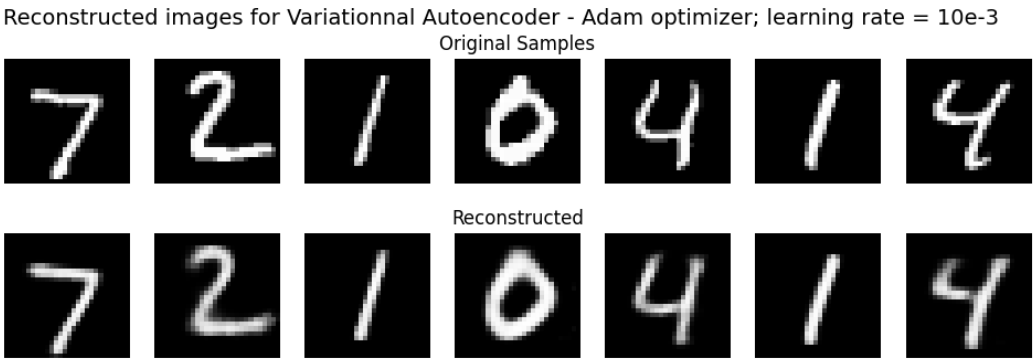

Figure 5 – Images reconstruites par notre structure définie dans Section 4.1

afin d'observer les changements. L'objectif principal est de déterminer des configurations de modèle (en nombre de neurones) plus compactes tout en garantissant une précision satisfaisante pour la reconstruction et la génération d'images, que nous avons nommée **structure optimisée**.

4.2 Expansion en nombre de neurones

Pour trouver la structure optimisée, nous avons essayé d'ajouter de nouveaux neurones pendant l'entraînement, puis comparé les performances de la structure en expansion avec celles d'un modèle à structure fixe, que nous désignons comme structure cible, tant visuellement qu'analytique.

Nous avons adopté deux approches d'implémentation. Dans un premier temps, nous utilisons le modèle défini dans Figure 2 comme référence pour comparer avec le modèle en expansion pendant l'apprentissage. Nous entraînons les deux modèles pendant **200 itérations**, avec $L = 20$. Nous utilisons l'optimiseur *Adam* [5] avec un taux d'apprentissage de $1e^{-3}$, une taille de mini-lots de $B = 128$ et la fonction de perte basée sur l'entropie binaire croisée (BCE).

4.2.1 Approche 1

Notre intention est de débuter avec une architecture réduite pour la partie **Encodeur** du modèle, en utilisant initialement **32 neurones** dans la première couche cachée et **16 neurones** dans la seconde. Toutes les 25 itérations, nous effectuons une expansion en ajoutant de nouveaux neurones dans chacune des deux couches du modèle. Après 7 expansions, le modèle en croissance atteint la même taille que le modèle à structure fixe — que nous appelons structure cible (voir Table 1). Le temps total d'entraînement est de 4034,42 secondes.

Structure optimisée initialement de l'approche 1 :

- **Encodeur :** $\mathbb{R}^{784\times1} \to \mathbb{R}^{32\times1} \to \mathbb{R}^{16\times1}$
- **Vecteur latent :** $\mathbb{R}^{20\times1}$
- **Décodeur :** $\mathbb{R}^{128\times1} \to \mathbb{R}^{256\times1} \to \mathbb{R}^{784\times1}$

Figure 6 illustre l'évolution de la fonction de perte (exprimée en fonction du nombre de points de données) ainsi que celle de la borne inférieure d'évidence (exprimée en nombre de mini-lots) sur 200 époques d'apprentissage, pour les deux modèles : celui en expansion (avec une taille variable) et celui à structure fixe (structure cible). Nous constatons que les deux modèles aboutissent à des valeurs de perte identiques à la fin de l'apprentissage, sans qu'aucune amélioration significative ne soit observée pour le modèle en expansion par rapport au modèle fixe.

Époque de Croissance	Couche Cachée 1	Couche Cachée 2
Etat initial	32	16
1	64	32
2	96	48
3	128	64
4	160	80
5	192	96
6	224	112
7	256	128

Table 1 – Évolution du nombre de neurones dans les deux couches cachées (Approche 1)

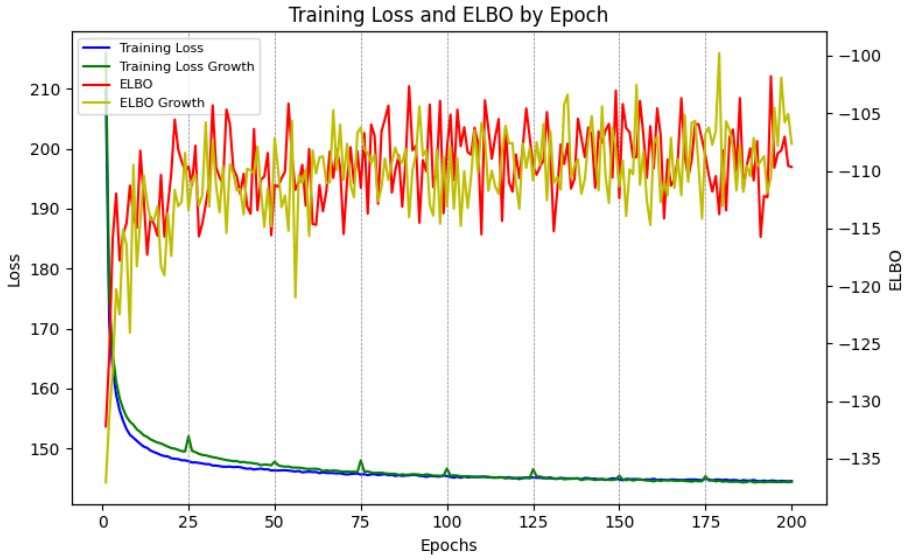

Figure 6 – Cette figure présente l'évolution de la fonction de perte et de l'ELBO lors de l'entraînement du modèle en expansion (Approche 1) comparé au modèle de référence (structure cible - courbe bleue et rouge). On observe que, lors des moments d'expansion des neurones (indiqués par des marques verticales), la courbe de perte affiche des pics ponctuels avant de rapidement se stabiliser.

Par ailleurs, en ce qui concerne la génération d'images, nous ne pouvons pas distinguer visuellement les productions du modèle en expansion de celles du modèle fixe. Bien que nous ayons évalué la variance de l'ELBO sur Figure 6, celle-ci ne permet pas de tirer une conclusion définitive concernant la performance comparée des deux modèles.

4.2.2 Approche 2

Dans cette deuxième approche, nous commençons par un modèle de taille cible. À partir de l'itération 25, nous appliquons un changement : le modèle en expansion est initialisé à partir du modèle cible, mais il conserve uniquement 32 neurones dans la première couche cachée et 16 neurones dans la deuxième.

Structure optimisée initialement d'approche 2 :

— **Encodeur :** $\mathbb{R}^{784\times1} \to \mathbb{R}^{256\times1} \to \mathbb{R}^{128\times1}$

— **Vecteur latent :** $\mathbb{R}^{20\times1}$

— **Décodeur :** $\mathbb{R}^{128\times1} \to \mathbb{R}^{256\times1} \to \mathbb{R}^{784\times1}$

Époque de Croissance	Couche Cachée 1	Couche Cachée 2
Etat initial	256	128
1	64	32
2	96	48
3	128	64
4	160	80
5	192	96
6	224	112
7	256	128

Table 2 – Extension progressive des tailles des couches cachées de l'encodeur (Approche 2)

Dans la même époque de croissance, nous ajoutons respectivement 32 et 16 nouveaux neurones aux deux couches, ce qui donne, dès la première phase d'expansion, une structure optimisée comportant 64 neurones dans la première couche et 32 dans la deuxième (voir Table 2).

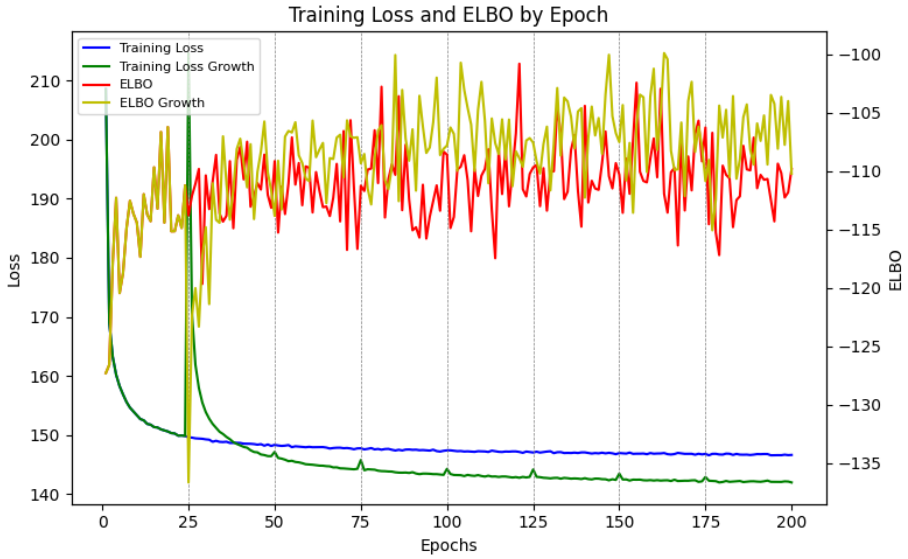

Figure 7 – Évolution de la fonction de perte et de l'ELBO pour l'approche 2. On observe qu'un important pic de la perte apparaît lors de l'ajout initial de neurones, suivi d'une amélioration continue dès l'itération 37 par rapport à la structure cible.

Sur la courbe d'entraînement visualisé dans le 7, nous constatons qu'au premier épisode d'expansion, le modèle en croissance réagit fortement, affichant un important pic de la fonction de perte en raison de l'augmentation soudaine du nombre de neurones.
À partir de la 37e itération, la fonction de perte du modèle en expansion chute de manière significative et continue de diminuer par rapport à celle du modèle à structure cible, tandis que l'ELBO s'améliore également. Cette amélioration suggère que l'expansion de la structure peut potentiellement conduire à de meilleures performances, même si des investigations supplémentaires sont nécessaires pour confirmer cette tendance.

5 Points à améliorer

Dans ce projet de fin d'études, nous avons réussi à appliquer le concept de Deep Learning et d'Auto-encodeur Variationnel [2, 7, 6] en Python et Pytorch, ainsi qu'à développer une

méthode de croissance pour cette structure VAE. Cependant, nous n'avons pas encore progressé suffisamment sur l'évaluation probabilistique des performances des modèles, comme nous l'avions mentionné dans la Section 3.4. De plus, nous n'observons aucune amélioration significative de la fonction de perte lors de l'expansion, comme indiqué dans la Section 4.2.1, contrairement à nos attentes initiales.

Par ailleurs, le phénomène constaté avec l'approche 2 (Section 4.2.2) nous amène à nous interroger sur l'impact de chaque neurone des couches latentes, leur positionnement ainsi que leur influence sur la qualité des images générées. Nous souhaiterions également mettre en œuvre une méthode permettant de mesurer la performance en calculant la variance de la valeur de la borne inférieure (3.4) en fonction du nombre L d'échantillons du vecteur latent, et ce, en l'implémentant sous Python et Pytorch.

6 Conclusion

Dans ce rapport, nous avons examiné l'expansion en nombre de neurones dans les couches de l'encodeur d'un Auto-encodeur Variationnel appliqué sur le jeu de données MNIST. Nos expérimentations, basées sur deux approches différentes, montrent que l'ajout de neurones peut modifier le comportement de la fonction de perte et l'ELBO. Nous avons constaté des réactions immédiates lors des phases d'expansion, sans toujours observer des améliorations claires. Par ailleurs, l'évaluation probabilistique des performances ainsi que l'impact de chaque neurone restent à approfondir. Ces résultats nous ouvrent de nouvelles pistes tant pour affiner notre méthode que pour mieux comprendre l'influence du dimensionnement sur la qualité de la génération d'images.

Table des figures

Liste des tableaux

Références

[1] C.M. Bishop and H. Bishop. *Deep Learning : Foundations and Concepts*. Springer International Publishing, 2023.

[2] Carl Doersch. Tutorial on variational autoencoders, 2021.

[3] Ian Goodfellow et al. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers : Surpassing human-level performance on imagenet classification, 2015.

[5] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[6] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4) :307–392, 2019.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[8] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. 2005.

[9] Yifan Wang. Réseaux de neurones légers pour le machine learning 'sobre'. 2024.