

TP3 : AutoEncoder

Apprentissage non supervisé

SINADINOVIC Marko

Semestre 6 : 2024-2025

Contents

1	Motivation	1
2	Introduction	1
3	Exercice 1	2
3.1	Auto-encoder d'espace latent de dimension dimension p	2
3.1.1	Auto-encodeur avec une couche intermédiaire	3
3.2	Comparaisons	6
3.2.1	Qualité de reconstruction des chiffres à partir de l'espace latent	6
3.3	Erreur quadratique moyenne	9
3.4	Cluster	10

1 Motivation

Dans le cadre du cours de Recherche Opérationnelle (Machine Learning), j'ai été amené à réaliser le TP numéro 3, qui portait sur l'apprentissage non supervisé. Mon objectif était d'implémenter un réseau de neurones auto-encodeur. Passionné par cet univers mêlant mathématiques et informatique, j'ai pris l'initiative de documenter mon travail afin de mieux comprendre les concepts abordés et de faciliter ma préparation pour l'examen.

2 Introduction

Le but de ce TP est d'implémenter un réseau de neurones auto-encodeur. Les auto-encodeurs sont des algorithmes conçus pour réaliser des tâches de compression et de décompression de données. Ces modèles reposent sur des réseaux de neurones qui apprennent automatiquement à accomplir cette tâche, sans nécessiter de labélisation préalable des données, ce qui en fait une méthode d'apprentissage non supervisé.

Le principe de l'auto-encodeur repose sur la représentation des données initiales par des vecteurs $\mathbf{X}_i \in \mathbb{R}^n$ que l'on souhaite compresser en des vecteurs réduits $\mathbf{Z}_i \in \mathbb{R}^p$, avec $p < n$. À cet effet, on construit un réseau de neurones divisé en deux parties :

- **Un encodeur** $E_r : \mathbb{R}^n \rightarrow \mathbb{R}^p$, chargé de compresser les données initiales.
- **Un décodeur** $D_r : \mathbb{R}^p \rightarrow \mathbb{R}^n$, qui reconstruit les données compressées vers leur forme originale.

L'objectif est d'entraîner ce réseau de neurones de manière à ce que la composition des deux parties soit proche de l'identité:

$$D_r \circ E_r \approx \text{Id}.$$

Dans ce TP, cette méthode sera appliquée à la reconnaissance de chiffres calligraphiés, en utilisant un ensemble de données adapté. Cette exploration permettra d'examiner les performances de compression et de reconstruction des auto-encodeurs ainsi que leur application au clustering dans un espace latent réduit.

3 Exercice 1

L'encodeur prend en entrée n valeurs (avec $n = 28 \times 28 = 784$, correspondant au nombre de pixels d'une image) et les réduit dans un vecteur de \mathbb{R}^p . Plus p est petit, plus la compression sera importante. Par exemple, nous pouvons choisir $p = 2$ pour une forte compression ou $p = 16$ pour une compression plus modérée.

Le décodeur, quant à lui, prend en entrée le vecteur compressé dans \mathbb{R}^p et reconstruit un vecteur dans \mathbb{R}^n , dans le but de retrouver une approximation de l'image d'origine. La qualité de reconstruction dépend à la fois de la valeur de p et de l'architecture interne des couches du réseau.

Nous allons comparer les performances du réseau en considérant deux configurations pour l'espace latent: $p = 2$ et $p = 16$. Ces comparaisons permettront de mettre en évidence les compromis entre le degré de compression et la fidélité de la reconstruction.

3.1 Auto-encoder d'espace latent de dimension p

Un espace latent est une représentation compressée et abstraite des données initiales. C'est la sortie de l'encodeur, qui transforme des données de haute dimension en un vecteur de dimension réduite, noté p . Cet espace latent préserve les caractéristiques principales des données tout en éliminant les détails non essentiels pour leur reconstruction. Cet espace latent a plusieurs propriétés essentielles. Dans un premier temps, la **compression de données**, en effet, il "réduit" la taille des données tout en conservant leurs principaux attributs. Ensuite nous avons **l'extraction de motifs** qui permet un output des données (motifs) comme elles étaient dans un espace augmenté.

De ces faits, le choix de la dimension de l'espace latent (p) a un impact significatif sur les performances du réseau :

- **Petite dimension ($p = 2$)** : Une faible dimension est synonyme de forte compression. Le défaut d'une petite dimension est la perte d'information, qui est importante. Cela rend difficile la différenciation de données qui se ressemblent, par exemple ici le chiffre 5 avec le chiffre 8 ou le 4 avec 9.

- **Grande dimension ($p = 16$)** : Une dimension plus élevée permet de conserver davantage d'informations. La reconstruction des données est donc plus fidèle. Or, une grande dimension rend le modèle plus complexe à entraîner (temps de calculs et mémoire plus important) et la compression des données est moins élevée mais l'objectif principal des auto-encodeurs est de compresser les données. Donc une très grande dimension de l'espace de latence diminue voir annule l'intérêt de l'utilisation d'un auto-encodeur.

3.1.1 Auto-encodeur avec une couche intermédiaire

Nous disposons d'un jeu de données de 60000 items de taille 28x28, soit 784 pixels. Construisons notre réseau de neurones constitué de deux parties, la partie encodeur, et la partie décodeur, expliqué dans l'introduction. En remplaçant p par 2, nous obtenons le code suivant :

```

1 import torch.nn as nn
2
3 class AutoEncoder(nn.Module):
4     def __init__(self):
5         super(AutoEncoder, self).__init__()
6         # Definition de l'encodeur ( $R^n \rightarrow R^p$  avec une couche intermediaire)
7         self.encoder = nn.Sequential(
8             nn.Linear(28 * 28, 128), # Couche intermediaire
9             nn.Tanh(),               # Fonction d'activation Tanh
10            nn.Linear(128, 2)         # Espace latent de dimension 2
11        )
12        # Definition du decodeur ( $R^p \rightarrow R^n$  avec une couche intermediaire)
13        self.decoder = nn.Sequential(
14            nn.Linear(2, 128),        # Couche intermediaire
15            nn.Tanh(),               # Fonction d'activation Tanh
16            nn.Linear(128, 28 * 28), # Reconstruction vers  $R^n$ 
17            nn.Sigmoid()             # Fonction d'activation Sigmoid
18        )
19
20    def forward(self, x):
21        encoded = self.encoder(x)    # Encodage
22        decoded = self.decoder(encoded) # Decodage
23        return encoded, decoded

```

Listing 1: Auto-encodeur avec une couche intermédiaire et espace latent de dimension deux

Ce code comporte qu'une seule couche intermédiaire, qui est insuffisant pour bien traiter notre problème. Pour l'instant nous nous restreignons à cette structure de code. Par le même procédé, pour $p = 16$:

```

1 import torch.nn as nn
2
3 class AutoEncoder(nn.Module):
4     def __init__(self):
5         super(AutoEncoder, self).__init__()
6         # Definition de l'encodeur (R^n -> R^p avec une couche intermediaire)
7         self.encoder = nn.Sequential(
8             nn.Linear(28 * 28, 128), # Couche intermediaire
9             nn.Tanh(),               # Fonction d'activation Tanh
10            nn.Linear(128, 16)        # Espace latent de dimension 16
11        )
12        # Definition du decodeur (R^p -> R^n avec une couche intermediaire)
13        self.decoder = nn.Sequential(
14            nn.Linear(16, 128),       # Couche intermediaire
15            nn.Tanh(),               # Fonction d'activation Tanh
16            nn.Linear(128, 28 * 28), # Reconstruction vers R^n
17            nn.Sigmoid()             # Fonction d'activation Sigmoid
18        )
19
20    def forward(self, x):
21        encoded = self.encoder(x)    # Encodage
22        decoded = self.decoder(encoded) # Decodage
23        return encoded, decoded

```

Listing 2: Auto-encodeur avec une couche intermédiaire et espace latent de dimension seize

Maintenant que nous avons en notre possession 2 auto-encodeurs de dimensions latentes $p = 2$ et $p = 16$, nous pouvons les entraîner. Pour ce faire, nous allons adopté une approche d'optimisation basée sur l'utilisation de la fonction coût MSE qui mesure l'erreur quadratique moyenne entre l'entrée initiale et la sortie reconstruite. Mathématiquement, elle est représenté de la manière suivante :

Définition 1. Erreur Quadratique Moyenne

$$MSE = \mathbb{E} \left[(Y - \hat{Y})^2 \right]$$

où:

- Y est la variable aléatoire représentant les valeurs réelles.
- \hat{Y} est la variable aléatoire représentant les prédictions faites par le modèle.
- \mathbb{E} désigne l'espérance (ou la moyenne).

L'erreur quadratique moyenne (ou Mean Squared Error, MSE) mesure la différence moyenne entre les valeurs réelles Y et les valeurs prédites par un modèle \hat{Y} :

$$MSE = \mathbb{E} \left[(Y - \hat{Y})^2 \right]$$

où:

- Y est la variable aléatoire représentant les valeurs réelles.
- \hat{Y} est la variable aléatoire représentant les prédictions faites par le modèle.

- \mathbb{E} désigne l'espérance (ou la moyenne).

L'erreur quadratique moyenne peut être décomposée en trois termes : le biais, la variance, et l'erreur irréductible. Cette décomposition, appelée théorème du biais-variance, s'exprime ainsi:

$$\text{MSE} = \underbrace{\left(\mathbb{E}[\hat{Y}] - Y\right)^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}\left[(\hat{Y} - \mathbb{E}[\hat{Y}])^2\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur irréductible}}$$

- **Le biais** représente l'erreur systématique introduite par le modèle. Il correspond à la différence entre la vraie valeur Y et la moyenne des prédictions $\mathbb{E}[\hat{Y}]$.
- **La variance** décrit la variabilité des prédictions \hat{Y} autour de leur moyenne $\mathbb{E}[\hat{Y}]$. Une variance élevée indique que le modèle est très sensible aux variations des données d'entraînement.
- **L'erreur irréductible** σ^2 est due au bruit inhérent dans les données, qui ne peut être réduit par aucun modèle.

En outre, elle évalue la qualité de reconstruction du réseau et guide l'apprentissage en minimisant l'erreur.

Pour optimiser les paramètres du réseau, nous avons utilisé l'algorithme **Adam** qui ajuste les taux d'apprentissage pour chaque paramètre, ce qui accélère la convergence. Cependant, étant donné le grand nombre d'images (60 000) dans l'ensemble d'entraînement, il serait coûteux de calculer la fonction de coût sur l'intégralité des données à chaque itération. Nous avons donc opté pour une optimisation par *batches*, où les données sont divisées en lots de taille fixe à l'aide de la fonction `DataLoader` de PyTorch. Le code d'optimisation est donné ci-dessous :

```

1 BATCH_SIZE = 64
2 train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
   shuffle=True)
3
4 EPOCH = 100
5 LEARNING_RATE = 0.005
6
7 autoencoder = AutoEncoder() # Modele a entrainer
8 criterion = nn.MSELoss() # Fonction de cout
9 optimizer = torch.optim.Adam(autoencoder.parameters(), lr=LEARNING_RATE) # ADAM
10
11 # Boucle d'entrainement
12 for epoch in range(EPOCH):
13     for step, (x, _) in enumerate(train_loader):
14         inputX = x.view(-1, 28 * 28) # Transformation en vecteurs
15         _, decoded = autoencoder(inputX) # Encodage et reconstruction
16
17         loss = criterion(decoded, inputX) # Calcul de la perte
18         optimizer.zero_grad() # Reinitialisation des gradients
19         loss.backward() # Calcul des gradients
20         optimizer.step() # Mise a jour des parametres
21

```

Listing 3: Optimisation de l'auto-encodeur avec ADAM

À l'issue de l'entraînement, les performances des modèles avec $p = 2$ et $p = 16$ seront comparées. Cette analyse portera sur :

- La qualité de reconstruction des chiffres à partir de l'espace latent.
- La représentation des clusters dans l'espace latent, en visualisant les différences entre $p = 2$ et $p = 16$ (plus tard, je n'ai pas eu le temps désolé).
- La convergence de la fonction de coût pour chaque configuration .

Ces comparaisons permettront d'évaluer l'impact de la dimension de l'espace latent sur les performances de l'auto-encodeur.

3.2 Comparaisons

Nous allons comparer les performances du réseau dans deux configurations : $p = 2$ et $p = 16$. Cette analyse permettra de mettre en évidence les compromis entre le degré de compression et la qualité de reconstruction, et la différence entre une et plusieurs couches de neurones.

3.2.1 Qualité de reconstruction des chiffres à partir de l'espace latent

A) Auto-encodeur avec une couche intermédiaire



Figure 1: Reconstruction avec un auto-encodeur d'une couche intermédiaire, de dimension d'espace latent 2 et 100 EPOCH.



Figure 2: Reconstruction avec un auto-encodeur d'une couche intermédiaire, de dimension d'espace latent 16 et 100 EPOCH.

Nous remarquons que dans la Figure 1 les images décodées ont comme un effet de flou, comme un bruit, et certains chiffres ressemblent à d'autres comme le 5 qui ressemble à un 8 ou le 4 qui ressemble à un 9. C'est l'inconvénient d'un espace latent de petite dimension, on a bien la compression des

données, mais il existe une perte d'information et une confusion. Contrairement à la Figure 2, d'espace latent de dimension 16 où on distingue parfaitement les chiffres décodés mais on se doute qu'ils ne sont pas autant compressé que dans le cas de l'espace latent de plus petite dimension.

B) Auto-encodeur avec une deux couches intermédiaires

Introduisons notre code avec deux couches intermédiaires avec des fonctions d'activations pour un espace latent p quelconque à définir :

```

1 import torch.nn as nn
2
3 class AutoEncoder(nn.Module):
4     def __init__(self):
5         super(AutoEncoder, self).__init__()
6         # Definition de l'encodeur ( $R^n \rightarrow R^p$  avec deux couches intermediaires)
7         self.encoder = nn.Sequential(
8             nn.Linear(28 * 28, 256), # Premiere couche intermediaire
9             nn.Tanh(), # Fonction d'activation Tanh
10            nn.Linear(256, 128), # Deuxieme couche intermediaire
11            nn.Tanh(), # Fonction d'activation Tanh
12            nn.Linear(128, p) # Espace latent de dimension p
13        )
14        # Definition du decodeur ( $R^p \rightarrow R^n$  avec deux couches intermediaires)
15        self.decoder = nn.Sequential(
16            nn.Linear(p, 128), # Premiere couche intermediaire
17            nn.Tanh(), # Fonction d'activation Tanh
18            nn.Linear(128, 256), # Deuxieme couche intermediaire
19            nn.Tanh(), # Fonction d'activation Tanh
20            nn.Linear(256, 28 * 28), # Reconstruction vers  $R^n$ 
21            nn.Sigmoid() # Fonction d'activation Sigmoid
22        )
23
24    def forward(self, x):
25        encoded = self.encoder(x) # Etape d'encodage
26        decoded = self.decoder(encoded) # Etape de decodage
27        return encoded, decoded

```

Listing 4: Auto-encodeur avec deux couches intermédiaires

Après l'entraînement, nous obtenons :



Figure 3: Reconstruction avec un auto-encodeur de 2 couche intermédiaire, de dimension d'espace 2 deux et 100 EPOCH.

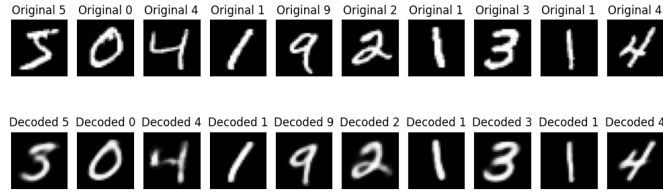


Figure 4: Reconstruction avec un auto-encodeur de 2 couche intermédiaire, de dimension d'espace latent 8 et 100 EPOCH.

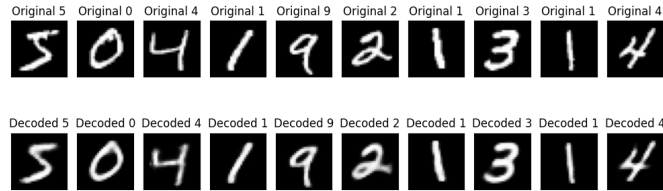


Figure 5: Reconstruction avec un auto-encodeur de 2 couche intermédiaire, de dimension d'espace latent 16 et 100 EPOCH.

Nous remarquons que, même avec un EPOCH relativement élevé (epoch = 100), la compression de l'auto-encodeur avec 2 couches intermédiaires et une dimension d'espace latent de 2, des imprécisions persistent pour les chiffres ayant les mêmes caractéristiques. Or, comme la théorie le suggère pour les dimensions élevées, nous distinguons bien les chiffres, mais on se rend compte que le but de l'auto-encodeur n'est pas pleinement rempli: nous n'avons pas réellement une compréhension complète de l'image.

C) Auto-encodeur avec une deux couches intermédiaires en utilisant des couches batch



Figure 6: Reconstruction avec un auto-encodeur de 2 couche intermédiaire, de dimension d'espace latent 2, avec des couches intermédiaires batch et 100 EPOCH.

Il y a de l'amélioration mais certaines confusions persistent. En effet, la partie haute du chiffre 4 se confond encore, de manière légère avec le haut du chiffre 9. Cette confusion peut être expliquée par la dimension de l'espace latent mais aussi à cause des fonctions d'activation Tanh et ReLU qui ne sont peut-être pas adaptés à ce genre de modèle.

3.3 Erreur quadratique moyenne

Un des principaux caractères qu'on se doit d'étudier lorsqu'on réalise un apprentissage non supervisé (et même supervisé) est la convergence de la fonction coût, en d'autres termes nous étudions la qualité de reconstruction du réseau. Cette notion est essentielle car elle reflète la performance du modèle tout au long de l'entraînement. Une bonne convergence de la fonction MSE montre que le modèle parvient à apprendre efficacement à partir des données fournies, en revanche, une absence de convergence voire une vitesse de convergence lente peut indiquer des problèmes d'optimisation ou du learning rate.

Nous allons étudier deux graphiques, deux courbes, représentant la convergence de la fonction coût pour deux réseaux différents. Ces graphiques permettent de visualiser l'évolution de la perte (loss) au fil des itérations. Dans un premier temps, nous examinerons la convergence de la fonction coût d'un auto-encodeur possédant un espace latent de dimension 2 et 2 couches intermédiaires de neurones et dans un second temps, la convergence de la fonction MSE d'un auto-encodeur possédant un espace latent de dimension 2 et 4 couches intermédiaires de neurones.

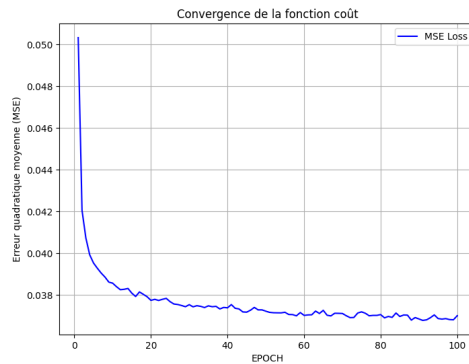


Figure 7: Convergence de la fonction coût de l'auto-encodeur avec un espace latent de dimension 2 et avec 2 couches de neurones intermédiaires.

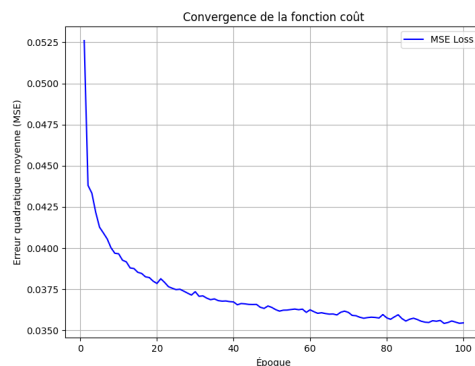


Figure 8: Convergence de la fonction coût de l'auto-encodeur avec un espace latent de dimension 2, avec 2 couches de neurones intermédiaires et des couches batch intermédiaires.

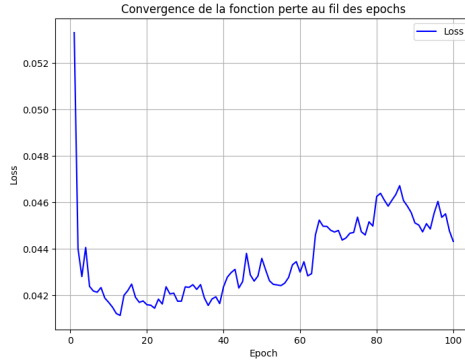


Figure 9: Convergence de la fonction coût de l’auto-encodeur avec un espace latent de dimension 2 et avec 4 couches de neurones intermédiaires.

Nous pouvons remarquer que la convergence avec un espace latent de 2 dimensions et 2 neurones, qui est presque une configuration minimale, l’auto-encodeur traite un nombre restreint de paramètres et donc la convergence est plus rapide à cause de cette simplicité. Or, ce modèle qui dispose peu de calculs, ne peut pas traiter des informations complexes, et la fonction coût reste stable mais avec une valeur encore élevée après 100 EPOCH, ici $\text{loss} = 0.037$.

En ce qui concerne la convergence avec un espace latent de 2 dimensions et 2 neurones avec des couches intermédiaires batch, nous obtenons une stabilité de l’entraînement. Nous voyons sur le graph Figure 8 que la convergence se fait de manière rapide, sans avoir de grande variation grâce à ces couches intermédiaires batch. La courbe est plus lisse comparé à la Figure 7. On obtient un $\text{loss} = 0.0358$, plus faible que pour le cas numéros 1 pour le même nombre d’itérations.

Enfin, en augmentant le nombre de neurones dans l’espace latent, le réseau dispose de plus de capacité pour encoder les caractéristiques des données au prix d’une convergence plus lente avec de fortes variations pour un nombre d’itération $\text{EPOCH} = 100$ et un coût de calcul plus important. On peut aussi expliquer ces variations par l’utilisation de fonction d’activation Tanh et ReLU.

3.4 Cluster

Un cluster est un regroupement ou un ensemble d’éléments qui présentent des similitudes ou des caractéristiques communes. Dans le contexte de l’analyse de données, un cluster désigne un groupe de points de données qui sont plus proches ou plus semblables les uns aux autres qu’aux points appartenant à d’autres groupes. [”Pattern Recognition and Machine Learning” par Christopher M. Bishop (2006)].

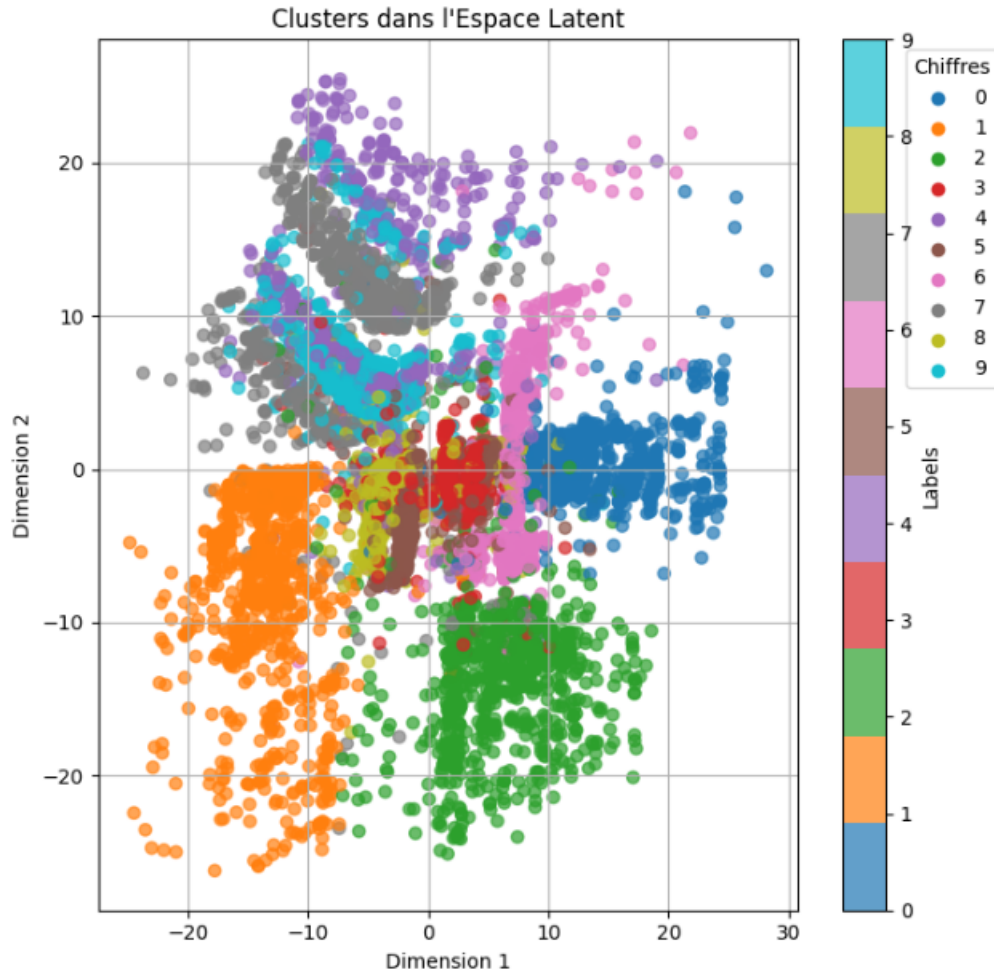


Figure 10: Représentation cluster dans l'espace latent de dimension 2 avec 2 couches de neurones avec 100 itérations

Ce graphique illustre la projection des données dans un espace latent à deux dimensions obtenu à partir d'un auto-encodeur. Chaque point représente une image de chiffre compressée en deux dimensions, et les couleurs correspondent aux différentes classes (chiffres de 0 à 9). On observe des regroupements bien définis pour certaines classes, comme les chiffres 1 et 2 ou encore 0 avec le 1 ce qui montre que le réseau a bien appris à les différencier. Cependant, les chiffres 4 et 9 montrent des regroupements moins distincts, ce qui pourrait indiquer que ces classes partagent des caractéristiques similaires dans l'espace latent. Comme ce qu'on a remarqué pour les représentations dans la section 3.2) Comparaison.

Le but premier de cette interprétation graphique est de comprendre la manière dont les données sont organisées dans l'espace latent, qui aide à évaluer la capacité de l'auto-encodeur à effectuer la compression et la reconstruction. En améliorant notre auto-encodeur, en utilisant d'autres fonctions d'activations, d'autres méthodes de reconnaissance comme le zoom, la rotation et autres, certaines classes pourraient bénéficier d'une meilleure distinction.