Enseignement d'approfondissement - Projet VQ-VAE Images

Vincent-Adam Alimi, Felix Rosseeuw, Jules Cognon

Sommaire

I

Introduction du problème, présentation de l'article

L'article — Mai 2018

Neural Discrete Representation Learning

Aaron van den Oord
DeepMind
avdnoord@google.com

Oriol Vinyals
DeepMind
vinyals@google.com

Koray Kavukcuoglu
DeepMind
korayk@google.com

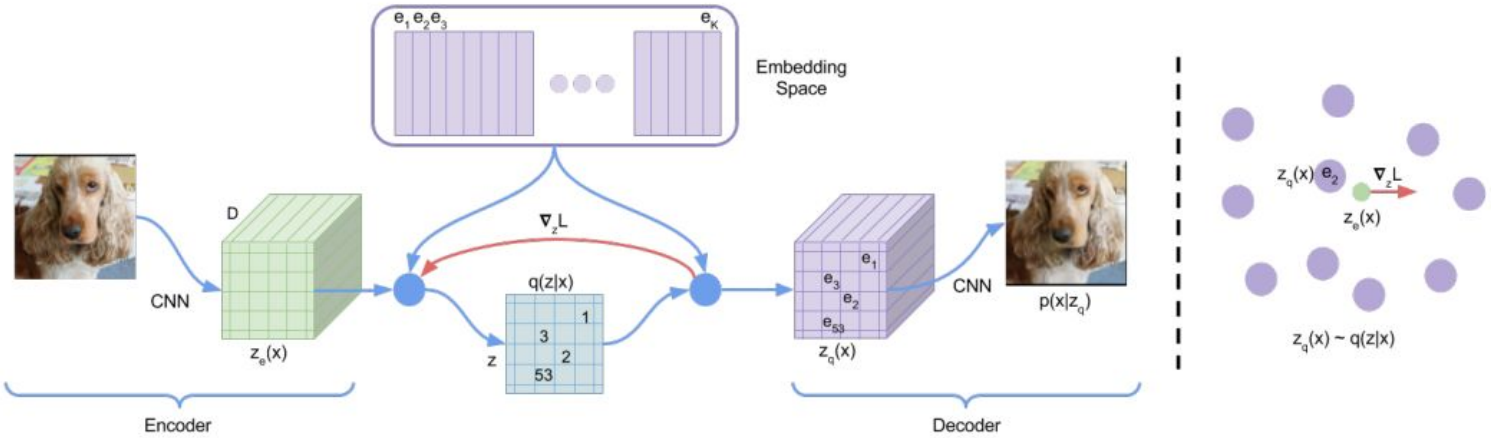
Contexte & Motivations

Modèles continus :
- Historiquement les plus représentés
- Recherche de décodeur le plus performant
- Les espaces latents continus pas toujours les plus intuitifs

Problème : “Posterior Collapse” :
- A lieu dans le cas d’un décodeur trop puissant, utilisé avec un espace latent continu
- Génération d’une image générique qui ne prend pas en compte les détails

Notre approche:
- Introduction de latents discrets, plus adaptés pour des tâches notamment liées à la vision.
- Préserver les caractéristiques importantes des données dans l’espace latent tout en optimisant le maximum de vraisemblance
- Le VQ-VAE quantifie les latents en les associant à un ensemble fini de codes discrets

Architecture de VQ-VAE



- Définition d'un espace d'embeddings *e* de taille *K*, où chaque vecteur est de dimension *D*
- L'encodeur transforme l'entrée en une représentation continue
- La représentation est quantifiée en sélectionnant l'élément du dictionnaire *e* le plus proche

$$z_q(x) = e_k, \quad \text{where} \quad k = \text{argmin}_j \|z_e(x) - e_j\|_2$$

- Le décodeur reconstruit ensuite l'entrée à partir de cet embedding discret

II

Entrainement du modèle

Première étape : Entrainement de l'auto-encodeur

Problème : La quantification n'est pas différentiable, ce qui bloque la rétropropagation du gradient.

Solution : Straight-Through Estimator (STE) : Lors de la phase de rétropropagation, le gradient du décodeur est copié directement vers l'encodeur, ignorant l'opération de quantification. Cela permet d'entraîner l'encodeur tout en conservant des latents discrets

Fonction de perte :

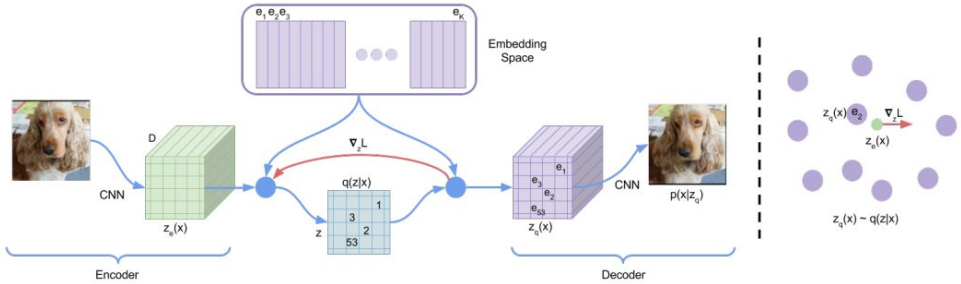
$$L = \underbrace{\log p(x|z_q(x))}_{\text{Erreur de reconstruction}} + \underbrace{\|sg[z_e(x)] - e\|_2^2}_{\text{Perte de quantification}} + \underbrace{\beta\|z_e(x) - sg[e]\|_2^2}_{\text{Pénalité d'engagement}}$$

Partie générative : Choix du Prior

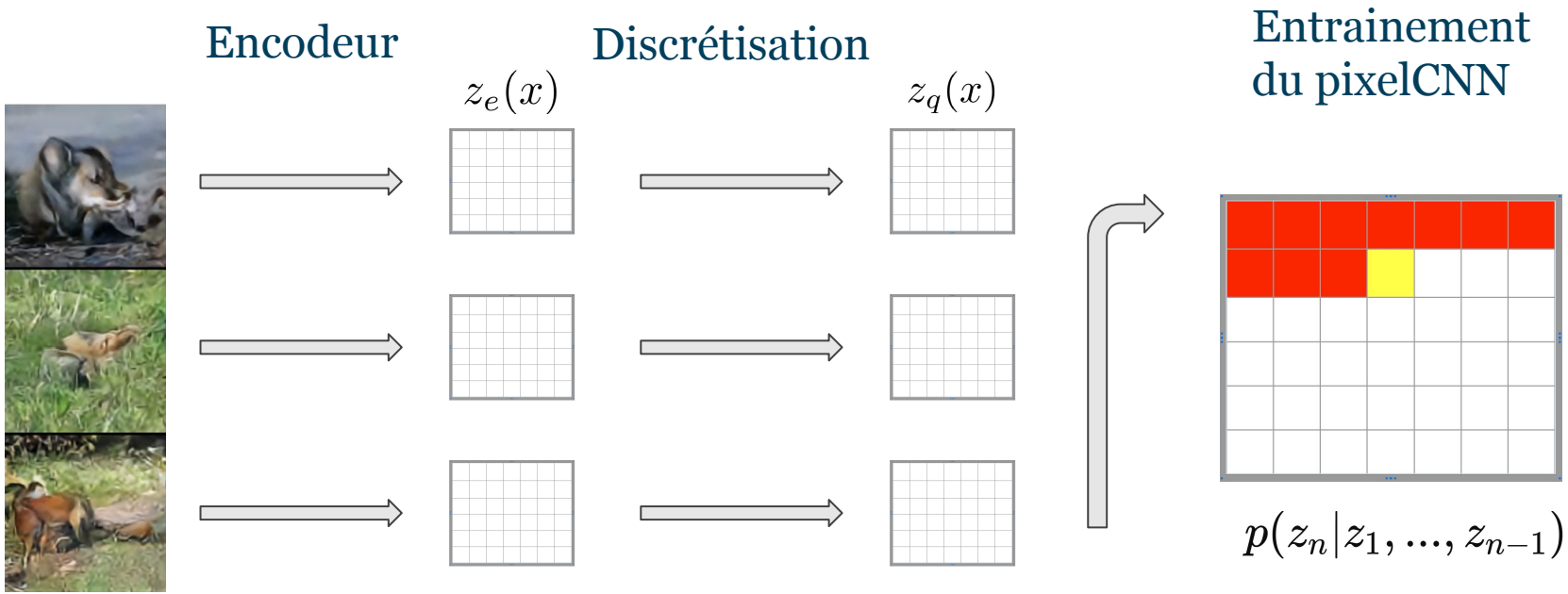
Dans notre cas, on n'utilise pas de prior gaussien, mais on introduit le prior de la manière suivante :

- Pendant l'entraînement, le prior est gardé constant et uniforme
- Après l'entraînement, on génère x via un prior calculé avec une **distribution autorégressive :**

$$p(z) = \prod_i p(z_i|z_{<i}) \quad \text{(Prior PixelCNN)}$$



Deuxième étape : entrainement du PixelCNN



Génération d'image

z_1

$p(z_n|z_1, \ldots, z_{n-1})$

Décodeur

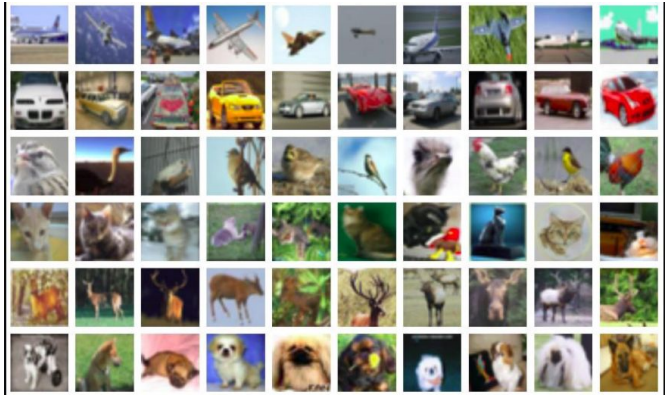
III

Implémentations

Datasets FashionMNIST et Cifar10

- Images 28x28 en nuance de gris
  -> transformation nécessaire
- Disponible sur torchvision.datasets
- 70 000 images

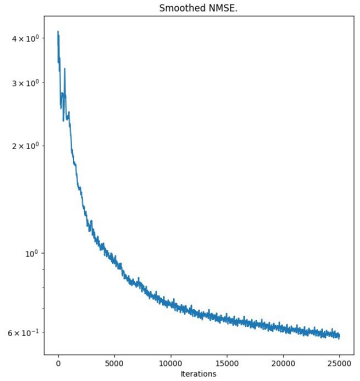
- Images 32x32 en couleur
- Disponible sur torchvision.datasets
- 60 000 images

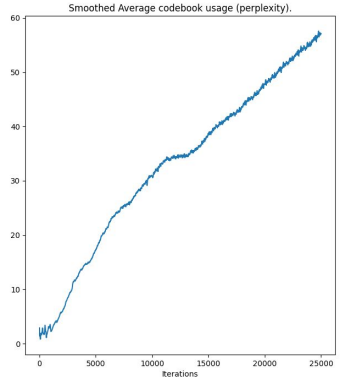


Exécution sur Cifar10 et FashionMNIST

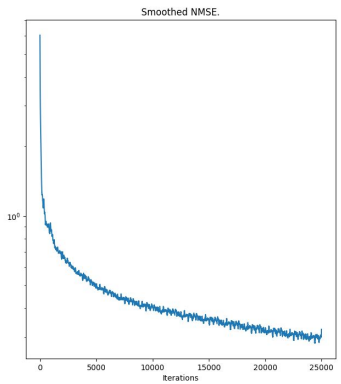
- tâche de régénération
- 25 000 itérations
- K = 64

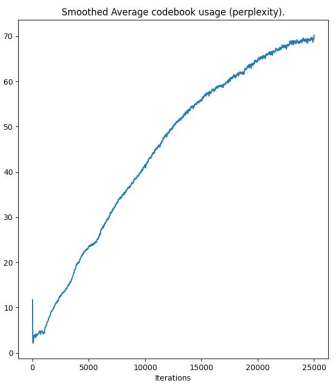
Suivi de la loss et de la perplexité





Cifar10

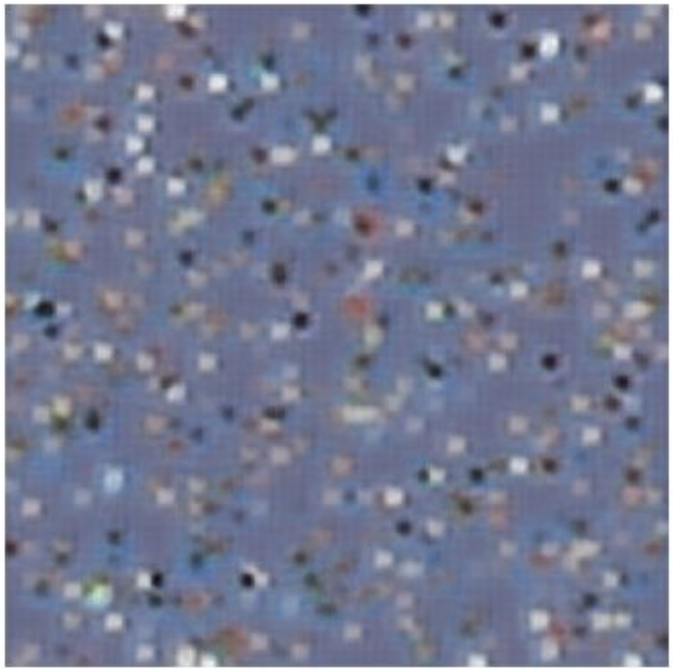




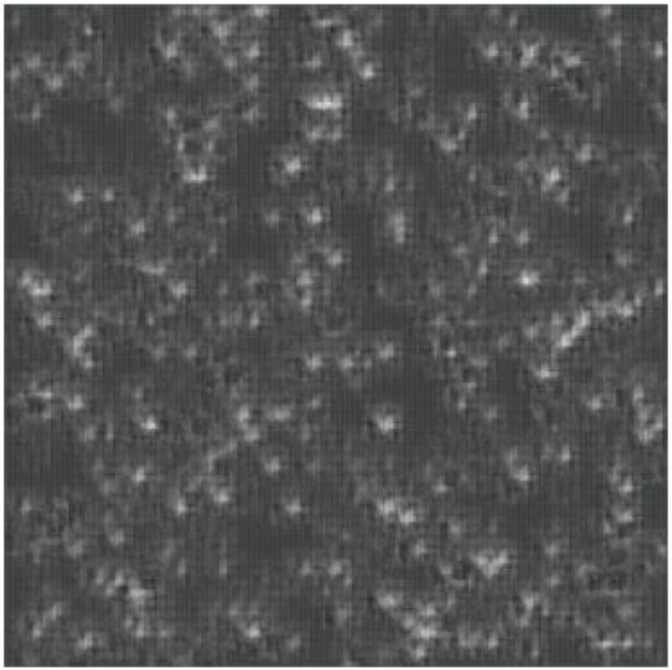
FashionMNIST

Étude de l'impact de K

Génération d'image avec prior uniforme



prior uniforme décodé - cifar10



prior uniforme décodé - FashionMNIST

Perspectives d'approfondissement

- Optimiser d’autres paramètres que K
- Implémenter PixelCNN
- Entraîner le prior avec d’autres méthodes
  - La tâche de régénération peut s’assimiler à une tâche de falsification, l’utilisation d’un GAN peut être pertinente
- Changer le 1-NN de la quantization en k-NN et optimiser k