TP: Variational AutoEncoder

L3-IA, Méthodes d'Apprentissage

Introduction

Le but de ce TP est d'implémenter un Variational AutoEncoder (VAE). Vous commencerez par implémenter un AutoEncoder simple, puis vous ajouterez le nécessaire pour le transformer en un Variational AutoEncoder.

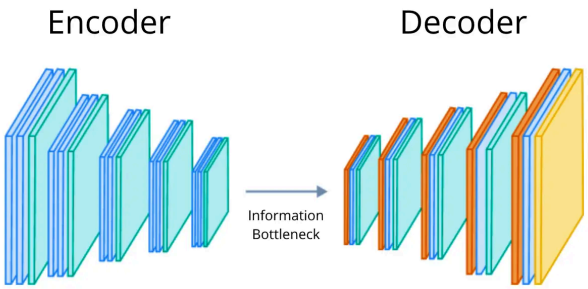

Figure 1: Schéma d'un AutoEncoder

Un AutoEncoder est un modèle entraîné à compresser son input (typiquement, une image). Il est divisé en deux modules : un encodeur et un décodeur. L'encodeur projette l'input dans un espace de dimension plus faible. Le décodeur part de la représentation latente produite par l'encodeur et la projette dans l'espace original.

Les deux modules sont entraînés conjointement de manière auto-supervisé. On souhaite simplement que la sortie du décodeur soit la même image que celle donnée en entrée dans l'encodeur. La loss est simplement la MSE appliqué sur l'entrée et la sortie du modèle :

$$h = E(x)$$
$$\hat{x} = D(h)$$
$$L = \|x - \hat{x}\|^2$$

L'encodeur et le décodeur sont symmétriques. Ce sont des CNNs contenant des séries de residual blocks et des couches de compression/décompression.

Le Variational AutoEncoder est similaire à un AE sauf qu'il considère la représentation latente h comme étant une variable aléatoire gaussienne. De plus, une loss additionnelle de régularisation force le modèle à produire des représentations latentes proches de l'origine (la distribution normale).

Cette loss de régularisation est ce qui nous permet de tirer des points h aléatoires autour de l'origine et de les donner au décodeur afin de générer une nouvelle image, interprétée comme si h provenait d'une image compressée par l'encodeur.

Vous allez tester votre modèle sur le dataset suivant: https://www.kaggle.com/datasets/splcher/animefacedataset.

Vous devrez vous faire un compte sur le site https://wandb.ai afin de tracker vos entraînements.

AutoEncoder

Commencez par implémenter l'AutoEncoder classique. Implémentez les couches dans `src/model.py` telles qu'elles sont décrites en commentaire. Implémentez aussi les fonctions `Trainer.batch_update` et `Trainer.batch_metrics`.

Vous pouvez ensuite tester votre modèle sur le dataset et chercher des hyperparamètres qui fonctionnent bien. De manière général, vous devriez commencer par de petits modèles ($< 250,000$ paramètres) sur de petites images ($\sim 32 \times 32$ pixels).

Votre modèle est-il performant dans sa reconstruction ? Qu'en est-il pour la génération d'images ?

Variational AutoEncoder

Le VAE associe chaque image à une distribution gaussienne localisée proche de l'origine de l'espace latent. Cela nous permet de s'assurer que les points autour de l'origine sont effectivement interprétables par notre décodeur et augmente ainsi grandement la performance en terme de génération d'image.

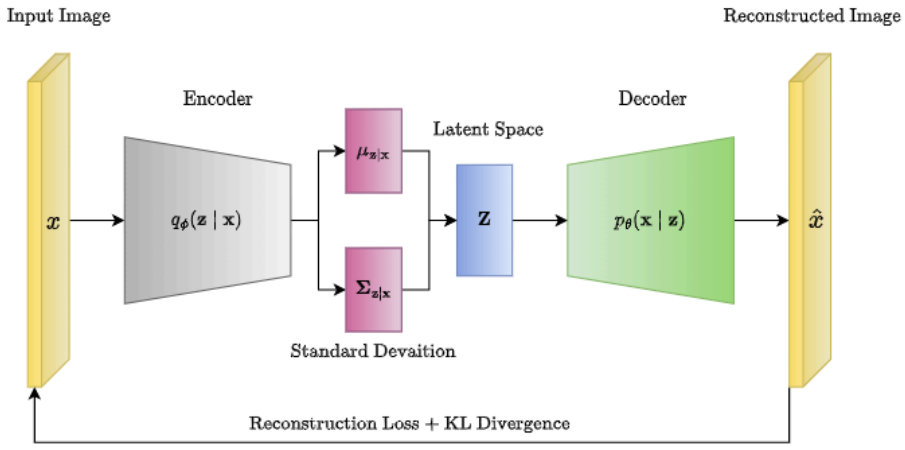

Figure 2: Schéma d'un VAE

Vous devez donc modifier légèrement votre implémentation. Vous devez maintenant utiliser la couche `VAE.project_latent` afin de générer deux vecteurs latents par image: `(mu, logvar)`, représentant respectivement la moyenne et la log-variance de la gaussienne associée à chaque image. Utilisez la fonction `VAE.reparameterize` pour sampler un point suivant les gaussiennes prédites afin de les redonner au decoder. Enfin, ajoutez la loss de régularisation dans les fonctions `trainer.batch_update` et `trainer.batch_metrics`. Pensez à utiliser le coefficient de régularisation `trainer.kl_weight` afin de contrôler la force de régularisation. La loss de régularisation est donnée par l'expression suivante :

$$\mathrm{KL} = -\frac{1}{2} * (1 + \mathrm{logvar} - \mathrm{mu}^2 - \exp(\mathrm{logvar}))$$

Pour donner une intuition à propos de cette loss, elle mesure la distance entre la gaussienne prédite par le modèle et une loi normale. Minimiser cette loss revient à pousser le modèle à prédire des gaussiennes similaires à des lois normales.

Enfin une fois que cela est fait, vous pouvez chercher de bons hyperparamètres pour entraîner votre modèle ! Comme pour l'AE, commencez par une petite version (petit modèle, petites images), progressivement augmentez la tailles des images et du modèle afin de trouver ce qui fonctionne bien.

Expérimentez avec plusieurs valeurs `kl_weight`. Que se passe-t-il lorsque ce paramètre est trop petit ou trop grand ?

Je vous conseille de manière générale de tout faire sur votre ordinateur sur CPU jusqu'à ce que votre code fonctionne et que vous souhaitez entraîner un gros modèle. Votre plus gros modèle ne devrait pas dépasser les 20M paramètres, et ce qui devrait tenir sur un GPU avec au moins 6Go de VRAM.