Due Date: May 8th 23:00, 2023
Instructions

- Ce devoir est difficile - commencez le bien en avance.
- Pour toute les questions, montrez votre travail!
- Soumettez votre rapport (PDF) et votre code électroniquement via la page Gradescope du cours.
- Les TAs pour ce devoir sont: **Reza Bayat** et **Johan S. Obando C.**

Dans ce devoir, vous devrez mettre en œuvre deux modèles génératifs différents qui sont largement populaires dans la littérature du Machine Learning, à savoir les Autoencodeurs variationnels (VAE) et les Modèles de diffusion et l'algorithme de Simsiam, un modèle d'appretissage auto-supervisé. Vous expérimenterez les VAE et les modèles de diffusion sur le jeu de données SVHN dataset, composé d'images de numéros de maisons vues de la rue et qui est similaire au célèbre jeu de données MNIST. Pour la partie SSL, vous allez utiliser CIFAR10 dataset..

Pour tous les modèles, vous avez été fourni avec les "starters codes" ainsi que les scripts de normalisation et de chargement des données. Votre travail consistera à compléter certaines parties manquantes dans chacune des implémentations des modèles (détails dans les notebooks de code et dans chacune des parties des questions) afin de permettre un entraînement adéquat des modèles génératifs.

Instructions de codage: Vous devrez utiliser PyTorch pour répondre à toutes les questions. De plus, ce travail **exige l'exécution des modèles sur GPU** (sinon cela prendra un temps incroyablement long) ; si vous n'avez pas accès à vos propres ressources (par exemple, votre propre machine, un cluster), veuillez utiliser Google Colab. Pour la plupart des questions, à moins qu'on ne vous le demande expressément, ne codez la logique qu'en utilisant les opérations de base de `torch` au lieu d'utiliser les bibliothèques torch avancées (par exemple `torch.distributions`).

Pour tous les modèles fournis, nous vous encourageons à vous entraîner plus longtemps pour obtenir des résultats encore plus intéressants.

Important : Avant de soumettre le code à Gradescope, veuillez supprimer la ligne `!pip install ...` de tous les fichiers python solution (`.py`). Pour soumettre le code VAE, renommez votre fichier python en `vae_solution.py`, pour le modèle de diffusion / code DDPM, `ddpm_solution.py`, et pour la partie SSL, `q3_solution.py`.

Problem 1

La tâche consiste à mettre en œuvre un autoencodeur variationnel sur l'ensemble de données SVHN. Les VAE sont une classe de modèles génératifs à variables latentes qui travaillent sur l'optimisation

de l'ELBO, qui est défini comme suit:

$$ELBO(\theta, \phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(z|x_i)}[logp_\theta(x_i|z)] + \mathbb{KL}[q_\phi(z|x_i)||p(z)]$$

où nous disposons d'un ensemble de données $\{x_i\}_{i=1}^N$ et $p_\theta(x|z)$ est la vraisemblance conditionnelle, $p(z)$ est le prior et $q_\phi(z|x)$ est la distribution variationnelle approximative. L'optimisation se fait en maximisant l'ELBO ou en minimisant sa valeur négative. C'est-à-dire,

$$\theta^*, \phi^* = \text{argmin } \ ELBO(\theta, \phi)$$

Bien qu'il existe de nombreux choix de distributions, nous nous concentrerons dans ce travail sur le cas où

$$\begin{aligned} q_\phi(z|x) &= \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x)) \\ p_\theta(x|z) &= \mathcal{N}(\mu_\theta(z), I) \\ p(z) &= \mathcal{N}(0, I) \end{aligned}$$

où $\Sigma_\phi(x)$ est une **matrice de covariance diagonale** dont les éléments hors diagonale sont à 0.

Pour mettre en œuvre la VAE, vous devrez:

1. Implémenter la classe `DiagonalGaussianDistribution`, qui constituera notre structure de base pour paramétrer toutes les distributions gaussiennes avec des matrices de covariance diagonales, et qui sera utile lorsque nous mettrons en œuvre la VAE elle-même.

   - (1 pts) Implémenter la fonction `sample` qui échantillonne à partir de la distribution gaussienne donnée en utilisant **reparameterization trick**, de sorte que les gradients puissent être rétropropagés à travers elle. (*Réparamétrage : vous pouvez échantillonner à partir d'une distribution gaussienne avec une moyenne μ et une variance σ^2 en effectuant une correspondance **déterministe** d'un échantillon de $\mathcal{N}(0, 1)$*)
   - (2 pts) Implémenter la fonction `kl` qui calcule la divergence de Kulback Leibler entre la distribution gaussienne donnée et la distribution normale standard.
   - (2 pts) Mettre en oeuvre la fonction `nll` qui calcule la valeur négative de la log-vraisemblance de l'échantillon d'entrée en fonction des paramètres de la distribution gaussienne.
   - (1 pts) Implémenter la fonction `mode` qui renvoie le mode de cette distribution gaussienne.

2. Implémenter la classe `VAE`, qui est le modèle principal permettant d'encoder l'entrée dans l'espace latent, de la reconstruire, de générer des échantillons et de calculer les losses basées sur ELBO et les calculs de vraisemblance logarithmique basés sur l'échantillonnage d'importance.

   - (2 pts) Implémenter la fonction `encode` qui prend en entrée un échantillon de données et renvoie un objet de la classe `DiagonalGaussianDistribution` qui paramétrise le posterior approximatif avec la moyenne et le log de la variance qui sont obtenus par le biais du `encodeur` et `self.mean`, `self.logvar`. Réseaux de neurones.

- (2 pts) Implémenter la fonction `decode` qui prend en entrée un échantillon de l'espace latent et renvoie un objet de la classe `DiagonalGaussianDistribution` qui paramètre la distribution de vraisemblance conditionnelle avec la moyenne obtenue par le décodeur et la variance établie à identité.
- (2 pts) Implémenter la fonction `sample` qui prend une taille de lot en entrée et renvoie le mode de $p_\theta(x|z)$. Pour ce faire, il faut d'abord générer z à partir du priori, puis utiliser la fonction `decode` pour obtenir la distribution de vraisemblance conditionnelle et renvoyer son mode.
- (4 pts) Implémenter la fonction `log_likelihood` qui prend en entrée les données ainsi qu'un hyperparamètre K et calcule une approximation de la log-vraisemblance, qui est une métrique populaire utilisée dans de tels modèles génératifs. Pour calculer la log-vraisemblance approximative, nous devons calculer

$$\log p_\theta(x) \approx \log \frac{1}{K} \sum_{i=1}^{K} \underbrace{\frac{p(x_i, z_k)}{q(z_k|x_i)}}_{\Gamma}$$

  où $z_k \sim q(z|x_i)$. Pour ce faire, nous devrions en fait calculer $\log \Gamma$ puis utiliser la technique du log-sum-exp. Il est également recommandé d'utiliser la classe `DiagonalGaussianDistribution` (que vous avez codée ci-dessus) dans les solutions de cette partie.
- (2 pts) Enfin, implémenter la fonction `forward` qui prend en entrée un échantillon de données, l'encode dans une distribution variationnelle, en tire un échantillon reparamétrable, le décode et renvoie le mode de la distribution décodée. Elle doit également renvoyer la log-vraisemblance négative conditionnelle de l'échantillon de données sous $p_\theta(x|z)$ et la divergence KL avec la normale standard.
- (2 pts) Enfin, terminez le code fourni dans `interpolate` pour fournir une visualisation de la méthodologie. Il s'agit d'interpoler (ou de se déplacer linéairement) dans l'espace latent entre deux points et de voir comment de tels effets de leur espace latent conduisent à des transitions (éventuellement lisses ?) dans l'espace observé.

3. Pendant et après l'entraînement du modèle, nous vous demandons de fournir les résultats supplémentaires suivants de vos expériences. Veuillez fournir

   (a) (2 pts) Graphique montrant le temps de l'horloge de la procédure d'apprentissage.

   (b) (2 pts) Plusieurs échantillons générés au cours de l'entraînement après toutes les 5 epochs. (Si vous avez effectué l'entraînement pendant plus de périodes par défaut, il convient de préciser à partir de quelle période les modèles utilisés pour les échantillons ont été générés).

   (c) (4 pts) Quelques reconstructions à partir de l'ensemble de test chaque 5 epochs.

   (d) (4 pts) Échantillons du modèle VAE à la fin de l'entraînement. À quoi ressemblent les échantillons ? Voyez-vous des motifs de chiffres ; les échantillons sont-ils flous ?

   (e) (6 pts) Montrez un profil de variables latentes pour chaque classe de l'ensemble de données en faisant la moyenne des représentations latentes de quelques échantillons de l'ensemble d'apprentissage correspondant à chaque classe. Créez des diagrammes à barres pour

chaque classe afin de visualiser la distribution des variables latentes. Recherchez des modèles dans les représentations et identifiez les pics dans certaines variables latentes.

(f) (2 pts) Images des résultats de l'interpolation à partir du code. L'interpolation entre deux points est-elle régulière ? Les images changent-elles de façon régulière ?

Problem 2

Il s'agit ici d'implémenter le modèle probabiliste de débruitage par diffusion (DDPM) sur le jeu de données SVHN. Les modèles de diffusion sont une classe émergente de modèles génératifs qui s'appuient sur un processus de diffusion vers l'avant connu, qui détruit progressivement la structure des données jusqu'à ce qu'il converge vers un bruit non structuré, par exemple $\mathcal{N}(0, I)$ et un processus vers l'arrière paramétré (par un réseau de neurones !) qui supprime itérativement le bruit jusqu'à ce que vous ayez obtenu un échantillon de la distribution des données.

En particulier, on construit le processus de diffusion vers l'avant comme suit

$$q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

avec les β_t définissant les schémas de bruit, typiquement 0,0001 à $t = 1$ et 0,02 à $t = T$ avec un schéma linéaire entre les deux. On peut voir ce processus comme l'ajout itératif de bruit à l'échantillon et la destruction de sa structure.

Le processus à rebours paramètre ensuite une distribution

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t), \tilde{\beta}_t I)$$

où $\tilde{\beta}_t$ est la variance de la distribution $q(x_{t-1}|x_t, x_0)$, et en apprenant le paramètre θ, on espère **dénoiser** légèrement de x_t à x_{t-1}. Avec un peu d'algèbre et en effectuant quelques calculs, cela conduit à paramétrer un modèle de bruit au lieu de la moyenne, c'est-à-dire $\epsilon_\theta(x_t, t)$, ce qui conduit de manière équivalente à la distribution

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right), \tilde{\beta}_t I)$$

L'objectif de l'apprentissage se résume donc à la prédiction du bruit,

$$\mathbb{E}_{t\sim\mathcal{U}(1,T),x_0,\epsilon_t}\left[||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t, t)||^2\right]$$

et l'échantillonnage est itératif, un échantillon de données étant obtenu en échantillonnant $x_T \sim \mathcal{N}(0, I)$ puis en échantillonnant progressivement $x_t \sim p_\theta(x_{t-1}|x_t)$. Pour plus de détails, veuillez vous référer à la théorie fournie dans le notebook colab, ainsi qu'à l'article original de DDPM et à un article de blog populaire, tous deux référencés dans le notebook.

1. Implémenter le modèle DDPM ainsi que son entraînement en suivant les étapes suivantes :

- (4 pts) Calculer à l'avance les coefficients / variables utiles suivants : β_t, α_t, $\frac{1}{\sqrt{\alpha_t}}$, $\bar{\alpha}_t$, $\sqrt{\bar{\alpha}_t}$, $\sqrt{1-\bar{\alpha}_t}$, $\bar{\alpha}_{pt}$ (qui est $\bar{\alpha}_t$ décalé vers la droite et complété par 1), et $\tilde{\beta}_t$. Les détails sont fournis dans le Notebook.
- (4 pts) Compléter la fonction `q_sample` qui implémente l'échantillonnage à partir de la distribution $q(x_t|x_0)$ en utilisant l'astuce de reparamétrage.
- (4 pts) Compléter la fonction `p_sample` qui implémente l'échantillonnage à partir de la distribution $p(x_{t-1}|x_t)$ en utilisant l'astuce de reparamétrage.
- (2 pts) Compléter la fonction `p_sample_loop` qui utilise la fonction `p_sample` et débruite itérativement un bruit complètement aléatoire de $t = T$ à $t = 1$.
- (4 pts) Implémenter la fonction `p_losses` qui génère un bruit aléatoire, utilise ce bruit aléatoire pour obtenir un échantillon bruyant, obtient sa prédiction de bruit, et renvoie ensuite la **perte lissée** L_1 entre le bruit prédit et le bruit réel.
- (2 pts) Enfin, implémentez l'échantillonnage aléatoire des pas de temps dans la fonction `t_sample`.

2. Pendant et après l'entraînement du modèle, nous vous demandons de fournir les résultats supplémentaires suivants et les conclusions issues de vos expériences. Veuillez fournir

   (a) (2 pts) Graphique montrant le temps de l'horloge de la procédure d'apprentissage.

   (b) (2 pts) Plusieurs échantillons générés au cours de l'entraînement après toutes les 5 époques. (Si vous avez effectué l'entraînement pendant plus de périodes par défaut, il convient de préciser à partir de quelle période les modèles utilisés pour les échantillons ont été générés).

   (c) (6 pts) Quelques reconstructions à partir de l'ensemble de test. Dans cette partie, ajoutez du bruit aux échantillons au pas de temps $t = t'$ et essayez de reconstruire l'échantillon en utilisant le processus de diffusion inverse (idée similaire à l'échantillonnage, mais pas initié à partir d'un bruit aléatoire). Essayer différents niveaux de bruit pour déterminer le pas de temps spécifique $t = t^*$ auquel les échantillons originaux ne peuvent plus être restaurés, ce qui entraîne la génération d'échantillons différents. (Il existe certaines applications de ce processus, telles que l'article sur purifying adversarial examples.)

   (d) (4 pts) Quelques échantillons générés à partir du modèle de diffusion à la fin de l'apprentissage. À quoi ressemblent ces échantillons ? Voyez-vous des motifs de chiffres ; les échantillons sont-ils flous ? Comparez-les aux échantillons VAE.

3. (Optional) Bien que cette section ne soit pas obligatoire et ne fasse pas l'objet d'une évaluation, nous vous recommandons vivement de la parcourir. Notre objectif est d'explorer plus en profondeur les modèles de diffusion.

   (a) (0 pts) Entraîner deux modèles de diffusion supplémentaires avec T=500 et T=1500 et étudier les effets de l'augmentation du nombre de pas de temps sur le temps d'entraînement/d'échanti et la qualité de l'échantillonnage. Que pensez-vous de ce compromis ?

   (b) (0 pts) Certaines études ont utilisé la représentation intermédiaire de U-Net pour des tâches autres que son objectif initial. Vous pouvez vous référer à ce document qui utilise

de telles représentations pour la segmentation sémantique. Veuillez préciser votre point de vue sur l'applicabilité de concepts similaires à d'autres tâches.

Problem 3

Les méthodes d'apprentissage auto-supervisé apprennent une représentation des données en résolvant des tâches préalables afin d'alléger le processus d'étiquetage supervisé coûteux. Les méthodes d'apprentissage contrastif, une sous-catégorie de l'apprentissage auto-supervisé (SSL), apprennent une représentation en minimisant la distance entre la représentation de deux visions différentes du même échantillon tout en maximisant la distance entre la représentation de la vision de deux échantillons différents. Cependant, les méthodes SSL non contrastives ont montré des résultats comparables sans utiliser un grand nombre d'échantillons négatifs. Dans cette question, vous chercherez à savoir pourquoi ces réseaux ne se réduisent pas à une solution triviale en mettant en œuvre l'algorithme Simsiam et en expérimentant les facteurs clés de sa performance.

Simsiam fonctionne étonnamment bien sans plusieurs stratégies de prévention de dégradation. Ce modèle maximise directement la similarité entre les deux visions d'une image, sans utiliser de paires négatives ni d'encodeur de momentum. Ici, vous commencerez par implémenter une fonction permettant d'estimer la similarité en cosinus négatif. Ensuite, vous implémenterez les fonctions de perte et d'avance Simsiam. Enfin, vous réaliserez des expériences pour analyser les performances du modèle dans différentes conditions.

Dataset et dataloader pour Simsiam [1] Dans cette question, vous allez effectuer une tâche de classification d'objets pour l'ensemble de données **CIFAR10**. [2]. Cet ensemble de données se compose d'images $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}$ de 10 classes. Nous fournissons des échantillonneurs pour générer les différentes distributions dont vous aurez besoin pour cette question. Dans le même répertoire, nous fournissons également l'architecture d'un réseau de neurones f : $\mathcal{X} \to \mathcal{Z}$ and h : $\mathcal{Z} \to \mathcal{P}$ s.t. $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}, \mathcal{Z} \subset \mathbb{R}^d, \mathcal{P} \subset \mathbb{R}^d$ (d est la feature dimension, la valeur par défaut de Simsiam est 2048).

Hyperparamètres & Pointeurs d'entraînement Nous fournissons le code pour le réseau SSL ainsi que les hyperparamètres que vous devriez utiliser. Nous vous demandons de coder la procédure d'entraînement du Simsiam ainsi que l'exploration qualitative que vous inclurez dans votre rapport.

[1] Un réseau de neurones Simsiam est une classe d'architectures de réseaux neuronaux qui contiennent au moins deux sous-réseaux identiques. (D'après here)

[2] Les données CIFAR10 peuvent être téléchargées à https://www.cs.toronto.edu/ kriz/cifar.html. Veuillez noter que Pytorch CIFAR10 Dataset peut télécharger le jeu de données, vous n'avez donc pas besoin de le télécharger séparément.

1. (2 pts) Implémenter les fonctions forward du Simsiam dans 'tt q3$_s olution.py'. Cette fonction reçoit deux vu
et x_2 à partir d'un échantillon d'entrée x et calcule les sorties du réseau, qui sont les suivantes :

$$z_1 \triangleq f(x_1), \quad p_1 \triangleq h(f(x_1)) \tag{1}$$

$$z_2 \triangleq f(x_2), \quad p_2 \triangleq h(f(x_2)) \tag{2}$$

Notez que vous devez également effectuer Le grading stopping dans cette étape.

2. (2 pts) Implémenter la fonction '`cosine similarity`' dans '`q3_solution.py`' pour calculer la similarité entre deux entrées. La similarité cosinus entre deux variables A et B peut être définie comme suit

$$S_c(A, B) = \frac{A}{\|A\|_2} \cdot \frac{B}{\|B\|_2} \tag{3}$$

3. (2 pts) Implémenter les fonctions de perte Simsiam dans '`q3_solution.py`' pour calculer la fonction objective du Simsiam, en utilisant la similarité cosinus ci-dessus. La fonction objective de Simsiam est définie comme suit :

$$L = 0.5 * \mathcal{D}(p_1, stop - grad(z_2)) + 0.5 * \mathcal{D}(p_2, stop - grad(z_1)) \tag{4}$$

Où $\mathcal{D}$ est la similarité en cosinus négatif.

4. (8 pts) Entraînez le modèle pour 100 époques avec et sans arrêt du gradient. Tracer la perte d'entraînement et la précision Knn en fonction des époques d'entraînement.

5. (9 pts) Étudiez l'effet du réseau prédicteur (MLP) en expérimentant le(s) paramètre(s) ci-dessous. Tracez la perte d'entraînement et la précision du KNN en fonction des époques d'entraînement.

   (a) Supprimer le prédicteur en le remplaçant par une identité.

   (b) Que se passe-t-il lorsque la dimensionnalité de la dernière couche du réseau de projecteurs est augmentée ? De 2048 à 4096 ? Évaluez SimSiam avec une taille de lot égale à 4096 et indiquez les performances pour les deux valeurs de taille de batch.

   (c) D'après l'observation précédente, pouvez-vous vous attendre à ce que le modèle de Barlow Twins se comporte de la même manière ? Expliquez votre raisonnement.

6. (4 pts) L'incorporation de lourdes augmentations dans SimSiam entraîne des performances instables, voire un risque d'effondrement. Comment pourriez-vous atténuer ce problème ? Utilisez-vous l'augmentation des données dans le modèle SimSiam actuel ? Si oui, où ? Conseil : vérifiez DSSL [3].

7. (4 pts) Le principal défi des architectures d'embedding conjointe est d'éviter un effondrement dans lequel les deux branches ignorent les entrées et produisent des vecteurs de sortie identiques et constants. Pourriez-vous expliquer comment éviter les problèmes d'effondrement ? Indice : vérifiez VicReg [4].

[3] Directional Self-supervised Learning for Heavy Image Augmentations: https://arxiv.org/abs/2110.13555

[4] VICReg: Variance-Invariance-Covariance Regularization For Self-Supervised Learning: https://arxiv.org/pdf/2105.04906.pdf

8. (5 pts) Quelles sont les principales différences entre les méthodes BYOL et SimSiam ? Quel est leur rôle dans chaque modèle ?

9. (4 pts) Pourquoi l'exploitation de l'asymétrie pour l'apprentissage de la représentation de SimSiam est pertinente ? Comment les algorithmes peuvent-ils en bénéficier ?