Date remise : le 5 décembre, 2022, 23h

Instructions

- *Ce devoir est difficile – commencez le bien en avance.*
- *Montrez votre démarche pour toutes les questions !*
- *votre rapport (PDF) et votre code électroniquement via la page Gradescope du cours. Votre rapport doit contenir des réponses au problème 1.3, problème 2.2 - 2.4, problème 4(toutes les questions).*
- *Pour les expériences ouvertes, vous n'avez pas besoin de soumettre de code - un rapport suffira.*
- *Les TAs pour ce devoir sont* ***Arian Khorasani et Sarthak Mittal***

Dans ce devoir, vous devrez implémenter trois modèles génératifs différents qui sont largement populaires dans Apprentissage automatique, à savoir (a) Auto-encodeur variationnel (VAE), (b) Réseaux antagonistes génératifs (GAN), et (c) modèles de diffusion (Diffusion Model). Toutes les stratégies seront testées sur SVHN dataset qui se compose d'images de rues vues des numéros de maison, et est similaire au populaire ensemble de données MNIST.

Pour tous les modèles, vous avez reçu les codes de démarrage ainsi que les données de normalisation et chargement de scripts. Votre travail serait de remplir certaines parties manquantes dans chacune des implémentations de modèle (détails dans les carnets de codes et chacune des parties de la question) afin de permettre une formation adéquate des modèles génératifs.

Instructions de codage : Vous devrez utiliser PyTorch pour répondre à toutes les questions. De plus, ce devoir **nécessite d'exécuter les modèles sur GPU** (autrement cela prendra un temps incroyablement long) ; si vous n'avez pas accès à vos propres ressources (p. ex., votre propre machine, un cluster), veuillez utiliser Google Colab ou les crédits GCP fournis. Pour la plupart des questions, à moins qu'elles ne soient expressément posées, Veuillez coder la logique à l'aide des opérations de `torche` de base au lieu d'utiliser des bibliothèques de torches avancées (p. ex. `torch.distributions`).

Pour tous les modèles fournis, vous êtes encouragés à vous entraîner plus longtemps pour voir des résultats encore meilleurs.

Important : Avant de soumettre le code à Gradescope, veuillez supprimer le `!pip install ...` ligne de toute la solution python (`.py`) fichers. En outre, pour soumettre le code VAE, renommez votre fichier python comme `vae.py`, pour le code du GAN `gan.py` et pour le code de Diffusion / DDPM, `ddpm.py`.

Question 1. (**VAE**) [VAE Notebook]

la tâche consiste à Implémenter un Auto-encodeur variationnel sur le donnés SVHN. Les VAE sont une classe de modèles génératifs à variable latente qui travaillent à optimiser l'ELBO, qui est défini comme

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

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

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

Bien qu'il puisse y avoir de nombreux choix de telles distributions, dans cette tâche, nous limiterons notre attention au 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 avec les éléments hors diagonale comme 0.

Pour Implémenter la VAE, vous devrez

1. Implémenter la classe `DiagonalGaussianDistribution`, qui formera notre colonne vertébrale pour paramétrer toutes les distributions gaussiennes avec matrices de covariance diagonales, et sera utile lorsque nous mettrons en œuvre la VAE elle-même.
   - (2 pts) Implémenter la fonction `sample` que les échantillons de la distribution gaussienne donnée en utilisant le **reparameterization** Pour que les gradients puissent être inversés. (*Reparameterization Trick : Vous pouvez échantillonner à partir d'une distribution gaussienne avec moyenne μ et variance σ^2 en faisant une **deterministic** cartographie d'un échantillon de $\mathcal{N}(0, 1)$*)
   - (3 pts) Implémenter la fonction `kl` qui calcule le Divergence de Kullback-Leibler entre la distribution guassienne donnée et la distribution normale standard.
   - (3 pts) Implémenter la fonction `nll` qui calcule le négatif de la probabilité logarithmique 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 qui prend soin d'encoder l'entrée dans l'espace latent, sa reconstruction, générer des échantillons, et le calcul des pertes ELBO requises et les calculs de vraisemblance logarithmique stylisés par échantillonnage de l'importance.
   - (3 pts) Implémenter la fonction `encode` qui prend comme entrée un échantillon de données et retourne un objet de la classe `DiagonalGaussianDistribution` qui paramétre le postérieur approximatif avec la moyenne et le log de la variance qui sont obtenus par le `encoder` et `self.mean`, `self.logvar` réseaux neuronaux.
   - (3 pts) Implémenter la fonction `decode` qui prend comme entrée un échantillon de l'espace latent et retourne un objet de la classe `DiagonalGaussianDistribution` qui paramétre la distribution de vraisemblance conditionnelle avec la moyenne obtenu à partir du décodeur et de la variance fixée comme identité.
   - (2 pts) Implémenter la fonction `sample` qui prend une taille de lot comme entrée et sorties le mode de $p_\theta(x|z)$. Pour ce faire, commencez par générer z de la précédente, puis utiliser la fonction `decode` pour obtenir la distribution de vraisemblance conditionnelle, et retourne son mode.

- (5 pts) Implémenter la fonction `log_likelihood` qui prend en entrée les données ainsi qu'un hyperparamètre K et calcule une approximation de la log-likelihood, qui est une métrique populaire utilisée dans de tels modèles génératifs. Pour calculer la log-likelihood 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 calculer cela, on calculerait en fait $\log \Gamma$ et ensuite utiliser le log-sum-exp que nous avons utilisé dans Softmax dans devoir 1 (partie théorique) pour le rendre plus stable. Vous pouvez utiliser la fonctionnalité log-sum-exp de PyTorch pour ce faire. Il est également recommandé d'utiliser la classe `DiagonalGaussianDistribution` (que tu as si minutieusement codé au-dessus) dans les solutions à cette partie.
- (3 pts) Finalement, Implémenter la fonction `forward` qui prend comme entrée un échantillon de données, l'encode dans une variational distribution, en tire un échantillon réparable, le décode et renvoie le mode de la distribution décodée. Il devrait également renvoyer la log-likelihood négative conditionnelle de l'échantillon de données sous $p_\theta(x|z)$ et le KL-Divergence avec norme normale.
- (2 pts) Finalement, terminer le code fourni dans `interpolate` pour fournir une certaine visualisation de la méthodologie. Ceci est censé interpoler (ou déplacer linéairement) dans l'espace latent entre deux points et voir comment ces effets de leur espace latent conduisent à (éventuellement lisse ?) transitions dans l'espace observé.

3. Après avoir formé le modèle,, nous vous demandons de fournir les résultats supplémentaires suivants de vos expériences. Veuillez fournir
   (a) (3 pts) Quelques reconstructions de l'ensemble d'essai et des échantillons de modèle VAE en fin de formation.
   (b) (2 pts) Comment sont les échantillons ? Voyez-vous des modèles de chiffres ; les échantillons sont-ils flous ?
   (c) (2 pts) Les images de l'interpolation proviennent du code. L'interpolation entre deux points est-elle fluide ? Voyez-vous les images changer en douceur ?

Question 2. (**GAN**) [GAN Notebook]

La tâche ici est implémenter le Réseaux antagonistes génératifs (GAN) sur l'ensemble de données SVHN. Les GAN sont une catégorie de modèles génératifs implicites qui ne fonctionnent pas sur la probabilité maximale likelihood mais s'appuient plutôt sur une formation contradictoire, où un discriminateur est formé pour faire la distinction entre les données réelles et les données générées à partir d'un générateur, qui à son tour tente de tromper le discriminant. Formellement, cela conduit à un jeu à 2 joueurs et le problème d'optimisation min-max suivant

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]$$

En pratique, le discriminateur (D) n'est pas entraîné à l'optimialité avant de faire un pas sur le générateur. Au lieu de cela, nous alternons entre un nombre défini d'étapes pour le discriminateur

suivi d'une étape d'entraînement du générateur. Dans ce devoir, nous allons juste alterner entre un pas de discriminateur et un pas de discriminateur mais en pratique, effectuer plus d'étapes sur le discriminateur pour chaque étape du générateur a montré quelques améliorations.

De plus, pour permettre un meilleur entraînement et éviter les problèmes de gradient (problème des gradients saturants), le générateur est plutôt formé dans son étape pour maximiser $\log D(G(z))$ par opposition à minimiser $\log(1 - D(G(z)))$.

1. Implémenter le modèle GAN et la formation en suivant les étapes suivantes
   - (3 pts) Définir le `optimizers` en utilisant des fonctions intégrées de PyTorch pour les réseaux générateur et discriminateur. Chacun des optimiseurs doit Être `Adam` avec les taux d'apprentissage fournis dans le Colab notebook et l'hyperparamètre défini sur $(0.5, 0.999)$.
   - (2 pts) Définir le `criterion` en utilisant les criterions de PyTorch afin qu'il soit adapté à la formation de l'objectif GAN.
   - (5 pts) Complétez la méthode `discriminator_train` qui prend en entrée des échantillons réels et faux et renvoie la perte pour le discriminateur.
   - (5 pts) Complétez la méthode `generator_train` qui prend en entrée de faux échantillons et renvoie la perte pour le générateur.
   - (2 pts) Complétez la méthode `sample`, qui échantillonne le nombre `num_samples` d'échantillons à l'aide du générateur.
   - (2 pts) Implémentez la fonction `interpolate` qui interpole entre les deux points en utilisant le nombre `n_samples` de points et renvoie l'image générée correspondant à chacun.
2. Après avoir entraîné le modèle, nous vous demandons de fournir les résultats supplémentaires suivants et les informations issues de vos expériences. Veuillez fournir
   (a) (3 pts) Certains ont généré des échantillons du modèle GAN à la fin de la formation.
   (b) (2 pts) De quoi ont l'air les échantillons ? Sont-ils flous ? Comparez-les avec les échantillons VAE.
   (c) (2 pts) Fournir des images pour le résultat de l'interpolation entre deux points de la distribution du bruit. L'interpolation est-elle fluide ?
3. (3 pts) Dans la boucle d'entraînement (avant-dernière cellule), nous devions utiliser `.detach()` dans l'entraînement du discriminateur pour les fausses images générées. Est-ce indispensable, et si oui, pourquoi ?
4. (5 pts) Le modèle GAN, tel qu'il est actuellement, peut-il être utilisé pour (a) reconstruire des images d'entrée, (b) calculer (exactement ou approximativement) la log-likelihood, ou (c) l'apprentissage de la représentation ?

Question 3. (**Modèles de diffusion**) [DDPM Notebook]

La tâche ici est de mettre en œuvre le Denoising Diffusion Probabilistic Model (DDPM) sur le SVHN dataset. Les modèles de diffusion sont une classe montante de modèles génératifs qui reposent sur un processus de diffusion directe connu, qui détruit progressivement la structure des données jusqu'à ce qu'elle converge vers un bruit non structuré, par exemple. $\mathcal{N}(0, I)$ et un savant paramétré (par

un réseau de neurones !) processus rétroactif qui élimine itérativement le bruit jusqu'à ce que vous ayez obtenu un échantillon de la distribution des données.

En particulier, un construit le processus de diffusion en avant comme

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

avec le definition de β_t les horaires de bruit, généralement conservé comme 0.0001 à $t = 1$ et 0.02 at $t = T$ avec un calendrier linéaire entre les deux. On peut voir ce processus comme ajoutant itérativement plus de bruit à l'échantillon et détruire la structure.

Le processus en arrière paramétre 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 **dénoter** légèrement de x_t à x_{t-1}. Avec un peu d'algèbre et quelques calculs, cela conduit à paramétrer un modèle de bruit au lieu de la moyenne, C'est $\epsilon_\theta(x_t, t)$, ce qui conduit à la distribution

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

Cela conduit à l'objectif de formation étant juste 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 étant itératif, lorsqu'un échantillon de données est obtenu par échantillonnage $x_T \sim \mathcal{N}(0, I)$ puis échantillonnage progressif $x_t \sim p_\theta(x_{t-1}|x_t)$. Pour plus de détails, veuillez vous référer à la théorie fournie dans le Colab notebook, ainsi que le papier DDPM original et un blog populaire, les deux liés à dans le Colab notebook.

1. Implémenter le modèle DDPM et la formation en suivant les étapes données :
   - (5 pts) Calculer à l'avance (Pre-Compute) les variables et les coefficients utiles suivants : β_t, α_t, $\frac{1}{\sqrt{\alpha_t}}$, $\bar{\alpha}_t$, $\sqrt{\bar{\alpha}_t}$, $\sqrt{1-\bar{\alpha}_t}$, $\bar{\alpha}_{pt}$ (lequel est $\bar{\alpha}_t$ right-shifted and padded avec 1), et $\tilde{\beta}_t$. Les détails sont fournis dans le Colab notebook.
   - (5 pts) Complètez la fonction `q_sample` qui met en œuvre l'échantillonnage de la distribution $q(x_t|x_0)$ en utilisant le reparameterization trick.
   - (5 pts) Complètez la fonction `p_sample` qui met en œuvre l'échantillonnage de la distribution $p(x_{t-1}|x_t)$ en utilisant le reparameterization trick.
   - (2 pts) Complètez la fonction `p_sample_loop` qui utilise la fonction `p_sample` et désigne itérativement le bruit complètement aléatoire de $t = T$ à $t = 1$.
   - (5 pts) Implémenter la fonction `p_losses` qui génère des bruits aléatoires, utilise ce bruit aléatoire pour obtenir un échantillon bruyant, obtient sa prédiction de bruit, puis retourne le **smoothed L_1 loss** entre le bruit prédit et le bruit réel.
   - (1 pts) Enfin, Implémenter l'échantillonnage aléatoire des pas-temps dans la fonction `t_sample`.
2. après avoir formé le modèle, nous vous demandons de fournir les résultats et les observations supplémentaires suivants de vos expériences. veuillez fournir

(a) (3 pts) Certains ont généré des échantillons du modèle Diffusion à la fin de la formation.

(b) (2 pts) Comment sont les échantillons ? Comparez-les avec les échantillons VAE et GAN.

Question 4. (**Modèle génératif ; avantages et inconvénients**)

Maintenant que vous avez mis en œuvre les trois modèles génératifs populaires : VAE, GAN and DDPM, pouvez-vous commenter ce qui suit

1. (6 pts) Quels sont les principaux points forts et points faibles (en termes de qualité des échantillons, de charge de calcul à la formation et d'inférence, etc.) de
   - modèle VAE ?
   - modèle GAN ?
   - modèle DDPM ?