Date limite: 23:00, 26 Avril 2024

Instructions

- *Ce devoir est difficile – veuillez le commencer en avance.*
- *Pour toutes les questions, montrez votre travail!*
- *Soumettez votre rapport (PDF) et votre code par voie électronique via la page Gradescope du cours..*
- *Vous ne pouvez pas utiliser ChatGPT pour ce devoir. Vous êtes encouragé à poser des questions sur la Piazza ou par courriel/Slack au TA pour ce devoir (**Tianyu Zhang**, Email:* `tianyu DOT zhang AT mila DOT quebec`).

Résumé : Dans ce devoir, vous allez compléter l'implémentation de SimCLR et de l'algorithme DDPM qui a été abordé en cours. Veuillez consulter l'instruction détaillée ci-dessous. Vous devez soumettre à la fois le code et vos rapports pour les deux problèmes.

Instructions de codage : Vous devrez utiliser PyTorch pour répondre à toutes les questions. De plus, ce devoir **nécessite l'exécution des modèles sur GPU** (sinon cela prendrait 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 ou Kaggle (vous devez vous inscrire et vérifier votre identité). Pour certaines questions, il vous sera demandé de ne pas utiliser certaines fonctions de PyTorch et de les implémenter vous-même en utilisant des fonctions primitives de `torch`.

Problem 1

Implémentation d'une méthode d'apprentissage contrastive : SimCLR (60 pts) L'apprentissage auto-supervisé est apparu comme une technique puissante pour apprendre des représentations sans nécessiter de données étiquetées. SimCLR (Cadre Simple pour l'Apprentissage Contrastif des Représentations Visuelles, https://arxiv.org/abs/2002.05709) est une telle approche qui utilise l'apprentissage contrastif pour apprendre des représentations visuelles. Ce rapport décrit l'implémentation de SimCLR en utilisant le dataset STL-10 dans PyTorch.

Vous devez compléter la définition de la classe Net, la définition de la classe de dataset SimCLR-Dataset et la perte SimCLR dans la classe Trainer. Vous devez exécuter la boucle d'entraînement, sauvegarder le meilleur modèle d'entraînement et évaluer en utilisant la tâche de classification de sonde linéaire. Comme nous n'avons pas assez de ressources GPU et que l'algorithme d'apprentissage contrastif comme SimCLR a généralement besoin d'environ '1000' époques pour s'entraîner (nous n'avons que 70 époques), vous n'obtiendrez peut-être pas les meilleures performances. Ainsi, du côté des performances, tant que vous voyez que la perte diminue (jusqu'à environ 7.4 à 70 époques) et que la précision augmente, c'est bon.

Note :

- Remplir la définition de la classe Net (5 points).
- Remplir la définition de la classe de dataset SimCLRDataset (10 points).
- Remplir la perte SimCLR dans la classe Trainer (20 points).
- Enregistrer la perte d'entraînement dans les 70 époques, plus elle est basse, mieux c'est (5 points).
- Enregistrer la précision de la sonde linéaire, plus elle est haute, mieux c'est (5 points).
- Rédiger un rapport incluant comment vous sélectionnez l'augmentation des données (transformation) dans le pool de transformations, comment vous implémentez la perte SimCLR et expliquer pourquoi votre perte SimCLR est efficace en termes de calcul et équivalente à la fonction de perte dans l'article. Veuillez également inclure la courbe de perte d'entraînement et les précisions top1 et top5 en aval (15

  points). Veuillez noter que la logique de journalisation n'est pas fournie, veuillez l'implémenter avant de commencer l'entraînement.

Pour plus de détails, veuillez voir les instructions fournies dans le notebook Colab.

Veuillez NE PAS changer la configuration fournie. Ne changez le code donné que si vous êtes sûr que le changement est nécessaire. Il est recommandé d'utiliser une session CPU pour déboguer lorsque le GPU n'est pas nécessaire puisque Colab ne donne que 12 heures d'accès GPU gratuit à la fois. Si vous utilisez toutes les ressources GPU, vous pouvez envisager d'utiliser les ressources GPU de Kaggle. Merci et bonne chance!

Problem 2

Implémentation d'une classe DDPM pour générer des images de style MNIST. (40 pts) Dans ce problème, vous implémenterez une classe DDPM (https://arxiv.org/abs/2006.11239) sur le dataset MNIST en utilisant PyTorch selon les directives. L'objectif est de minimiser la fonction de perte et d'entraîner le modèle pour générer des images MNIST.

Les classes Train et UNet sont déjà implémentées pour vous (elles se trouvent dans les blocs cachés, n'oubliez pas de les lancer avant de commencer à travailler). Vous devez implémenter la classe DDPM (voir les détails ci-dessous). Les images générées par le modèle seront automatiquement affichées conformément à l'implémentation de la classe Trainer. Assurez-vous que les images générées sont affichées dans la sortie, cela sera évalué.

Note :

- Implémentation de la classe DDPM (20 points).
- Entraîner le modèle pour générer des images MNIST raisonnables dans les 20 époques (10 points).
- Rédiger un rapport pour décrire inclure les images échantillons générées par chaque époque (10 points). Veuillez noter que la fonction pour générer l'image est déjà fournie.

Pour plus de détails, veuillez voir les instructions fournies dans le notebook Colab.

Veuillez NE PAS changer le code fourni, ajoutez uniquement votre propre code où indiqué. Il est recommandé d'utiliser une session CPU pour déboguer lorsque le GPU n'est pas nécessaire puisque Colab ne donne que 12 heures d'accès GPU gratuit à la fois. Si vous utilisez toutes les ressources GPU, vous pouvez envisager d'utiliser les ressources GPU de Kaggle. Merci et bonne chance!