Documentation rapide 'score crawl'

clement.brochet

8 Novembre 2022

Comment calculer les métriques "à la chaîne" ...

1 Créer les bons répertoires

1.1 Répertoires

Il faut se créer un répertoire 'Set d'expérience', organisé comme ça :
Set_nn : (nn = le numéro d'expé)

- monNomDeDossier_bs_1_lrD_lrG/ :(bs : taille de batch utilisée, lrD, lrG = learning rates de D/G 1 = nombre d'iter de D pour 1 iter de G)
  - Instance_xx/ (xx : le numéro de "l'instance", ie la même expérience relancée xx fois)
    - log/ ← ce dossier est important, il contient les sorties des calculs
    - models/
    - samples/ ← ce dossier est important, il contient les échantillons à tester
    - ReadMe_xx.txt ← ce fichier est important, il contient des paramètres de l'expérience
  - Instance_yy/
- resnet_128_wgan-hinge_64_bs2_1_lrD2_lrG2/ etc... etc... selon les hyper-paramètres

Le dossier *monNomDeDossier* peut avoir un nom compliqué, le défaut est */resnet_128_wgan-hinge_64*, mais on peut tout aussi bien metre 'progan'. Les données réelles sont sur le dossier '/scratch/mrmn/brochetc/GAN_2D/Sud_Est_Baselines_IS_1_1.0_0_0_0_0_0_0_256_done/' qui contient les fichiers d'entraînement. On peut le laisser là.

1.2 Contenu

Le fichier ReadMe_xx.txt (!!! attention à l'orthographe et respecter la casse !!!) doit contenir :

———————-
crop_indexes!tab!:!tab![78,206,55,183]
var_names!tab!:!tab![’u’,’v’,’t2m’]

Où !tab! signifie Tabulation (et non espace). La première ligne correspond aux coordonnées extraites et la deuxième aux variables étudiées?

Le dossier samples/ doit contenir les ’samples’ qui sont des fichiers .npy répondant à la dénomination :

’_Fsample_aa_b.npy’ avec aa = l’itération à laquelle les data ont été générées et b = le numéro du fichier.

Ces fichiers doivent contenir les échantillons, sous la forme $N_{ech} \times C \times H \times W$. Pour chaque itération aa, il y a donc $b \times N_{ech}$ échantillons au max. Ils seront ’chargés’ au moment de calculer les métriques (on peut en prendre moins que le max).

Si on n’est pas intéressé par l’itération, on peut choisir aa=0.

Le dossier log/ peut être vide a priori, mais il sera rempli à la fin. Les sorties se présentent sous la forme de dictionnaires, avec l’extension .p

Pour un dictionnaire donné, on a la forme :
{ ’nomDeLaMétrique’ : arrayNumpy, ’nomDeLaMétriqueNuméro2’ : arrayNumpy, etc... }

On les ouvre avec le module *pickle* de python :

res = pickle.load(open(’monDico.p’, ’rb’))

Et res[’nomDeLaMétrique’] contient l’array Numpy.

2 Syntaxe

2.1 Configuration

S’assurer qu’on est dans le bon environnement. Les métriques utilisent abondamment torch, numpy et parfois pandas. Il faut être sur l’environnement *preproc* a minima (si on utilise des ondelettes il faut un environnement custom → autre histoire).

Il y a deux types de métriques :

- Les ’stand-alone’ : on a pas besoin de données ’réelles’ de référence pour les calculer
- Les ’distance’ : on a besoin de données réelles pour les calculer.

Le fichier metrics4arome/__init__.py recense toutes ces métriques et leur type. Il est utilisé pour vérifier que le calcul d’une métrique est bien légal.

Pour préparer le lancement, il faut donc spécifier, pour chaque type, la liste de métriques qu'on veut calculer.

Ensuite, il y a l'initialisation d'un 'MetricsCalculator', une sorte de gros machin qui va exécuter les calculs et stocker les résultats. Un 'MetricsCalculator' est défini par liste de métriques. Le deuxième paramètre de cet objet est une chaîne de caractère qui donnera un préfixe explicite au fichier de sortie (pour 's'y retrouver').

Les autres paramètres sont les directories d'appel (qui chapeautent tous les directories utilisés dans le calcul) et le nombre d'échantillons. Il ya aussi la variable 'programme' : on peut la modifier pour faire du 'bootstrap', mais ce n'est pas recommandé pour un usage basique.

2.2 Exécution

Le fichier *metric_tests_exec.py* peut-être appelé avec des mots-clés optionnels (voir la source score_crawl/configurate.py) :

1. *–glob_name=...* contient le nom de dossier qui référence l'expérience (eg 'progan').
2. *–batch_sizes=[...]* contient les tailles de batch (liste !)
3. *–expe_set=nn* contient le numéro de l'expé que l'on traite (correspond au Set_nn du début)
4. *–lr0=[...]* correspond au learning rate du discriminitateur/générateur (liste !)
5. *–variables=['u','v','t2m']* correspond aux variables sur lesquelles on veut calculer les métriques (un sous-ensemble des variables de l'expérience, on peut en calculer moins que ce qu'il y a dans les échantillons !)
6. *–instance_num=[...]* (liste !) correspond aux instances sur lesquelles lancer l'expérience

Par exemple, l'appel :

```
python3 metric_tests_exec.py --glob_name='progan' --expe_set=38 --batch_sizes=[8,16,32]
 --lr0=[0.001] --instance_num=[1,2,3,4] --variables=['u','v']
```

Calculera les métriques présentes dans le fichier *metric_tests_exec.py* sur le Set_38, contenant des fichiers intitulés 'progan' pour les tailles de batch 8,16,32, le learning rate 0.001, les 4 premières instances et les variables 'u' et 'v'. L'algo va créer un produit cartésien de ces paramètres, trouver toutes les expériences correpondantes grâce aux noms de fichiers, et lancer successivement les calculs de métriques dessus.

A noter que chacun de ces paramètres est initialisé à une valeur par défaut. Si on oublie de mentionner un paramètre dans la ligne de commande, et que sa valeur par défaut correspond à une expérience inexistante, il ne se passera rien (voir le paragraphe ci-dessous).

2.3 Erreurs possibles

Si un fichier manque dans l'arborescence, ou que le nommage est incorrect, l'algo ne s'arrête pas, il dit juste : 'Il me manque un fichier là' et il passe à l'expérience suivante. Il faut donc vérifier précisément tous les noms, toutes les extensions, la présence du fichier ReadMe, l'accessibilité des directories.

2.4 Efficacité

1. Si on a beaucoup de métriques à calculer, il vaut mieux les calculer toutes dans un seul appel, car on ne charge les données qu'une fois par appel.

2. Mieux vaut utiliser parallel=True dans la mesure du possible; surtout si on a plusieurs itérations (aa) à calculer, ça va nettement plus vite.

3 Aperçu du code source

Le code est contenu dans le dossier score_crawl, qui est organisé comme suit :

- configurate.py fait appel à evaluation_backend.py, est appelé par metric_tests_exec.py Ce fichier contient les fonctions pour initialiser et parcourir les expériences voulues. C'est lui qui lit les arguments donnés en ligne de commande et le fichier ReadMe_xx.txt

- evaluation_frontend.py est appelé par metric_tests_exec.py : c'est lui qui définit la classe 'MetricsCalculator' et définit les logiques de calcul (standalone VS distance, parallele ou non, etc...) qui seront appliquées pendant l'exécution. Il appelle le fichier evaluation_backend.py

- evaluation_backend.py est appelé par configurate.py et evaluation_frontend.py : il contient quelques paramètres généraux (var_dict en particulier), et définit le chargement des fichiers, leur normalisation, les fonctions de calcul de métriques distance VS standalone, dont les paramètres sont modulés par le frontend.

Il y a plein d'autres fichiers dans score_crawl/, mais honnêtement pas tous utiles car très très ad'hoc. Pour les plot, plutôt se référer au dossier metrics4arome.