AI Frameworks. TP-Google Cloud/Docker.

Brendan Guillouet

3 décembre 2018

Contents

Introduction

Au cours de ce TP nous allons voir les différentes étapes permettant d'exécuter un code python sur un serveur distant à l'aide de *Google Cloud* et de *Docker* afin d'utiliser des puissances de calcul supérieures.

Nous allons utiliser pour cela les données du TP *Cats Vs Dogs*.

Voici les différentes étapes qui seront exécutées dans ce TP :

- Création de 2 scripts python fonctionnels en local permettant d'apprendre un modèle de *Deep Learning* et d'effectuer des prédictions à partir de ce modèle.
- Configuration d'une instance *Google Cloud*
- Envoie des données de *CatsVsDogs*, ainsi que les scripts python sur l'instance.
- Exécution du script sur l'instance.
- Récupération des résultats et des différentes observations sur votre machine en local.
- Réitérer ce processus à l'aide d'un container *Docker*.

Exercice : Avant de commencer ce TP. Assurez-vous que votre répertoire git *AI-Framework* est à jour. Clonez le ou mettez le à jour si ce n'est pas ke cas.

1 Script *python.*

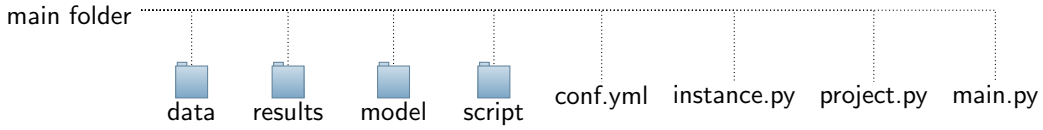
L'utilisation d'un serveur distant peut s'avérer coûteux. En effet les sociétés telles que *Google*, *Amazon* ou *Microsoft* facturent l'utilisation de leur machine à l'heure.

Il est alors très vivement déconseillé d'effectuer le travail d'exploration des données directement sur ces machines. Même si cela est possible, l'utilisation d'outils tel que *Jupyter*, n'est pas adapté. Une bonne pratique vise donc à effectuer le travail d'exploration des données sur une machine en locale et sur un jeu de données réduit, si cela est nécessaire. On écrit ensuite un script, toujours en local, permettant d'effectuer toutes les opérations désirées et de produire les résultats souhaitées. Une fois que le script fonctionne en local, on pousse et on exécute ce script sur une instance du serveur distant.

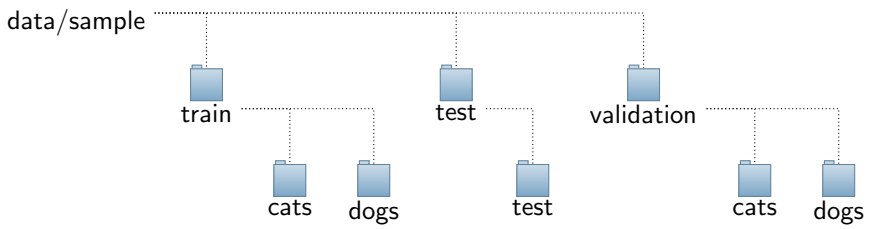
1.1 Créer l'arborescence

La première étape consiste à créer, en local, l'arborescence qui sera reproduite sur l'instance. Il est très important de définir rigoureusement cette organisation.

Voici celle qui sera utilisée dans ce TP:



- **data** : Ce dossier va contenir les données sur lesquelles votre algorithme sera exécuté. De façon plus général, il doit contenir toutes les données nécessaires à l'exécution de votre code. Dans ce dossier data, on placera un dossier *sample*, qui contient les données *train*, *test* et *validation* organisées de la sorte :



- **model** : Les différents modèles seront stockés dans ce dossier. De façon plus général, il doit contenir toutes les données qui doivent être sauvegardées au moins temporairement, mais que l'on ne souhaite pas récupérer sur notre machine en local en fin d'éxécution.

- **results** : Ce dossier va contenir différentes informations et statistiques que votre script va générer. De façon plus général, il doit contenir toutes les données que vous souhaitez récupérer pour les explorer en local.
- **script** : Code python permettant d'effectuer l'apprentissage et de générer des prédictions.
- **conf.yml, instances.py, project.py, main.py** : Code python permettant d'utiliser, et d'éxécuter les codes du scripts sur le serveur (Section 3.1 et 4.5.)

ExerciceDans le dossier *GoogleCloud* du répertoire git *AI-Frameworks*, cette organisation est déjà partiellement recrée. Les dossiers *metadata*, *model* et *script* sont manquants; ajoutez les.

Dans le dossier *GoogleCloud/data* du répertoire git un fichier *sample_2.zip* est présent. Décompresser le dans ce dossier vérifier que la structure des données présentée plus haut est bien présente.

1.2 Écriture des scripts

Nous allons maintenant écrire les deux scripts qui nous permettront d'atteindre nos objectifs :

- **learning.py** : Dans ce script nous allons définir et entraîner un nouveau modèle. Nous allons le sauvegarder dans le dossier *model* et sauvegarder les différents résultats que nous souhaitons étudier en fin d'exécutions dans le dossier *results*.
- **prediction.py** : Dans ce script nous allons télécharger le modèle entraîné précédemment et l'utiliser pour effectuer la prédiction sur les données tests, générer le fichier *.csv* contenant les réponses et le sauvegarder dans le dossier *results*.

Avant de commencer l'écriture de ces scripts, nous rappelons quelques bonne pratiques :

1.2.1 Argument en paramètre

L'exécution de vos scripts dans un terminal s'effectue de façon très simple et de la façon suivante :

```
python learning.py
```

Cependant votre script doit être assez souple pour prendre en compte différents paramètres au moment de l'éxécution. Ceux-ci peuvent-être classés en deux types :

- Paramètres du modèle (liste non exhaustive) : Nombre d'epochs, taille du batch, Nombre de neurones, architecture, etc...
- Paramètres de l'environnement : Ces paramètres changeront en fonction de la machine sur lequel le code va tourner: direction des données, du modèle, etc..

On utilisera la librairie **argparse** (*cf. diapos*) de python pour gérer ce type d'argument.

NB: Une autre possibilité consiste à boucler sur les paramètres à l'intérieur du script.

1.2.2 Résultats à sauvegarder

Il est très important d'anticiper quels seront les résultats et informations que vous souhaitez récupérer en fin d'exécution de votre code afin de pouvoir les analyser. Si vous avez oublié de sauvegarder certaines informations, celles-ci seront perdues en fin d'exécution du script.

Aussi avant d'envoyer votre script sur le *cloud*, veillez a ce que toutes les données et informations que vous aimeriez pouvoir étudier une fois l'exécution terminée soit sauvegardée dans le dossier *results*.

Vous pouvez sauvegarder ces informations dans un *dictionnaire* python à l'aide de la librairie **pickle** (*cf. diapos*).

1.2.3 Exercice

Dans le dossier *GoogleCloud/script*, ouvrez les fichiers *learning.py* et *prediction.py*. Complétez-les en suivant les consignes écrites dans chacun de ces fichiers afin qu'ils exécutent les objectifs énoncés section 1.2.

Assurez vous de pouvoir faire tourner ces deux scripts sur votre machine (sur l'échantillon *sample_2*) en ayant sauvegarder le modèle, la prédiction et les résultats dans les dossiers désirés.

Quand c'est le cas. Appelez votre responsable pour validation.

NB: Des solutions à ces exercices sont disponibles dans les scripts *learning_solution.py* et *prediction_solution.py*. Il est vivement conseillé de ne pas regarder ces solutions dans un premier temps.

2 Configuration d'une machine *Google Cloud.*

Aidez-vous des impressions écrans présentes dans les diapos du cours pour plus de précisions.

Par défaut, un projet est déjà créé. Vous allez donc pouvoir créer une instance sur le compte de ce projet.

2.1 Configurer une machine

- Dans le menu principal (Cliquez sur le *Hambuger*), dans le sous menu *Compute Engine* cliquez sur la section *instance de VM* (*virtual machine*).
- Cliquez alors sur l'onglet *Créer une instance*. Vous êtes alors sur le menu de configuration de votre machine:
  - Choisissez un nom et sélectionnez *europe-west1* comme région.
  - Dans le menu **Type de machine**, cliquez sur *Personnaliser*, sélectionnez 4 coeurs CPU et 1 GPU de type Tesla K80. *Vous pourrez sélectionnez des cartes plus puissantes plus tard.*
  - Dans le menu **Disque de Démarrage** sélectionnez *Ubuntu 16.04 LTS*. Dans ce menu vous pouvez également augmenter la taille du disque de démarrage, configurez-le à une taille de 50Go.
  - Dans le menu **Pare-feu**, cochez les cases *Autoriser le trafic HTTP* et *Autoriser le trafic HTTPS*.
  - Cliquez sur créer.

Attention : Votre VM démarre automatiquement à sa création.

2.2 Configuration de gcloud

gcloud est un outil de Google Cloud, permettant de gérer une instance depuis votre terminale. C'est-à-dire que vous pouvez envoyer des fichiers sur cette instances, des commandes, *etc.* directement depuis votre machine en local.

Un peu comme avec *git*, vous devez dans un premier temps, "initialiser" votre compte gcloud depuis votre machine en local pour que celle-ci soit connectée avec votre compte Google Cloud.

- Suivre les indications de https://cloud.google.com/sdk/docs/quickstart-debian-ubuntu de la partie *Initialize the SDK*
- Connectez-vous sur votre instance en exécutant la commande suivante sur **le terminal de votre machine en local**:

```
gcloud compute ssh NomDeVotreMachine
```

2.3 Installation de la machine

Vous êtes maintenant connecté sur le **terminal de votre instance**. Vous pouvez constater que celle-ci est presque entièrement vierge. Il vous faut donc installer les différents outils et logiciels nécessaires à l'exécution de votre code et notamment *Python3* et *Cuda* afin d'exécuter les version GPU de Tensorflow. Dans le dossier *GoogleCloud/utils/bash_util_on_gpu* de votre machine en local, vous trouverez différents scripts permettant l'installation de ces différents outils.

Exercice: Ouvrez un **terminal en local**. Completez la commande suivante depuis votre terminal en local afin d'envoyer ce dossier sur votre machine distante.

```
gcloud compute scp --recurse --zone europe-west1-b ACOMPLETER/bash_util_on_gpu/
instance-1:~ --ssh-key-file ACOMPLETER/.ssh/google_compute_engine
```

Une fois ce dossier envoyé sur votre instance. Retournez sur **terminal de votre instance** et constatez que le dossier est bien présent. Exécutez ensuite sur ce terminal ces différents scripts afin d'installer :

- Cuda,

  ```
  sh bash_util_on_gpu/bash_cuda_install.sh
  ```

- Python3 et autres utilitaires.

  ```
  sh bash_util_on_gpu/bash_python3.sh
  ```

Votre machine est maintenant prête à exécuter vos scripts.

NB: Ces scripts contiennent différentes commandes qui peuvent très facilement être trouvées sur internet sur les pages officiel de ces différents outils. N'hésitez pas à les parcourir.

3 Exécuter le code

3.1 Pipeline d'exécution

Afin d'exécuter vos scripts sur votre machine distante vous devez effectuer les différentes actions suivantes : Envoyer la dernière version de votre code sur la machine, possiblement mettre à jour vos données, exécuter vos différents scripts avec les différents paramètres souhaités, ramener sur votre **terminal, en local**, les différents résultats à analyser, penser à éteindre votre machine *etc...* Ces opérations ne sont pas compliquées, mais elles peuvent vite entraîner des erreurs de manipulation (oublier de mettre vos codes à jour, écraser d'anciens fichiers, oublier d'éteindre la machine, *etc...*), et représenter un travail fastidieux. Par exemple, lancer l'exécution d'un code python sur la machine distante se traduit par la commande suivante :

```
gcloud compute ssh nom_instance --zone europe-west-1 --ssh-key-file DirKeyFile
    --command 'python3 learning.py --data_dir DirData -- results_dir DirResults
                                    --epochs 10 --batch_size 128'
```

Afin d'éviter ce travail à chaque nouveau test, on va alors écrire un script permettant d'automatiser ces étapes.

3.2 Exercice

- Complétez le fichier **conf.yml** en indiquant les chemins des différents dossiers dans votre arborescence **en local**, et en indiquant les chemins des dossiers sur la machine distante telle que vous souhaitez les voir apparaître.
- Passez un peu de temps à explorer le fichier *main.py* et à comprendre l'utilité de chacune des fonctions appelées. Complétez les arguments indiqués *"ACOMPLETER"*.
- Exécutez le script *main.py* **sur votre terminal, en local** et interprétez les différentes sorties du terminal.
- Ouvrez un jupyter, en local. Téléchargez les résultats de votre exécution à l'aide de pickle.
- Affichez l'évolution du *loss* en fonction du nombre d'*epochs* à l'aide de *matplotlib*.
- **Bonus:** Téléchargez le jeu de données complet de CatsVsDogs sous format *zip* en suivant ce lien. Envoyez, à l'aide de la commande *gcloud compute scp*, le fichier zip sur l'instance. Faites tourner l'algorithmes sur l'ensemble des données en changeant uniquement l'argument de la commande *projectManager.update_data* du script *main.py*.

4 Docker

Dans cette partie nous allons voir comment exécuter notre code dans un container *Docker* sur la machine distante. Pour la suite, **toutes les commandes *"nvidia-docker"* sont a exécuter sur l'instance.**

4.1 Installation de la machine

Dans un premier temps il faut installer les outils *Docker* et *Nvidia-Docker* sur votre instance. Cette étape sera effectuée à l'aide des scripts bash contenu dans le dossier *bash_util_on_gpu* que vous avez précédemment envoyé sur votre instance).

Exécuter ces différents scripts sur le **terminal de votre instance** :

- docker

```
sh bash_util_on_gpu/bash_docker_install.sh
```

- nvidia-docker

```
sh bash_util_on_gpu/bash_nvidia_docker.sh
```

4.2 Dockerfile

Dans le dossier *GoogleCloud/utils/Docker* vous trouverez différents *Dockerfile* permettant de construire des images docker.

Exercice: Ouvrez le fichier *Dockerfile* et décrivez très rapidement la fonction des trois lignes qui le compose.

Un fichier *Dockerfile.devel-gpu* est également présent dans le dossier. Il ne correspond pas à l'image *tensorflow* directement utilisée par notre *Dockerfile*, mais il est très similaire. Vous pouvez le parcourir pour avoir une idée des installations effectuées.

Depuis **votre terminal en local**, utilisez la commande *gcloud compute scp* afin d'envoyer ce dossier sur votre machine distante afin de pouvoir les utiliser.

```
gcloud compute scp --recurse --zone europe-west1-b ACOMPLETER/Docker/
instance-2:~  --ssh-key-file ACOMPLETER/.ssh/google_compute_engine
```

Constatez que le dossier est bien présent sur votre instance.

4.3 Image

Nous allons maintenant construire une image docker à partir du Dockerfile en notre possession. Pour cela, exécutez la commande suivante depuis le **terminal de votre instance** en complétant par les arguments nécessaires.

```
sudo nvidia-docker build -t ACOMPLETER -f ACOMPLETER ACOMPLETER
```

Une fois l'image construite, cela peut prendre un moment la première fois, exécutez la commande suivante qui vous permet de lister les images disponibles sur votre machine.

```
sudo nvidia-docker image ls
```

Question Combien y a t-il d'images? A quoi correspondent elles?

4.4 Container

Vous avez maintenant construit votre propre *image*. La prochaine étape consiste donc à lancer un *container* à partir de cette image dans lequel vous allez pouvoir effectuez vos opérations et calculs.

4.4.1 Container interactif

Exercice: Complétez la commande suivante afin de lancer un container de façon interactive et sans volume.

```
sudo nvidia-docker run ACOMPLETER --name ACOMPLETER ACOMPLETER
```

Vous êtes maintenant à l'intérieur d'un container docker. Ouvrez une console python. Vérifier que *tensorflow* est installé et que c'est bien la version GPU qui est installée à l'aide de la commande suivante.

```
from tensorflow.python.client import device_lib
MODE = "GPU" if "GPU" in [k.device_type for k in device_lib.list_local_devices()
```

Quittez la console python.

Tous nos outils sont en place excepté le fait que le container docker n'a, pour le moment, pas accès aux données du TP! Avant de régler ce problème (Section 4.4.2), nous allons voir comment quitter et relancer un container:

- Quittez le container avec la commande *ctrl+d*.
- Lister les container à l'aide de la commande suivante :

  ```
  sudo nvidia-docker container ls
  ```

  Qu'observez vous? Réitérer la commande en y ajoutant l'option *-a*. Qu'observez-vous?

- En quittant un container, celui-ci se ferme automatiquement. Pour ré-utiliser le container en mode interactif, vous devez le re-démarrer

```
sudo nvidia-docker start container_it
```

- Une fois redémarré, vous devez vous *attacher* à ce container :

```
sudo nvidia-docker attach container_it
```

- Quittez de nouveau le container puis supprimer le définitivement :

```
sudo nvidia-docker rm container_it
```

4.4.2 Mounted Volume

Dans cette partie nous allons voir comment "monter" un volume dans le container. C'est-à-dire rendre disponible dans le container, un dossier situé sur la machine. L'argument *-v* permet cette opération.

Exercice: Complétez la commande suivante afin de lancer un container de façon interactive et qui aura accès au dossier *CatsVSDogs/sample*. On notera que, par défaut, le container s'ouvre dans le dossier *root*.

```
sudo nvidia-docker run -it --name ACOMPLETER -v
ACOMPLETER/CatsVsDogs/:/root/ACOMPLETER tfimg
```

Constatez que les données sont maintenant disponibles dans votre container. Créez un dossier *tmp* dans le dossier *CatsVSDogs/sample* au sein de votre container. Quittez le container. Ouvrez le dossier *CatsVSDogs/sample* sur votre instance. Qu'observez vous?

Avec cette manipulation, vous êtes maintenant capable de lancer un container, de lancer des opérations au sein de ce container et de sauvegarder des résultats auxquels vous aurez accès sur votre instance.

4.4.3 Background Mode

La dernière étape consiste a lancer un container en mode *background*. En effet, si le mode *interactif* est utile pour développer, comprendre le fonctionnement du container, il est utile de pouvoir lancer des calculs sur le container sans avoir à garder un terminal ouvert.

Exercice: Complétez la commande suivante afin de lancer un container en background et qui aura accès au dossier *CatsVSDogs/sample*.

```
sudo nvidia-docker run ACOMPLETER --name ACOMPLETER
-v ACOMPLETER
```

Votre container est lancé en mode *background*. Vérifiez-le en exécutant la commande suivante:

```
sudo nvidia-docker container ls
```

Qu'observez-vous?

Maintenant que vous savez que le container est correctement lancé et en train de tourner, vous pouvez lancer des exécutions au sein de ce container à l'aide de la commande *exec*. Pour commencer, listez les fichiers présents dans le dossier /root/ de votre container à l'aide de la commande suivante:

```
sudo nvidia-docker exec cn ls
```

Exercice: Dans votre container qui tourne en *background*, envoyez, à l'aide de la commande *exec*, une commande qui permettra de créer un dossier dans le volume que vous avez monté dans le container. Vérifiez sur votre machine que ce dossier est bien accessible. Une fois ces manipulations terminées, stoppez puis supprimez ce container, et vérifiez que la suppression est bien effective.

4.5 Pipeline d'exécution

L'utilisation de container docker à l'aide de l'outil gcloud est très simple. En effet, au moment de l'appel à la fonction *gcloud compute ssh* il suffit d'ajouter l'option *–container* suivi du nom du container que l'on souhaite utilisez pour que le calcul soit lancé dans un container plutôt que directement sur la machine.

Pour reprendre l'exemple de la Section 3.1, la commande correspondante permettant de faire tourner le code dans le container *NameContainer* est la suivante :

```
gcloud compute ssh nom_instance --zone europe-west-1 --ssh-key-file DirKeyFile
    --container NameContainer
    --command 'python3 learning.py --data_dir DirData -- results_dir DirResults
                                    --epochs 10 --batch_size 128'
```

Ainsi pour adapter le fichier *main.py* afin que celui-ci utilise le container il suffit d'ajouter les fonctions permettant les étapes suivantes :

- Lancer un container en background,
- Lancer les codes en ajoutant l'option *–container*,
- Stopper puis supprimer le container.

Le fichier *main_docker.py* a été écrit afin d'adapter ces différentes étapes.

4.6 Exercice

- Passez un peu de temps à explorer le fichier *main_docker.py* et pour comprendre les différences avec le fichier *main.py* utilisé précédemment. Repérez les nouvelles fonctions et les nouveaux arguments utilisés. Complétez les arguments indiqués *"ACOMPLETER"*
- Exécutez le script *main_docker.py* et interprétez les différentes sorties du terminal.

5 Application

Appliquez ce que vous venez d'apprendre pour le défi IA.