Training and Deploying TensorFlow Models at Scale [1] - Docker

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교 HANKYONG NATIONAL UNIV.

일단 놀라운 예측을 수행하는 아름다운 모델을 만들었다면 이제 그 모델을 가지고 무엇을 할 것인가?

⇒ 모델을 제품으로 출시!!

옵션 1: (모델 웹 서비스화)

- 회사 인프라의 다양한 파트에서 **REST API**를 통해 언제든지 모델 사용 가능

시간이 지남에 따라 새로운 데이터에 대해 모델을 재학습시켜야 하고, 모델 버전을 관리해야 하고, 제품이 성공해서 QPS(Queries Per Second)가 증가하면 규모를 확장해야 한다.

옵션 2: (규모 확장을 위해 **TF Serving** 사용)

- **Docker -** 자체 HW 인프라에
- **Google Cloud AI Platform -** 클라우드 서비스를 통해

Serving a TensorFlow Model

TensorFlow 모델을 학습시켰다면 어떤 Python 코드에서도 사용 가능:

- `tf.keras` 모델인 경우 그냥 **`predict()`** 메소드 호출!

인프라가 커지면

- 모델을 독립적인 **서비스(microservice)**로 만들어 사용해야 하는 시점에 도달
- 서비스의 역할 => 예측 수행, 인프라의 나머지 파트에서 REST(or gRPC) API를 통해 질의(query)할 수 있도록 해 줌

  ⇒ 모델과 인프라의 나머지 파트를 분리해 줌

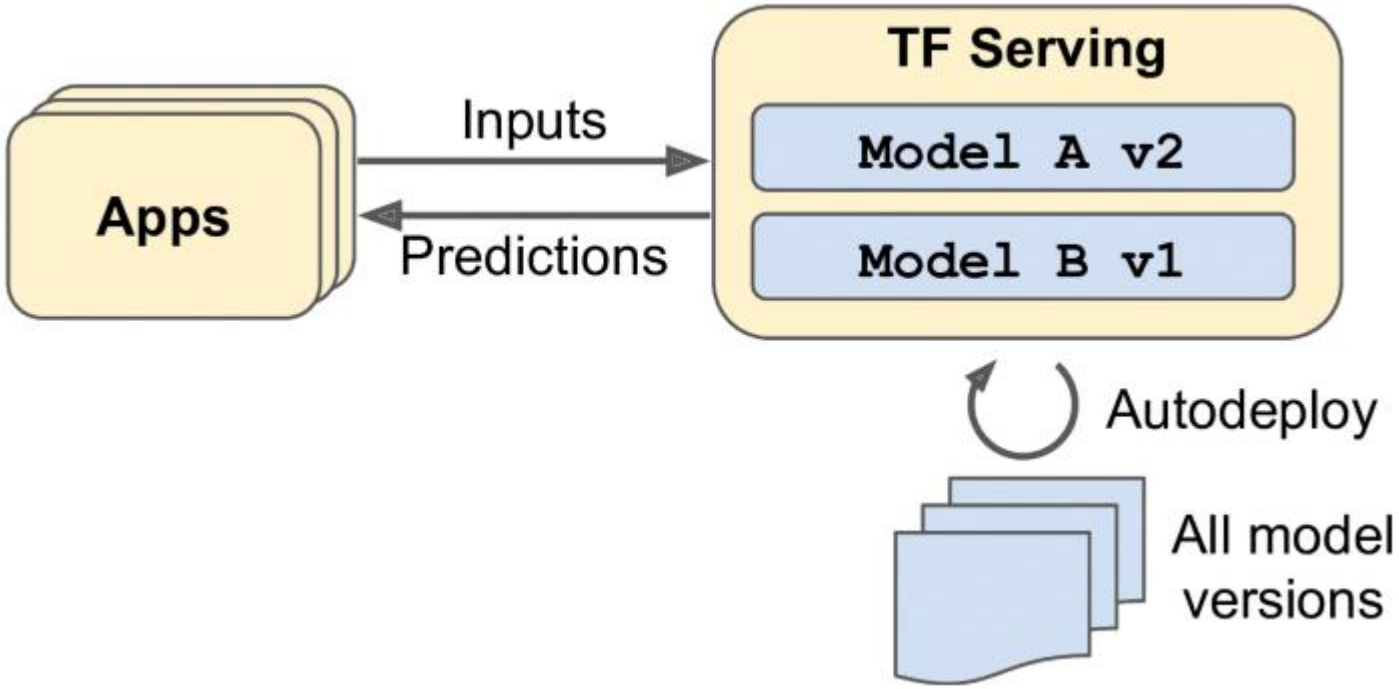
  ⇒ 모델 버전을 쉽게 스위칭 가능, 서비스 규모를 쉽게 확장 가능하게 해 줌
- 독립된 서비스 => 자신의 방법(예: Flask library)으로 개발 가능!!

Why reinvent the wheel when can just use TF Serving?

Using TensorFlow Serving

TF Serving

- C++로 만들어진 효율적이고 철저하게 테스트된 모델 서버
- 여러 개의 모델 버전 서비스 가능, 가장 최신 버전을 자동으로 배포 등



이제 `tf.keras`를 사용하여 MNIST 모델을 학습시키고 이를 TF Serving에 배치해 보자:

⇒ 맨 먼저 해야 할 일 : 모델을 TensorFlow의 **`SavedModel`** 포맷으로 export하는 것!!

Exporting SavedModels

TensorFlow => 모델을 SavedModel 형식으로 export할 수 있도록 **`tf.saved_model.save()`** 함수 제공

- save() 함수에게 모델(모델명, 버전 번호) 제공
- save() 함수 => 모델의 계산 그래프 및 가중치 저장:

```
model = keras.models.Sequential([...])
model.compile([...])
history = model.fit([...])

model_version = "0001"
model_name = "my_mnist_model"
model_path = os.path.join(model_name, model_version)
tf.saved_model.save(model, model_path)
```

Warning:
`SavedModel`은 계산 그래프를 저장하므로 배타적으로 TensorFlow 연산에 기반한 모델에만 사용될 수 있다.
동적 `tf.keras` 모델은 계산 그래프로 변환될 수 없기 때문에 다른 도구(예: **Flask**)를 사용해야 한다.

`SavedModel` => 사용자 모델의 한 버전:

- `saved_model.pb` 파일을 포함하는 디렉토리로서 저장됨(계산 그래프 정의)
- `variables` 서브 디렉토리
- `assets` 서브 디렉토리(여기서 사용 안함)

```
my_mnist_model
└── 0001
    ├── assets
    ├── saved_model.pb
    └── variables
        ├── variables.data-00000-of-00001
        └── variables.index
```

`tf.saved_model.load()` 함수를 이용하여 SavedModel 로딩:

- 반환된 객체 => Keras 모델이 아니라 **SavedModel**(계산 그래프와 변수 값들을 포함)
- 함수처럼 사용해 예측 수행:

```
saved_model = tf.saved_model.load(model_path)
y_pred = saved_model(tf.constant(X_new, dtype=tf.float32))
```

SavedModel을 직접 Keras 모델로 로딩하는 것도 가능:

- **`tf.models.load_model()`** 함수 이용

```
model = keras.models.load_model(model_path)
y_pred = model.predict(tf.constant(X_new, dtype=tf.float32))
```

- `predict()` 사용 => 예측

TensorFlow => saved_model_cli 커맨드라인 툴을 사용하여 SavedModel 검사:

```
$ export ML_PATH="$HOME/ml" # point to this project, wherever it is
$ cd $ML_PATH
$ saved_model_cli show --dir my_mnist_model/0001 --all
MetaGraphDef with tag-set: 'serve' contains the following SignatureDefs:
signature_def['__saved_model_init_op']:
  [...]

signature_def['serving_default']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['flatten_input'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 28, 28)
        name: serving_default_flatten_input:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['dense_1'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 10)
        name: StatefulPartitionedCall:0
  Method name is: tensorflow/serving/predict
```

`saved_model_cli` 커맨드 => 예측 수행에도 사용 가능:

- `X_new` => 예측을 수행할 3개의 필기체 숫자를 포함하는 NumPy 배열
- 먼저 NumPy의 `npy` 포맷으로 export:

```
np.save("my_mnist_tests.npy", X_new)
```

- `saved_model_cli` 커맨드 사용 => 예측:

```
$ saved_model_cli run --dir my_mnist_model/0001 --tag_set serve \
                      --signature_def serving_default \
                      --inputs flatten_input=my_mnist_tests.npy
[...] Result for output key dense_1:
[[1.1739199e-04 1.1239604e-07 6.0210604e-04 [...] 3.9471846e-04]
 [1.2294615e-03 2.9207937e-05 9.8599273e-01 [...] 1.1113169e-07]
 [6.4066830e-05 9.6359509e-01 9.0598064e-03 [...] 4.2495009e-04]]
```

Installing TensorFlow Serving (Docker option)

TF Serving을 설치하기 위한 방법으로 Docker 선택:

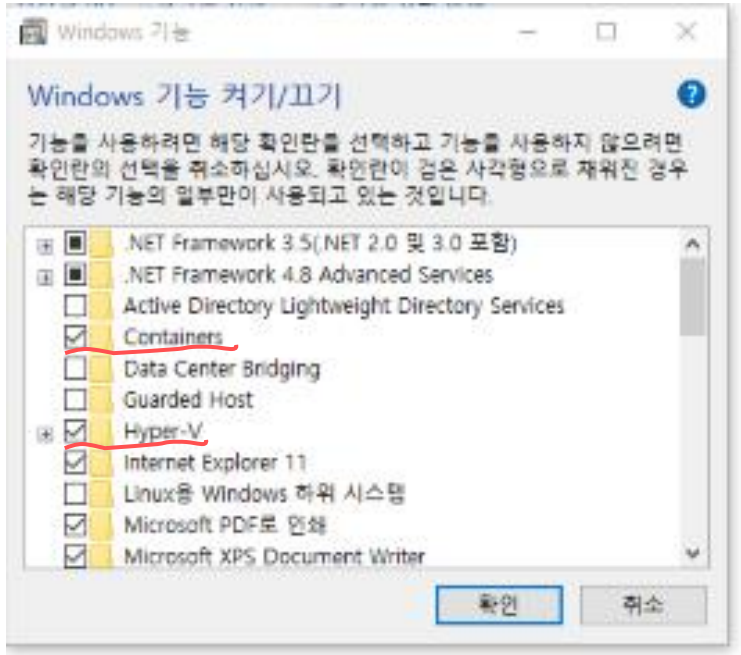
- Docker 설치 => TensorFlow 팀이 강력 권장!

Download for Windows: https://hub.docker.com/editions/community/docker-ce-desktop-windows

필수 설정:

윈도 키+Pause/Break > 제어판 홈> 프로그램 >

Windows 기능 켜기/끄기: **Hyper-V**, **Containers** 체크



공식 TF Serving Docker image downloading:

```
$ docker pull tensorflow/serving
```

Docker image를 실행하기 위한 Docker 컨테이너 생성:

```
$ docker run -it --rm -p 8500:8500 -p 8501:8501 \
             -v "$ML_PATH/my_mnist_model:/models/my_mnist_model" \
             -e MODEL_NAME=my_mnist_model \
             tensorflow/serving
[...]
2019-06-01 [...] loaded servable version {name: my_mnist_model version: 1}
2019-06-01 [...] Running gRPC ModelServer at 0.0.0.0:8500 ...
2019-06-01 [...] Exporting HTTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 237] RAW: Entering the event loop ...
```

*That's it! TF Serving is running. It loaded our MNIST model (version 1), and it is serving it through both gRPC (on port 8500) and **REST** (on port 8501).*

```
# Docker option
(aisam) E:\practice\ai\handson(pycharm)>docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
image operating system "linux" cannot be used on this platform

(aisam) E:\practice\ai\handson(pycharm)>cd C:\Program Files\Docker\Docker
(aisam) C:\Program Files\Docker\Docker>.\dockercli -SwitchDaemon
(aisam) C:\Program Files\Docker\Docker>docker pull tensorflow/serving

# Create a Docker container to run image:
$ docker run -it --rm -p 8500:8500 -p 8501:8501 -v
"e:/practice/ai/handson(pycharm)/my_mnist_model:/models/my_mnist_model" -e
MODEL_NAME=my_mnist_model tensorflow/serving
```

`-it`

Makes the container interactive (so you can press Ctrl-C to stop it) and displays the server's output.

`--rm`

Deletes the container when you stop it (no need to clutter your machine with interrupted containers). However, it does not delete the image.

`-p 8500:8500`

Makes the Docker engine forward the host's TCP port 8500 to the container's TCP port 8500. By default, TF Serving uses this port to serve the gRPC API.

`-p 8501:8501`

Forwards the host's TCP port 8501 to the container's TCP port 8501. By default, TF Serving uses this port to serve the REST API.

`-v "$ML_PATH/my_mnist_model:/models/my_mnist_model"`

Makes the host's `$ML_PATH/my_mnist_model` directory available to the container at the path */models/mnist_model*. On Windows, you may need to replace `/` with `\` in the host path (but not in the container path).

`-e MODEL_NAME=my_mnist_model`

Sets the container's `MODEL_NAME` environment variable, so TF Serving knows which model to serve. By default, it will look for models in the */models* directory, and it will automatically serve the latest version it finds.

`tensorflow/serving`

This is the name of the image to run.

Querying TF Serving through the REST API

REST API를 사용하여 서버에게 쿼리 수행:

- 쿼리 생성 => 호출하고 싶은 함수 signature name 및 입력 데이터를 포함해야

```
import json

input_data_json = json.dumps({
    "signature_name": "serving_default",
    "instances": X_new.tolist(),
})
```

- JSON 포맷 => 100% text-based

  `X_new` NumPy 배열 => Python 리스트로 변환 => JSON으로 포맷!

```
>>> input_data_json
'{"signature_name": "serving_default", "instances": [[[0.0, 0.0, 0.0, [...]
0.3294117647058824, 0.7254901960784314, [...very long], 0.0, 0.0, 0.0, 0.0]]]}'
```

이제 입력 데이터를 TF Serving에 전달해보자 : HTTP POST 요청을 통해

- requests 라이브러리를 이용하여 해결 가능 (`$ pip install requests`)

```
import requests

SERVER_URL = 'http://localhost:8501/v1/models/my_mnist_model:predict'
response = requests.post(SERVER_URL, data=input_data_json)
response.raise_for_status() # raise an exception in case of error
response = response.json()
```

- Response => "**predictions**" 키를 포함하는 dictionary

  `predictions` 리스트: Python list => NumPy array로 변환해야

```
>>> y_proba = np.array(response["predictions"])
>>> y_proba.round(2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.96, 0.01, 0.  , 0.  , 0.  , 0.  , 0.01, 0.01, 0.  ]])
```

Deploying a new model version

새로운 모델 버전 생성 => SavedModel을 *my_mnist_model/0002* directory에 export!

```
model = keras.models.Sequential([...])
model.compile([...])
history = model.fit([...])

model_version = "0002"
model_name = "my_mnist_model"
model_path = os.path.join(model_name, model_version)
tf.saved_model.save(model, model_path)
```

TF Serving => 규칙적인 시간 간격으로 새 모델 버전 체크!!

- 새 버전이 발견되면 pending 요청은 구버전으로 처리, 새 요청은 새 버전으로 처리
- 모든 pending 요청이 완료되면 구 모델 버전은 unloaded!

```
[...]
reserved resources to load servable {name: my_mnist_model version: 2}
[...]
Reading SavedModel from: /models/my_mnist_model/0002
Reading meta graph with tags { serve }
Successfully loaded servable version {name: my_mnist_model version: 2}
Quiescing servable version {name: my_mnist_model version: 1}
Done quiescing servable version {name: my_mnist_model version: 1}
Unloading servable version {name: my_mnist_model version: 1}
```

실습과제 14-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

★ **반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.**

참고:

제 14강 실습과제 14-1 (model-prediction) Training and Deploying TensorFlow Models at Scale [1] - Docker.pdf