제 4장 TensorFlow 2.x와 Keras

TensorFlow와 Keras는 머신러닝과 딥러닝 분석에 특화된 독립적으로 개발되고 있는 오픈소스(open source) 라이브러리이다. 그러나 2019년 9월 30일 TensorFlow 2.0이 발표되면서 Keras는 TensorFlow의 표준고급(standard high-level) API가 되어 Keras와 TensorFlow의 장점을 모두 이용할 수 있게 되었다. 이는 기존의 Keras가 TensorFlow에 완전히 종속되었다는 의미는 아니며 여전히 독립적으로 개발되고 있지만, 2019년 9월 Keras 2.3.0을 발표하면서 Keras는 더 이상의 주요 update를 하지 않으므로 TensorFlow의 고급언어인 tf.keras로 교체할 것을 권고하고 있다. TensorFlow 2.x내의 Keras는 tf.keras로 명명하여 Keras와 구별하고 있으며 tf.keras와 Keras는 1:1 대응관계를 가지고 있지 않다. Keras는 Google TensorFlow, Microsoft CNTK, Amazon MxNet, 그리고 Theano 등 딥러닝 엔진을 backend로 사용하지만(기본적으로 TensorFlow를 backend로 사용함), tf.keras는 오직 TensorFlow내에서만 사용할 수 있다. 그러나 tf.keras는 TensorFlow 2.x가 가지고 있는 다양하고 유용한 기능을 사용할 수 있다는 장점을 가지고 있다.

TensorFlow 2.x에서는 TensorFlow 1.x에서 사용하였던 하위수준(low-level) API, 예를 들어, 계산그래프(computational graph)를 위한 tf.Variables과 tf.placeholder, 계산그래프를 실행하기 위한 tf.Session, run 등을 더 이상 사용하지 않아도 되게 되었다. TensorFlow 2.x에서는 @tf.function을 이용하여 tf.Session 역할을 하도록 하였을 뿐만 아니라, 즉시 계산이 가능하도록 하여 훨씬 간단하고 직관적인 딥러닝 프로그램을 구성하도록 하였다. 또한 @tf.function을 이용하여 built-in이 아닌 자체적 손실함수(customized loss function)를 정의할 수 있으며 이 손실함수를 최소화하기 위한 미분을 tf.GradientTape() 틀래스를 통해 실행할 수 있다. 그러나 Keras가 익숙한 사람에게는 tf.keras가 훨씬 더 간편하게 자체적 손실함수를 정의할 수 있고 제 11장 Autoencoder와 Variational Autoencoder에서 볼 수 있듯이 tf.keras의 add_loss 함수를 통해 어렵지 않게 자체적 손실함수를 정의하고 이를 최소화할 수 있는 미분을 자동적으로 구할 수 있는 예제를 제공하고 있다. 그러므로 @tf.function과 tf.GradientTape()에 관심있는 독자들은 googling을 통해 공부하길 바란다.

Keras에도 built-in 데이터셋이 있지만, TensorFlow 2.x는 매우 다양하고 풍부한 built-in 데이터셋을 제공하고 있다. TensorFlow 데이터셋을 사용하기 위해

```
pip install tensorflow-datasets
```

으로 tensorflow_datasets을 설치한 후 아래의 프로그램을 실행하여 tensorflow 데이터셋의 리스트를 출력할 수 있으며 총 173개(2020년 6월 기준)의 데이터셋이 built-in으로 제공되고 있음을 알 수 있다.

```
import tensorflow_datasets as tfds
builders=tfds.list_builders()
print(builders)
print(len(builders))
```

```
['abstract_reasoning', 'aeslc', 'aflw2k3d', 'amazon_us_reviews', 'arc', 'bair_robot_pushing_small', 'beans', 'big_patent', 'bigearthnet', 'billsum', 'binarized_mnist', 'binary_alpha_digits', 'blimp', 'c4', 'caltech101', 'caltech_birds2010', 'caltech_birds2011', 'cars196', 'cassava', 'cats_vs_dogs', 'celeb_a', 'celeb_a_hq', 'cfq', 'chexpert', 'cifar10', 'cifar100', 'cifar10_1', 'cifar10_corrupted', 'citrus_leaves', 'cityscapes', 'civil_comments', 'clevr', 'cmaterdb', 'cnn_dailymail', 'coco', 'coil100', 'colorectal_histology', 'colorectal_histology_large', 'common_voice', 'cos_e', 'crema_d', 'curated_breast_imaging_ddsm', 'cycle_gan', 'deep_weeds', 'definite_pronoun_resolution', 'dementiabank', 'diabetic_retinopathy_detection', 'div2k', 'dmlab', 'downsampled_imagenet', 'dsprites', 'dtd', 'duke_ultrasound', 'emnist', 'eraser_multi_rc', 'esnli', 'eurosat', 'fashion_mnist', 'flic', 'flores', 'food101', 'forest_fires', 'gap', 'geirhos_conflict_stimuli', 'german_credit_numeric', 'gigaword', 'glue', 'groove', 'higgs', 'horses_or_humans', 'i_naturalist2017', 'image_label_folder', 'imagenet2012', 'imagenet2012_corrupted', 'imagenet2012_subset', 'imagenet_resized', 'imagenette', 'imagewang', 'imdb_reviews', 'iris', 'kitti', 'kmnist', 'lfw', 'librispeech', 'librispeech_lm', 'libritts', 'ljspeech', 'lm1b', 'lost_and_found', 'lsun', 'malaria', 'math_dataset', 'mnist', 'mnist_corrupted', 'movie_rationales', 'moving_mnist', 'multi_news', 'multi_nli', 'multi_nli_mismatch', 'natural_questions', 'newsroom', 'nsynth', 'omniglot', 'open_images_challenge2019_detection', 'open_images_v4', 'opinosis', 'oxford_flowers102', 'oxford_iiit_pet', 'para_crawl', 'patch_camelyon', 'pet_finder', 'places365_small', 'plant_leaves', 'plant_village', 'plantae_k', 'qa4mre', 'quickdraw_bitmap', 'reddit', 'reddit_tifu', 'resisc45', 'robonet', 'rock_paper_scissors', 'rock_you', 'samsum', 'savee', 'scan', 'scene_parse150', 'scicite', 'scientific_papers', 'shapes3d', 'smallnorb', 'snli', 'so2sat', 'speech_commands', 'squad', 'stanford_dogs', 'stanford_online_products', 'starcraft_video', 'stl10', 'sun397', 'super_glue', 'svhn_cropped', 'ted_hrlr_translate', 'ted_multi_translate', 'tedlium', 'tf_flowers', 'the300w_lp', 'tiny_shakespeare', 'titanic', 'trivia_qa', 'uc_merced', 'ucf101', 'vgg_face2', 'visual_domain_decathlon', 'voc', 'voxceleb', 'waymo_open_dataset', 'web_questions', 'wider_face', 'wiki40b', 'wikihow', 'wikipedia', 'wmt14_translate', 'wmt15_translate', 'wmt16_translate', 'wmt17_translate', 'wmt18_translate', 'wmt19_translate', 'wmt_t2t_translate', 'wmt_translate', 'xnli', 'xsum', 'yelp_polarity_reviews']

173
```

위 173개 데이터셋 중 손글씨 데이터인 'mnist'데이터를 불러내어 학습데이터와 시험데이터를 구성하고 'mnist'데이터에 대한 정보를 출력하면 다음과 같다.

```
data, info=tfds.load('mnist', with_info=True)
train_data, test_data=data['train'],data['test']
print(info)
```

```
Dataset mnist downloaded and prepared to C:₩Users₩yspark₩tensorflow_datasets₩mnist₩3.0.1. Subsequent calls will reuse this data.
tfds.core.DatasetInfo(
    name='mnist',
    version=3.0.1,
    description='The MNIST database of handwritten digits.',
    homepage='http://yann.lecun.com/exdb/mnist/',
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
    },
    supervised_keys=('image', 'label'),
    citation="""@article{lecun2010mnist,
      title={MNIST handwritten digit database},
      author={LeCun, Yann and Cortes, Corinna and Burges, CJ},
      journal={ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist},
      volume={2},
      year={2010}
    }""",
    redistribution_info=,
)
```

저자와 같이 딥러닝을 Keras로 시작하면 tf.keras를 사용하기 위해 TensorFlow 2.x를 새롭게 공부해야 하나? 하는 두려움이 있을 것이지만, 기존의 Keras와 동일한 프로그램을 사용하므로 전혀 걱정할 필요가 없다. 다만, 기존의 Keras 프로그램에서 keras를 tensorflow.keras로만 수정하면 error 없이 잘 실행되는 것을 확인할 수 있을 것이다. TensorFlow 2.x에서 기존의 Keras를 표준고급 API로 채택하여 tf.keras로 출시하였기 때문이다.

tf.keras를 본격적으로 논의하기 이전에 몇가지 중요한 기술적인 문제를 정리 요약하기로 하자. 기존의 딥러닝 관련 Keras 프로그램을 공부하다보면

```
__future__ import print_function, division, absolute_import
```

를 볼 수 있을 것이다.

이는 2008년 12월 python3가 발표된 이후, python2와 python3를 동시에 동작할 수 있도록 하는 모듈로 python2 문법을 python3 문법으로 전환시키는 역할을 한다. print_function은 python3에서 print는 반드시 ()를 쓰도록 하였고, division은 python2에서 8/7=1, 8//7=1로 동일한 나눗셈을 하였으나 python3에서는 8/7=1.1428, 8//7=1이 되도록 구분하였다. 또한 alsolute_import는 표준모듈과 자체모듈(customized module)을 동시에 사용할 수 있도록 python3에서는 허용하고 있다. 그러므로 __future__는 python3 이전에 작성된 또는 python2로 작성된 프로그램을 실행하거나 수정·보완할 때 유용한 모듈이다.

딥러닝 모형은 빅데이터를 사용하고 추정해야할 모수는 많게는 수천만 개에 이르므로 직렬연산장치인 CPU (central processing unit)보다는 병렬연산장치인 GPU (graphic processing unit)를 이용하여 계산속도를 높일 필요가 있다. 2020년 5월 TensorFlow 2.2.0이 출시된 후

```
pip install tensorflow
```

또는 이미 tensoeflow가 설치되어 있으면

```
pip install –upgrade tensorflow
```

으로 update하면 tf.keras를 CPU와 GPU로 실행할 수 있다. 참고로 tf.keras를 CPU로만 실행하려면

```
pip install tensorflow-cpu
```

를 anaconda prompt에 입력·실행하면 된다. TensorFlow 2.2.0 이전 version에서는 tensorflow-cpu와 tensorflow-gpu를 각각 설치해야 했고 이로 인한 시스템적 충돌이 자주 발생하여 많은 불편이 발생하였다. 그러므로 현재 사용하는 TensorFlow가 2.2 이전 version이면 update하기를 추천한다. tensorflow-gpu로 딥러닝 모형을 실행하면 OOM (out of memory) 에러가 자주 발생한다. 메모리의 조각화를 방지하기 위해 gpu 전체에 메모리를 할당하기 때문이다. 이를 방지하기 위해

```
import tensorflow as tf
config=tf.ConfigProto()
config.gpu_options.allow_growth=True
tf.keras.backend.set_session(tf.Session(config=config))
```

를 tf.keras 프로그램 맨 앞에 입력하여 실행하면 GPU 메모리가 메모리 수요에 따라 적절하게 증감하게 된다. tensorflow-gpu를 사용할 때 무조건 입력하기를 권고한다. 이러한 메모리

조정에도 불구하고 배치크기가 약간만 커도 OOM이 발생하여 딥러닝모수 추정시 배치크기를 줄일 필요가 있다. 제 13장의 CycleGAN에서 이러한 현상이 발생하여 배치크기를 2 또는 4로 줄이게 되었다. 이러한 문제 때문에 tensorflow-gpu 환경에서 시간이 많이 걸리더라도 cpu만을 이용하여 딥러닝모수를 추정하고자 할 때는

```
import os
os.environ['CUDA_DEVICE_ORDER']='PCI_BUS_ID'
os.environ['CUDA_VISIBLE_DEVICE']='-1'
```

을 tf.keras 프로그램 시작부분에 입력하면 된다.