


# DNN\_01\_2. MLP\_tensorflow

🕒 생성일	@2022년 6월 10일 오후 1:50
🏷️ 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

## Tensorflow 저수준 api 에서 MLP 모델 구현.

저수준 api 에서 MLP 모델을 작성하는 법에 관한 내용이다. 코드 + 설명으로 진행한다. 코딩 환경은 구글 코랩 진행했으며 구글 드라이브를 저장소로 사용했다.

개발에 필요한 패키지들을 txt 파일로 작성해두었다. /content/drive/Mydrive/source 위치에 “requirements.txt” 파일명으로 저장.

```
pytorch-lightning==1.3.8
torch-optimizer==0.1.0
hydra-core==1.1
wandb==0.11.1
torchtext==0.10.0
spacy==2.2.4
efficientnet_pytorch==0.7.1
tensorflow-addons==0.14.0
```

위의 패키지를 설치하기 위해 구글 드라이브 접근 및 시스템 path 설정 → !pip 를 통해 패키지 설치

```
from google.drive import drive
drive.mount("/content/drive")

import os
import sys
sys.path.append("/content/drive/Mydrive/source")
!pip install -r "/content/drive/Mydrive/source/requirements.txt"
```

## 추가 라이브러리 설치

코랩에서 제공하는 머신러닝 관련 라이브러리를 import.

```
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_addons as tfa
import wandb
```

## 하드웨어 체크 및 GPU 설정.

코랩에서는 48 시간까지 연속으로 GPU 를 사용할 수 있다. GPU 를 사용하기 위한 세팅은 다음과 같다.

```
tf.config.list_physical_devices()
# define GPUs strategy
mirrored_strategy = tf.distribute.MirroredStrategy() # gpu 병렬 처리할 수 있다.
```

## data : normalization / split / dataloader

```
with mirrored_strategy.scope():
    # 사용할 데이터셋은 fashion_mnist 이다.
    fashion_mnist = tf.keras.datasets.fashion_mnist
    (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

    # 이미지 데이터 -> 255 로 나누어서 0-1 사이로 정규화
    x_train = x_train / 255.0 # float type
    x_test = x_test / 255.0

    # split 비율 미리 지정
    train_size = int(len(x_train) * 0.9)
    val_size = len(x_train) - train_size

    # dataset 을 만들기 위해 tensor 객체로 변환 후 이미지와 라벨을 함께 전달.
    dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(buffer_size=1024)
    test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).shuffle(buffer_size=1024)

    # dataset split 과정
    train_dataset = dataset.take(train_size)
    val_dataset = dataset.skip(train_size)

    # dataloader 정의

    train_batch_size = 100
    val_batch_size = 10
    test_batch_size = 100

    train_dataloader = train_dataset.batch(train_batch_size, drop_remainder = True)
    val_dataloader = val_dataset.batch(val_batch_size, drop_remainder = True)
    test_dataloader = test_dataset.batch(test_batch_size, drop_remainder = True)
```

입력 파이프라인 빌드 관련 내용은 링크를 참고하면 된다.(이후 관련 내용 다룰 예정.) :

### tf.data: TensorFlow 입력 파이프라인 빌드 | TensorFlow Core

tf.data API를 사용하면 간단하고 재사용 가능한 조각으로 복잡한 입력 파이프라인을 빌드할 수 있습니다. 예를 들어, 이미지 모델의 파이프라인은 분산된 파일 시스템의 파일에서 데이터를 집계하고 각 이미지에 임의의 퍼터베이션을 적용하며 무작위로 선택한 이미지를 학습을 위한 batch로 병합할 수 있습니다.

 <https://www.tensorflow.org/guide/data?hl=ko>



다음은 모델 세션이다.

## model : MLP (with Dropout)

```
class MLP(tf.keras.Model):
    def __init__(self, input_dim:int, h1_dim:int, h2_dim:int, out_dim:int, dropout_prob: float):
        super().__init__()
        self.flatten = tf.keras.layers.Flatten()
        self.linear1 = tf.keras.layers.Dense(input_dim=input_dim, units=h1_dim)
        # units : 출력값의 크기
        self.linear2 = tf.keras.layers.Dense(units=h2_dim)
        self.linear3 = tf.keras.layers.Dense(units=out_dim)
        self.dropout = tf.keras.layers.Dropout(dropout_prob)
        # activation
        self.relu = tf.nn.relu

    def call(self, input, training=False):
```

```

# 모델에서 실질적으로 작동하는 부분.
x = self.flatten(input)
x = self.relu(self.linear1(x))
x = self.dropout(x, training=training)
x = self.relu(self.linear2(x))
x = self.dropout(x, training=training)
out = self.linear3(x)
out = tf.nn.softmax(out)
return out

def train_step(self, data):
    # 손실 함수를 통과시켜서 손실 계산하고 그레디언트 업데이트 하는 과정.
    images, labels = data

    with tf.GradientTape() as tape:
        outputs = self(images, training=True)
        preds = tf.argmax(outputs, 1)

        loss = self.compiled_loss(labels, outputs)

    # compute gradients
    trainable_vars = self.trainable_variables
    gradients = tape.gradient(loss, trainable_vars)

    # update weights
    self.optimizer.apply_gradients(zip(gradients, trainable_vars))

    # update the metrics
    self.compiled_metrics.update_state(labels, preds)

    # return a dict mapping metrics names to current values
    logs = {m.name: m.result() for m in self.metrics}
    logs.update({"loss": loss})
    return logs

# test_step 의 경우 일단 train_step 과 동일.
def test_step(self, data):
    images, labels = data
    outputs = self(images, training=True) # call 함수를 불러온다.
    preds = tf.argmax(outputs, 1) # 예측확률 계산, 가장 높은 값 불러오기

    loss = self.compiled_loss(labels, outputs)

    # update the metrics
    self.compiled_metrics.update_state(labels, preds)

    # return a dict mapping metrics names to current values
    logs = {m.name: m.result() for m in self.metrics}
    logs.update({"test_loss": loss})
    return logs

```

- `tf.GradientTape()`

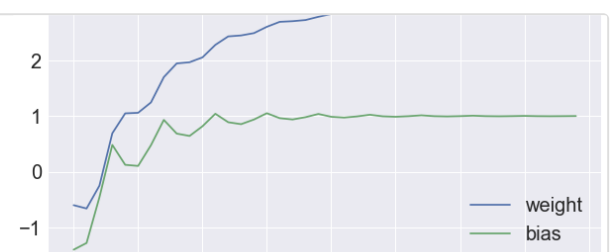
딥러닝에서는 prediction 과 loss 를 연산하는 forward propagation 과 partial derivative 와 vector chain rule 을 이용한 back propagation 이 존재한다. Tensorflow 와 같은 프레임워크를 이용하는 가장 큰 이유는 이 foward/backward propagation 을 쉽게 구현할 수 있도록 도와주는 Automatic differentiation 기능 때문이다.

back propagation 을 연산할 때 하나하나 gradient 를 구하는 것이 아니라 foward propagation 이 진행될 때 이 값들을 저장해두면 back propagation에서 이 저장된 값들을 이용하여 훨씬 빠르게 연산이 가능하다. 이때 `tf.GradientTape` 에는 forward propagation 이 진행되면서 역전파에 필요한 값들을 저장해둔다.

#### [Tensorflow2 강의자료] 4. Gradient Tape(`tf.GradientTape`)

Neural network를 만든다는 것은 곧 computational graph를 만든다는 것이다. 즉 node와 edge를 만들어 network computation을 만든다. 일반적인 computational graph와는 다르게 deep learning에서는 prediction과 loss를 연산하는 forward propagation과 partial derivative와 vector chain rule을 이용한 backpropagation이 존재한다. 우리가

🔗 <https://shinslab.tistory.com/110>



- `tf.GradientTape.gradient(loss, trainable_variables)`

`tf.Module` 의 모든 서브 클래스는 `Module.trainable_variables` 속성에서 변수를 집계하므로 몇 줄의 코드로 이러한 그레디언트를 계산할 수 있다. 이러한 기능을 `GradientTape` 에서도 사용가능하며, loss function 에 대한 gradient 계산을 `trainable_variables` 변수를 추가함으로써 가능하다.

- optimizer.apply\_gradients(zip(grad, model.trainable\_variables))


업데이트를 통해 경사하강법 1 step 을 진행한다. 진행의 결과는, loss function 을 최솟값으로 만드는 변수들의 value.(=업데이트된 가중치들)

- self.compiled\_metrics.update\_state(y, y\_pred)

update metrics includes the metric that tracks the loss.

#### Model.fit의 동작 사용자 정의하기 | TensorFlow Core

감독 학습을 수행할 때 fit() 를 사용할 수 있으며 모든 것이 원활하게 작동합니다. 훈련 루프를 처음부터 작성해야 하는 경우, GradientTape 를 사용하여 모든 세부 사항을 제어할 수 있습니다. 그러나 사용자 정의 훈련 알고리즘이 필요하지만 콜백, 내장 배포 지원 또는 단계 융합과 같은 fit() 의 편리한 특성을 계속 활용하려면 어떻게 해야 할까요?

 [https://www.tensorflow.org/guide/keras/customizing\\_what\\_happens\\_in\\_fit?hl=ko](https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit?hl=ko)



## compile setting

```
n_class = 10
max_epoch = 50

with mirrored_strategy.scope():
    model = MLPWithDropout(28*28*1, 128, 64, n_class, dropout_prob=0.3)
    model_name = type(model).__name__

    # define loss function
    loss_function = tf.losses.SparseCategoricalCrossentropy()

    # define learning rate
    # if learning_scheduler exists, use it.
    lr = 1e-3
    scheduler = LinearWarmupLRScheduler(lr, 1500)
    scheduler_name = type(scheduler).__name__ if scheduler is not None else "no_scheduler"
    if scheduler is None:
        scheduler = lr

    # define optimizer
    optimizer = tf.optimizers.Adam(learning_rate=scheduler)
    optimizer_name = type(optimizer).__name__

    # model compile
    model.compile(
        loss=loss_function,
        optimizer=optimizer,
        metrics=[tf.keras.metrics.Accuracy()],
    )

model.build((1, 28*28*1))
model.summary()
```

## logging & callbacks

logging 은 모델의 학습 진행과정, 학습 이후 모델 선별 및 가중치 업데이트 등의 결과를 확인하거나 개선점을 파악하기 위해 꼭 필요한 부분이다. 텐서보드와 wandb 를 통해 logging 작업을 수행할 수 있다. 둘 다 외부 프레임워크이기 때문에 연동 코드가 필요하다.

```
from datetime import datetime
drive_project_root = "/content/drive/Mydrive/source"
# 사실 위의 project root 설정은 sys.path 에서 다뤄야 한다.
# 모델 저장/관리를 위해 다뤄야 한다.

log_interval = 100

# 아래 제시된 run_name 은 optimizr 에 따라, learning rate 에 따른 실험을 위해 만든 이름 형식이다.
```

```

run_name = f"{datetime.now()}-{model_name}-{optimizer_name}-optim-{lr}-lr"
run_dirname = "dnn-tutorial-fashion-mnist-runs-tf"

log_dir = os.path.join(drive_project_root, "runs", run_dirname, run_name)

# callbacks - Tensorboard, Earlystopping
tb_callback = tf.keras.callbacks.TensorBoard(log_dir,
                                              update_freq=log_interval)
early_stop_callback = tf.keras.callbacks.EarlyStopping(patience=5, verbose=True)

```

```

# wandb 연동 전에 물론 wandb 로그인은 되어 있어야 한다.
# wandb setup
project_name = "fastcampus_fashion_mnist_tutorials_tf"
run_tags = {project_name}
wandb.init(
    project=project_name
    name=run_name
    tags=run_tags
    config={
        "lr": lr,
        "model_name": model_name,
        "optimizer_name": optimizer_name
    },
    reinit=True,
    sync_tensorboard=True
)

```

```

%load_ext tensorboard
%tensorboard --logdir /content/drive/Mydrive/source/runs

model.fit(
    train_data_loader,
    validation_data=val_data_loader,
    epochs=max_epoch,
    callbacks=[tb_callback, early_stop_callback]
)

```

## model testing(수정중)

```

# 모델 평가
model.evaluate(test_data_loader)

# calculate accuracy
test_labels_list = []
test_preds_list = []
test_outputs_list = []

for i, (test_images, test_labels) in enumerate(tqdm(test_data_loader, position=0, leave=True, desc='testing')):
    with mirrored_strategy.scope():
        test_outputs = model(test_images)
        test_preds = tf.argmax(test_outputs, 1)

        final_outs = test_outputs.numpy()
        test_outputs_list.extend(final_outs)
        test_preds_list.extend(test_preds.numpy())
        test_labels_list.extend(test_labels.numpy())

test_preds_list = np.array(test_preds_list)
test_labels_list = np.array(test_labels_list)

test_accuracy = np.mean(test_preds_list == test_labels_list)
print(f"\nacc: {test_accuracy*100}")

```