

MNIST 손글씨 TF 튜토리얼

TF2 Quickstart Beginners

<https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb?hl=ko#scrollTo=3wF5wszaj97Y>

파일

- `my_beginner.ipynb`





초보자를 위한 가이드

+ 코드
+ 텍스트
드라이브로 복사
연결
수정 가능

Copyright 2019 The TensorFlow Authors.

[] Licensed under the Apache License, Version 2.0 (the "License");

TensorFlow 2 quickstart for beginners


[View on TensorFlow.org](#)

[Run in Google Colab](#)

[View source on GitHub](#)

[Download notebook](#)

This short introduction uses [Keras](#) to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a [Google Colaboratory](#) notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime > Run all*.

MNIST 손글씨

```
[ ] 1 import tensorflow as tf
```



Load and prepare the [MNIST dataset](#). Convert the samples from integers to floating-point numbers:

```
[ ] 1 mnist = tf.keras.datasets.mnist
    2
    3 (x_train, y_train), (x_test, y_test) = mnist.load_data()
    4 x_train, x_test = x_train / 255.0, x_test / 255.0
```

Build the `tf.keras.Sequential` model by stacking layers. Choose an optimizer and loss function for training:

```
[ ] 1 model = tf.keras.models.Sequential([
    2     tf.keras.layers.Flatten(input_shape=(28, 28)),
    3     tf.keras.layers.Dense(128, activation='relu'),
    4     tf.keras.layers.Dropout(0.2),
    5     tf.keras.layers.Dense(10)
    6 ])
```

손실 함수 SparseCategoricalCrossentropy

- **tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)**
 - 뉴런 결과가 softmax를 취한 확률 값이 아닌 경우, from_logits=True
 - 기본은 softmax를 취한 결과로 from_logits=False
 - 정답은 바로 정수

```
[30] 1 cce = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True)
      2
      3 target = [[0], [1], [2]]
      4 output = [[1.2, 0.3, 0.1], [.15, .89, .26], [.25, .41, .94]]
      5
      6 loss = cce(target, output)
      7 print(loss.numpy())
```

0.66293424

```
[32] 1 cce = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False)
      2
      3 target = [[0], [1], [2]]
      4 output = [[1.0, 0., 0.], [.05, .89, .06], [.05, .01, .94]]
      5
      6 loss = cce(target, output)
      7 print(loss.numpy())
```

0.059469808

손실 함수 CategoricalCrossentropy

- **tf.keras.losses.CategoricalCrossentropy(from_logits=True)**
 - 뉴런 결과가 softmax를 취한 확률 값이 아닌 경우, from_logits=True
 - 정답은 분류의 원핫인코딩

```
[33] 1 cce = tf.keras.losses.CategoricalCrossentropy(from_logits = True)
      2
      3 target = [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]
      4 output = [[1.2, 0.3, 0.1], [.15, .89, .26], [.25, .41, .94]]
      5
      6 loss = cce(target, output)
      7 print(loss.numpy())
```

↪ 0.66293424

```
[34] 1 cce = tf.keras.losses.CategoricalCrossentropy(from_logits = False)
      2
      3 target = [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]
      4 output = [[1.0, 0., 0.], [.05, .89, .06], [.05, .01, .94]]
      5
      6 loss = cce(target, output)
      7 print(loss.numpy())
```

↪ 0.059469786

모델의 최적화 방법 및 손실 함수 지정

```
model.compile(optimizer='adam',  
              loss=loss_fn,  
              metrics=['accuracy'])
```


모델로 예측한 로짓

- 첫 학습 데이터를 model로 직접 로짓(logit) 값 생성
 - 각 자릿수의 벡터 값 반환
 - 그 자릿수일 가능성에 대한 값, 클수록 높은 가능성
 - 맞으면, 최대 값의 위치가 정답

```
[36] 1 print(y_train[0])
```

```
↳ 5
```

```
[37] 1 predictions = model(x_train[:1]).numpy()
      2 predictions
```

```
↳ array([[ -10.200226 ,  -5.6334195,  -4.217148 ,   6.610473 , -23.137548 ,
           9.5274315, -16.32289 ,  -7.115164 ,  -5.709602 ,  -4.5382705]],
        dtype=float32)
```

```
[38] 1 tf.math.argmax(predictions, 1).numpy()
```

```
↳ array([5])
```

결과 벡터를 확률 값으로 변환

• 함수 softmax

– 각 자릿수의 확률 값

• 합이 1

The `tf.nn.softmax` function converts these logits to "probabilities" for each class:

```
[39] 1 #학습 데이터 첫 번째의 결과 확률
      2 sm = tf.nn.softmax(predictions)
      3 sm.numpy()
```

```
↳ array([[2.5674698e-09, 2.4708382e-07, 1.0184110e-06, 5.1321477e-02,
          6.1787178e-15, 9.4867623e-01, 5.6294552e-12, 5.6147538e-08,
          2.2895931e-07, 7.3868807e-07]], dtype=float32)
```

```
[40] 1 tf.reduce_sum(sm).numpy()
```

```
↳ 0.99999994
```

```
[41] 1 max = tf.math.reduce_max(sm).numpy()
      2 print(max)
```

```
↳ 0.9486762
```

손실 값 계산

• 다음 두 손실 값이 동일

- 우리가 정의한 손실 함수로 직접 계산
 - `loss_fn`
- 맞는 자릿수 값의 $-\log(\text{logits})$
 - `-\log(\text{max_logit_of_softmax})`

```
[42] 1 loss_fn(y_train[:1], predictions).numpy()
```

```
↳ 0.052687697
```

$-\log(H(x))$

```
[43] 1 print(-tf.math.log(max).numpy())
```

```
↳ 0.05268771
```

• 다음 값도 동일

- 기본 인자가 `from_logits=False`이므로 예측 값을 softmax를 취한 결과인 `sm`을 사용해야 함

다음 값도 같다!

```
tf.keras.losses.sparse_categorical_crossentropy(y_train[:1], sm).numpy()
```

훈련과 평가

```
[45] 1 model.fit(x_train, y_train, epochs=5)
```

```
↳ Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0678 - accuracy: 0.9783
Epoch 2/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0594 - accuracy: 0.9811
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0535 - accuracy: 0.9829
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0470 - accuracy: 0.9845
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0442 - accuracy: 0.9853
<tensorflow.python.keras.callbacks.History at 0x7f7f24890cc0>
```

The `Model.evaluate` method checks the models performance, usually on a "[Validation-set](#)" or "[Test-set](#)".

```
[46] 1 model.evaluate(x_test, y_test, verbose=2)
```

```
↳ 313/313 - 0s - loss: 0.0667 - accuracy: 0.9796
[0.06672389805316925, 0.9796000123023987]
```

테스트 데이터의 확률 값

• 확률 모델 생성

- 모델 마지막에 softmax() 적용 모델로 테스트 데이터 첫 5개 예측
- 이전 model에 softmax() 함수를 적용한 결과와 동일
 - 첫 테스트 데이터만 시험함

```
[47] 1 probability_model = tf.keras.Sequential([
      2     model,
      3     tf.keras.layers.Softmax()
      4 ])
```

```
[48] 1 probability_model(x_test[:5])
```

```
↳ <tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[1.71166925e-09, 2.80625913e-12, 2.00822560e-05, 3.72063914e-05,
        5.71995717e-15, 1.46413814e-09, 1.19693216e-19, 9.99942660e-01,
        2.89911783e-09, 8.75712871e-08],
       [1.10604292e-09, 2.27550729e-04, 9.99772489e-01, 1.42703460e-10,
        6.73729146e-17, 1.28637279e-08, 3.16502771e-08, 2.46536054e-14,
        1.15961338e-10, 7.55421454e-19],
       [5.60639535e-09, 9.99642730e-01, 1.32333284e-04, 2.52610647e-07,
        1.57154159e-06, 1.51096913e-08, 9.58559326e-07, 1.83527300e-04,
        3.86024112e-05, 1.00909245e-08],
       [9.99941230e-01, 3.68927328e-12, 4.38338429e-05, 1.02480013e-09,
        3.84527752e-08, 1.15736176e-08, 3.21393912e-07, 1.52054167e-06,
        2.69515765e-09, 1.31156594e-05],
       [5.06234016e-07, 4.74564669e-13, 3.75828591e-07, 5.23128318e-10,
        9.63448048e-01, 5.77253090e-09, 1.09736881e-07, 2.25496726e-04,
        5.69125291e-07, 3.63250226e-02]], dtype=float32)>
```

```
[49] 1 predictions = model(x_test[:1]).numpy()
      2 sm = tf.nn.softmax(predictions)
      3 sm.numpy()
```

```
↳ array([[1.7116725e-09, 2.8062591e-12, 2.0082256e-05, 3.7206391e-05,
        5.7199572e-15, 1.4641381e-09, 1.1969322e-19, 9.9994266e-01,
        2.8991176e-09, 8.7571451e-08]], dtype=float32)
```

로그 오즈(log-odds)

- 어떤 이벤트가 일어날 가능성의 로그

- 이벤트가 이진 확률을 의미하는 경우의 가능성은 성공 확률(p) 대 실패 확률(1-p)의 비율을 의미
- 특정 이벤트의 성공 확률이 90%, 실패 확률이 10%라고 가정

$$\text{가능성} = \frac{p}{(1-p)} = \frac{.9}{.1} = 9$$

$$\text{로그 오즈} = \ln(9) = 2.2$$

- 시그모이드 함수의 역함수

TF2 로 직접 훈련시키는
MNIST 손글씨 구현

MNIST 데이터 로드

```
[1] 1 import tensorflow as tf
    2
    3 from tensorflow.keras.layers import Dense, Flatten
    4 from tensorflow.keras import Model
    5
    6 print(tf.__version__, tf.executing_eagerly())
```

↪ 2.2.0 True

```
[2] 1 #####
    2 # ① 문제와 정답 데이터 지정
    3 mnist = tf.keras.datasets.mnist
    4
    5 (x_train, y_train), (x_test, y_test) = mnist.load_data()
    6 x_train, x_test = x_train / 255.0, x_test / 255.0
```


DataSet과 모델 생성

```
[12] 1 #####
      2 # tf.data를 사용해 훈련 데이터와 검증 데이터를 나눔
      3 train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(32)
      4 test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

```
[ ] 1 #####
     2 # ② 인공신경망 모델 생성(구성)
     3 model = tf.keras.models.Sequential([
     4     tf.keras.layers.Flatten(input_shape=(28, 28)),
     5     tf.keras.layers.Dense(128, activation='relu'),
     6     tf.keras.layers.Dropout(0.2),
     7     tf.keras.layers.Dense(10)
     8 ])
```

손실 함수와 최적화 방법

```
[4] 1 #####
2 # ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
3 loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
4 optimizer = tf.keras.optimizers.Adam()
5
6 #####
7 # ④ 생성된 모델로 훈련 데이터 학습
8 # 손실과 정확도를 위한 객체
9 train_loss = tf.keras.metrics.Mean(name='train_loss')
10 train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')
11
12 test_loss = tf.keras.metrics.Mean(name='test_loss')
13 test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

학습 과정과 테스트 과정의 함수

```
[5] 1 #####
2 # GradientTape()로 모델을 훈련하는 함수
3 @tf.function
4 def train_step(images, labels):
5     with tf.GradientTape() as tape:
6         # training=True is only needed if there are layers with different
7         # behavior during training versus inference (e.g. Dropout).
8         predictions = model(images, training=True)
9         loss = loss_object(labels, predictions)
10        gradients = tape.gradient(loss, model.trainable_variables)
11        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
12
13    train_loss(loss)
14    train_accuracy(labels, predictions)
```

손실 값을 최소화하는 패러미터
를 계산해 최적화 과정을 수행

```
[6] 1 # 모델을 테스트 또는 검증하기 위한 함수
2 @tf.function
3 def test_step(images, labels):
4     # training=False is only needed if there are layers with different
5     # behavior during training versus inference (e.g. Dropout).
6     predictions = model(images, training=False)
7     t_loss = loss_object(labels, predictions)
8
9     test_loss(t_loss)
10    test_accuracy(labels, predictions)
```

5회 학습

```

[9] 1 # 실제 전체 훈련 데이터의 5회를 훈련
    2 EPOCHS = 5
    3
    4 for epoch in range(EPOCHS):
    5     # Reset the metrics at the start of the next epoch
    6     train_loss.reset_states()
    7     train_accuracy.reset_states()
    8     test_loss.reset_states()
    9     test_accuracy.reset_states()
    10
    11     for images, labels in train_ds:
    12         train_step(images, labels)
    13
    14     for test_images, test_labels in test_ds:
    15         test_step(test_images, test_labels)
    16
    17     # template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}'
    18     template = '{} Epoch 손실={:.4f}, 정확도={:.4f}, 테스트손실={:.4f}, 테스트검정={:.4f}'
    19     print(template.format(epoch + 1,
    20                           train_loss.result(),
    21                           train_accuracy.result() * 100,
    22                           test_loss.result(),
    23                           test_accuracy.result() * 100))

```

➞ 1 Epoch 손실=0.0666, 정확도=97.8383, 테스트손실=0.0759, 테스트검정=97.7900
 2 Epoch 손실=0.0588, 정확도=98.1417, 테스트손실=0.0728, 테스트검정=97.9200
 3 Epoch 손실=0.0536, 정확도=98.2450, 테스트손실=0.0734, 테스트검정=97.8200
 4 Epoch 손실=0.0485, 정확도=98.4200, 테스트손실=0.0788, 테스트검정=97.7900
 5 Epoch 손실=0.0454, 정확도=98.5100, 테스트손실=0.0803, 테스트검정=97.8200

테스트 성능 평가

```
[10] 1 #####  
      2 # ⑤ 테스트 데이터로 성능 평가  
      3 test_step(x_test, y_test)  
      4 print('테스트 데이터 정확도(Accuracy): %.4f' % (test_accuracy.result() * 100))
```

➡ 테스트 데이터 정확도(Accuracy): 97.8200