KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

스마트교통 빅데이터 분석

CNN 실습 (FMNIST 데이터)

2022.8.30

이 홍 석 (hsyi@kisti.re.kr)

학습 목표

- 학습 내용
  - ✓CNN 알고리즘의 기본 개념을 이해한다.
  - ✓케라스(텐서플로우)를 이용한 CNN 기본 모델을 구현한다.
  - ✓텐서보드(Tensorboard)를 이용한 결과 분석을 학습한다.

tf.keras.losses 클래스 소개

`class BinaryCrossentropy`: Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between `y_true` and `y_pred`.

`class Hinge`: Computes the hinge loss between `y_true` and `y_pred`.

`class Huber`: Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss`: Loss base class.

`class MeanAbsoluteError`: Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError`: Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError`: Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError`: Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson`: Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction`: Types of loss reduction.

`class SparseCategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge`: Computes the squared hinge loss between `y_true` and `y_pred`.

keras와 tensorflow 2 적용

- 이번 장의 목표
  - ✓fashion NMIST dataset
  - ✓keras.datasets에 있는 fashion NMIST 데이터 세트의 이미지 분류에 대하여 tensorflow2.+이상을 적용해 보자

```
In [21]: import tensorflow as tf
         from tensorflow import keras

In [22]: tf.__version__
Out[22]: '2.0.0'

In [23]: keras.__version__
Out[23]: '2.2.4-tf'
```

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

패션 NMINST 데이터(1)

- 패션 NMINST는 총 60,000개 훈련 데이터, 픽셀 크기는 28x28
- 테스트 데이터 개수는 10,000개

```
In [18]: X_train_full.shape
Out[18]: (60000, 28, 28)
```

```
In [29]: X_valid.shape
Out[29]: (5000, 28, 28)
```

```
In [21]: X_test.shape
Out[21]: (10000, 28, 28)
```

```
In [30]: X_test.shape
Out[30]: (10000, 28, 28)
```

```
In [19]: X_train_full.dtype
Out[19]: dtype('uint8')
```

- 검증 데이터 5000개 사용
- 훈련은 55,000개 모두 0~1로 스케일

```
In [20]: X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
         y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
         X_test = X_test / 255.
```

패션 NMINST 데이터(2)

```
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```

```
plt.imshow(X_train[2], cmap="binary")
plt.axis('off')
plt.show()
```

```
In [23]: y_train
Out[23]: array([4, 0, 7, ..., 3, 0, 5], dtype=uint8)
```

```
In [24]: class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
                       "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

So the first image in the training set is a coat:

```
In [28]: class_names[y_train[0]]
Out[28]: 'Coat'
```

패션 NMINST 데이터(3)


```
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
```

케라스 Sequential 모델 (1)

- ## 텐서플로 2.0의 권장 사항

  1)작은 함수로 코드를 리팩토링 하자

  - 텐서플로 1.x의 일반적인 사용패턴은 모든 연산을 결합하여 준비한 다음
  - session.run()을 텐서를 평가 함
  - 텐서플로 2.x에서는 필요할 때 호출할 수 있는 작은 함수로 코드를 리팩토링 함

  2) 케라스 층과 모델을 사용해 변수를 관리하자

  - 케라스 모델과 층은 재귀적으로 의존하는 모든 변수를 수집하여 variables와 trainable_variables 속성으로 제공한다. 지역 범위로 변수 관리가 쉽다.

- ## 케라스 사용

```
import tensorflow as tf
from tensorflow import keras
```

- ## Sequential 모델 만들기

  ✓케라스에서는 층(layer)을 조합하여 모델을 만듬. 모델은 층의 그래프이다.

  - 가장 흔한 모델 구조는 층을 차례대로 쌓는 tf.keras.Sequential 모델이다.

케라스 모델 구성 및 층 설정 (3)

05. 케라스 사용하기

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

층을 차례로 쌓는 모델을 만들자

input을 일렬화 한다. 1개의 연속 메모리 784개 1차원 배열로 변환. 학습되는 가중치는 없고 데이터를 변환하기만 한다.

keras.layers.Dense 층이 연속적으로 연결되어 완전연결(fully-connected layer) 층이라고 부름.

첫 번째 Dense 층은 300개 노드(뉴런)을 가진다.
두 번째 층은 100개, 마지막 층은 10개의 뉴런이 소프트맥스 활성함수를 통해서 확률적으로 반환함. 전체 확률합은 당연히 1이다.

케라스 모델 (4)

```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

케라스는 고수준의 API를 모듈 방식으로 제공하며, 여러 다른 백엔드 엔진들을 연결함. clear 세션은 현재 케라스 그래프를 없애고, 새로운 그래프를 만들자

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

케라스 모델 summary (5)

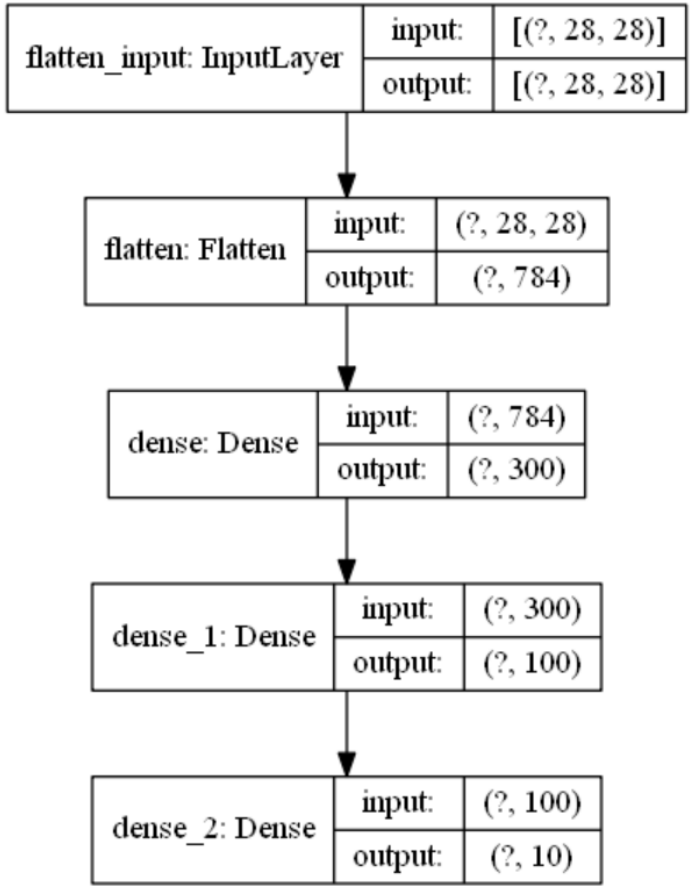

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

케라스 모델 구조 (6)

pydot과 graphviz가 설치가 안되어 있다면 ! pip install pydot graphviz

```
keras.utils.plot_model(model, "my_mnist_model.png", show_shapes=True)
```



케라스 Sequential 모델 (7)

In [43]:
```
hidden1 = model.layers[1]
hidden1.name
```

Out[43]:
```
'dense'
```

In [44]:
```
model.get_layer(hidden1.name) is hidden1
```

Out[44]:
```
True
```

In [45]:
```
weights, biases = hidden1.get_weights()
```

In [46]:
```
weights
```

Out[46]:
```
array([[ 0.02448617, -0.00877795, -0.02189048, ..., -0.02766046,
         0.03859074, -0.06889391],
       [ 0.00476504, -0.03105379, -0.0586676 , ...,  0.00602964,
        -0.02763776, -0.04165364],
       [-0.06189284, -0.06901957,  0.07102345, ..., -0.04238207,
         0.07121518, -0.07331658],
       ...,
       [-0.03048757,  0.02155137, -0.05400612, ..., -0.00113463,
         0.00228987,  0.05581069],
       [ 0.07061854, -0.06960931,  0.07038955, ..., -0.00384101,
         0.00034875,  0.02878492],
       [-0.06022581,  0.01577859, -0.02585464, ..., -0.00527829,
         0.00272203, -0.06793761]], dtype=float32)
```

케라스 Sequential 모델 (9)


```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy])
```

```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
Epoch 27/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2362 - accuracy: 0.9145 - val_loss: 0.3002 - val_accuracy: 0.8890
Epoch 28/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2331 - accuracy: 0.9157 - val_loss: 0.3021 - val_accuracy: 0.8926
Epoch 29/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2289 - accuracy: 0.9182 - val_loss: 0.2985 - val_accuracy: 0.8924
Epoch 30/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2254 - accuracy: 0.9181 - val_loss: 0.3329 - val_accuracy: 0.8806
```

케라스 Sequential 모델 (10)

In [52]:
```
history.params
```

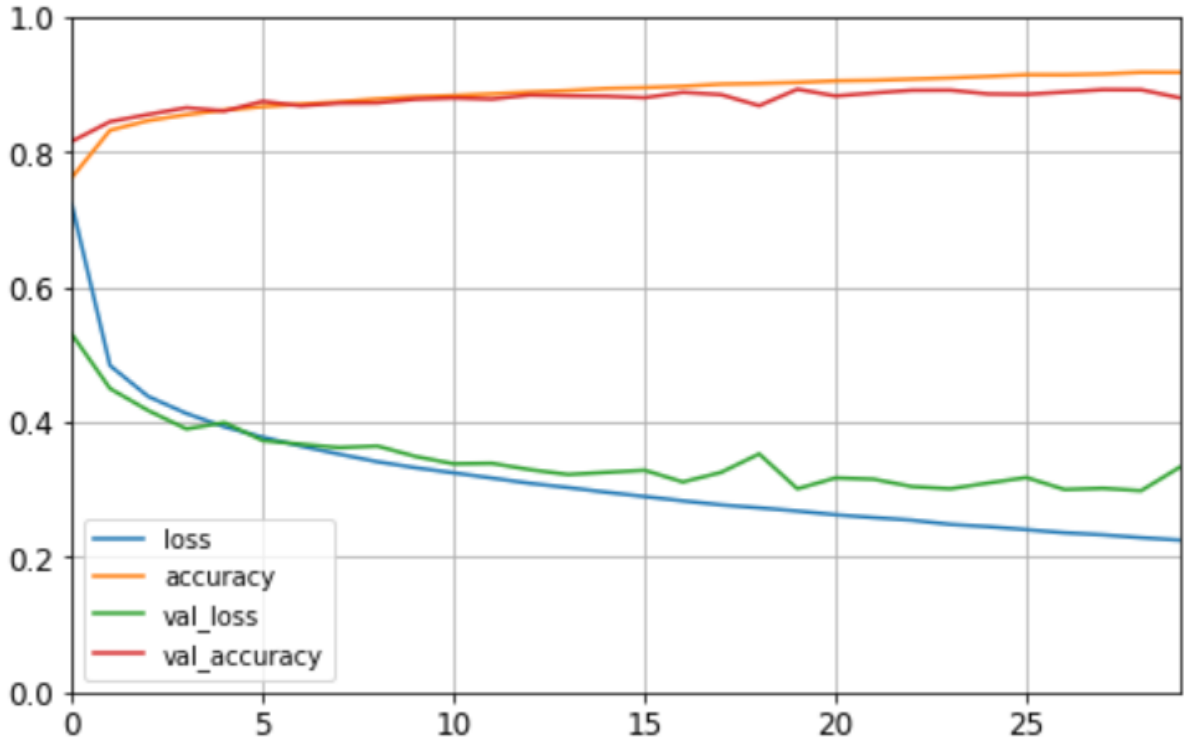
Out[52]:
```
{'batch_size': 32,
 'epochs': 30,
 'steps': 1719,
 'samples': 55000,
 'verbose': 0,
 'do_validation': True,
 'metrics': ['loss', 'accuracy', 'val_loss', 'val_accuracy']}
```

In [53]:
```
print(history.epoch)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

케라스 Sequential 모델 (11)


```
import pandas as pd
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



케라스 Sequential 모델 (12)


```
In [60]: model.evaluate(X_test, y_test)
```

```
39us/sample - loss: 0.3276 - accuracy: 0.8664

[0.3684497040987015, 0.8664]
```

```
In [61]: X_new = X_test[:3]
         y_proba = model.predict(X_new)
         y_proba.round(2)
```

```
Out[61]: array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
                [0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
                [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
               dtype=float32)
```

케라스 Sequential 모델 (13)


```
In [69]: y_pred = model.predict_classes(X_new);y_pred
Out[69]: array([9, 2, 1], dtype=int64)
```

```
In [63]: np.array(class_names)[y_pred]
Out[63]: array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

```
In [68]: y_new = y_test[:3];y_new
Out[68]: array([9, 2, 1], dtype=uint8)
```

케라스 Sequential 모델 (14)

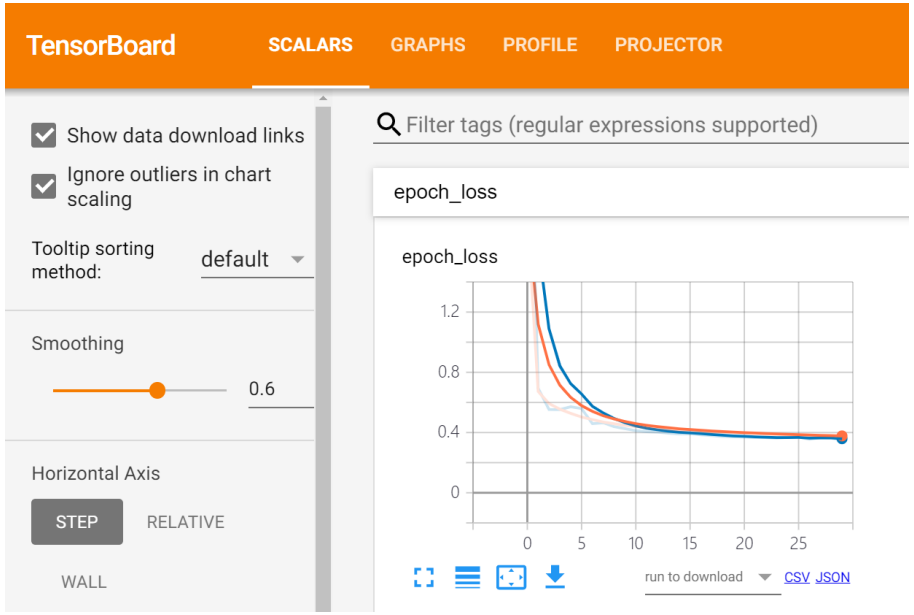
In [66]:

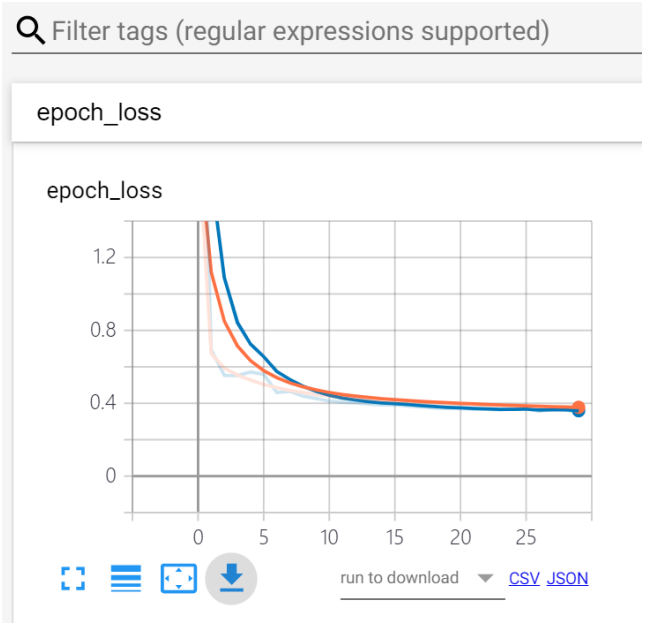
```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)

plt.show()
```

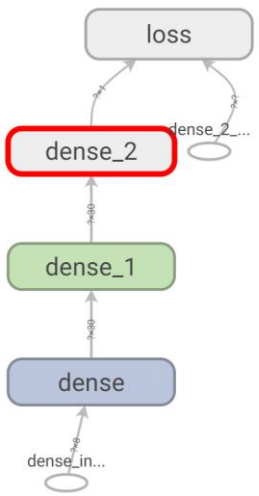
09. tensorboard()

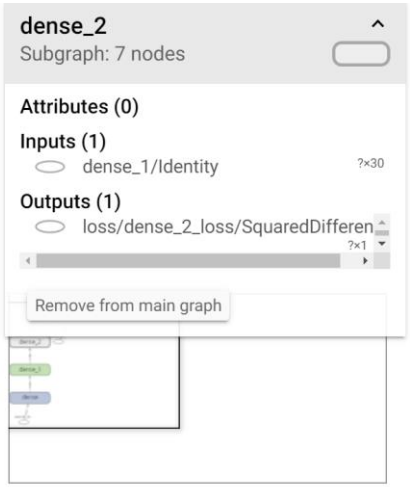
- 텐서보드 실행
  - ✓$ tensorboard --logdir=./my_logs --port=6006
- 웹브라우저 (MS 엣지, 크롬) 주소 입력 localhost:6006

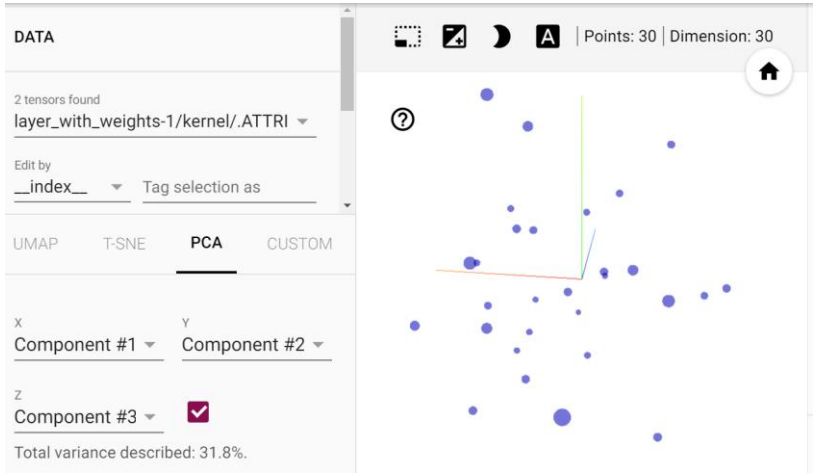




tensorboard()







2022 TRUST KISTI

Korea Institute of Science
and Technology Information