TF2.0 신경망 만들기

- fashion2.0 데이터 셋을 이용한 신경망 만들기
- 개발 환경 : tf 버전 2.x (2020/12)

학습내용

- 성능을 업그레이드한다.

In [1]:

```
import tensorflow as tf
```

In [2]:

```
print(tf.__version__)
```

2.4.0

In [3]:

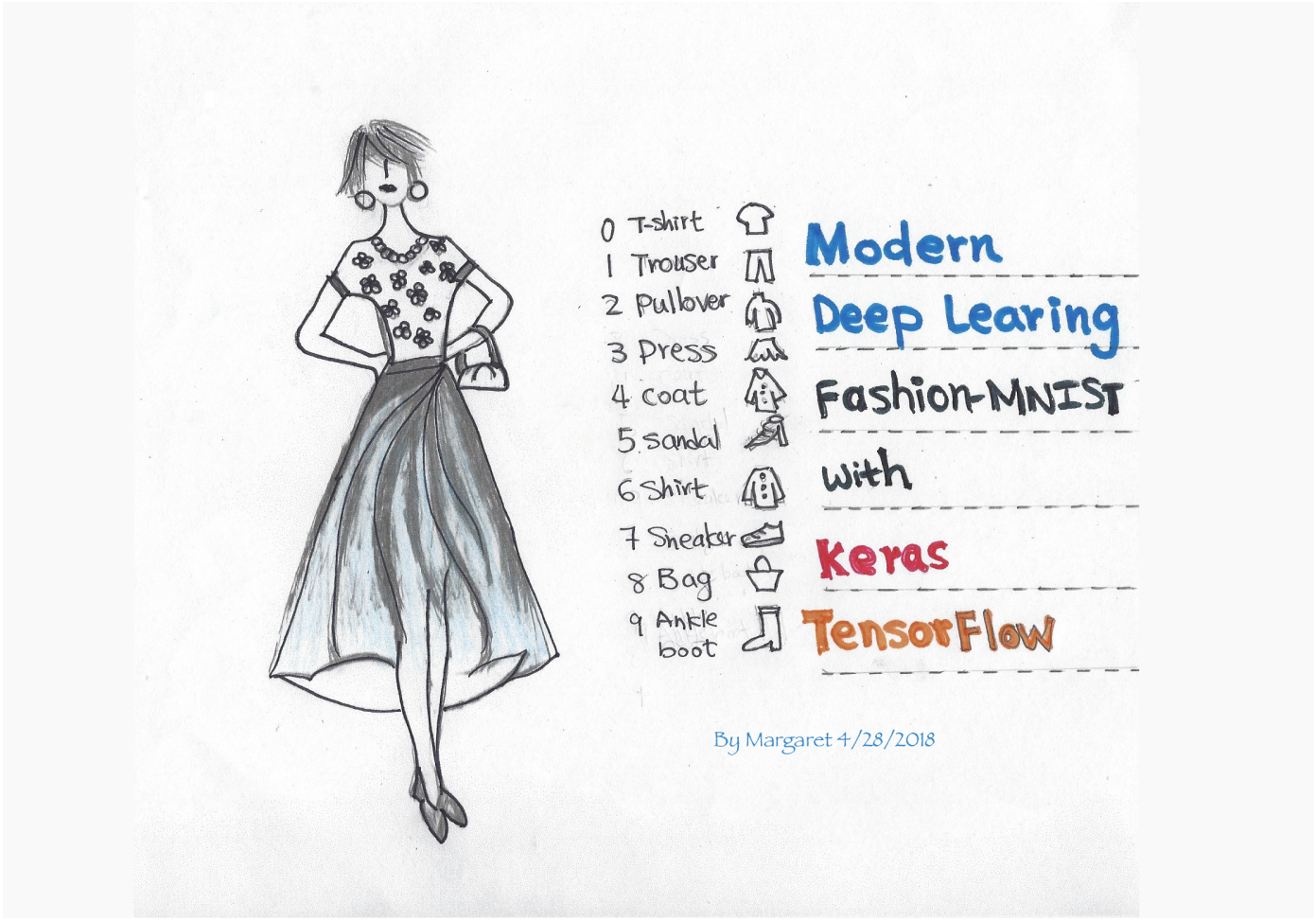
```
# !pip install -q tensorflow-gpu==2.0.0-rc1
```

In [4]:

```
# tensorflow와 tf.keras를 임포트합니다
import tensorflow as tf
from tensorflow import keras

# 헬퍼(helper) 라이브러리를 임포트합니다
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
print(np.__version__)
```

2.4.0
1.19.4

Fashion MNIST DataSet



In [57]:

```
fashion_mnist = keras.datasets.fashion_mnist


# 4개의 데이터 셋 반환(numpy 배열)
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

In [58]:

```
print("학습용 데이터 : x: {}, y:{}".format(X_train.shape, y_train.shape) )
print("테스트 데이터 : x: {}, y:{}".format(X_test.shape, y_test.shape) )
```

```
학습용 데이터 : x: (60000, 28, 28), y:(60000,)
테스트 데이터 : x: (10000, 28, 28), y:(10000,)
```

In [59]:

```
# 훈련셋과 검증셋 분리
X_val = X_train[50000:]
y_val = y_train[50000:]
X_train = X_train[:50000]
y_train = y_train[:50000]

# 훈련셋, 검증셋 고르기
train_rand_idxs = np.random.choice(50000, 25000)
val_rand_idxs = np.random.choice(10000, 5000)

X_train = X_train[train_rand_idxs]
y_train = y_train[train_rand_idxs]
X_val = X_val[val_rand_idxs]
y_val = y_val[val_rand_idxs]
```

In [60]:

```
print("학습용 데이터 : x: {}, y:{}".format(X_train.shape, y_train.shape) )
print("검증용 데이터 : x: {}, y:{}".format(X_val.shape, y_val.shape) )
print("테스트 데이터 : x: {}, y:{}".format(X_test.shape, y_test.shape) )
```

```
학습용 데이터 : x: (25000, 28, 28), y:(25000,)
검증용 데이터 : x: (5000, 28, 28), y:(5000,)
테스트 데이터 : x: (10000, 28, 28), y:(10000,)
```

In [61]:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

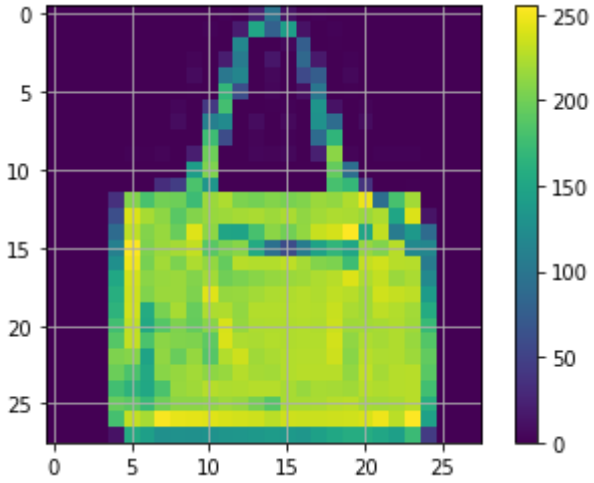
In [62]:

```
print("학습용 데이터의 레이블 ", np.unique(y_train) )
```

```
학습용 데이터의 레이블  [0 1 2 3 4 5 6 7 8 9]
```

In [63]:

```
plt.figure()
plt.imshow(X_train[0])   # 첫번째 이미지 데이터
plt.colorbar()     # 색깔 표시바
plt.grid(True)   # grid 선
plt.show()
```



In [64]:

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

이미지 확인

In [65]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)  # 그래프의 표시 위치
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)  # 그리드선
    plt.imshow(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

데이터 나누기

In [66]:

```
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [72]:

```
X_train_len = X_train.shape[0]
X_val_len   = X_val.shape[0]
X_test_len  = X_test.shape[0]
X_train_len, X_val_len, X_test_len
```

Out[72]:

```
(25000, 5000, 10000)
```

In [74]:

```
X_train = X_train.reshape(X_train_len, 784).astype('float32') / 255.0
X_val = X_val.reshape(X_val_len, 784).astype('float32') / 255.0
X_test = X_test.reshape(X_test_len, 784).astype('float32') / 255.0

# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

In [75]:

```
print(X_train.shape, y_train.shape)
print(X_val.shape, y_val.shape)
print(X_test.shape, y_test.shape)
```

```
(25000, 784) (25000, 10)
(5000, 784) (5000, 10)
(10000, 784) (10000, 10)
```

모델 생성

In [85]:

```
model = keras.Sequential()
model.add( keras.layers.Flatten() )
model.add( keras.layers.Dense(128, activation='relu') )
model.add( keras.layers.Dense(10, activation='softmax') )

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

조기종료

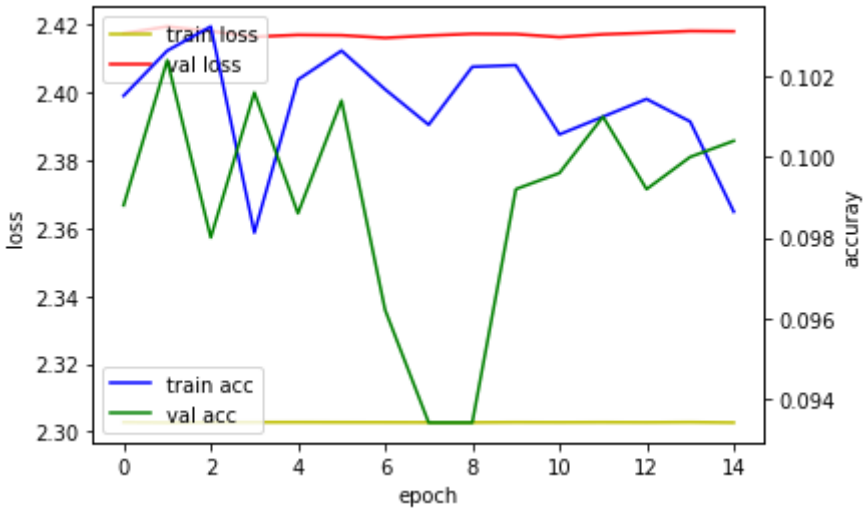
In [87]:

```
from keras.callbacks import EarlyStopping
# early_stopping = EarlyStopping() # 조기종료 콜백함수 정의
early_stopping = EarlyStopping(patience = 20) # 조기종료 콜백함수 정의
hist = model.fit(X_train, y_train,
                 epochs=15,
                 batch_size=10,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping])
```

```
Epoch 1/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1015 - val_loss: 2.4175 - val_accuracy: 0.0988
Epoch 2/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3027 - accuracy:
0.1026 - val_loss: 2.4195 - val_accuracy: 0.1024
Epoch 3/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1032 - val_loss: 2.4181 - val_accuracy: 0.0980
Epoch 4/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.0981 - val_loss: 2.4164 - val_accuracy: 0.1016
Epoch 5/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1019 - val_loss: 2.4170 - val_accuracy: 0.0986
Epoch 6/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1026 - val_loss: 2.4169 - val_accuracy: 0.1014
Epoch 7/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1017 - val_loss: 2.4161 - val_accuracy: 0.0962
Epoch 8/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1008 - val_loss: 2.4168 - val_accuracy: 0.0934
Epoch 9/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3027 - accuracy:
0.1022 - val_loss: 2.4173 - val_accuracy: 0.0934
Epoch 10/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1023 - val_loss: 2.4172 - val_accuracy: 0.0992
Epoch 11/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1006 - val_loss: 2.4164 - val_accuracy: 0.0996
Epoch 12/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1010 - val_loss: 2.4172 - val_accuracy: 0.1010
Epoch 13/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1014 - val_loss: 2.4176 - val_accuracy: 0.0992
Epoch 14/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3028 - accuracy:
0.1009 - val_loss: 2.4181 - val_accuracy: 0.1000
Epoch 15/15
2500/2500 [==============================] - 4s 2ms/step - loss: 2.3027 - accuracy:
0.0986 - val_loss: 2.4181 - val_accuracy: 0.1004
```

In [88]:

```
# 5. 모델 학습 과정 표시하기
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')

acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



In [90]:

```
test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)

print('₩n테스트 정확도:', test_acc)
```

```
313/313 - 1s - loss: 2.3028 - accuracy: 0.0959

테스트 정확도: 0.09589999914169312
```

성능 개선시키기

- adam, DropOut

In [91]:

```
from tensorflow.keras.layers import Dense, Dropout, Flatten
```

In [92]:

```
model = keras.Sequential()
model.add( Flatten() )
model.add( Dense(128, activation='relu') )
model.add( Dropout(0.2) )                    # Dropout적용
model.add( Dense(128, activation='relu') )
model.add( Dropout(0.2) )
model.add( Dense(10, activation='softmax') )

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

In [93]:

```
hist = model.fit(X_train, y_train,
                 epochs=15,
                 batch_size=10,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping])
```

```
Epoch 1/15
2500/2500 [==============================] - 6s 2ms/step - loss: 2.3030 - accuracy:
0.1036 - val_loss: 7.4411 - val_accuracy: 0.1860
Epoch 2/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3024 - accuracy:
0.1065 - val_loss: 7.7366 - val_accuracy: 0.1732
Epoch 3/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3028 - accuracy:
0.0972 - val_loss: 7.7517 - val_accuracy: 0.1728
Epoch 4/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3028 - accuracy:
0.1000 - val_loss: 7.7553 - val_accuracy: 0.1726
Epoch 5/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3028 - accuracy:
0.1038 - val_loss: 7.7540 - val_accuracy: 0.1726
Epoch 6/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3027 - accuracy:
0.1021 - val_loss: 7.7572 - val_accuracy: 0.1730
Epoch 7/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3027 - accuracy:
0.0992 - val_loss: 7.7418 - val_accuracy: 0.1728
Epoch 8/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.0973 - val_loss: 7.7467 - val_accuracy: 0.1726
Epoch 9/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.1003 - val_loss: 7.7573 - val_accuracy: 0.1728
Epoch 10/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.1050 - val_loss: 7.7398 - val_accuracy: 0.1728
Epoch 11/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.1025 - val_loss: 7.7688 - val_accuracy: 0.1726
Epoch 12/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.1040 - val_loss: 7.7385 - val_accuracy: 0.1728
Epoch 13/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3025 - accuracy:
0.1061 - val_loss: 7.7660 - val_accuracy: 0.1726
Epoch 14/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3028 - accuracy:
0.1009 - val_loss: 7.7346 - val_accuracy: 0.1728
Epoch 15/15
2500/2500 [==============================] - 5s 2ms/step - loss: 2.3026 - accuracy:
0.0954 - val_loss: 7.7589 - val_accuracy: 0.1728
```

성능 개선시키기

- CNN, 데이터 키우기

In [109]:

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
```

In [122]:

```
# 4개의 데이터 셋 반환(numpy 배열)
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# 훈련셋과 검증셋 분리
X_val = X_train[50000:]
y_val = y_train[50000:]
X_train = X_train[:50000]
y_train = y_train[:50000]

# 훈련셋, 검증셋 고르기
train_rand_idxs = np.random.choice(50000, 50000)
val_rand_idxs = np.random.choice(10000, 10000)

X_train = X_train[train_rand_idxs]
y_train = y_train[train_rand_idxs]
X_val = X_val[val_rand_idxs]
y_val = y_val[val_rand_idxs]

X_train.shape, X_val.shape, X_test.shape
```

Out[122]:

```
((50000, 28, 28), (10000, 28, 28), (10000, 28, 28))
```

In [123]:

```
X_train = X_train.reshape( (-1, 28, 28, 1) )
X_train = X_train.astype('float32') / 255

X_val = X_val.reshape( (-1, 28, 28, 1) )
X_val = X_val.astype('float32') / 255

X_test = X_test.reshape( (-1, 28, 28, 1) )
X_test = X_test.astype('float32') / 255

y_train = to_categorical(y_train)
y_val =  to_categorical(y_val)
y_test = to_categorical(y_test)
```

In [124]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D( pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D( pool_size=(2, 2) ))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

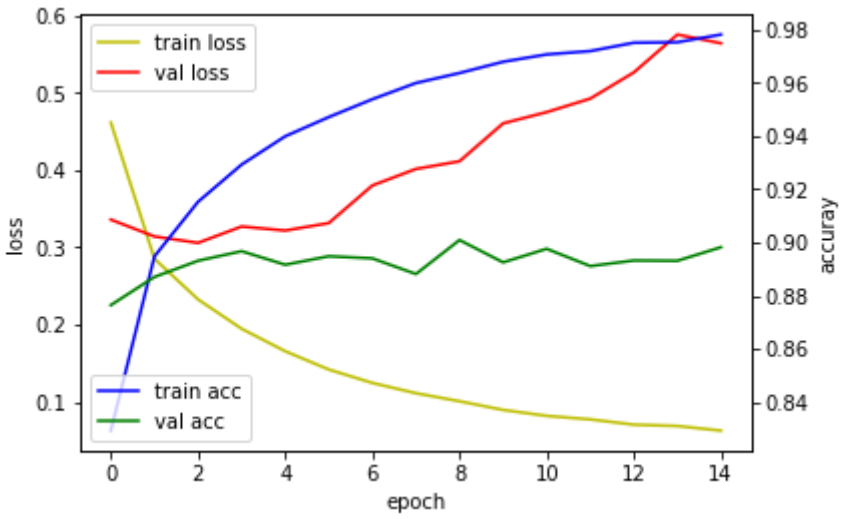
In [125]:

```
%%time

model.compile(optimizer='adam',
              loss='categorical_crossentropy',  # sparse_categorical_crossentropy
              metrics=['accuracy'])

hist = model.fit(X_train, y_train,
                 epochs=15,
                 batch_size=10,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping])
```

```
Epoch 1/15
5000/5000 [==============================] - 52s 10ms/step - loss: 0.6417 - accurac
y: 0.7597 - val_loss: 0.3361 - val_accuracy: 0.8764
Epoch 2/15
5000/5000 [==============================] - 49s 10ms/step - loss: 0.3002 - accurac
y: 0.8900 - val_loss: 0.3142 - val_accuracy: 0.8870
Epoch 3/15
5000/5000 [==============================] - 49s 10ms/step - loss: 0.2411 - accurac
y: 0.9113 - val_loss: 0.3059 - val_accuracy: 0.8931
Epoch 4/15
5000/5000 [==============================] - 49s 10ms/step - loss: 0.1954 - accurac
y: 0.9291 - val_loss: 0.3271 - val_accuracy: 0.8967
Epoch 5/15
5000/5000 [==============================] - 49s 10ms/step - loss: 0.1586 - accurac
y: 0.9427 - val_loss: 0.3219 - val_accuracy: 0.8916
Epoch 6/15
5000/5000 [==============================] - 50s 10ms/step - loss: 0.1363 - accurac
y: 0.9481 - val_loss: 0.3315 - val_accuracy: 0.8948
Epoch 7/15
5000/5000 [==============================] - 50s 10ms/step - loss: 0.1230 - accurac
y: 0.9543 - val_loss: 0.3803 - val_accuracy: 0.8940
Epoch 8/15
5000/5000 [==============================] - 50s 10ms/step - loss: 0.1065 - accurac
y: 0.9620 - val_loss: 0.4017 - val_accuracy: 0.8881
Epoch 9/15
5000/5000 [==============================] - 50s 10ms/step - loss: 0.0946 - accurac
y: 0.9652 - val_loss: 0.4116 - val_accuracy: 0.9009
Epoch 10/15
5000/5000 [==============================] - 51s 10ms/step - loss: 0.0847 - accurac
y: 0.9689 - val_loss: 0.4608 - val_accuracy: 0.8925
Epoch 11/15
5000/5000 [==============================] - 50s 10ms/step - loss: 0.0761 - accurac
y: 0.9719 - val_loss: 0.4753 - val_accuracy: 0.8976
Epoch 12/15
5000/5000 [==============================] - 54s 11ms/step - loss: 0.0720 - accurac
y: 0.9738 - val_loss: 0.4928 - val_accuracy: 0.8911
Epoch 13/15
5000/5000 [==============================] - 48s 10ms/step - loss: 0.0636 - accurac
y: 0.9774 - val_loss: 0.5268 - val_accuracy: 0.8932
Epoch 14/15
5000/5000 [==============================] - 48s 10ms/step - loss: 0.0632 - accurac
y: 0.9767 - val_loss: 0.5758 - val_accuracy: 0.8931
Epoch 15/15
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0514 - accurac
y: 0.9818 - val_loss: 0.5646 - val_accuracy: 0.8982
```

In [127]:

```
# 5. 모델 학습 과정 표시하기
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')

acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



배치 정규화, Depthwise

In [131]:

```
from tensorflow.keras.layers import BatchNormalization, SeparableConv2D
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
```

In [138]:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())

model.add(SeparableConv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())

model.add(SeparableConv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
Wall time: 175 ms
```

In [139]:

```
%%time

opt = Adam(lr=0.0001, decay=1e-5)
early_stopping = EarlyStopping(patience=5)

model.compile(optimizer=opt,
              loss='categorical_crossentropy',  # sparse_categorical_crossentropy
              metrics=['accuracy'])

hist = model.fit(X_train, y_train,
                 epochs=15,
                 batch_size=10,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping])
```

```
Epoch 1/15
5000/5000 [==============================] - 57s 11ms/step - loss: 1.1562 - accurac
y: 0.6146 - val_loss: 0.5657 - val_accuracy: 0.7930
Epoch 2/15
5000/5000 [==============================] - 60s 12ms/step - loss: 0.5794 - accurac
y: 0.7935 - val_loss: 0.4930 - val_accuracy: 0.8181
Epoch 3/15
5000/5000 [==============================] - 60s 12ms/step - loss: 0.5020 - accurac
y: 0.8195 - val_loss: 0.4608 - val_accuracy: 0.8318
Epoch 4/15
5000/5000 [==============================] - 61s 12ms/step - loss: 0.4615 - accurac
y: 0.8340 - val_loss: 0.4373 - val_accuracy: 0.8388
Epoch 5/15
5000/5000 [==============================] - 64s 13ms/step - loss: 0.4432 - accurac
y: 0.8403 - val_loss: 0.4216 - val_accuracy: 0.8456
Epoch 6/15
5000/5000 [==============================] - 78s 16ms/step - loss: 0.4139 - accurac
y: 0.8516 - val_loss: 0.4135 - val_accuracy: 0.8466
Epoch 7/15
5000/5000 [==============================] - 64s 13ms/step - loss: 0.3982 - accurac
y: 0.8572 - val_loss: 0.4105 - val_accuracy: 0.8505
Epoch 8/15
5000/5000 [==============================] - 57s 11ms/step - loss: 0.3800 - accurac
y: 0.8626 - val_loss: 0.3958 - val_accuracy: 0.8528
Epoch 9/15
5000/5000 [==============================] - 56s 11ms/step - loss: 0.3764 - accurac
y: 0.8633 - val_loss: 0.3889 - val_accuracy: 0.8580
Epoch 10/15
5000/5000 [==============================] - 70s 14ms/step - loss: 0.3601 - accurac
y: 0.8697 - val_loss: 0.3882 - val_accuracy: 0.8512
Epoch 11/15
5000/5000 [==============================] - 60s 12ms/step - loss: 0.3538 - accurac
y: 0.8716 - val_loss: 0.3867 - val_accuracy: 0.8556
Epoch 12/15
5000/5000 [==============================] - 61s 12ms/step - loss: 0.3525 - accurac
y: 0.8709 - val_loss: 0.3785 - val_accuracy: 0.8594
Epoch 13/15
5000/5000 [==============================] - 57s 11ms/step - loss: 0.3456 - accurac
y: 0.8719 - val_loss: 0.3740 - val_accuracy: 0.8600
Epoch 14/15
5000/5000 [==============================] - 63s 13ms/step - loss: 0.3337 - accurac
y: 0.8797 - val_loss: 0.3860 - val_accuracy: 0.8538
Epoch 15/15
```

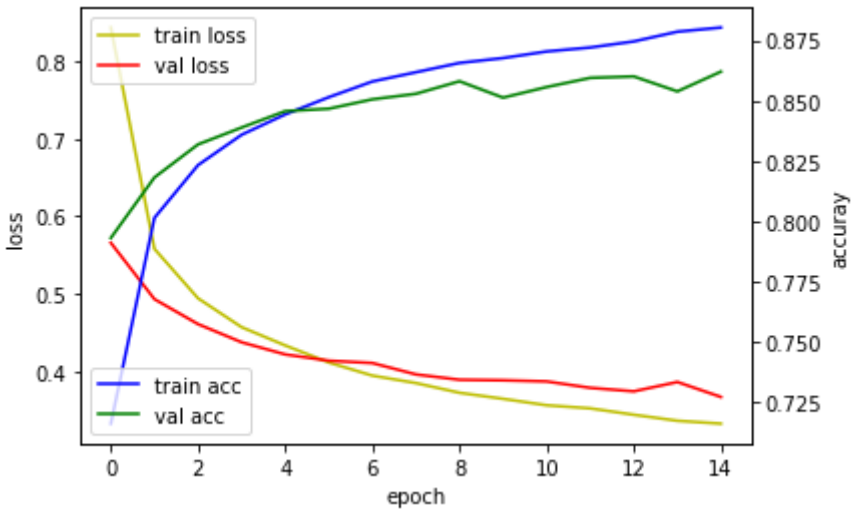
```
5000/5000 [==============================] - 62s 12ms/step - loss: 0.3288 - accurac
y: 0.8813 - val_loss: 0.3667 - val_accuracy: 0.8620
Wall time: 15min 33s
```

In [143]:

```
### 훈련 후, 모델 저장
model.save('fashion_mnist_1.h5')
```

In [145]:

```
# 5. 모델 학습 과정 표시하기
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')

acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



In [146]:

```
# Evaluation on test dataset
test_loss, test_score = model.evaluate(X_test, y_test, batch_size=16)
print("Loss on test set: ", test_loss)
print("Accuracy on test set: ", test_score)
```

```
625/625 [==============================] - 3s 5ms/step - loss: 0.3861 - accuracy: 0.
8620
Loss on test set:  0.38605040311813354
Accuracy on test set:  0.8619999885559082
```

더해보기

- 하이퍼 파라미터 변경
- 전이학습(사전 훈련된 네트워크) 사용해보기
- 세부 파라미터 변경 및 설정
- 이미지 제너레이터를 활용한 데이터 증식
- ...

REF

- fashion 2.0 TF : https://www.tensorflow.org/tutorials/keras/classification (https://www.tensorflow.org/tutorials/keras/classification)
- BatchNormalization layer : https://keras.io/api/layers/normalization_layers/batch_normalization/ (https://keras.io/api/layers/normalization_layers/batch_normalization/)
- SeparableConv2D layer : https://keras.io/api/layers/convolution_layers/separable_convolution2d/ (https://keras.io/api/layers/convolution_layers/separable_convolution2d/)

History

- 2020/12/28 tf 2.x (ver 1.1)