**Coding과제 2**

목차

1. MNIST 모델 소스코드
2. 실행 결과

학번 20140269 이름 고혁훈



* 과제의 목적

Mnist 부분데이터만 가지고 모델을 학습해 본다.

one-hot 형태의 label을 사용해본다.

* 배운 점

부분데이터와 전체데이터를 똑같은 모델로 학습했을 때, 0~6이 0~9보다 쉬운 task임을 알 수 있었다. 클래스의 개수가 많아서 잘 못 맞추는 지 아니면 7~9가 더 어려운 task인지 알기 위해서 4~9도 해보았다. 그랬더니 정확도가 93%에 근접해서 0~6이랑 크게 차이가 나지 않았다. 나머지 조합으로 6개 만들었을 때도 성능이 93% 근처에 머무는 것을 알 수 있었다. 결과적으로 성능이 떨어진 이유가 클래스 개수 때문이라고 추측해볼 수 있다. 이는 클래스가 많아질수록 레이어도 늘어나야 하며, 데이터셋을 구분할 수 있는 모델은 클래스와 레이어가 비례해야 된다고 추측할 수 있다.

* 소스코드

1) 임포트

import numpy as np

import tensorflow as tf

import pandas as pd

2) 데이터 load

mnist = tf.keras.datasets.mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

start = 0 # start point of label

n = 6 # this is for how many labels type that I will use.

3) 데이터 preprocess

filters = np.isin(y\_train,list(range(n))) # filter index for what we will use for labels

# true /false type per each index

# 선형 레이어를 사용할 것이기에 이미지를 늘어 놓는다.

x\_new,y\_new = x\_train[filters].reshape(-1,28\*28),y\_train[filters] #

y\_new -= start # if not, when convert it into one\_hot, I won’t be the vector you expect.

# one hot 형태로 바꾼다

y\_one\_new = tf.one\_hot(y\_new,depth=n)

# 분류하려는 개수에 따라 output이 달라질 예정이기에 n으로 놓는다.

model=tf.keras.models.Sequential([tf.keras.layers.Dense(n,activation='softmax')])

# adam 옵티마이저와 CategoricalCrossentropy 사용했다.(one-hot형태라서)

# sgd 대신에 adam을 쓴 이유는 lr을 튜닝하는 수고를 덜기 위해서이다.

model.compile(optimizer='adam', loss='CategoricalCrossentropy',metrics=["accuracy"])

# xdata, ydata(one-hot), epochs=100

history= model.fit(x\_new,y\_new\_one,epochs=100)

* 실행결과

1. 모델 summary

Model: "sequential\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_2 (Dense) (None, 6) 4710

=================================================================

Total params: 4,710

Trainable params: 4,710

Non-trainable params: 0

1. epoch결과 일부

Epoch 61/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.7282 - accuracy: 0.9467

Epoch 62/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.6288 - accuracy: 0.9489

Epoch 63/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.7350 - accuracy: 0.9479

Epoch 64/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.8395 - accuracy: 0.9462

Epoch 65/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.8048 - accuracy: 0.9464

Epoch 66/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.7079 - accuracy: 0.9479

Epoch 67/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.8147 - accuracy: 0.9465

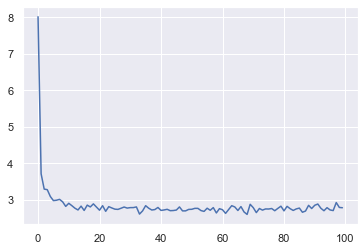
Epoch 68/100

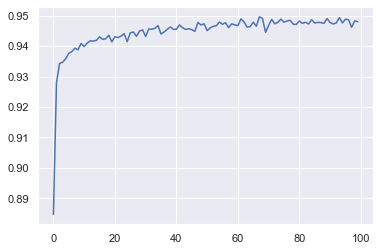
1126/1126 [==============================] - 1s 1ms/step - loss: 2.6684 - accuracy: 0.9496

Epoch 69/100

1126/1126 [==============================] - 1s 1ms/step - loss: 2.6002 - accuracy: 0.9491

1. fit 결과 도표(x축은 epoch, y1은 loss, y2는 accuracy)





1. 예측결과

model.predict(x\_data[:10])

array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.37390135e-36,

0.00000000e+00, 1.00000000e+00],

[1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

1.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

1.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,

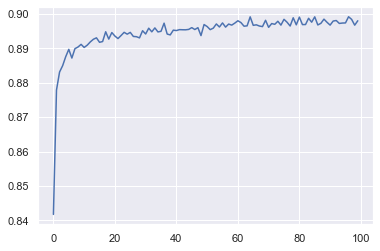
0.00000000e+00, 0.00000000e+00]], dtype=float32)

1. **Accuracy**

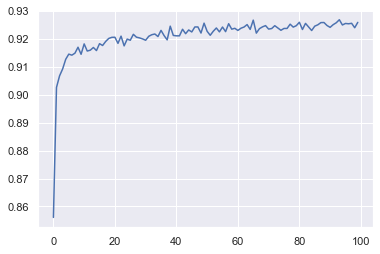
* 최고 accuracy

0.9496349096298218

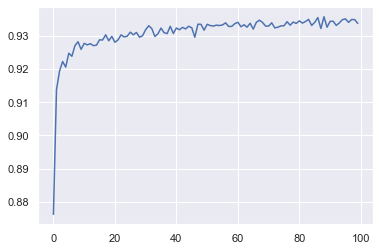
* 실행결과 – 0~9

-

* 실행결과 – 4~9

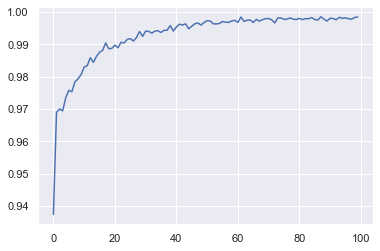


* 실행결과 – 2~7



* 실행결과 – layer 한 개 추가(2~7)

model = tf.keras.models.Sequential([tf.keras.layers.Dense(x\_new.shape[-1] /2 ,activation='relu'),tf.keras.layers.Dense(n,activation='softmax')])



* 실행결과 – layer 한 개 추가(0~10)

Epoch 98/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0818 - accuracy: 0.9851

Epoch 99/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0966 - accuracy: 0.9846

Epoch 100/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.1067 - accuracy: 0.9843