딥러닝

Report_1

컴퓨터공학과

2019305059

이현수

문제 1>

교재 예제 3.7 을 참조하여 균일분포로 난수를 얻은 후 다음 2 가지 형식의 tensor 를 만들어 출력한다.

① shape [8, 3]

```
import tensorflow as tf

rand = tf.random.uniform([8,3],0,1)
print(rand)
```

```
tf.Tensor(
[[0.45779598 0.74105585 0.35280168]
 [0.15629208 0.02582407 0.22230637]
 [0.90625584 0.4588387  0.4914322 ]
 [0.5788727  0.3699106  0.04805267]
 [0.3257321  0.8881749  0.37883353]
 [0.50118446 0.16218722 0.5027509 ]
 [0.06168485 0.01416314 0.71326923]
 [0.5464822  0.6418642  0.6588141 ]], shape=(8, 3), dtype=float32)
```

②rank 3, shape [4, 2, 3]

```
import tensorflow as tf

rand = tf.random.uniform([4,2,3],0,1)
print(rand)
```

```
tf.Tensor(
[[[0.43708205 0.98200095 0.8271593 ]
  [0.32923615 0.8683387  0.83414614]]

 [[0.94342065 0.4735409  0.35782194]
  [0.8040042  0.8935957  0.05787885]]

 [[0.33474207 0.7360755  0.7039399 ]
  [0.44566917 0.6163107  0.9876845 ]]

 [[0.12055421 0.35236144 0.4693203 ]
  [0.3967569  0.99675214 0.4308275 ]]], shape=(4, 2, 3), dtype=float32)
```

문제 2>

①교재 예제 3.27 의 신경망을 모델을 이용하여 AND 신경망을 수행하고 예제 3.29 와 같이 예측결과 값을 출력한다.

```
import tensorflow as tf
import numpy as np

x=np.array([[1,1],[1,0],[0,1],[0,0]])
y=np.array([[1],[0],[0],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
array([[0.921054  ],
       [0.05399007],
       [0.05415323],
       [0.00321528]], dtype=float32)
```

② ①의 예측 출력값과 예제 3.16 의 결과값을 비교하고 그 차이점과 원인을 설명하시오.

예제 3.16 의 결과값이 ①의 예측결과값 보다 출력값 y 에 더 근접하다.

즉 예제 3.16 이 ①보다 오차가 더 적다.

그 원인은 예제 3.16 에서 AND 연산의 수행결과가 이미 충분히 가중치 w1, w2, 편향 b 의 차이가 분명해 특징이 충분하기 때문이다. 그래서 굳이 ①번문제처럼 두개의 레이어(계층)을 만들어서 연산을 하게되면 필요이상으로 과하게 학습되어 오차가 더 커지는 결과가 나온다.

문제 3>

①교재 예제 3.27 에서 입력 x 를 [x1, x2, x3] 3 개 입력으로 바꾸고 출력 y 를 8 개로 바꾼다.(입력이 3 개이니 입력이 8 가지 경우가 되고 따라서 출력도 8 개)

이때, 출력 y 는 임의로 정하시오.(단, 출력 '0'을 3~4 개, '1'을 3~4 개 되도록)

[1,1,1]	[1,1,0]	[1,0,1]	[1,0,0]	[0,1,1]	[0,1,0]	[0,0,1]	[0,0,0]
[1]	[0]	[0]	[1]	[0]	[1]	[1]	[0]

②①에서 입력에 대한 출력을 잘 예측하도록 신경망 모델을 설계하는데 다음표와 같이 여러 경우로 설계한다.

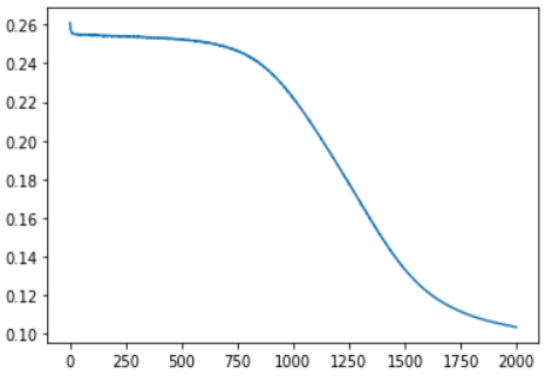
③ ②번의 9 개 case 에 대하여 예제 3.27 과 같은 model.compile 을 사용하고 예제 3.28 과 같이 model.fit 를 실행 한 후 예제 3.34 와 같이 'loss'값을 그래프로 출력한다. 9 개 case 에 대한 코드와 그래프출력을 제출한다.

Case1)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
[ ]  model.predict(x)
```

```
[ ]  plt.plot(history.history['loss'])
```

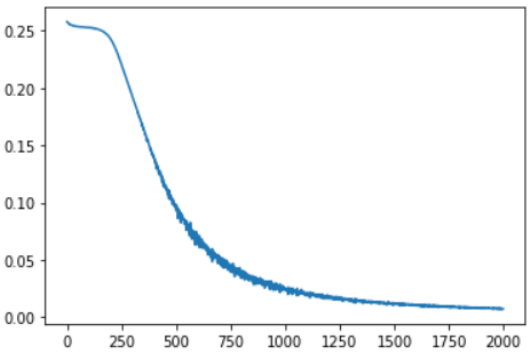


Case2)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='tanh', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
[ ]  model.predict(x)
```

```
[ ]  plt.plot(history.history['loss'])
```

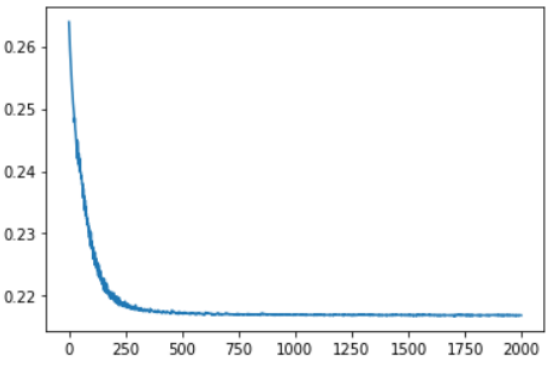


Case3)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
[ ]  model.predict(x)
```

```
[ ]  plt.plot(history.history['loss'])
```

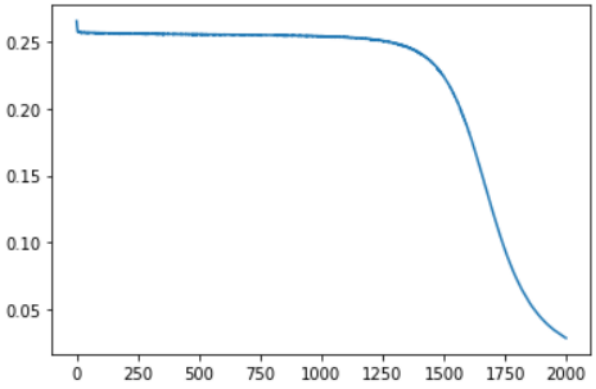


Case4)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=4, activation='sigmoid', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
plt.plot(history.history['loss'])
```

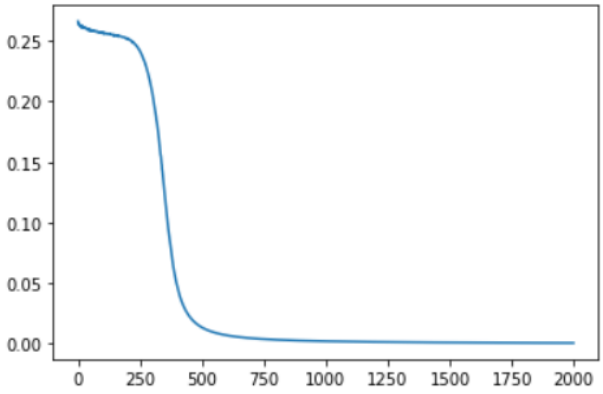


Case5)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=4, activation='tanh', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
plt.plot(history.history['loss'])
```

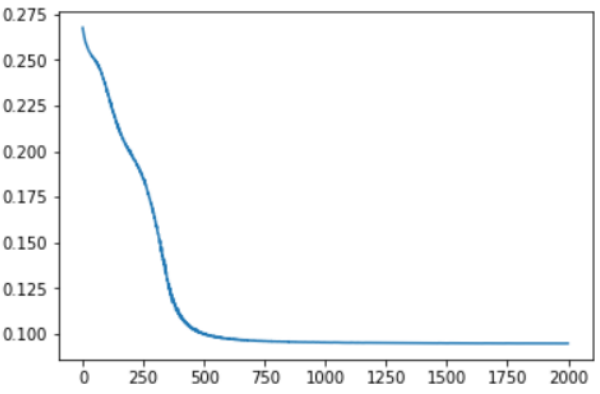


Case6)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=4, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
plt.plot(history.history['loss'])
```

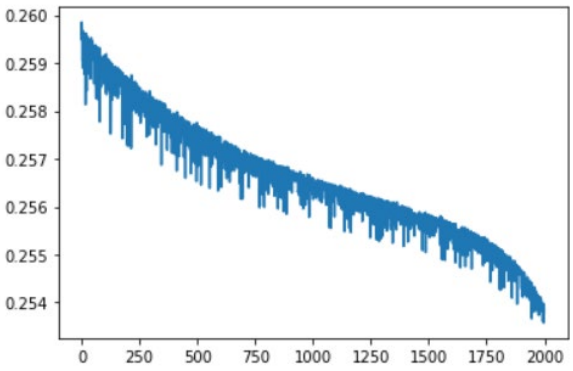


Case7)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=8, activation='sigmoid', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
plt.plot(history.history['loss'])
```

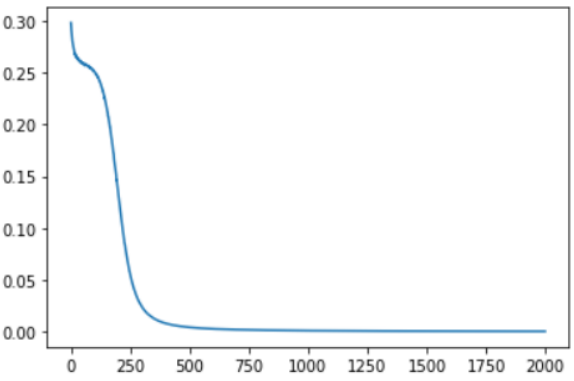


Case8)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=8, activation='tanh', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

```
plt.plot(history.history['loss'])
```

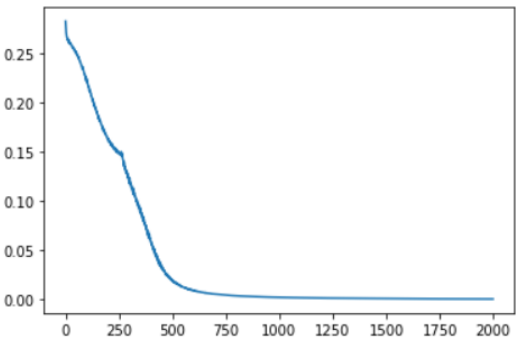


Case9)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0],[0,1,1],[0,1,0],[0,0,1],[0,0,0]])
y=np.array([[1],[0],[0],[1],[0],[1],[1],[0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=8, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')

history = model.fit(x,y,epochs=2000, batch_size=1)
```

```
model.predict(x)
```

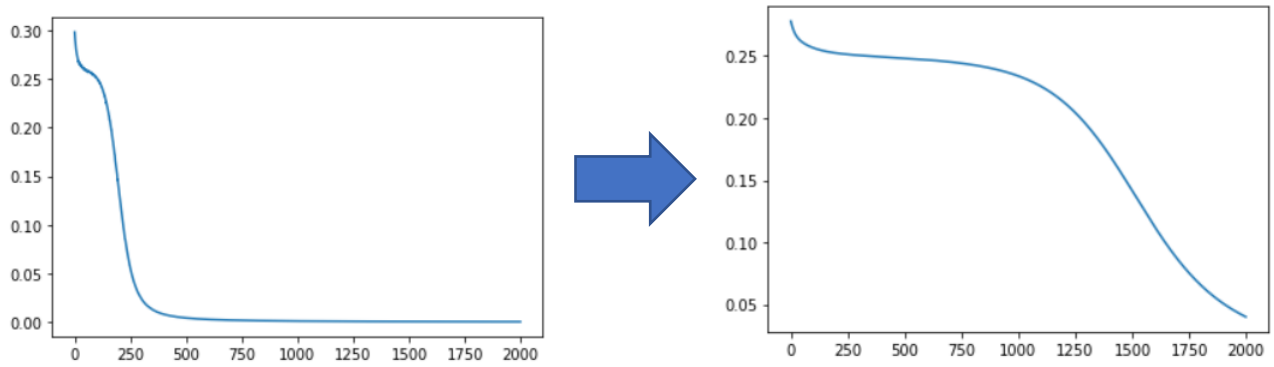
```
[5] plt.plot(history.history['loss'])
```



④ ③번의 결과에서 학습이 종료된 후 loss 값이 가장 작은 경우와 가장 큰 경우가 어떤 case 인지 설명하고 그러한 차이가 나는 이유에 대한 본인 의견을 설명하시오.

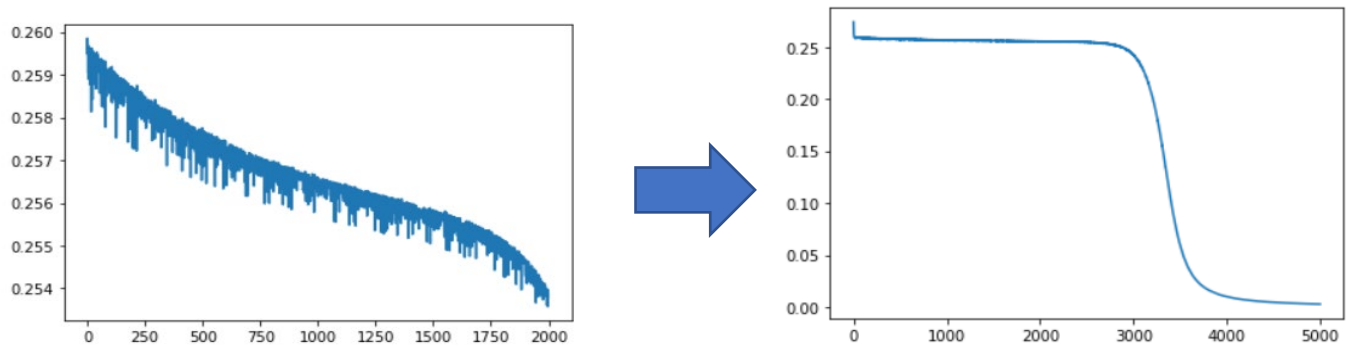
loss 값이 가장 큰 경우는 case7이고, lose값은 0.2538이다.
loss 값이 가장 작은 경우는 case8이고 lose값은 3.6098e-04이다.
이러한 차이가 나는 이유는 레포트문제 2번에서 AND신경망을 구현할 때 sigmoid에서 필요이상으로 과하게 구현하면 오차가 커지듯이 case7번 역시 필요이상으로 뉴런수가 많아 loss값이 가장 크게 나온거 같다.
case8의 경우 일반적으로 뉴런수가 많을수록 성능이 좋아지고(오차가 작아지고) 특히 첫번째 layer의 뉴런수가 같을 때 sigmoid, tanh, relu 세가지 경우중에서 tanh가 성능이 가장 좋다. 그래서 뉴런수도 많고 tanh라서 loss값이 가장 작게 나온 것 같다.

⑤ ④번에서 loss 가 가장 작은 경우에 대하여 model.fit 의 batch_size=10 으로 하면 loss 가 어떻게 변화하는지 그래프출력을 보이고 epochs 와 loss 변화를 보고 비교 설명하시오.



case8 loss값이 3.6098e-04로 9개의 케이스 중에서 가장 작았다.
batch_size를 10으로 한 후에는 lose값이 0.0400로 loss값이 커졌다.
왼쪽그래프와 비교해서 batch_size를 10으로 늘린 오른쪽그래프는 초반에 loss값을 급격하게 줄이지 못하고 완만하게 loss값을 줄인다.

⑥ ④번에서 loss 가 가장 큰 경우에 대하여 model.fit 의 epochs=5000 으로 하면 loss 가 어떻게 변화하는지 그래프출력을 보이고 epochs 와 loss 변화를 보고 비교 설명하시오.



case7 loss값이 0.2538로 9개의 케이스 중에서 가장 컸다.
epochs를 5000으로 한 후에는 lose값이 0.0028으로 loss값이 작아졌다.
오른쪽 그래프를 보면 왼쪽 그래프에서 나타나듯이 2000번까지 loss값 변화가 지체되어있다. 그러다 2000중 후반으로 넘어가면서 loss값이 대폭감소하기 시작한다. epochs값을 늘려 학습량을 늘리면 loss값은 줄어든다.