생체인증보안 HW 1

1971063 김윤서

성능 향상을 위해 변경한 변수들

1. 노드 개수 (num_units): 2,4, ...
2. 경사하강법의 학습율(GradientDescentOptimizer): 0.001, 0.0001, 0.00001, 0.000008...
3. 반복 횟수: 1000, 5000, 10000...
4. 은닉계층 활성화 함수: tanh, relu, ...

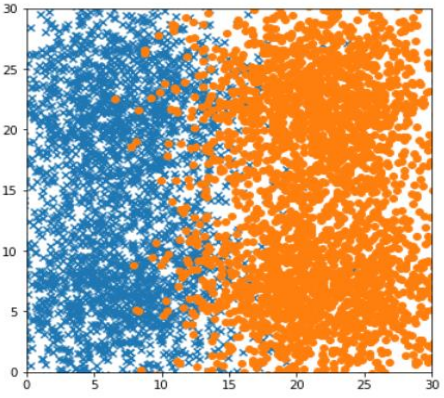
1. 예제 그대로 변수 설정할 경우

- 노드 개수 = 2
- GradientDescentOptimizer= **0.001**
- 반복 횟수 = 1,000
- tanh 함수 사용

```
Step: 100, Loss: nan, Accuracy: 0.000000
Step: 200, Loss: nan, Accuracy: 0.000000
Step: 300, Loss: nan, Accuracy: 0.000000
Step: 400, Loss: nan, Accuracy: 0.000000
Step: 500, Loss: nan, Accuracy: 0.000000
Step: 600, Loss: nan, Accuracy: 0.000000
Step: 700, Loss: nan, Accuracy: 0.000000
Step: 800, Loss: nan, Accuracy: 0.000000
Step: 900, Loss: nan, Accuracy: 0.000000
Step: 1000, Loss: nan, Accuracy: 0.000000
```

➔ Loss 는 숫자가 아닌 값, Accuracy 는 **모두 0.00** 으로 제대로 학습이 안 되었다.

```
<string>:6: UserWarning: Warning: converting a masked element to nan.
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: UserWarning:
  return array(a, dtype, copy=False, order=order)
```



2. 경사하강법의 학습율(GradientDescentOptimizer)을 **0.0001** 로 줄인 경우

```
t = tf.placeholder(tf.float32, [None, 1])
loss = -tf.reduce_sum(t*tf.log(p) + (1-t)*tf.log(1-p))
train_step = tf.train.GradientDescentOptimizer(0.0001).minimize(loss) # 이 부분
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
Step: 100, Loss: nan, Accuracy: 0.000000
Step: 200, Loss: nan, Accuracy: 0.000000
Step: 300, Loss: nan, Accuracy: 0.000000
Step: 400, Loss: nan, Accuracy: 0.000000
Step: 500, Loss: nan, Accuracy: 0.000000
Step: 600, Loss: nan, Accuracy: 0.000000
Step: 700, Loss: nan, Accuracy: 0.000000
Step: 800, Loss: nan, Accuracy: 0.000000
Step: 900, Loss: nan, Accuracy: 0.000000
Step: 1000, Loss: nan, Accuracy: 0.000000
```

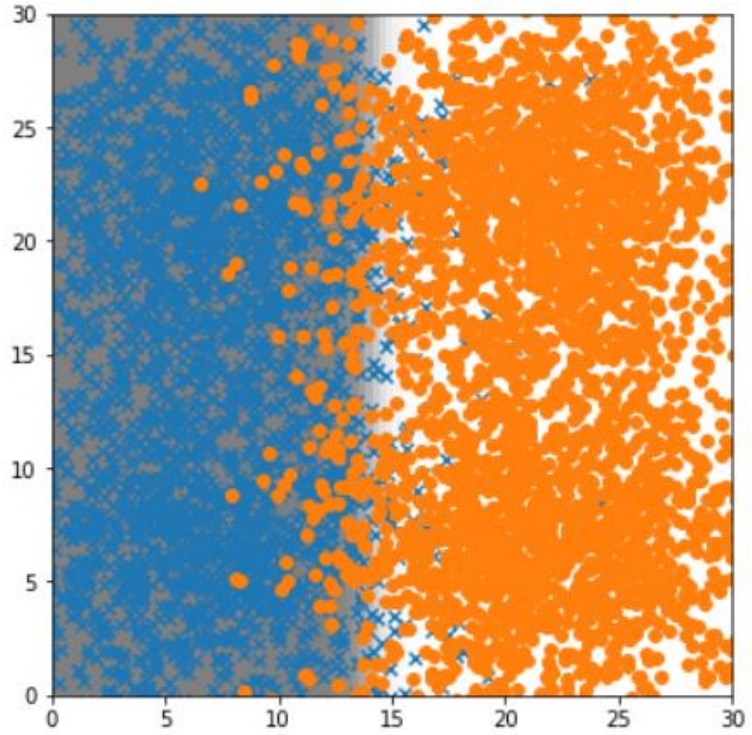
➔ 여전히 결과가 나오지 않음

3. GradientDescentOptimizer 을 **0.00001** 로 줄인 경우

```
Step: 100, Loss: 3508.053955, Accuracy: 0.500000
Step: 200, Loss: 1262.053711, Accuracy: 0.945667
Step: 300, Loss: 1160.163696, Accuracy: 0.945667
Step: 400, Loss: 1107.622925, Accuracy: 0.945667
Step: 500, Loss: 1095.405762, Accuracy: 0.945833
Step: 600, Loss: 1273.229370, Accuracy: 0.929833
Step: 700, Loss: 1067.012939, Accuracy: 0.945833
Step: 800, Loss: 1069.255371, Accuracy: 0.945667
Step: 900, Loss: 1072.472046, Accuracy: 0.945833
Step: 1000, Loss: 1076.317627, Accuracy: 0.945667
```

➔ 정확도와 오차가 유의미한 값으로 나타남

```
<matplotlib.image.AxesImage at 0x7f1e1804acd0>
```



➔ 시각화했을 때도 어느 정도 분리가 된 모습

4. GradientDescentOptimizer= 0.00001 로 유지, 반복 횟수를 **5000** 번으로 늘린 경우

```
[19] i = 0
     for _ in range(5000): #5000번 반복
       i+=1
       sess.run(train_step, feed_dict={x:train_x, t:train_t})
       if i % 100 == 0:  ##100번 단위로 출력해준다
```

```
Step: 4300, Loss: 1056.515137, Accuracy: 0.946000
Step: 4400, Loss: 1060.921387, Accuracy: 0.945667
Step: 4500, Loss: 1076.951538, Accuracy: 0.945833
Step: 4600, Loss: 1051.619507, Accuracy: 0.945500
Step: 4700, Loss: 1059.760010, Accuracy: 0.945667
Step: 4800, Loss: 1063.823486, Accuracy: 0.945833
Step: 4900, Loss: 1065.695923, Accuracy: 0.945833
Step: 5000, Loss: 1069.982666, Accuracy: 0.945833
```

➔ 정확도가 약 0.0002 정도 늘어남

5. 노드 개수를 **4** 로 늘린 경우 (다른 변수는 4 번과 동일)

```
num_units = 4  # 노드 개수: 4
mult = train_x.flatten().mean()

x = tf.placeholder(tf.float32,
```

```
Step: 4600, Loss: 892.760010, Accuracy: 0.944500
Step: 4700, Loss: 890.628967, Accuracy: 0.944500
Step: 4800, Loss: 888.704041, Accuracy: 0.944333
Step: 4900, Loss: 886.963745, Accuracy: 0.944500
Step: 5000, Loss: 885.391602, Accuracy: 0.944667
```

➔ 정확도가 0.001 정도 줄어들었으므로, 노드 개수는 2 일 때가 더 정확하다.

6. 노드 개수 4, 은닉 계층 함수를 **relu** 로 변경한 경우

```
hidden1 = tf.nn.relu(tf.matmul(x, w1)+ b1*mult)  #relu로도 해보고
#hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1*mult) # tanh로도 해보기
```

```
Step: 4700, Loss: 852.381531, Accuracy: 0.946167
Step: 4800, Loss: 852.380127, Accuracy: 0.946167
Step: 4900, Loss: 852.378967, Accuracy: 0.946167
Step: 5000, Loss: 852.377686, Accuracy: 0.946167
```

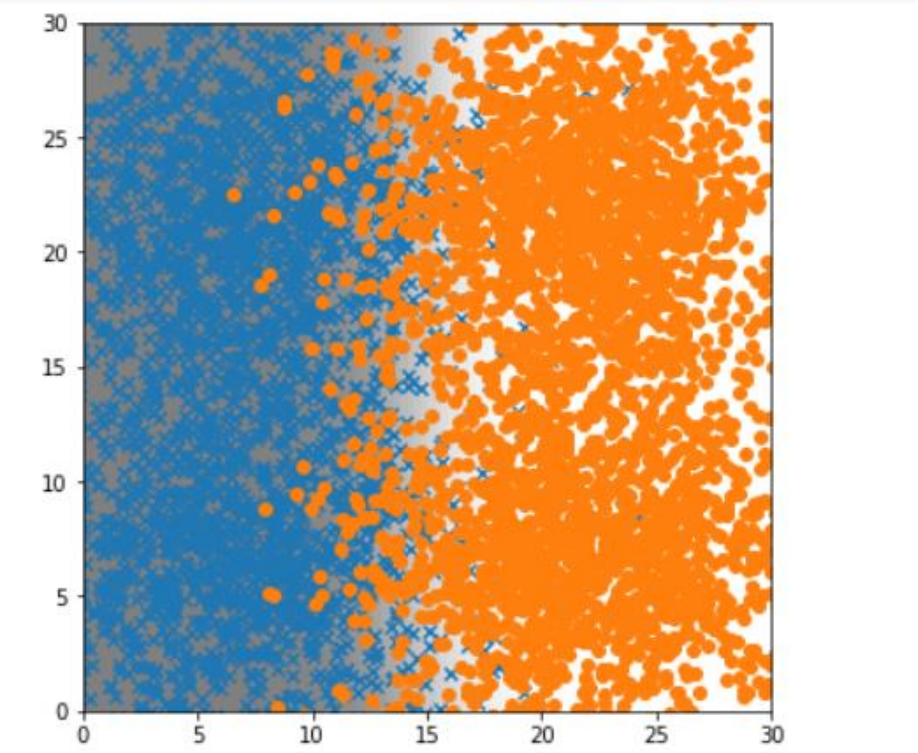
➔ 정확도가 0.946167 로 현재까지 최대 성능을 보임

7. 6 번과 동일 조건+ **GradientDescentOptimizer= 0.000008** 로 줄인 경우

```
t = tf.placeholder(tf.float32, [None, 1])
loss = -tf.reduce_sum(t*tf.log(p) + (1-t)*tf.log(1-p))
train_step = tf.train.GradientDescentOptimizer(0.000008).minimize(loss)
```

```
Step: 4600, Loss: 849.518921, Accuracy: 0.946667
Step: 4700, Loss: 849.515259, Accuracy: 0.946667
Step: 4800, Loss: 849.511597, Accuracy: 0.946667
Step: 4900, Loss: 849.508057, Accuracy: 0.946667
Step: 5000, Loss: 849.504517, Accuracy: 0.946667
```

➔ **0.946667** 으로 최대 성능 갱신



■ 최종 결과

정확도 **0.946667**, 오차 **849.5** 정도로 도출

<변수 설정>

1. 노드 개수 = **4**
2. GradientDescentOptimizer= **0.000008**
3. 반복 횟수= **5,000**
4. **relu** 함수 사용

■ 최종 코드 보기

https://github.com/ottl-seo/biometric-security/blob/main/hw1_%EA%B9%80%EC%9C%A4%EC%84%9C_1971063.ipynb