

인공지능 학기말 프로젝트
(텐서플로 딥러닝)

진민주
202001834
컴퓨터공학과

1. 서론(문제 설명)

이번 과제는 tic-tac-toe data(csv)를 이용하여 tensorflow로 학습하는 것이다.

1.1. 게임 설명

우선 tic-tac-toe 게임은 두 명이 번갈아가면서 O, X를 3x3판에 써서 자신의 글자를 가로, 세로, 혹은 대각선 상에 놓이도록 하는 게임이다.

1.2. 파일(데이터) 설명

tic-tac-toe data(csv)는 X가 먼저 플레이한 것으로 가정하고, 목표는 "X를 위한 승리"이다. 파일 내용으로 row 첫 번째 줄을 보면 TL, TM, TR, ML, MM, MR, BL, BM, BR, class가 있다.

*L, TM, TR, ML, MM, MR, BL, BM, BR(X)*는 각각 앞글자 T(top)는 3x3의 상단을 의미하며, M(middle)은 가운데, B(bottom)은 하단이다. 각각 뒷글자 L(left)은 3x3의 왼쪽, M(middle)은 가운데, R(right)은 오른쪽을 의미하며, 해당 칸을 선택한 값들이 들어있다. X나 O, b가 들어가 있는데, X는 X가 선택한 자리, O는 O가 선택한 자리, b는 공백이라는 의미이다.

*class(y_true)*는 해당 게임에 이긴 패가 적혀있다. TRUE일 때 X가 이겼고, FLASE일 때 X가 진 것을 의미한다.

예를 들어 데이터가 아래와 같을 경우

TL	TM	TR	ML	MM	MR	BL	BM	BR	class
x	x	x	x	o	o	o	b	b	TRUE

3x3에는

x x x

x o o

o b b

와 같이 들어가 있기 때문에 X가 이겼다는 의미로 class에 TRUE가 들어가는 형태의 예시이다.

그리고 해당 파일은 라벨을 제외한 데이터의 수는 958이다.

2. 본론(구현내용을 코드, 그래프 등 사용하여 설명)

본론에 앞서서 설정한 학습환경이다.

train:val:test: 6:2:2 or 8:1:1

optimizers: Adam or RMSprop

층 갯수: 2개

은닉층의 뉴런 개수: 2 or 10

모델의 평가 방법(metrics): accuracy or binary_accuracy

손실함수: categorical_crossentropy or binary_crossentropy

epoch: 1000(계속 변경)

learning_rate: 0.01 or 0.001

2.1. 데이터 불러오기, 전처리

데이터를 불러오면서 게임 한 판 당 TL, TM, TR, ML, MM, MR, BL, BM, BR(X)에 들어간

값들 x, o, b는 0, 1, 2로 값을 치환, class(y_true)의 값들 TRUE, FALSE는 각각 1, 0으로 치환해서 사용했다.

```

1. def load_ttt(shuffle=True):
2.     game_board = {'x': 0, 'o': 1, 'b': 2}
3.     game_win = {'false': 0, 'true': 1}
4.
5.     data = np.loadtxt("./tic-tac-toe.csv", skiprows=1, delimiter=',',
6.                       converters={0: lambda board: game_board[board.decode()],
7.                                   1: lambda board: game_board[board.decode()],
8.                                   2: lambda board: game_board[board.decode()],
9.                                   3: lambda board: game_board[board.decode()],
10.                                  4: lambda board: game_board[board.decode()],
11.                                  5: lambda board: game_board[board.decode()],
12.                                  6: lambda board: game_board[board.decode()],
13.                                  7: lambda board: game_board[board.decode()],
14.                                  8: lambda board: game_board[board.decode()],
15.                                  9: lambda winner: game_win[winner.decode()]})
16.
17.     print(data[0], data[637]) # 치환 됐는지 확인
18.
19.     if shuffle:
20.         np.random.shuffle(data)
21.     return data

```

치환이 원하는대로 잘 되었는지 확인하기 위해서 print(data[0], data[637])을 사용해서 확인해보니까 잘 변경된 것을 확인했다.

x	x	x	x	o	o	x	o	o	TRUE
x	x	o	x	o	b	o	b	b	FALSE

```

6/6 - 0s - loss: 0.8646 - accuracy: 0.7969 - 28ms/epoch - 5ms/step
PS C:\Users\freet\Desktop\3학년2학기\인공지능\과제\삼목게임> & C:/Use
[0. 0. 0. 0. 1. 1. 0. 1. 1. 1.] [0. 0. 1. 0. 2. 0. 1. 1. 1. 0.]
X.shape: (958, 9)

```

데이터를 받아오고 손실함수는 categorical_crossentropy나 binary_crossentropy로 사용할 것이기 때문에 y_true는 원-핫 인코딩을 해준다. 처음에는 train:validation:test는 6, 2, 2의 비율로 fit 할 것이기 때문에 train, test set을 나눠준다.

```

1. ttt_data = load_ttt()
2. X = ttt_data[:, :-1] # class 제외
3. y_true = ttt_data[:, -1] # class만
4. y_true = tf.keras.utils.to_categorical(y_true) # 원-핫으로 변경
5. # print(y_true[0])
6. # print("X.shape:", X.shape) # 958x9 데이터
7. # print("y_true.shape:", y_true.shape) # 958x2 데이터
8.
9. # train, test = 8:2 or 9:1
10. X_train, X_test, y_train, y_test = train_test_split(

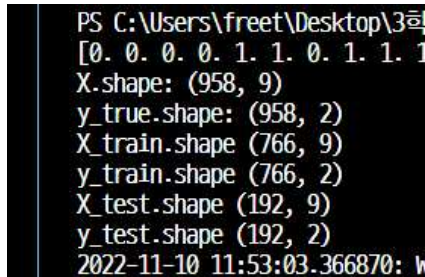
```

```

11. X, y_true, test_size=0.2, stratify=y_true, random_state=1)
12.
13.
14. # print("X_train.shape", X_train.shape)
15. # print("y_train.shape", y_train.shape)
16. # print("X_test.shape", X_test.shape)
17. # print("y_test.shape", y_test.shape)

```

잘 나뉜 것을 확인한다.



```

PS C:\Users\freet\Desktop\3회>
[0. 0. 0. 0. 1. 1. 0. 1. 1. 1.]
X.shape: (958, 9)
y_true.shape: (958, 2)
X_train.shape (766, 9)
y_train.shape (766, 2)
X_test.shape (192, 9)
y_test.shape (192, 2)
2022-11-10 11:53:03.366870: W

```

앞서 설정한 학습 환경에 맞춰서 짰 코드이다.

input_dim은 TL, TM, TR, ML, MM, MR, BL, BM, BR로 총 9개가 들어올 것이고, output layer는 class가 categorical 형태라서 출력층의 units은 2로 설정했다.

```

1. # 2층 신경망
2. # n = 2
3. n = 10
4. model = tf.keras.Sequential()
5. model.add(tf.keras.layers.Dense(units=n, input_dim=9, activation="sigmoid"))
6. model.add(tf.keras.layers.Dense(units=2, activation='softmax'))
7. model.summary()
8.
9. # opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
10. opt = tf.keras.optimizers.Adam(0.001)
11.
12. model.compile(optimizer=opt, loss='binary_crossentropy', # categorical_crossentropy
13.               metrics=['binary_accuracy']) # accuracy
14.
15. ret = model.fit(X_train, y_train, epochs=3000, verbose=2,
16.               validation_split=0.2, batch_size=64) # validation_split=0.1

```

어떤 환경이 더 좋은지 판단하기 위해서 train, test의 데이터의 손실과 정확도를 구했으며, train, val의 loss, acc 그래프를 그렸다. 그리고 잘 측정되었는지 보기위해 원래 값과 예측한 값을 30개만 추출해서 비교했다.

```

1. train_loss, train_acc = model.evaluate(X_train, y_train, verbose=2)
2. test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
3.
4. y_pred = model.predict(X_train[:30], verbose=0)
5. y_pred = np.around(y_pred.astype(int)).flatten()
6. y_act = np.around(y_train[:30]).astype(int).flatten()
7.

```

```

8. fig, ax = plt.subplots(1, 2, figsize=(10, 6))
9. ax[0].plot(ret.history['loss'], "b-", label="train loss")
10. ax[0].plot(ret.history['val_loss'], "r-", label="var loss")
11. ax[0].set_title("loss")
12. ax[0].set_xlabel("epochs")
13. ax[0].set_ylabel("loss")
14.
15. ax[1].plot(ret.history['binary_accuracy'], "b-",
16.           label="train accuracy") # accuracy
17. ax[1].plot(ret.history['val_binary_accuracy'], "r-",
18.           label="val accuracy") # val_accuracy
19. ax[1].set_title("accuracy")
20. ax[1].set_xlabel("epochs")
21. ax[1].set_ylabel("accuracy")
22. plt.legend(loc="best")
23. fig.tight_layout()
24. plt.show()

```

3. 실험결과(화면 덤프 및 결과 분석 설명)

train:val:test는 6:2:2 비율로 고정했다.

+) 원래는 binary_crossentropy가 아닌 categorical_crossentropy만 사용했다가 친구와 이야기를 하다가 실험에 binary_crossentropy 조건이 추가되었습니다. 어떤 crossentropy에 따라서 평가방법은 아래와 같이 다르며, categorical로 했던 것부터 시작하겠습니다.

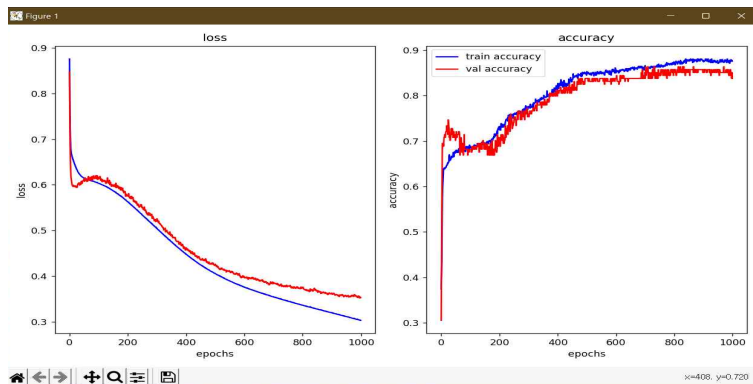
- 손실함수: categorical_crossentropy or binary_crossentropy
- 모델의 평가 방법(metrics): accuracy or binary_accuracy

3.1 첫 번째 실험(optimizer과 epoch, learning_rate를 계속 바꿔가면서 좋은 환경 찾기)

우선은 수업시간에 주로 했던 기본 세팅으로 해보았다.

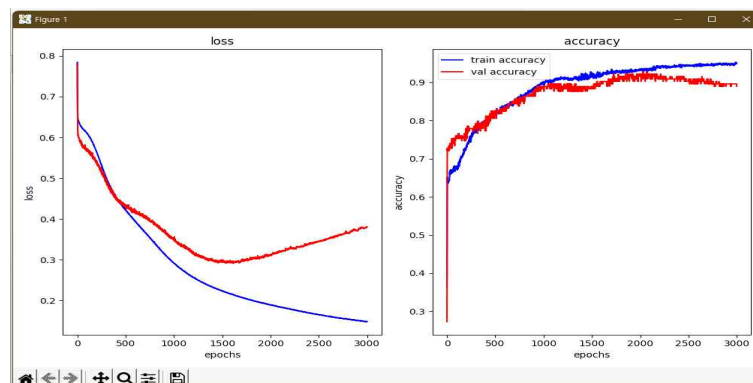
- train:val:test: 6:2:2, 은닉층의 뉴런 개수: 10, epoch: 계속 변경, learning_rate: 0.01 or 0.001, optimizers: RMSprop or adam

optimizer를 RMSprop일 때 val_loss가 상승하는 과적합 상태가 발생했다. adam으로 변경해서 진행해보았더니 RMSprop처럼 수직 상승은 아니지만 너무 loss율이 중구난방이고, 상승 구간이 있었다. 그래서 RMSprop나 adam을 learning rate를 0.001로 변경해보았으며 수업 시간이라 다르게 adam 하고 싶었기에 adam으로 하려고한다. 그렇게 그린 그래프는 아래와 같다.



-> 그래프는 loss가 보기 좋게 하락하고 acc도 상승하는 것을 볼 수 있지만 acc가 0.9를 넘지 못하는 모습을 볼 수 있다.

혹여나 환경을 바꿨으니 3000번 돌렸을 때 더 잘 학습되지 않을까 싶어 돌려보았다.



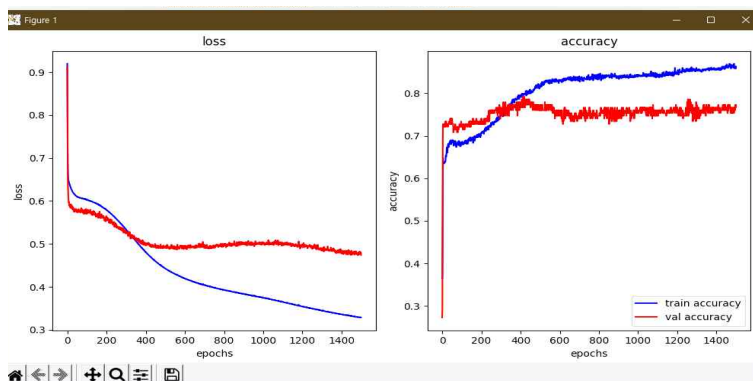
-> train 데이터는 여전히 잘 학습되고 있지만 val을 보면 loss는 1500을 넘으니 상승했고, acc가 정체되어있는 것을 볼 수 있다.

=> 다음 실험에서는 epoch은 1000 ~ 1500 중에서 더 좋은 것을 찾으려고 한다.

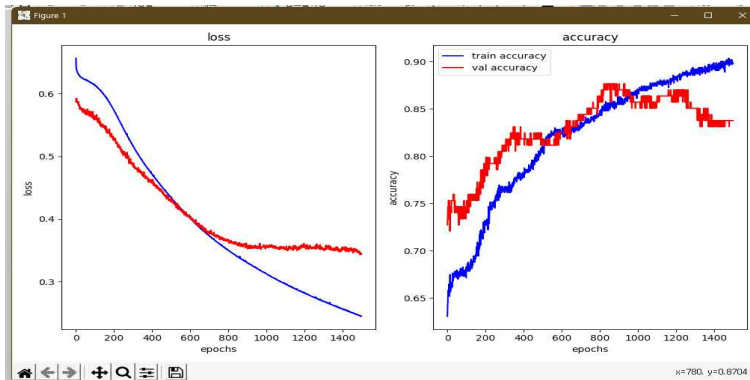
3.2 두 번째 실험(에폭을 바꿔가면서 좋은 환경 찾기)

- optimizers: Adam, learning_rate: 0.001, train:val:test: 6:2:2, 은닉층의 뉴런 개수: 10, epoch: 1000 ~ 1500

우선 epoch을 1000과 1500으로 두고 살펴보면서 어느 정도가 괜찮을지 한번 더 살펴봤다.



-> val loss가 1000쯤에서 올라가긴 했지만 다시 하락하기 때문에 1500이 적당할 것 같지만 데이터가 shuffle로 되어 있고 돌릴 때마다 다르게 set이 나뉘지다보니 한 번 더 돌려봤다.



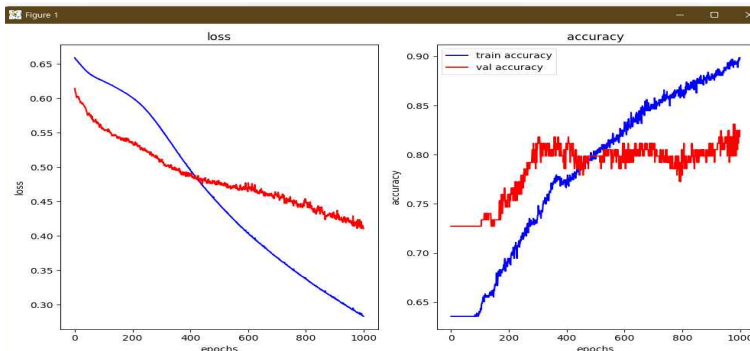
-> train loss, acc는 여전히 잘 되고 있고 val loss는 1000 정도에서 정체, val_acc는 1000정도의 값이 좋게 나온다.

1500으로 혹시 돌려보았더니 한번 더 돌려봤는데 1000구간에서도 상승하기도 한다. 그런데 계속 돌려봤지만 1000에서 올라가는 일은 적어서 epoch은 1000으로 고정해서 사용하기로 했다.

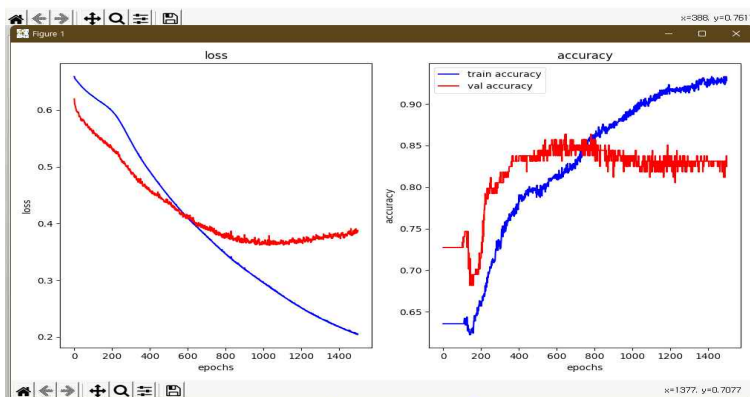
3.3 세 번째 실험(층 개수 바꿔보기)

- optimizers: Adam, learning_rate: 0.001, train:val:test: 6:2:2, epoch: 1000, 층 개수: 2층 신경망 or 3층 신경망, 은닉층의 뉴런 개수: 10 or 2

현재까지 설정한 환경에서 2층에서 3층 신경망으로 변경했다.



-> 1000번을 돌렸을 때 val_loss가 평평한 느낌이 없어서 혹시 싶어 1500번을 계속 돌려봤다.



-> 1500번을 여러번 돌려봤는데 2층 신경망이었을 때랑 비슷하게 val_loss가 상승할 듯하다.

3000번도 돌려봤는데 3층 신경망도 1000 epoch이 오버피팅도 없는 적당한 위치인 것 같다.

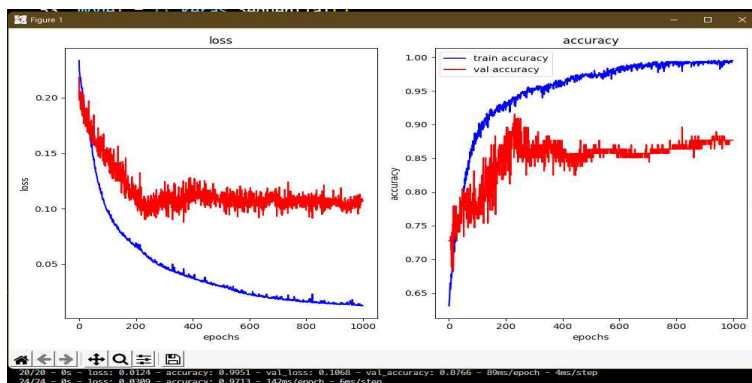
=> 2층에서 3층으로 1개의 층만 추가해봤긴했지만 별 차이가 없다.

=> 뉴런도 2에서 10으로 변경해서 해봤었는데, 2인 상태에서 여러 가지 맞춘 환경이라 그런지 더 안 좋게 나왔다.

3.4 네 번째 실험(손실함수 바꿔보기)

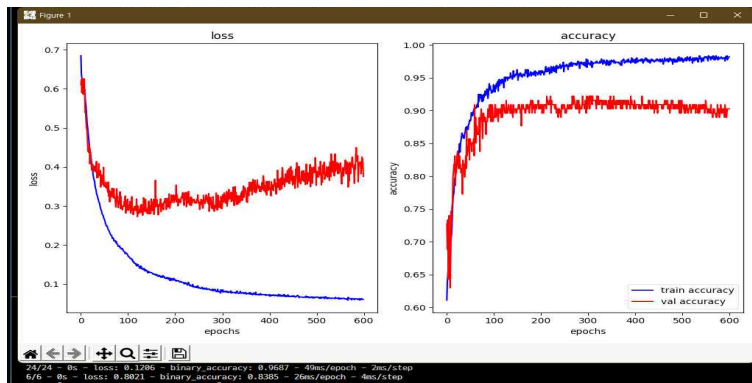
친구랑 과제 이야기를 하다가 acc가 1까지 올라갔다고 들었다. 들어보니까 손실함수를 mse를 사용했다고 한다.

우선 $learning_rate=0.01$, 손실함수=mse로 해보았다.

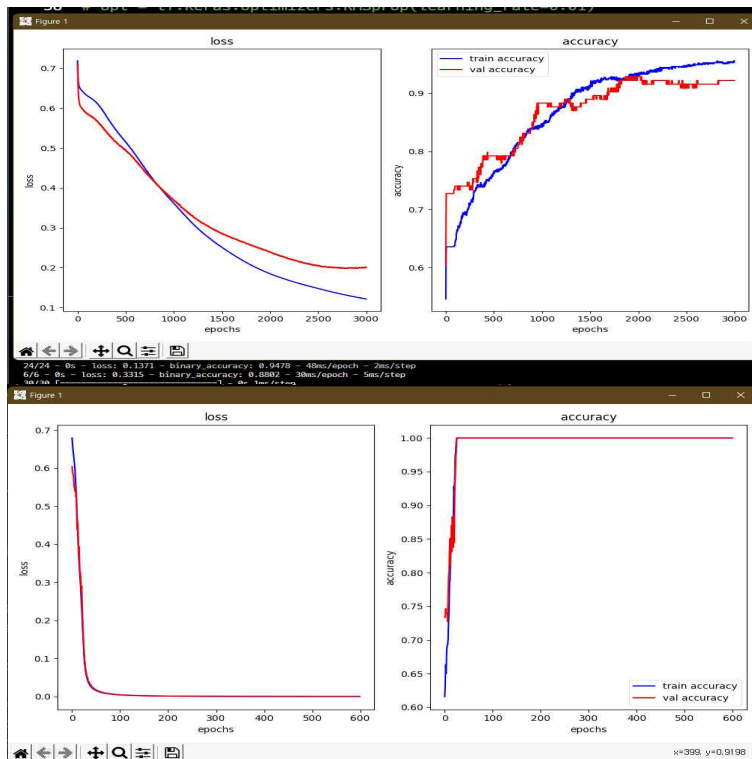


-> loss도 잘 내려가고 acc도 test는 0.84로 나왔지만 잘 나왔다. 이때까지 한 파라미터를 고치면 괜찮게 뜰 것 같긴하다

근데 데이터 자체를 생각해보면 y의 값은 0, 1로 나뉘지니까 binary crossentropy가 맞지 않을까 싶었다. 그리고 친구가 mse로 했으니까 binary로 하고 싶었다.



-> 해당 그래프는 loss는 binary_crossentropy, metrics는 binary_accuracy로 했다. epoch은 600, batch_size를 기본 32로 하니까 애매했었다. 참고문헌2를 보니까 batch size를 크게 하면 일반화 성능이 다소 감소한다해서 자주 사용한다는 것 중 64로 맞춰서 해보았다. 그랬더니 acc는 잘 나왔는데, val loss가 올라갔다.(learning_rate를 0.1, 0.01로 했을 때도 val loss가 상승했다.)



-> learning_rate=0.001, epoch은 600으로 했다가 더 내려갈 수 있을 것 같아서 3000으로 확 올렸다. 그래프상으로 전부 안정적인데 3000번은 너무 많긴하다.

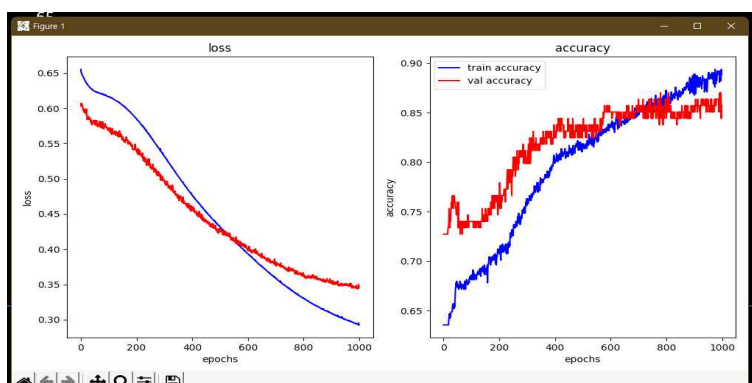
-> 혹시 몰라서 learning_rate를 = 0.1로 했더니.. 확올라갔다.(0.1로 하니깐 다시 할 때마다 그래프가 들쭉날쭉하다..)

이후로 batch_size, epoch, learning_rate를 변경해보았는데 binary crossentropy일 때는 전전 그래프처럼 3000번 돌리고 안전하게 하는게 나을 것 같다.

4. 결론(결론, 과제 수행 느낌 등)

결론적으로 적절한 환경 설정으로는

- 손실함수: categorical_crossentropy
 - optimizers: Adam, learning_rate: 0.001, train:val:test = 6:2:2, epoch: 1000, 은닉층의 뉴런 개수: 10, 층의 개수: 2, batch_size: 32가 괜찮았다!
- 해당하는 환경 설정의 결과는 아래와 같다.



그래프를 보면 train, val loss가 계속해서 내려가는 모습을 보여서 잘 train하고 있다는 것을 볼 수 있다. 또한 acc도 상승하는 모습을 보여준다!

밑에 보이는 것처럼 최종 train_loss는 0.29, test_loss는 0.45로 0에 가깝게 내려갔다. train_acc와 test_acc도 0.88, 0.80으로 1에 가까워지는 것을 볼 수 있다.

예측한 것을 비교했을 때 잘 되어진 것을 볼 수 있다.

-
- The figure consists of two side-by-side line plots. The left plot, titled 'loss', shows the training loss (blue line) and validation loss (red line) over 3000 epochs. Both losses decrease, with the training loss reaching approximately 0.13 and the validation loss reaching approximately 0.20. The right plot, titled 'accuracy', shows the training accuracy (blue line) and validation accuracy (red line) over 3000 epochs. Both accuracies increase, with the training accuracy reaching approximately 0.95 and the validation accuracy reaching approximately 0.93. A status bar at the bottom provides specific metrics for the current epoch (24/24) and the next epoch (6/6).
- | Epoch | loss | binary_accuracy | time/epoch | time/step |
|-------|--------|-----------------|------------|-----------|
| 24/24 | 0.1371 | 0.9478 | 48ms/epoch | 2ms/step |
| 6/6 | 0.3315 | 0.8882 | 30ms/epoch | 5ms/step |

```
Prediction: [1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0
1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1]
Actual: [1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0
1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1]
```

느낌점

categorical로만 했을 때는 acc를 1에 가깝게 더 설정하고 싶었는데 어느 구간부터 val_loss가 상승해서 어쩔 수 없이 적당히 train 했는데 아쉬웠었다. 친구랑 의논하고 binary로 해봤는데 1에 가까운 값을 가질 수 있어서 좋았다. 근데 친구는 acc가 1로 나왔다고 했는데 잘 학습한건지는 모르겠다... 그래도 열심히 했습니다!

참고문헌

- 김동근, 『텐서플로 딥러닝 프로그래밍』, 가메출판사, p85~p207
- 예비개발자, 네이버 블로그, batch size 적절하게 조절하기,

<https://blog.naver.com/qbxlvnf11/221449595336>

코드

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def load_ttt(shuffle=True):
    game_board = {'x': 0, 'o': 1, 'b': 2}
    game_win = {'false': 0, 'true': 1}
    data = np.loadtxt("./tic-tac-toe.csv", skiprows=1, delimiter=',',
                      converters={0: lambda board: game_board[board.decode()],
                                   1: lambda board: game_board[board.decode()],
                                   2: lambda board: game_board[board.decode()],
                                   3: lambda board: game_board[board.decode()],
                                   4: lambda board: game_board[board.decode()],
                                   5: lambda board: game_board[board.decode()],
                                   6: lambda board: game_board[board.decode()],
                                   7: lambda board: game_board[board.decode()],
                                   8: lambda board: game_board[board.decode()],
                                   9: lambda winner: game_win[winner.decode()]})

    # print(data[0], data[637]) # 치환 됐는지 확인
    if shuffle:
        np.random.shuffle(data)
    return data

ttt_data = load_ttt()
X = ttt_data[:, :-1] # class 제외
y_true = ttt_data[:, -1] # class만
y_true = tf.keras.utils.to_categorical(y_true) # 원-핫으로 변경
# print(y_true[0])
# print("X.shape:", X.shape) # 958x9 데이터
# print("y_true.shape:", y_true.shape) # 958x2 데이터
# train, test = 8:2 or 9:1
X_train, X_test, y_train, y_test = train_test_split(
    X, y_true, test_size=0.2, stratify=y_true, random_state=1)
# print("X_train.shape", X_train.shape)
# print("y_train.shape", y_train.shape)
# print("X_test.shape", X_test.shape)
# print("y_test.shape", y_test.shape)

# 2층 신경망
# n = 2
n = 10
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=n, input_dim=9, activation="sigmoid"))
model.add(tf.keras.layers.Dense(units=2, activation='softmax'))
model.summary()
# opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
opt = tf.keras.optimizers.Adam(0.001)
model.compile(optimizer=opt, loss='binary_crossentropy', #
```

```

categorical_crossentropy
        metrics=['binary_accuracy']) # accuracy
ret =model.fit(X_train, y_train, epochs=3000, verbose=2,
               validation_split=0.2, batch_size=64) # validation_split=0.1
train_loss, train_acc =model.evaluate(X_train, y_train, verbose=2)
test_loss, test_acc =model.evaluate(X_test, y_test, verbose=2)
y_pred =model.predict(X_train[:30], verbose=0)
y_pred =np.around(y_pred).astype(int).flatten()
y_act =np.around(y_train[:30]).astype(int).flatten()
print("pred:", y_pred)
print("act:   ", y_act)

fig, ax =plt.subplots(1, 2, figsize=(10, 6))
ax[0].plot(ret.history['loss'], "b-", label="train loss")
ax[0].plot(ret.history['val_loss'], "r-", label="var loss")
ax[0].set_title("loss")
ax[0].set_xlabel("epochs")
ax[0].set_ylabel("loss")
ax[1].plot(ret.history['binary_accuracy'], "b-",
           label="train accuracy") # accuracy
ax[1].plot(ret.history['val_binary_accuracy'], "r-",
           label="val accuracy") # val_accuracy
ax[1].set_title("accuracy")
ax[1].set_xlabel("epochs")
ax[1].set_ylabel("accuracy")
plt.legend(loc="best")
fig.tight_layout()
plt.show()

```