에이쁠원하조 실습과제

Neural Network

Data & Preprocessing

	PlayerName	Club	DistanceCovered(InKms)	Goals	MinutestoGoalRatio	ShotsPerGame	AgentCharges	BMI	Cost	PreviousClubCost	Height	Weight	Score
0	Braund, Mr. Owen Harris	MUN	3.96	7.5	37.5	12.3	60	20.56	109.1	63.32	195.9	78.9	19.75
1	Allen, Mr. William Henry	MUN	4.41	8.3	38.2	12.7	68	20.67	102.8	58.55	189.7	74.4	21.30
2	Moran, Mr. James	MUN	4.14	5.0	36.4	11.6	21	21.86	104.6	55.36	177.8	69.1	19.88
3	McCarthy, Mr. Timothy J	MUN	4.11	5.3	37.3	12.6	69	21.88	126.4	57.18	185.0	74.9	23.66
4	Palsson, Master. Gosta Leonard	MUN	4.45	6.8	41.5	14.0	29	18.96	80.3	53.20	184.6	64.6	17.64
...	...	...	...	...	...	...	...	...	...	...	...	...	...
197	Ryan, Mr. Patrick	LIV	4.90	7.6	45.6	16.0	90	27.56	67.2	82.00	183.9	93.2	11.79
198	Saad, Mr. Amin	LIV	5.66	8.3	50.2	17.7	38	23.76	56.5	72.00	183.5	80.0	10.05
199	Saad, Mr. Khalil	LIV	5.03	6.4	42.7	14.3	122	22.01	47.6	68.00	183.1	73.8	8.51
200	Saade, Mr. Jean Nassr	LIV	4.97	8.8	43.0	14.9	233	22.34	60.4	63.00	178.4	71.1	11.50
201	Sadlier, Mr. Matthew	LIV	5.38	6.3	46.0	15.7	32	21.07	34.9	72.00	190.8	76.7	6.26

```
[114] dataset = dataset.drop(columns=['PlayerName','Club','Height','Weight','PreviousClubCost','AgentCharges'])
```

- **Data**: EPL 축구선수들의 몸값에 영향을 미치는 요인들
- **sample size**: 202
- **target** : 변수 'Score'에 대한 regression
- **제외한 변수**: PlayerName, Club, Height, Weight, PreviousClubCost, AgentCharges
- 데이터 변수 설명 ->

PlayerName : Player Name
Club : Club of the player
MUN:Manchester United F.C.
CHE: Chelsea F.C.
LIV: Liverpool F.C.
DistanceCovered(InKms): Average Kms distance covered by the player in each game
Goals: Average Goals per match
MinutestoGoalRatio: Minutes
ShotsPerGame: Average shots taken per game
AgentCharges: Agent Fees in h
BMI: Body-Mass index
Cost: Cost of each player in hundread thousand dollars
PreviousClubCost: Previous club cost in hundread thousand dollars
Height: Height of player in cm
Weight: Weight of player in kg
Score: Average score per match

```
[449] dataset.describe()
```

	DistanceCovered(InKms)	Goals	MinutestoGoalRatio	ShotsPerGame	BMI	Cost	Score
count	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000
mean	4.718614	7.108663	43.091584	14.566337	22.955891	69.021782	13.507426
std	0.457976	1.800549	3.662989	1.362451	2.863933	32.565333	6.189826
min	3.800000	3.300000	35.900000	11.600000	16.750000	28.000000	5.630000
25%	4.372500	5.900000	40.600000	13.500000	21.082500	43.850000	8.545000
50%	4.755000	6.850000	43.500000	14.700000	22.720000	58.600000	11.650000
75%	5.030000	8.275000	45.575000	15.575000	24.465000	90.350000	18.080000
max	6.720000	14.300000	59.700000	19.200000	34.420000	200.800000	35.520000

data describe 결과 각 변수간 scale이 많이 차이나므로 scaling 진행

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)
```

Train/Test data 8:2 분리

```
[452] # train: test = 8:2 분리
      from sklearn.model_selection import train_test_split
      train_features, test_features , train_target, test_target = train_test_split(
          features, target, test_size = 0.2, random_state = 2021)
```

```
print(len(train_features))
print(len(train_target))

print(len(test_features))
print(len(test_target))
```

```
161
161
41
41
```

Model 생성, 결과

모델 1) hidden layer = 1, node = 16, learning rate = 0.01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.912340	0.218443	0.390225	0.082550	-0.816883	-0.365530	-0.192804	-0.134349	0.383808	0.306161	-0.406389	-0.514943	-0.290274	0.281638	0.174377	0.283489

Epoch 459: early stopping

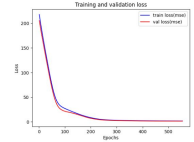
test R squared: 0.922
test MSE: 3.193
tset MAPE: 0.105

모델 2-1)hidden layer =1, nodes =8, learning rate = 0.01

	0	1	2	3	4	5	6	7
0	0.504289	-0.065140	0.189214	0.196162	-0.969830	-0.205919	0.120056	-0.309530

Epoch 554: early stopping

test R squared: 0.934
test MSE: 2.690
tset MAPE: 0.088

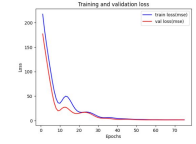


모델 2-2)hidden layer =1, nodes =8, learning rate = 0.1

	0	1	2	3	4	5	6	7
0	-0.075818	-0.967834	0.665693	-0.149890	-0.724824	0.503973	0.084422	-0.720856

Epoch 75: early stopping

test R squared: 0.935
test MSE: 2.672
tset MAPE: 0.105

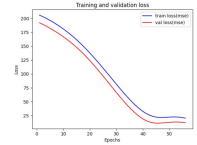


모델3-1)hidden layer = 2, nodes = 4,8, **learning** rate = 0.01

	0	1	2	3
0	0.512785	-0.497460	0.185284	-0.072461

Epoch 56: early stopping

test R squared: 0.489
test MSE: 20.947
tset MAPE: 0.275

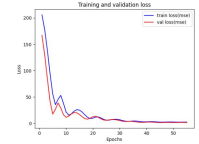


모델3-2)hidden layer = 2, nodes = 4,8, **learning** rate = 0.1

	0	1	2	3
0	-0.035122	-0.170474	-0.262611	0.164996

Epoch 55: early stopping

test R squared: 0.925
test MSE: 3.069
tset MAPE: 0.099

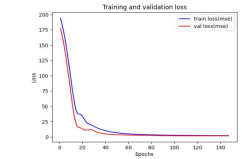


모델4-1) hidden layer = 2, node=8,8,learning rate = 0.01

	0	1	2	3	4	5	6	7
0	0.046655	-0.176287	0.139822	0.101463	-0.432385	-0.024470	0.237378	-0.394762

Epoch 147: early stopping

test R squared: 0.917
test MSE: 3.422
tset MAPE: 0.105

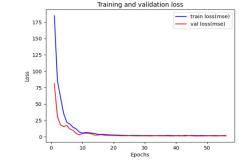


모델4-2) hidden layer = 2, node=8,8, learning rate = 0.1

	0	1	2	3	4	5	6	7
0	-0.475604	-0.540169	0.055706	0.255476	-0.411088	0.229565	0.262250	-0.614154

Epoch 56: early stopping

test R squared: 0.927
test MSE: 2.978
tset MAPE: 0.090

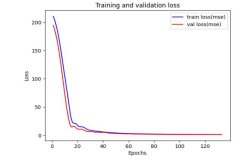


모델5-1)hidden layer = 3, nodes 16,4,8, learning rate = 0.01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.123431	-0.107715	0.122882	0.307856	-0.460514	-0.027155	0.198087	-0.058222	0.186993	-0.252857	0.214876	-0.371222	0.132294	-0.321692	0.148701	-0.076604

Epoch 133: early stopping

test R squared: 0.936
test MSE: 2.614
tset MAPE: 0.091

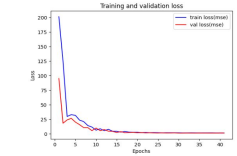


모델5-2)hidden layer = 3, nodes 16,4,8, learning rate = 0.1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-0.251421	-0.360066	0.439925	-0.387005	-0.560211	-0.252269	-0.072096	-0.278004	-0.064533	-0.643864	-0.160264	-0.235399	0.121463	-0.700380	-0.111796	0.043219

Epoch 41: early stopping

test R squared: 0.948
test MSE: 2.124
tset MAPE: 0.091

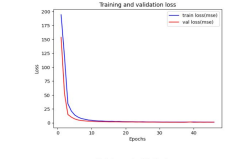


모델6-1) hidden layer = 3, node=16,4,4,learning rate = 0.01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.140838	-0.199111	0.011634	-0.329483	-0.525269	-0.013447	0.208697	-0.118693	0.271123	-0.211208	0.338193	-0.363441	0.031813	-0.221111	0.326704	-0.136745

Epoch 46: early stopping

test R squared: 0.938
test MSE: 2.529
tset MAPE: 0.090

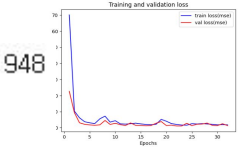


모델6-2) hidden layer = 3, node=16,4,4,learning rate = 0.1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-0.375060	-0.135872	0.245511	0.337233	-0.357262	0.009100	0.281996	-0.088212	0.247447	-0.582629	0.321832	-0.469481	0.125514	-0.324959	0.377189	-0.112931

Epoch 32: early stopping

test R squared: 0.948
test MSE: 2.120
tset MAPE: 0.093



결과 해석

모델 1:
표를 참고하면 0,10 번 노드에 높은 가중치가 부여되어 있고, 0,3,14 번 노드가 크게 편향되어 있음을 알 수 있다. 3번 노드는 음의 방향으로 편향되어 있어 역방향으로 영향을 준다는 것을 알 수 있다. R-squared 값은 0.922이다.

모델 2-1:
0, 4번 노드에 높은 가중치가 부여되어 있고, 1,2번 노드가 음의 값으로 크게 편향되어 있음을 알 수 있다. R-squared값은 0.934이며 모델 1보다 개선된 결과를 보인다.

모델 2-2:
1,4,7 번 노드에 높은 가중치가 부여되어 있고, 2,5번 노드가 음의 값으로 매우 크게 편향되어 있다. R-squared값은 0.935이며 모델 2-1과 성능 면에서 크게 개선된 수준은 아니다.

모델 3-1:
0,1 번 노드에 높은 가중치가 부여되어 있고, 0,1번은 음의 값으로 크게 편향되어 있고 2번 노드는 양의 값으로 편향되어 있다. R-squared값은 0.489로 다른 모델에 비해 낮은 성능을 가진다.

모델 3-2:
0번 노드가 낮은 가중치를 가지고 나머지 노드는 고른 가중치를 가진다. 1번 노드가 음의 값으로 매우 크게 편향되어 있다. R-squared값은 0.925로 learning rate를 조절하여 성능이 크게 개선되었다.

모델 4-1:
0,5 번이 낮은 가중치를 가지고 나머지 노드는 고른 가중치를 가진다. 편향값은 다른 모델에 비해 비교적 고르게 가진다. R-squared값은 0.917이다.

모델 4-2:
2번이 낮은 가중치를 가지고 나머지는 노드는 고른 가중치를 가진다. 6번 노드의 편향값이 매우 작고 나머지는 고른 값을 가진다. R-squared값은 0.927로 learning rate를 조절하여 성능이 조금 개선되었다.

모델 5-1:
5,7,15번이 낮은 가중치를 가지고 나머지 노드는 고른 가중치를 가진다. 0,4,6,11,15번 노드의 편향값이 매우 작고 나머지는 비교적 고른 값을 가진다. R-squared값은 0,.936이다.

모델 5-2:
6,8,15번이 낮은 가중치를 가지고 나머지 노드는 고른 가중치를 가진다. -0.7~1.2 까지 노드의 편향값이 다양하게 분포한다.
R-squared값은 0.948로 learning rate를 조절하여 성능이 조금 개선되었다.

모델 6-1:
2,5,7,12번이 낮은 가중치를 가지고 나머지 노드는 고른 가중치를 가진다. 6번 노드가 매우 낮은 0에 가까운 편향값을 가진다.
R-squared값은 0.938이다.

모델 6-2:
5번 노드가 매우 낮은 가중치를 가진다. 3번 노드가 음의 값으로 큰 편향값을 가지고 4번 노드가 0에 가까운 편향값을 가진다. R-squared값은 0.948로 learning rate를 조절하여 성능이 개선되었고 모든 모델 중 가장 높은 값이다.

```
pd.DataFrame(model.get_weights()[0])
# weight와 bias 확인
## 0: input layer -> hidden layer1 의 weight
## 1: input layer -> hidden layer1 의 bias
## 2: hidden layer1 -> hidden layer2 의 weight
## 3: hidden layer1 -> hidden layer2 의 bias
## 4: hidden layer2 -> output layer 의 weight
## 5: hidden layer2 -> output layer 의 bias
```

위와 같은 방법으로 input layer에서 hidden layer1으로의 모델별 weight 확인

```
pd.DataFrame(model.get_weights()[0])
```

	0	1	2	3	4	5	6	7
0	0.504289	-0.065140	0.189214	0.196162	-0.969830	-0.205919	0.120056	-0.309530
1	0.512660	0.820465	-0.803091	0.575210	-0.483883	-0.424985	0.478441	-0.358041
2	1.101181	0.423566	-0.471245	1.830453	-0.178225	-0.033886	0.096514	0.017206
3	1.003033	0.328143	-0.479558	2.161221	-0.717934	0.360926	-0.586508	0.240158
4	0.263982	0.322454	-0.337852	0.532669	-0.840949	0.127287	0.035648	-0.422062
5	0.407216	0.544883	-0.261039	-0.247570	0.599476	1.691188	1.354437	-0.929179

-> 모델 2-1의 input layer->hidden layer1 노드별 weight

결론 및 한계점/개선점

결론:
learning rate를 0.01/0.1로 튜닝하여 모델을 작성한 결과 0.1의 경우일 때 성능이 조금 개선되었음을 확인할 수 있다. 또한 hidden layer의 개수, 노드의 개수와 형태에 따라 규칙적인 성능의 변화도 보이지 않았고 불규칙하게 성능이 개선되거나 저하되었기 때문에 어떤 모델 구조를 가지는지에 따라 성능이 변화하는 것으로 보인다. 하지만 hidden layer의 개수가 많아질 수록 적절한 모델의 Epoch 값은 점점 줄어드는 경향을 확인할 수 있었다. 또한 hidden layer의 노드 개수가 많아질 수록 weight, bias의 분포를 분석하기 난해하였고 hidden layer와 노드의 개수가 많아질 수록 모델의 분석 과정을 이해하기는 힘들것이라 생각되었다.

한계점/개선점
데이터 개수가 적어지면 모델이 일반화하기 어려워지며, Overfitting이 발생하기 쉬워진다. 하지만 이번 실습에서 사용한 데이터의 개수가 적었다. 또한 neural network는 Hyperparameter 튜닝에 따라 성능이 개선되지만, 적절한 Hyperparameter를 찾기 어렵다는 문제점이 있다. 더 나아가서 Neural Network 모델은 흑백으로 결과값만을 출력하는 것으로 모델이 내부적으로 어떻게 작동하는지를 이해하기 어렵다는 한계점이 있다.