인공지능 실습 Report

Predicting house prices:
a regression example 실습

201632230 컴퓨터공학과 이다은

2020. 05. 19

1-1. 실습문제 정의

Listing 3.25 Normalizing the data 코드가 없을 경우의 결과를 보여주고 분석하시오.

1-2. 문제해결 방법 제시

데이터 정규화 과정을 제외한 모델 구현을 진행한다.

1-3. 문서화된 소스 코드

서로 다른 스케일을 가진 데이터를 신경망에 바로 주입한다.

```
In [4]: train_targets
```

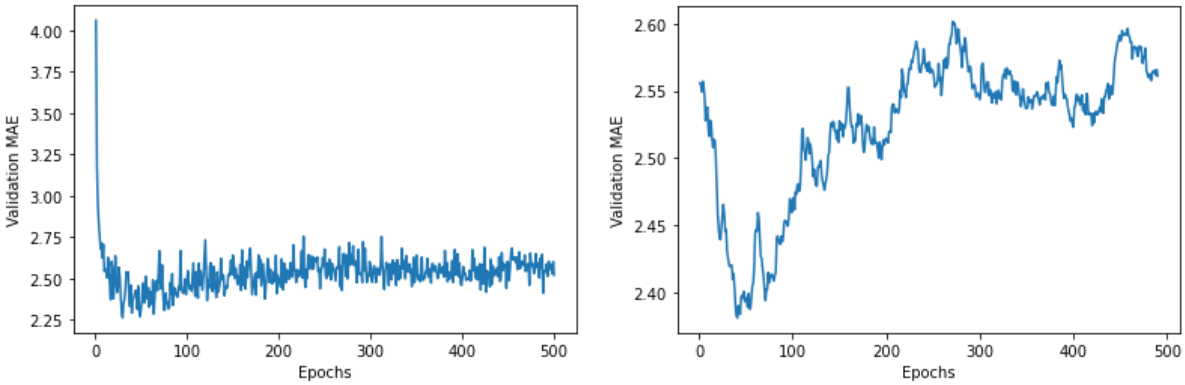
```
Out[4]: array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
               17.9, 23.1, 19.9, 15.7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
               32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
               23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
               12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7,  9.6, 31.5, 24.8, 19.1,
               22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3,
               15.6, 10.5,  6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5,  8.3,
               14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5,
               14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
               28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
               19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
               18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
               31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
               19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
               22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
               27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
                8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
               19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
               23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
```

```
In [5]: # model

        from keras import models
        from keras import layers

        def build_model():
            model = models.Sequential()

            model.add(layers.Dense(64, activation='relu',
                                   input_shape=(train_data.shape[1],)))
            model.add(layers.Dense(64, activation='relu'))
            model.add(layers.Dense(1))

            model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

            return model
```

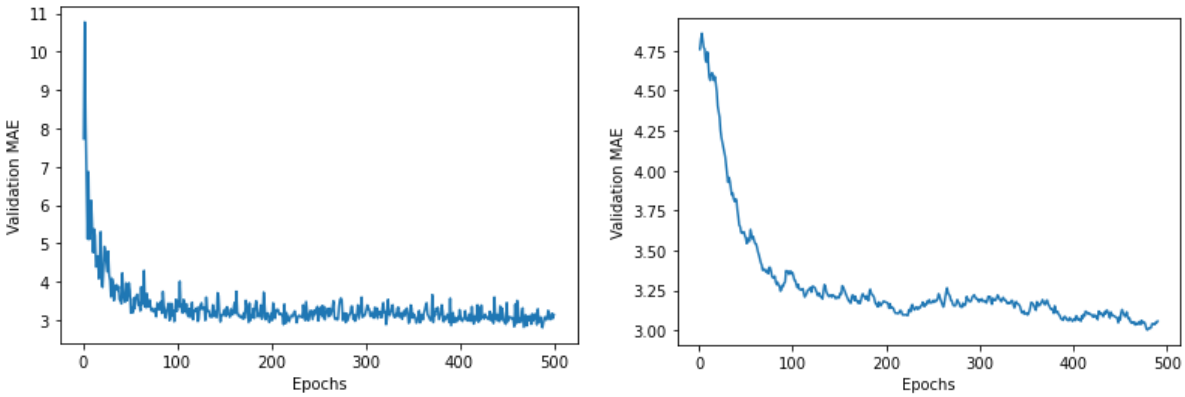
[그림 1-3-1] 데이터 정규화 과정이 생략된 소스코드

1-4. 실행 결과

왼쪽에는 지수이동평균 대체 전, 오른쪽에는 지수이동평균 대체 후 그래프를 첨부하였다.



[그림 1-4-1] 데이터 정규화 과정을 거친 학습곡선



[그림 1-4-2] 데이터 정규화 과정이 제외된 학습곡선

데이터 정규화 과정이 생략된 최종 모델의 훈련 결과를 살펴보았다.

```
In [14]: test_mae_score
Out[14]: 4.67919397354126
```

[그림 1-4-3] 최종 모델 훈련 결과

1-5. 실행 결과에 대한 설명 및 검토

지수이동평균 대체 전(그림 1-4-1, 그림 1-4-2 왼쪽)의 학습곡선 모두 변동이 심하지만, 데이터 정규화 적용의 학습곡선이 데이터 정규화 미적용의 학습곡선보다 변동이 더 크다. 또한, 데이터 정규화 적용보다 데이터 정규화 미적용의 검증 MAE는 더 높게 산출되었다. 따라서, 최종 결과에서 기존에는 약 2,799달러의 차이가 있었지만, 데이터 정규화 과정이 제외되면 약 4,679달러의 차이가 나타났다. 결국, 두 최종 결과 사이에는 약 2,000달러의 차이가 보였다.

1-6. 실습을 통한 이해 및 개선 방안

각 데이터 특성들이 상이한 스케일을 가진 경우, 데이터 정규화 과정이 생략되면 잘못된 최종 결과가 초래된다. 상대적으로 스케일이 큰 값이 스케일이 작은 값에 영향을 미치기 때문이다. 따라서, 각 특성 값들의 상대적인 크기 차이를 제거할 필요가 있다. 즉, 데이터 정규화 과정이 필수적으로 진행되어야 한다.

1-7. 결론

데이터 특성들이 서로 다른 스케일을 가졌을 때, 해당 값들을 신경망에 바로 주입시키면 문제가 발생한다. 각 데이터를 특성별로 정규화 하는 과정을 통해 이런 문제를 해결해야 된다. 데이터 정규화로 데이터의 상대적인 크기 차이를 제거하고 모델 구현을 진행하면 올바른 최종 결과를 얻을 수 있다.

2-1. 실습문제 정의

다른 정규화 방법을 찾아 구현하고 비교하시오(ex. 0~1 등).

2-2. 문제해결 방법 제시

z-distribution 대신 min-max normalization을 사용해본다.

2-3. 문서화된 소스 코드

min-max normalization은 (X – MIN) / (MAX – MIN) 공식으로 적용할 수 있다.

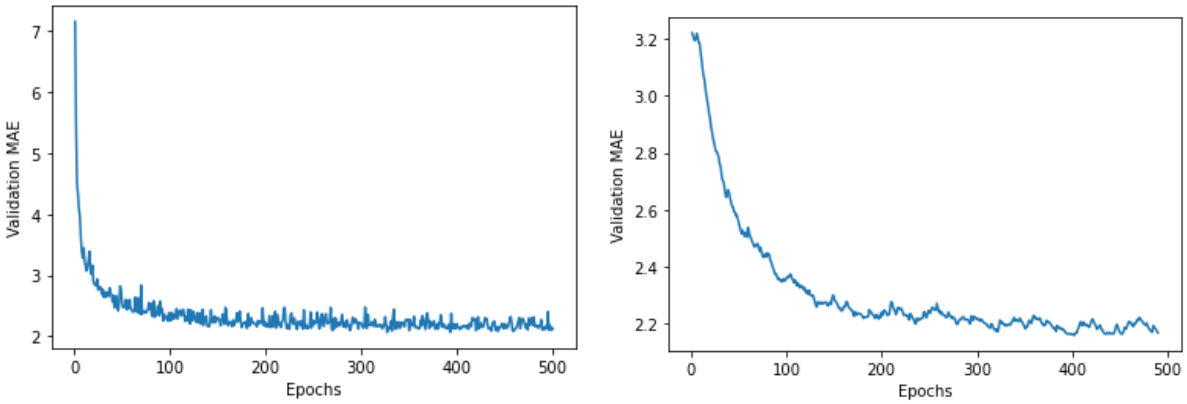
In [5]:
```
# feature scale normalization (train data)
# min-max normalization
# (X - MIN) / (MAX - MIN)

min_train_data = train_data.min(axis=0)
max_train_data = train_data.max(axis=0)

train_data -= min_train_data
temp_train_data = max_train_data - min_train_data

train_data /= temp_train_data
```

In [6]:
```
# feature scale normalization (test data)
# min-max normalization

test_data -= min_train_data
test_data /= temp_train_data
```

[그림 2-3-1] 데이터 정규화에서 min-max normalization이 사용된 소스코드

2-4. 실행 결과

왼쪽에는 지수이동평균 대체 전, 오른쪽에는 지수이동평균 대체 후 그래프를 첨부하였다. 두 학습곡선 모두 데이터 정규화 과정에서 min-max normalization이 사용되었다.



[그림 2-4-1] 지수이동평균 대체 전, 지수이동평균 대체 후 학습곡선

min-max normalization이 적용된 최종 모델의 훈련 결과를 살펴보았다.

```
In [16]: test_mae_score
Out[16]: 3.3511667251586914
```

[그림 2-4-2] 최종 모델 훈련 결과

2-5. 실행 결과에 대한 설명 및 검토

데이터 정규화 과정에서 min-max normalization을 적용한 결과 그래프의 곡선이 상당히 유해졌다. 하지만, 지수이동평균 대체 후의 학습곡선을 보면 기존보다 검증 MAE가 계속 줄어들어 과대적합을 파악하기 어렵다. 또한, 기존보다 검증 MAE의 값이 높게 산출되어 최종 결과에도 영향을 끼쳤다. 기존 z-distribution을 적용했을 땐 약 2,799달러의 차이가 있었으나, min-max normalization을 적용한 결과 약 3,351달러의 차이가 나타났다. 결국, 두 최종 결과 사이에는 약 500달러의 차이가 보였다.

2-6. 실습을 통한 이해 및 개선 방안

데이터 정규화 방법으로 min-max normalization을 사용하면 특성 값들을 강제적으로 0과 1 사이로 축소시킨다. 이런 경우, 데이터들의 상대적인 차이를 고려하지 않아 최종적으로 잘못된 결과가 산출된다. 따라서, 데이터의 평균과 표준편차를 활용하는 z-distribution을 데이터 정규화 방법으로 사용하는 것이 좋다.

2-7. 결론

서로 다른 스케일의 특성 값들을 정규화 하는 과정은 중요하지만, 그 전에 어떤 정규화 방법을 사용할지 결정하는 것도 중요하다. min-max normalization은 널리 쓰이는 정규화

방법 중 하나지만, 특성 값들을 강제적으로 0과 1 사이로 줄이기 때문에 원하는 결과를 도출하기 어려울 수 있다. 따라서, z-distribution을 통해 각 특성 값들의 상대적인 차이를 고려하며 데이터를 정규화 하는 게 바람직하다.

3-1. 실습문제 정의

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

→ model.compile(optimizer='rmsprop', loss='mae', metrics=['mse'])로 바꾼 결과를 비교분석하시오.

3-2. 문제해결 방법 제시

loss 함수는 mse에서 mae로, 모니터링 지표는 mae에서 mse로 변경한다.

3-3. 문서화된 소스 코드

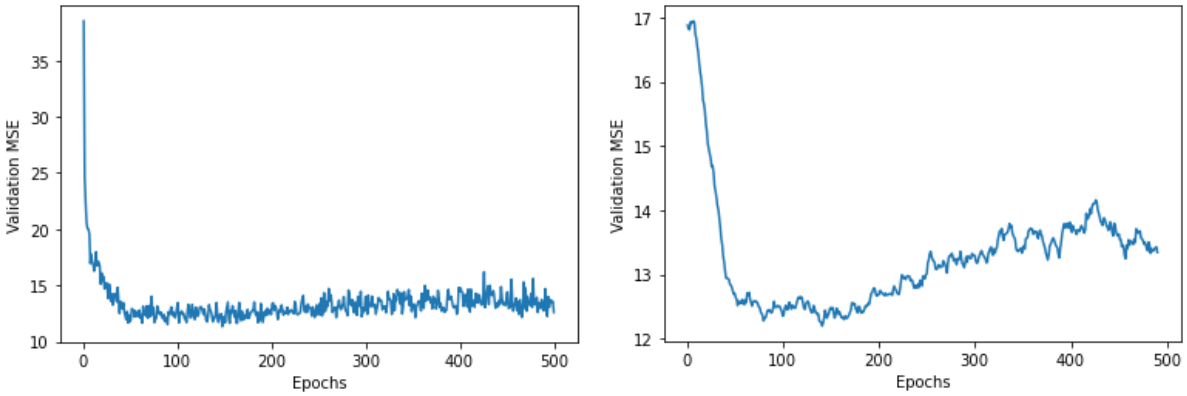
모델 compile에서 사용되는 loss 함수와 모니터링 지표를 서로 변경해준다.

In [14]:
```
# model

from keras import models
from keras import layers

def build_model():
    model = models.Sequential()

    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))

    model.compile(optimizer='rmsprop', loss='mae', metrics=['mse'])

    return model
```

[그림 3-3-1] loss 함수와 모니터링 지표가 변경된 소스코드

3-4. 실행 결과

왼쪽에는 지수이동평균 대체 전, 오른쪽에는 지수이동평균 대체 후 그래프를 첨부하였다. 두 학습곡선 모두 loss 함수는 mae, 모니터링 지표는 mse가 사용되었다.



[그림 3-4-1] 지수이동평균 대체 전, 지수이동평균 대체 후 학습곡선

loss 함수는 mae, 모니터링 지표는 mse가 적용된 최종 모델의 훈련 결과를 살펴보았다.

```
In [23]: test_mse_score
Out[23]: 18.524808883666992
```

[그림 3-4-2] 최종 모델 훈련 결과

3-5. 실행 결과에 대한 설명 및 검토

지수이동평균 대체 전 그래프의 변동이 loss 함수는 mse, 모니터링 지표는 mae를 사용했을 때보다 줄어들었다. 한편, 지수이동평균 대체 후 그래프를 통해 검증 MSE가 상당히 높은 값을 가지는 것을 볼 수 있다. 이는 최종 결과에도 영향을 준다. 결과를 살펴보면, 기존의 약 2,799달러의 차이가 약 18,524달러의 차이로 커진 것을 알 수 있다. 결국, 두 최종 결과 사이에 약 16,000달러의 차이가 나타났다.

3-6. 실습을 통한 이해 및 개선 방안

loss 함수와 모니터링 지표를 서로 변경하면 검증 MSE와 테스트 MSE에 영향을 미친다. 좋은 결과를 얻기 위해 loss 함수와 모니터링 지표를 그대로 사용하거나, 검증 그래프를 바탕으로 최종 모델 학습 epochs를 조절할 필요가 있다.

3-7. 결론

mse는 회귀에서 자주 사용되는 loss 함수로 오차 사이의 거리의 제곱을 계산한다. 한편, mae는 회귀에 일반적으로 사용되는 모니터링 지표로, 오차 사이의 정확한 거리를 계산한다. 각각의 함수는 주로 이용되는 경우가 있으므로 알맞게 활용하면 기대하는 결과를 얻을 수 있다.

4-1. 실습문제 정의

코드 3-31에서 지수이동평균 대신 5 epochs 이동평균을 사용한 결과를 보이시오.

- 자신과 이전 4개의 5개 포인트를 평균한 값을 출력
- 처음부터 5개보다 적을 때도 그 값들만의 평균을 출력

4-2. 문제해결 방법 제시

리스트의 길이가 5 미만인 경우와 5 이상인 경우 두 가지로 나누어서 코드를 작성한다.

4-3. 문서화된 소스 코드

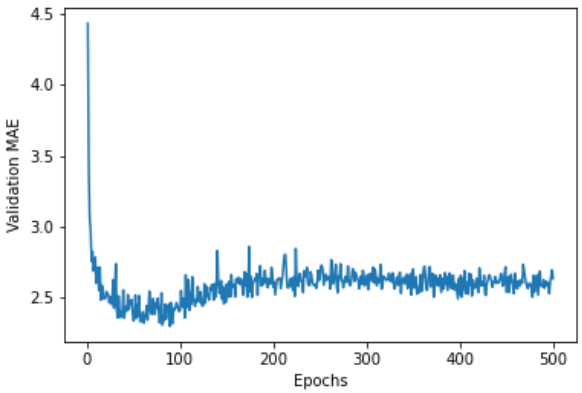
기존의 코드를 변형하여 5 epochs 이동평균을 사용해보았다.

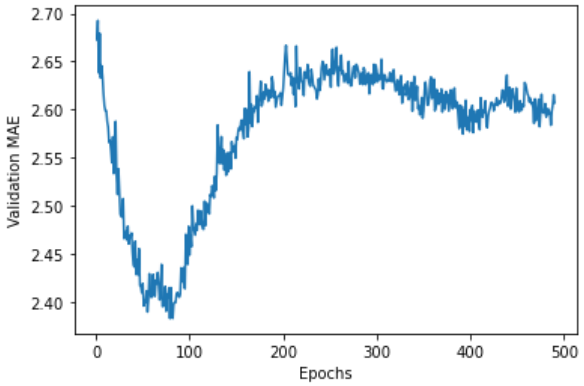
```
In [44]: # 처음 10개 데이터 포인트 제외한 검증 점수 그래프
         # 지수이동평균 대신 5 epochs 이동평균 사용

         def five_epochs_smooth_curve(points):
             smoothed_points = []

             for point in points:
                 if smoothed_points:
                     if len(smoothed_points) < 5:                                    # 포인트가 5개 미만인 경우
                         smoothed_points.append((point + sum(smoothed_points)) / (len(smoothed_points) + 1))
                     else:                                                           # 포인트가 5개 이상인 경우
                         temp_points = 0

                         for idx in range(len(smoothed_points)-1, len(smoothed_points)-5, -1):
                             temp_points += smoothed_points[idx]

                         smoothed_points.append((point + temp_points) / 5)
                 else:
                     smoothed_points.append(point)

             return smoothed_points

         smooth_mae_history = five_epochs_smooth_curve(average_mae_history[10:])

         plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)

         plt.xlabel('Epochs')
         plt.ylabel('Validation MAE')

         plt.show()
```

[그림 4-3-1] 5 epochs 이동평균 방식이 사용된 소스코드

4-4. 실행 결과

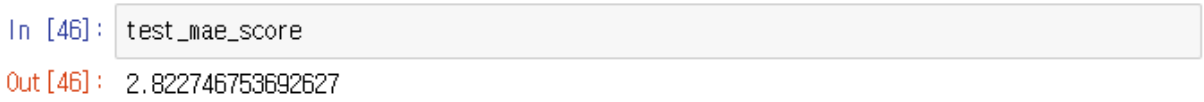
왼쪽에는 5 epochs 이동평균 대체 전, 오른쪽에는 5 epochs 이동평균 대체 후 그래프를 첨부하였다. 두 학습곡선 모두 지수이동평균은 사용되지 않았다.





[그림 4-4-1] 5 epochs 이동평균 대체 전, 5 epochs 이동평균 대체 후 학습곡선

5 epochs 이동평균이 적용된 최종 모델을 훈련 결과를 살펴보았다.



[그림 4-4-2] 최종 모델 훈련 결과

4-5. 실행 결과에 대한 설명 및 검토

5 epochs 이동평균을 사용하면 학습곡선의 변동이 부드럽긴 하지만, 지수이동평균보다는 변동이 심한 것을 볼 수 있다. 그러나, 그래프의 형태에는 큰 차이가 없고 거의 동일하다. 또한, 최종 훈련 결과도 비슷하게 나타났다. 기존에는 약 2,799달러의 차이가 있었지만, 현재는 약 2,822달러의 차이가 나타났다. 두 결과에는 약 20달러의 작은 차이만 보였다.

지수이동평균보다는 변동이 심한 것을 볼 수 있다. 그러나, 그래프의 형태에는 큰 차이가 없고 거의 동일하다. 5 epochs 이동평균 대체 후의 그래프를 통해 검증 MAE가 80번째 epoch 이후 줄어드는

4-6. 실습을 통한 이해 및 개선 방안

지수이동평균과 5 epochs 이동평균은 거의 동일한 결과를 산출했다. 그래프의 변동폭에 약간의 차이만 존재할 뿐, 형태와 최종 결과는 아주 비슷하였다. 5 epochs 이동평균 대체 후의 그래프를 통해서도 검증 MAE가 80번째 epoch 이후에 감소하는 게 멈추고 이 지점 이후에는 과대적합이 시작되는 걸 알 수 있다. 따라서, 두 방법 중 어느 방법을 사용하든 기대하는 최종 결과를 얻을 수 있다.

4-7. 결론

지수이동평균과 5 epochs 이동평균을 적용한 학습곡선에는 변동폭의 차이를 제외하고는 큰 차이가 없었다. 결국, 두 방법 모두 그래프의 변동을 조절하는 데에 유용한 방식이다. 따라서, 둘 중 어떤 방식을 활용해도 원하는 최종 모델 훈련 결과를 도출할 수 있다.