Machine Learning Practice

-11주차 chap.4-2

전자공학과

2022144007

김의진

Chap 4-2

1) 특징 미추출 case

```
'''mnist data 불러오는 for문'''
for i in range(1, 501):

      #주소 변수에 저장
    temp_name_0 = 'C:\\Users\\kim07\\Desktop\\Machinlearning_Workplace\\[배포용] MNIST Data\\0_' + str(i) + '.csv'
    temp_name_1 = 'C:\\Users\\kim07\\Desktop\\Machinlearning_Workplace\\[배포용] MNIST Data\\1_' + str(i) + '.csv'
    temp_name_2 = 'C:\\Users\\kim07\\Desktop\\Machinlearning_Workplace\\[배포용] MNIST Data\\2_' + str(i) + '.csv'

      #해당 주소 파일 dataframe 저장후 floating
    temp_image_0 = pd.read_csv(temp_name_0, header = None)
    temp_image_0 =  temp_image_0.to_numpy(dtype = 'float32')

    temp_image_1 = pd.read_csv(temp_name_1, header = None)
    temp_image_1 =  temp_image_1.to_numpy(dtype = 'float32')

    temp_image_2 = pd.read_csv(temp_name_2, header = None)
    temp_image_2 =  temp_image_2.to_numpy(dtype = 'float32')

      #각 데이터 array에 저장(flatten 이용해 모든 pixel 값 저장 28 x 28)
    class_0 = np.append(class_0, temp_image_0.flatten())
    class_1 = np.append(class_1, temp_image_1.flatten())
    class_2 = np.append(class_2, temp_image_2.flatten())

      #matrix로 만들어주고 data 라벨링 시켜줌
class_0 = np.resize(class_0, [784, 500])
class_0 = np.column_stack([class_0.T, y_0])
class_1 = np.resize(class_1, [784, 500])
class_1 = np.column_stack([class_1.T, y_1])
class_2 = np.resize(class_2, [784, 500])
class_2 = np.column_stack([class_2.T, y_2])

      #data set에 저장
data_set = np.row_stack([class_0, class_1, class_2])
```

1. MNIST(0, 1, 2에 대한 data) image 500개씩 총 1500개 받아옴

2. Flatten을 이용해 모든 pixel의 값을 x input으로 만듦

→ 한 이미지당 28 x 28 = 784개의 data

3. training, test set 7 : 3의 비율로 나눔

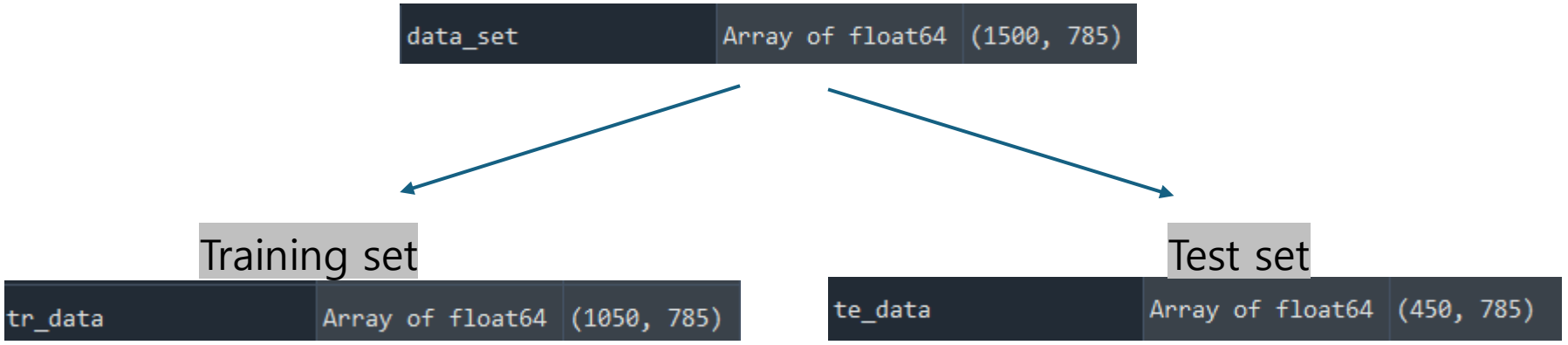
Chap 4-2

1) 특징 미추출 case

```
L = 16
epoch = 1000
LR = 0.01
tr_set = 7
va_set = 0
te_set = 3
```

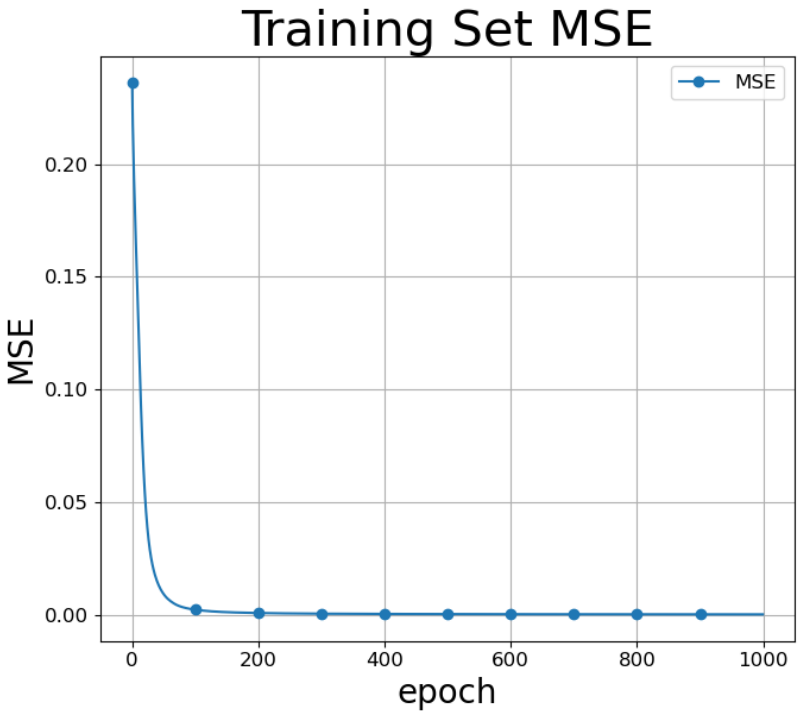
Hidden Layer의 Node 수: 16
Epoch : 1600
Learning Rate : 0.01
총 Input data 수(label 포함)
: 1500 x 785

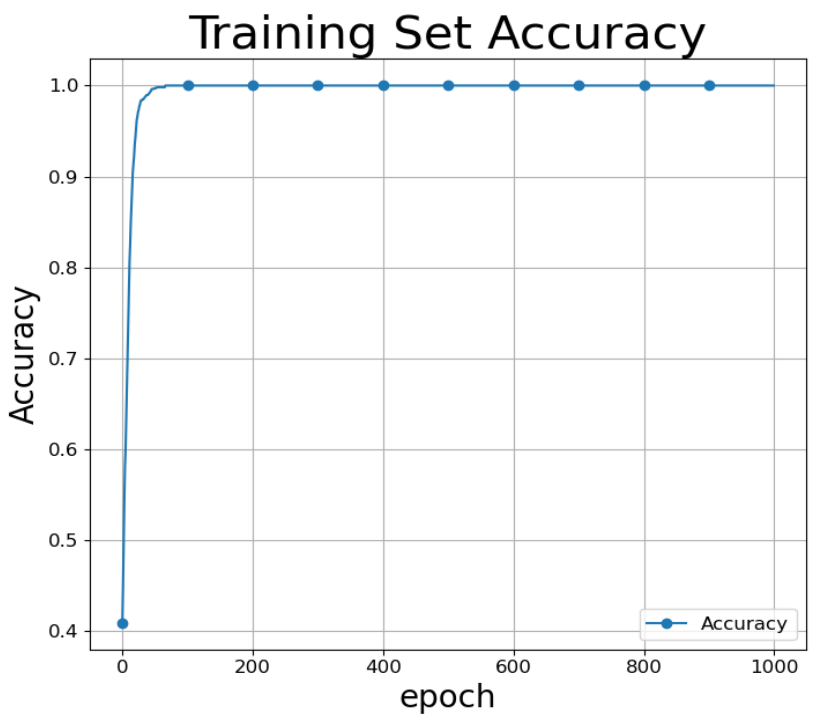
Training, test set 비율: 7 : 3



Chap 4-2

1) 특징 미추출 case





1. 저번주차의 batch size 1 two Layer Neural Network 이용
2. Hyper parameter tuning 으로 학습 시킴
3. MSE는 거의 0에 수렴, Accuracy는 1이 나온다.

→ Training set에 대하여 완벽히 학습한 모습

→ Test set을 이용해 검증

Chap 4-2

1) 특징 미추출 case

Test set
Confusion Matrix

	0	1	2	3
0	149	0	2	0.986755
1	0	157	0	1
2	1	0	141	0.992958
3	0.993333	1	0.986014	0.993333

precision

recall

accuracy

1. Training set으로 학습된 weight로 test set에 대해 forward propagation
2. Confusion matrix를 이용해 recall, precision, accuracy를 알아봄

→ Test set에 대해서도 거의 완벽한 분류를 해내고 있음

Chap 4-2

2) 특징 추출 case

```python
'''가로축 Expectation'''
def feature_1 (input_data):
    row_sum = np.sum(input_data, axis = 1)
    PSD = row_sum / np.sum(row_sum)
    E = np.sum(PSD * np.arange(len(row_sum)))

    return E

'''세로축 Variance'''
def feature_2 (input_data):
    column_sum = np.sum(input_data, axis = 0)
    PSD = column_sum / np.sum(column_sum)
    E = np.sum(PSD * np.arange(len(column_sum)))
    var = np.sum(PSD * (np.arange(len(column_sum)) - E) ** 2)

    return var

'''대각선 Expectation'''
def feature_3 (input_data):
    diag_components = np.diagonal(input_data)
    PSD = diag_components / np.sum(diag_components)
    E = np.sum(PSD * np.arange(len(diag_components)))

    return E

'''중앙의 0 개수'''
def feature_4 (input_data):
    center = input_data[10:18, 10:18]
    zero_cnt = np.sum(center == 0)

    return zero_cnt
```

```python
'''좌상단 0 개수'''
def feature_5 (input_data):
    center = input_data[1:10, 1:10]
    zero_cnt = np.sum(center == 0)

    return zero_cnt

'''대각성분 Variance'''
def feature_6 (input_data):
    diag_components = np.diagonal(input_data)
    PSD = diag_components / np.sum(diag_components)
    E = np.sum(PSD * np.arange(len(diag_components)))
    var = np.sum(PSD * (np.arange(len(diag_components)) - E) ** 2)

    return var

'''anti diogonal Variance'''
def feature_7 (input_data):
    Flip = np.fliplr(input_data)
    anti_diag_components = np.diagonal(Flip)
    PSD = anti_diag_components / np.sum(anti_diag_components)
    E = np.sum(PSD * np.arange(len(anti_diag_components)))
    var = np.sum(PSD * (np.arange(len(anti_diag_components)) - E) ** 2)

    return var

'''우 하단 0 개수'''
def feature_8 (input_data):
    center = input_data[15:21, 15:21]
    zero_cnt = np.sum(center == 0)

    return zero_cnt

'''top bottom rate'''
def feature_9 (input_data):
    top = np.sum(input_data[: 14, :])
    bottom = np.sum(input_data[14:, :])
    TBR = top / bottom

    return TBR
```

1. 9개의 특징 추출 방법 이용

→ 한 이미지당 9개의 data, 총 9 x 1500

Chap 4-2

2) 특징 추출 case

```python
# 각 class에 대한 data set의 배열을 만들고 크기를 다음음
x_0_set_f = np.array([], dtype = 'float32')
x_0_set_f = np.resize(x_0_set_f, (0, 9))
x_1_set_f = np.array([], dtype = 'float32')
x_1_set_f = np.resize(x_1_set_f, (0, 9))
x_2_set_f = np.array([], dtype = 'float32')
x_2_set_f = np.resize(x_2_set_f, (0, 9))

#데이터 파일 받고 특징추출하는 for 문
for i in range(1, 501):
    temp_name_f0 = 'C:\\Users\\kim07\\Desktop\\MachinLearning_Workplace\\[배포용] MNIST Data\\0_' + str(i) + '.csv'
    temp_name_f1 = 'C:\\Users\\kim07\\Desktop\\MachinLearning_Workplace\\[배포용] MNIST Data\\1_' + str(i) + '.csv'
    temp_name_f2 = 'C:\\Users\\kim07\\Desktop\\MachinLearning_Workplace\\[배포용] MNIST Data\\2_' + str(i) + '.csv'

    temp_image_f0 = pd.read_csv(temp_name_f0, header = None)
    temp_image_f1 = pd.read_csv(temp_name_f1, header = None)
    temp_image_f2 = pd.read_csv(temp_name_f2, header = None)

    temp_image_f0 = temp_image_f0.to_numpy(dtype = 'float32')
    temp_image_f1 = temp_image_f1.to_numpy(dtype = 'float32')
    temp_image_f2 = temp_image_f2.to_numpy(dtype = 'float32')

    # 0 data set에 대한 특징 추출
    x0_f0 = feature_1(temp_image_f0)
    x1_f0 = feature_2(temp_image_f0)
    x2_f0 = feature_3(temp_image_f0)
    x3_f0 = feature_4(temp_image_f0)
    x4_f0 = feature_5(temp_image_f0)
    x5_f0 = feature_6(temp_image_f0)
    x6_f0 = feature_7(temp_image_f0)
    x7_f0 = feature_8(temp_image_f0)
    x8_f0 = feature_9(temp_image_f0)

    # 1 data set에 대한 특징 추출
    x0_f1 = feature_1(temp_image_f1)
    x1_f1 = feature_2(temp_image_f1)
    x2_f1 = feature_3(temp_image_f1)
    x3_f1 = feature_4(temp_image_f1)
    x4_f1 = feature_5(temp_image_f1)
    x5_f1 = feature_6(temp_image_f1)
    x6_f1 = feature_7(temp_image_f1)
    x7_f1 = feature_8(temp_image_f1)
    x8_f1 = feature_9(temp_image_f1)

    # 2 data set에 대한 특징 추출
    x0_f2 = feature_1(temp_image_f2)
    x1_f2 = feature_2(temp_image_f2)
    x2_f2 = feature_3(temp_image_f2)
    x3_f2 = feature_4(temp_image_f2)
    x4_f2 = feature_5(temp_image_f2)
    x5_f2 = feature_6(temp_image_f2)
    x6_f2 = feature_7(temp_image_f2)
    x7_f2 = feature_8(temp_image_f2)
    x8_f2 = feature_9(temp_image_f2)

    #특징 합치기
    x_feature_f0 = np.array([x0_f0, x1_f0, x2_f0, x3_f0, x4_f0, x5_f0, x6_f0, x7_f0, x8_f0])
    x_feature_f0 = np.resize(x_feature_f0, (1, 9))
    x_0_set_f = np.concatenate((x_0_set_f, x_feature_f0), axis = 0)

    x_feature_f1 = np.array([x0_f1, x1_f1, x2_f1, x3_f1, x4_f1, x5_f1, x6_f1, x7_f1, x8_f1])
    x_feature_f1 = np.resize(x_feature_f1, (1, 9))
    x_1_set_f = np.concatenate((x_1_set_f, x_feature_f1), axis = 0)

    x_feature_f2 = np.array([x0_f2, x1_f2, x2_f2, x3_f2, x4_f2, x5_f2, x6_f2, x7_f2, x8_f2])
    x_feature_f2 = np.resize(x_feature_f2, (1, 9))
    x_2_set_f = np.concatenate((x_2_set_f, x_feature_f2), axis = 0)

#각 데이터 라벨링할 데이터 만들기
Y_0 = np.zeros([x_0_set_f.shape[0], 1])
Y_1 = np.ones([x_1_set_f.shape[0], 1])
Y_2 = np.array([2] * x_2_set_f.shape[0])

#각 데이터 셋 만들기
data0 = np.column_stack([x_0_set_f, Y_0])
data1 = np.column_stack([x_1_set_f, Y_1])
data2 = np.column_stack([x_2_set_f, Y_2])

#모든 class 데이터 합치기
data_f = np.row_stack([data0, data1, data2])

data_f = standard_data(data_f)                                    #특징 추출한 데이터 셋 정규화
```

Flattening해서 data set 만들 때 data 받는 코드
+
특징 추출한 후 한 data set에 모으는 코드

*특징 추출한 data set 정규화 꼭 해줘야 함
→ 특징 추출한 방법에 따라 값의 크기가 달라
다른 특징에 대한 값에 영향을 줄 수 있음

Chap 4-2

2) 특징 추출 case

정규화 x data

	0	1	2	3	4	5	6	7	8
0	13.5533	22.8111	13.1188	46	81	12.6761	37.0654	16	1.0893
1	13.6824	24.9043	14.1886	39	81	15.6639	26.3781	14	1.10572
2	14.2932	12.7841	14.1563	20	81	8.09746	18.7532	9	0.846866
3	14.0686	16.9763	14.0387	22	81	8.37016	22.6516	9	0.933485
4	14.292	27.2744	14.0711	46	77	21.8916	40.8539	17	0.987215
5	14.1873	26.6358	13.8214	47	74	26.9977	28.6849	31	1.03024
6	14.4947	24.9085	13.9854	19	81	12.437	24.0163	15	0.929931
7	13.6182	39.7984	14.5	64	70	43.6796	32.162	36	1.03693
8	14.3182	14.3447	13.7098	23	79	15.9927	18.3643	10	0.913065
9	14.2573	17.6454	14.9153	33	81	12.3954	27.5034	13	0.988374
10	14.2386	16.1449	13.7673	30	80	12.1037	33.0121	11	0.960838
11	13.5245	25.8299	14.0562	59	70	30.3227	27.6647	23	1.08777
12	13.849	15.5915	13.2405	31	80	11.395	23.8943	12	1.01146
13	13.7274	39.3633	14.6028	64	64	43.139	29.6769	29	0.966921
14	14.1875	33.6531	13.7339	46	75	19.8228	33.1151	15	0.996993
15	13.7764	20.2626	13.7491	48	80	15.7308	30.6706	20	1.01503
16	13.7373	27.5941	14.2656	35	80	13.7342	33.165	9	1.03311
17	13.6763	35.5944	14.0024	62	71	29.7208	33.4725	20	0.955952
18	13.9645	37.3068	13.658	61	78	20.3932	42.3085	15	0.969456
19	14.1283	13.4879	13.7529	27	81	11.4447	20.4475	11	0.935864
20	13.5763	19.9633	14.4532	47	80	16.5455	26.1328	9	0.944506
21	14.357	33.187	13.8548	60	74	30.6629	42.5954	27	0.945258
22	13.8058	32.0025	13.7934	64	68	31.2237	34.2408	23	0.949255

정규화 o data

	0	1	2	3	4	5	6	7	8
0	-1.5579	0.765192	2.82876	0.454754	0.235911	-0.750515	1.54302	-0.756795	-1.22727
1	1.24953	-0.892428	-0.128403	0.0491582	0.610571	-0.93556	-1.08038	1.41828	-0.12313
2	-0.869874	0.161903	1.33124	0.616993	0.0485809	-0.0395967	0.733763	-0.756795	-0.544573
3	1.63306	0.801015	1.73022	-0.437557	-1.26273	1.07606	0.414298	-1.74547	-1.53309
4	-0.151032	-0.275952	-0.519836	-0.194199	0.610571	-0.958103	-0.673803	1.61602	0.173452
5	-1.11952	-1.55369	-0.127873	-0.194199	0.610571	-0.884549	-1.15773	0.429611	0.655506
6	1.18217	0.256649	-0.569888	0.94147	0.423241	0.799392	0.670018	-0.163592	0.772072
7	1.29927	-0.126856	-0.338674	-0.031961	-1.82472	1.85624	0.878387	0.330744	-0.424217
8	-0.38991	1.85347	-0.971986	1.59042	0.423241	0.72287	2.11673	-0.460194	0.247476
9	1.76361	0.763472	0.531839	2.40162	-0.700739	1.81058	0.716171	-0.064725	-0.60057
10	-0.510221	-1.51902	-0.642141	-0.843153	0.610571	-0.845756	-1.13392	0.429611	0.348373
11	1.75291	-1.07149	-0.00204414	-0.599796	0.610571	-0.418898	-1.05859	-0.163592	-0.164749
12	0.655905	-1.23062	0.17593	-0.275319	0.610571	-0.923892	-1.11978	0.82508	0.733773
13	-0.568922	0.912516	1.1718	-0.275319	-0.513409	0.259439	1.36062	-1.6466	-1.26993
14	1.12866	0.114273	-0.314554	1.26595	0.610571	0.896422	0.285109	0.0341421	-0.188856
15	0.402964	0.34473	-0.168907	0.779231	0.423241	0.566696	1.08924	-1.35	0.0812743
16	-0.991534	0.100602	2.61557	-0.437557	-1.0754	-0.0789866	0.212367	-0.559061	-1.07836
17	-1.55217	1.51213	-1.01063	1.99602	-0.888069	1.11822	0.792709	-1.0534	0.885658
18	1.15697	-1.36588	-0.0312107	-1.08651	0.610571	-0.849586	-1.00194	0.429611	-0.12104
19	1.67142	-0.761932	-0.547429	0.0491582	0.235911	-0.955601	-0.83255	1.61602	0.522707
20	1.68673	-0.722562	0.196332	0.373635	0.423241	-0.950225	-1.133	1.31942	-0.294163
21	-1.35893	0.413813	0.269235	-0.518676	-3.13603	1.33118	-0.134782	-1.6466	-0.386131
22	0.557177	1.87849	0.361897	1.42819	0.235911	1.4487	1.15591	0.627345	1.25413

왼쪽 사진과 같이 큰 값을 가지는 특징이 있으면 상대적을 작은 특징을 무시해버릴 수도 있음

→ 정규화 시켜 scale을 비슷하게 맞춰줌

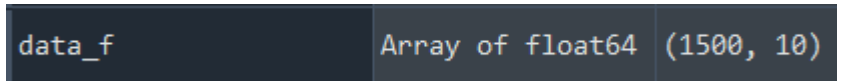
Chap 4-2

2) 특징 추출 case

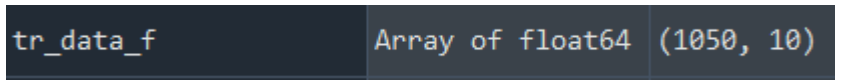
```
L = 16
epoch = 1000
LR = 0.01
tr_set = 7
va_set = 0
te_set = 3
```

Hidden Layer의 Node 수: 16
Epoch : 1600
Learning Rate : 0.01
총 Input data 수(label 포함)
: 1500 x 10

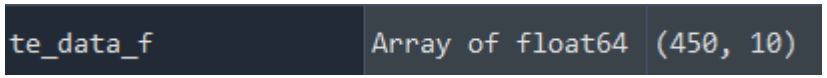
Training, test set 비율: 7 : 3



Training set



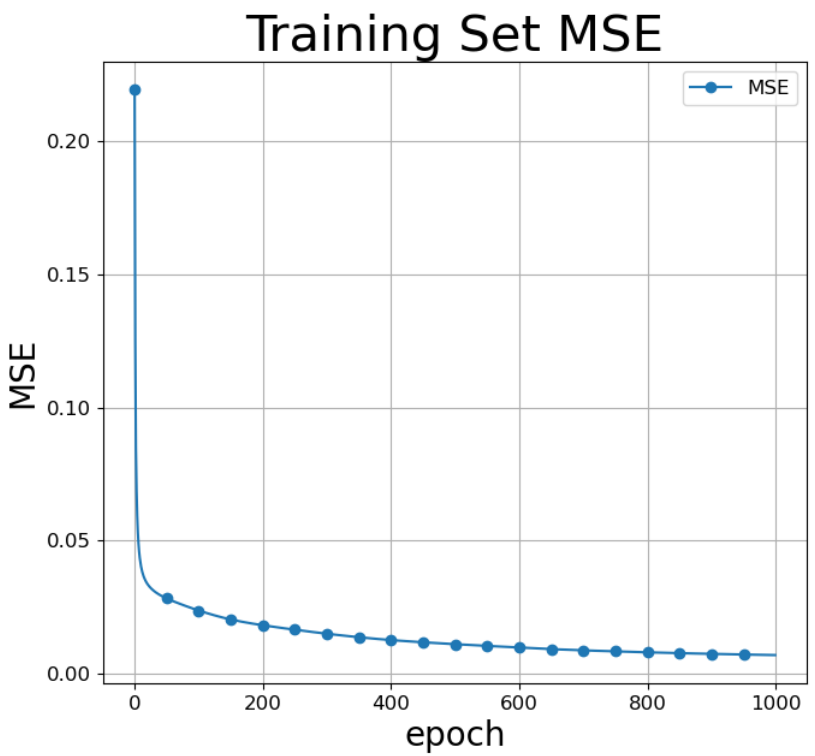
Test set

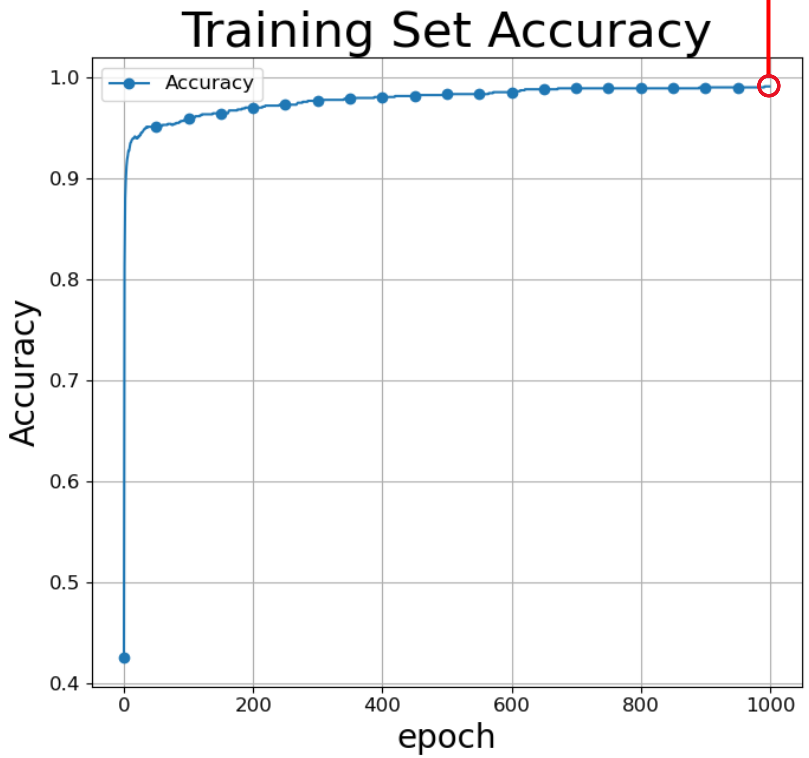


Chap 4-2

2) 특징 추출 case

999	float	1	0.9904761904761905





1. 특징 추출한 case에 대한Training set 학습
1) MSE가 0에 가까움
2) Accuracy역시 1에 거의 수렴함

→ overfitting 우려됨
→ Test set을 이용한 confusion matrix로 검증

Chap 4-2

2) 특징 추출 case

특징 수 5개

Test set
Confusion Matrix

	0	1	2	3
0	147	0	4	0.97351
1	1	128	2	0.977099
2	12	8	148	0.880952
3	0.91875	0.941176	0.961039	0.94

precision

recall

accuracy

특징 수 9개

	0	1	2	3
0	129	0	3	0.977273
1	0	152	2	0.987013
2	6	4	154	0.939024
3	0.955556	0.974359	0.968553	0.966667

특징 추출 방법 5개를 이용한 결과 (feature1, 2, 6, 7, 9)

1. 숫자 2에 대한 precision이 상대적으로 낮음
2. 숫자 0에 대한 recall이 상대적으로 낮음

특징 추출 방법 9개를 이용한 결과 (All feature)

1. 숫자 2에 대한 precision, 0에 대한 recall이 개선 됨

→ 특징 9개를 이용할 때 recall, precision값, accuracy 이 올랐으며 training test set accuracy에 차이가 별로 없어 overfitting없는 걸로 보임

→ 상황에 따라(효율 고려) 특징 수 채택

Chap 4-2

3) 특징 미 추출 case vs 특징 추출 case

특징 미 추출

	0	1	2	3
0	149	0	2	0.986755
1	0	157	0	1
2	1	0	141	0.992958
3	0.993333	1	0.986014	0.993333

특징 추출

	0	1	2	3
0	129	0	3	0.977273
1	0	152	2	0.987013
2	6	4	154	0.939024
3	0.955556	0.974359	0.968553	0.966667

특징 미 추출 case의 precision, recall, accuracy 모두 더 좋음

→ 특징 미 추출 case가 더 좋은 성능 가짐

Chap 4-2

3) 특징 미 추출 case vs 특징 추출 case

특징 미 추출

data_set	Array of float64	(1500, 785)

특징 추출

data_f	Array of float64	(1500, 10)

특징 미 추출 case가 연산량이 많음

→ 학습 시간이 상대적으로 더 느림

상황에 따라 두 경우 중 하나 골라서 사용 가능

1) 컴퓨터 성능에 제한이 없는 경우 → 특징 미 추출

1-1) 성능이 아무리 좋아도 data set양이 많거나 deep learning같은 복잡한 연산할 때 제약이 생길 수 있음

2) 컴퓨터 성능에 제한이 있을 경우 → 특징 추출

2-1) 특징 추출을 통해 효율을 추구할 수 있음