전자공학과 2020142001 곽종근

8주차 인공신경망 Two-Layer Neural Network Chap.4 - 2주차 실습

제출일: 2024.05.22.

Chap.4 전체 코드

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 시그모이드 함수 선언
def sigmoid(x):
    return 1 / (1 + np.exp(-x))  # 시그모이드 함수 정의

# 시그모이드 함수의 미분
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))  # 시그모이드 함수의 미분

# 순전파 함수
def forward_propagation(x_with_dummy, v, w):
    A = v @ x_with_dummy.T  # 입력과 가중치 v의 곱
    b = sigmoid(A)  # 활성화 함수 적용
    b_with_dummy = np.vstack([b, np.ones([1, len(x_with_dummy)])])  # 더미 변수를 포함한 b 생성
    B = w @ b_with_dummy  # 은닉층 출력과 가중치 w의 곱
    y_hat = sigmoid(B)  # 활성화 함수 적용
    return A, b, b_with_dummy, B, y_hat  # 순전파 결과 반환

# 역전파 함수
def backward_propagation(x_with_dummy, y_one_hot, A, b, b_with_dummy, B, y_hat, v, w):
    error = y_hat - y_one_hot.T  # 예측값과 실제값의 차이
    wmse = (error * sigmoid_derivative(B)) @ b_with_dummy.T / len(x_with_dummy)  # 출력층 가중치의 변화량 계산
    vmse = ((w[:, :-1].T @ (error * sigmoid_derivative(B))) * sigmoid_derivative(A)) @ x_with_dummy / len(x_with_dummy)
    return wmse, vmse  # 가중치 변화량 반환

# 데이터 분할 함수
def aug_data(data, train_ratio, test_ratio):
    # 데이터를 분할하는 것이므로 분할한 것들의 합이 1이 나와야 함
    assert train_ratio + test_ratio == 1  # 학습 데이터와 테스트 데이터 비율의 합이 1인지 확인

    # 데이터의 총 개수
    total_samples = len(data)

    # 각 세트의 크기 계산
    train_size = int(total_samples * train_ratio)

    # 데이터를 랜덤하게 섞음
    np.random.shuffle(data)

    # 데이터 분할
    train_set = data[:train_size]
    test_set = data[train_size:]

    return train_set, test_set  # 학습 세트와 테스트 세트 반환
```

```python
# confusion matrix 계산 함수
def compute_confusion_matrix(y_true, y_pred, num_classes):
    confusion_matrix = np.zeros((num_classes, num_classes))  # 초기화된 혼동 행렬
    for i in range(len(y_true)):
        row_index = int(y_pred[i]) - 1  # 예측값의 인덱스 계산
        col_index = int(y_true[i]) - 1  # 실제값의 인덱스 계산
        confusion_matrix[row_index, col_index] += 1  # 혼동 행렬 업데이트
    return confusion_matrix  # 혼동 행렬 반환

# 데이터 불러오기
fold_dir = "C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년 1학기\\머신러닝실습\\Machine-Learning\\NN_data.csv"
temp_data = pd.read_csv(fold_dir)  # CSV 파일에서 데이터 로드
temp_data = temp_data.to_numpy()  # numpy 배열로 변환

# 데이터 분할
train_data, test_data = aug_data(temp_data, 0.7, 0.3)  # 데이터를 70% 학습 데이터와 30% 테스트 데이터로 분할

# 데이터 분리
x_train = train_data[:, :3]  # 학습 데이터의 입력 값
y_train = train_data[:, 3].reshape(-1, 1)  # 학습 데이터의 출력 값

x_test = test_data[:, :3]  # 테스트 데이터의 입력 값
y_test = test_data[:, 3].reshape(-1, 1)  # 테스트 데이터의 출력 값

# 입력 속성 수와 출력 클래스 수 추출
M = x_train.shape[1]  # 입력 속성 수
output_size = len(np.unique(y_train))  # 출력 클래스 수

# hidden layer의 노드 수
hidden_size = 5  # 은닉층의 노드 수 설정

# weight 초기화
v = np.random.rand(hidden_size, M + 1)  # 은닉층 가중치 초기화
w = np.random.rand(output_size, hidden_size + 1)  # 출력층 가중치 초기화

# 학습 파라미터 설정
learning_rate = 0.1  # 학습률
epochs = 40  # 에포크 수

# One-Hot Encoding
y_train_one_hot = np.zeros((len(y_train), output_size))  # 초기화된 원-핫 인코딩 배열
for i in range(len(y_train)):
    y_train_one_hot[i, int(y_train[i]) - 1] = 1  # 원-핫 인코딩
```

Chap.4 전체 코드

```python
# 데이터에 더미 변수 추가
x_train_with_dummy = np.hstack((x_train, np.ones((len(x_train), 1))))  # 더미 변수를 포함한 학습 입력 데이터
x_test_with_dummy = np.hstack((x_test, np.ones((len(x_test), 1))))  # 더미 변수를 포함한 테스트 입력 데이터
total_samples = len(x_train)  # 학습 데이터의 총 샘플 수

# 정확도와 MSE를 저장할 리스트 초기화
accuracy_list = []  # 에포크 별 정확도를 저장할 리스트
mse_list = []  # 에포크 별 MSE를 저장할 리스트

# 최적의 가중치를 저장할 변수 초기화
best_accuracy = 0  # 최적의 정확도 초기화
best_v = v  # 최적의 은닉층 가중치 초기화
best_w = w  # 최적의 출력층 가중치 초기화

# 학습
for epoch in range(epochs):
    for step in range(total_samples):
        A, b, b_with_dummy, B, y_hat = forward_propagation(x_train_with_dummy[step:step+1], v, w)  # 순전파
        wmse, vmse = backward_propagation(x_train_with_dummy[step:step+1], y_train_one_hot[step:step+1], A, b, b_with_dummy, B, y_hat, v, w)  #
        w -= learning_rate * wmse  # 출력층 가중치 업데이트
        v -= learning_rate * vmse  # 은닉층 가중치 업데이트

    # 테스트 데이터에 대해 정확도 계산
    A_test, b_test, b_with_dummy_test, B_test, y_hat_test = forward_propagation(x_test_with_dummy, v, w)  # 순전파
    y_hat_test_index = np.argmax(y_hat_test, axis=0) + 1  # 예측값의 클래스 인덱스
    test_accuracy = np.mean(y_hat_test_index == y_test.flatten())  # 테스트 정확도 계산

    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy  # 최적의 정확도 갱신
        best_v = np.copy(v)  # 최적의 은닉층 가중치 갱신
        best_w = np.copy(w)  # 최적의 출력층 가중치 갱신

    A_train, b_train, b_with_dummy_train, B_train, y_hat_train = forward_propagation(x_train_with_dummy, v, w)  # 순전파
    predicted_labels = np.argmax(y_hat_train, axis=0) + 1  # 예측값의 클래스 인덱스
    accuracy = np.mean(predicted_labels == y_train.flatten())  # 학습 정확도 계산
    accuracy_list.append(accuracy)  # 학습 정확도 저장

    mse = np.mean((y_hat_train - y_train_one_hot.T) ** 2)  # 학습 MSE 계산
    mse_list.append(mse)  # 학습 MSE 저장

# 최적의 가중치로 모델 업데이트
v = best_v  # 최적의 은닉층 가중치 업데이트
w = best_w  # 최적의 출력층 가중치 업데이트
```

```python
# confusion matrix 계산
confusion_matrix = compute_confusion_matrix(y_test, y_hat_test_index, output_size)  # 혼동 행렬 계산

print(confusion_matrix)  # 혼동 행렬 출력

# 그래프 출력
plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), accuracy_list, label='Accuracy', color='blue')  # 정확도 그래프
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy over epochs')
plt.legend()
plt.grid(True)
plt.ylim(0, 1)

plt.subplot(1, 2, 2)
plt.plot(range(1, epochs+1), mse_list, label='MSE', color='red')  # MSE 그래프
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.title('MSE over epochs')
plt.legend()
plt.grid()
plt.ylim(0, 1)

plt.show()  # 그래프 표시
```

Chap.4 Error Back-Propagation 알고리즘 구현

```python
# 순전파 함수
def forward_propagation(x_with_dummy, v, w):
    A = v @ x_with_dummy.T  # 입력과 가중치 v의 곱
    b = sigmoid(A)  # 활성화 함수 적용
    b_with_dummy = np.vstack([b, np.ones([1, len(x_with_dummy)])])  # 더미 변수를 포함한 b 생성
    B = w @ b_with_dummy  # 은닉층 출력과 가중치 w의 곱
    y_hat = sigmoid(B)  # 활성화 함수 적용
    return A, b, b_with_dummy, B, y_hat  # 순전파 결과 반환

# 역전파 함수
def backward_propagation(x_with_dummy, y_one_hot, A, b, b_with_dummy, B, y_hat, v, w):
    error = y_hat - y_one_hot.T  # 예측값과 실제값의 차이
    wmse = (error * sigmoid_derivative(B)) @ b_with_dummy.T / len(x_with_dummy)  # 출력층 가중치의 변화량 계산
    vmse = ((w[:, :-1].T @ (error * sigmoid_derivative(B))) * sigmoid_derivative(A)) @ x_with_dummy / len(x_with_dummy)  # 은닉층 가중치의 변화량 계산
    return wmse, vmse  # 가중치 변화량 반환
```

히든노드=10
러닝레이트=0.01
Epochs=1000으로 설정시에
V = 10,4의 크기로
W = 6,11의 크기로 나타난다.

v - NumPy object array

	0	1	2	3
0	0.804664	0.536094	0.608871	0.627197
1	0.642328	0.256796	0.512402	0.300336
2	0.146378	-0.0908283	-0.108853	0.648488
3	0.295636	0.312304	0.5232	0.652167
4	0.54307	0.427392	0.756714	0.0481782
5	0.883037	0.889669	0.717363	0.766311
6	0.26713	0.989855	0.302966	0.0129478
7	0.839578	0.275428	0.378084	0.48953
8	-0.120017	-0.192171	-0.0576824	0.117221
9	0.244862	0.121171	0.578868	0.454762

w - NumPy object array

	0	1	2	3	4	5	6	7	8
0	0.735403	0.700773	0.78495	0.804891	0.17976	0.888939	0.0309019	0.318165	0.530962
1	0.388739	0.807005	0.357667	0.463494	0.473334	0.648222	0.625687	0.629982	0.185298
2	0.716464	0.751882	0.745489	0.730283	0.0670007	0.606086	0.747589	0.708323	0.326753
3	-0.278949	-0.0799522	0.382191	-0.389266	-0.133966	-0.18304	0.544978	-0.199709	0.382793
4	0.109862	0.907559	0.593505	0.709128	0.444528	0.726537	0.222927	0.633161	0.606185
5	0.399568	0.947501	0.967422	0.563131	0.654964	0.309006	0.294418	0.644081	0.690528

v	Array of float64	(10, 4)
w	Array of float64	(6, 11)

Chap.4 Two-Layer Neural Network – "Training"

```python
# 데이터 분할 함수
def aug_data(data, train_ratio, test_ratio):
    # 데이터를 분할하는 것이므로 분할한 것들의 합이 1이 나와야 함
    assert train_ratio + test_ratio == 1  # 학습 데이터와 테스트 데이터 비율의 합이 1인지 확인

    # 데이터의 총 개수
    total_samples = len(data)

    # 각 세트의 크기 계산
    train_size = int(total_samples * train_ratio)

    # 데이터를 랜덤하게 섞음
    np.random.shuffle(data)

    # 데이터 분할
    train_set = data[:train_size]
    test_set = data[train_size:]

    return train_set, test_set  # 학습 세트와 테스트 세트 반환
# 데이터 분할
train_data, test_data = aug_data(temp_data, 0.7, 0.3)  # 데이터를 70% 학습 데이터와 30% 테스트 데이터로 분할
```

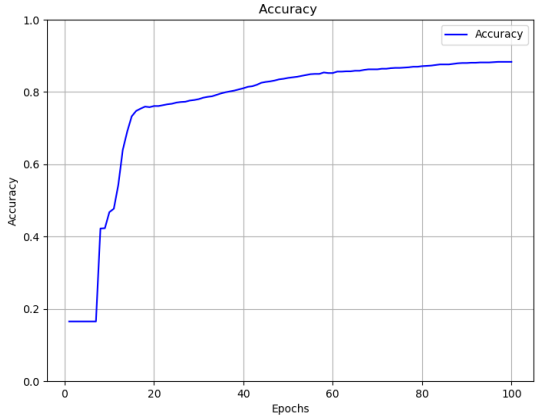
이 데이터 분할 함수를 이용해 기존 데이터를 셔플 후 7:3비율로 train_set과 test_set으로 분리

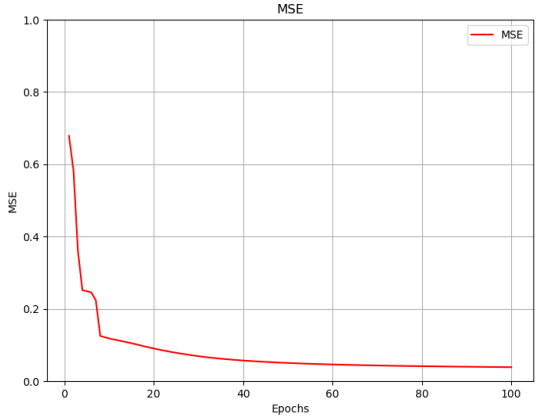
Chap.4 Two-Layer Neural Network – "Training"

```
# 학습
for epoch in range(epochs):
    for step in range(total_samples):
        A, b, b_with_dummy, B, y_hat = forward_propagation(x_train_with_dummy[step:step+1], v, w)  # 순전파
        wmse, vmse = backward_propagation(x_train_with_dummy[step:step+1], y_train_one_hot[step:step+1], A, b, b_with_dummy, B, y_hat, v, w)  # 역전파
        w -= learning_rate * wmse  # 출력층 가중치 업데이트
        v -= learning_rate * vmse  # 은닉층 가중치 업데이트


    A_train, b_train, b_with_dummy_train, B_train, y_hat_train = forward_propagation(x_train_with_dummy, v, w)  # 순전파
    predicted_labels = np.argmax(y_hat_train, axis=0) + 1  # 예측값의 클래스 인덱스
    accuracy = np.mean(predicted_labels == y_train.flatten())  # 학습 정확도 계산
    accuracy_list.append(accuracy)  # 학습 정확도 저장
    # 테스트 데이터에 대해 정확도 계산
    A_test, b_test, b_with_dummy_test, B_test, y_hat_test = forward_propagation(x_test_with_dummy, v, w)  # 순전파
    y_hat_test_index = np.argmax(y_hat_test, axis=0) + 1  # 예측값의 클래스 인덱스
    test_accuracy = np.mean(y_hat_test_index == y_test.flatten())  # 테스트 정확도 계산

    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy  # 최적의 정확도 갱신
        best_v = np.copy(v)  # 최적의 은닉층 가중치 갱신
        best_w = np.copy(w)  # 최적의 출력층 가중치 갱신
    mse = np.mean((y_hat_train - y_train_one_hot.T) ** 2)  # 학습 MSE 계산
    mse_list.append(mse)  # 학습 MSE 저장
```

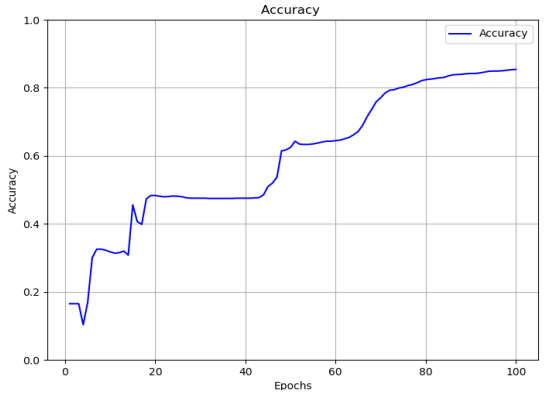
For문을 통해 처음 for문은 에폭을 그 다음은 스텝에 대한 for문을 나타낸다. 여기서 스텝에서는 데이터를 모두 다 확인 했을 때 1에폭이 실행되는 방향으로 코딩하였다.
여기서 최적의 가중치를 위해 if문으로 최적의 가중치를 저장하는 코드를 추가하였다.
각 스텝에 대한 값으로 정확도를 계산하므로 2번째 for문인 step 안으로 정확도와 mse를 계산하는 코드 삽입

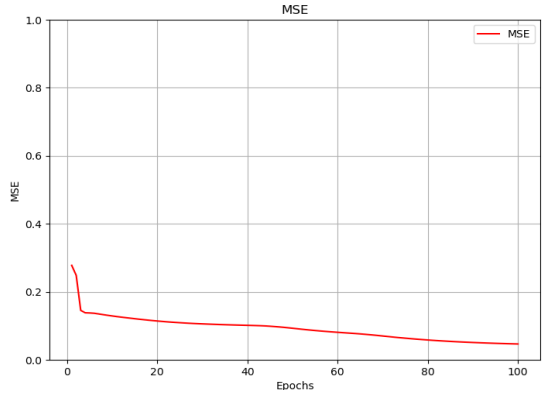
Chap.4 Two-Layer Neural Network – "Training"-히든노드 변경





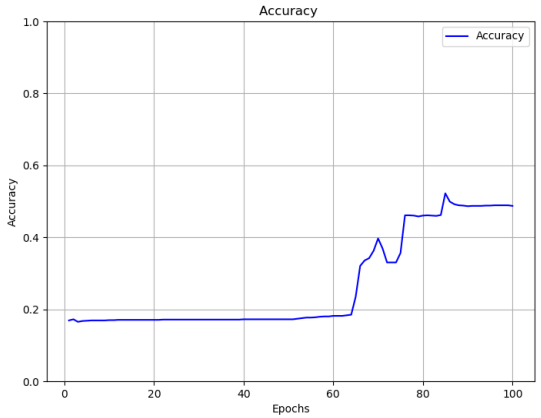
Hidden_Node수 : 10
Learning_rate = 0.01
Epochs = 100

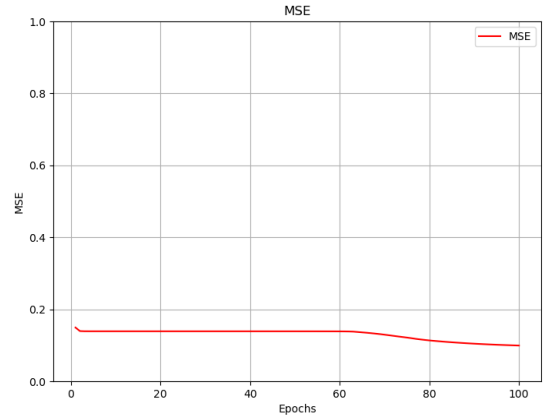




Hidden_Node수 : 7
Learning_rate = 0.01
Epochs = 100

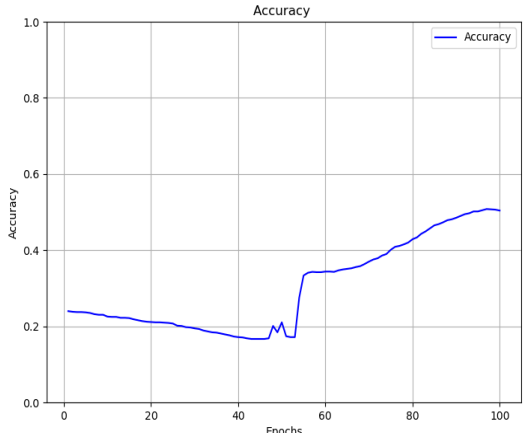
Hidden Node수만 변경했을 때
일단 값이 매우 작으면 정확도가
낮게 나오는 것을 알 수 있고,
어느정도 정확한 값을 주지 않으면
값이 매우 요동치는 것을 알 수 있다.

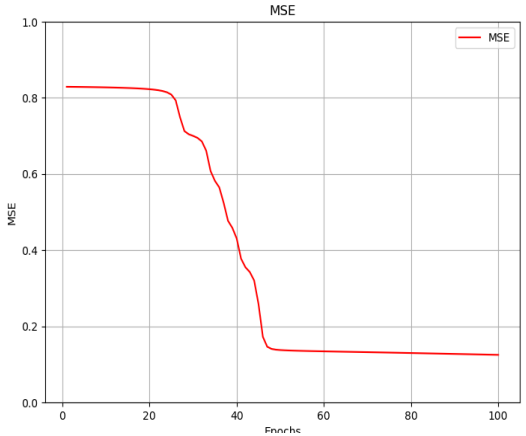




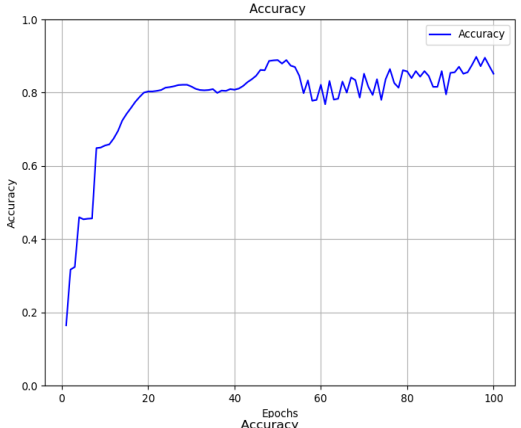
Hidden_Node수 : 3
Learning_rate = 0.01
Epochs = 100

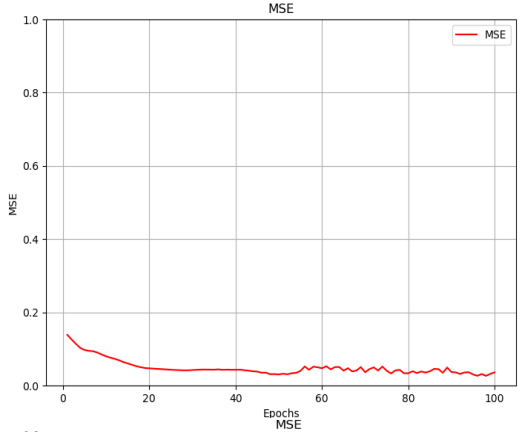
Chap.4 Two-Layer Neural Network – “Training”- LR변경





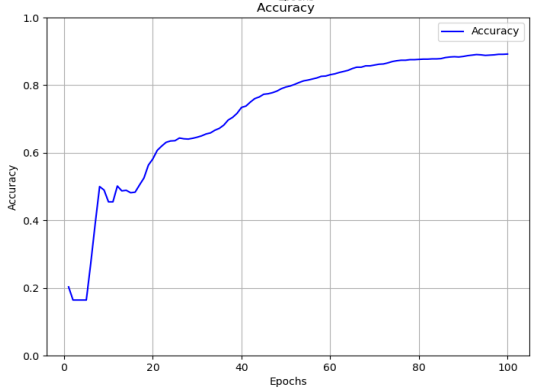
Hidden_Node수 : 10
Learning_rate = 0.001
Epochs = 100

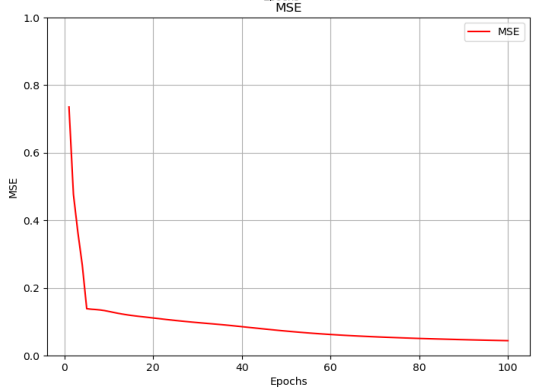




Hidden_Node수 : 10
Learning_rate = 0.1
Epochs = 100

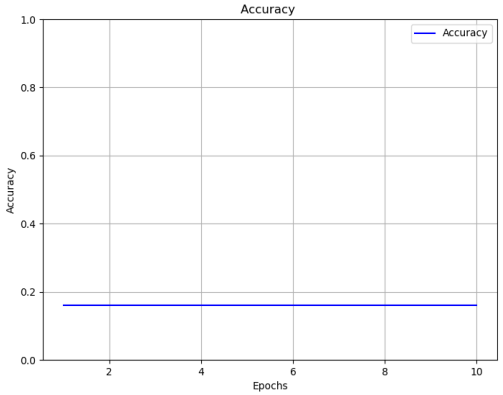
Learning rate만 변경했을 때
일단 값이 매우 작으면 정확도가
낮게 나오는 것을 알 수 있고,
어느정도 정확한 값을 주지 않으면
값이 매우 요동치는 것을 알 수 있다.
Mse는 0.1에서 일정하게 나타난다.

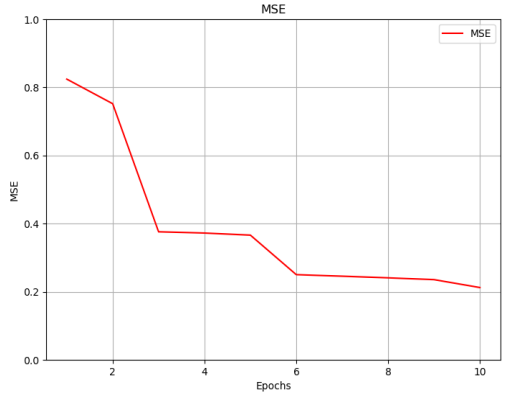




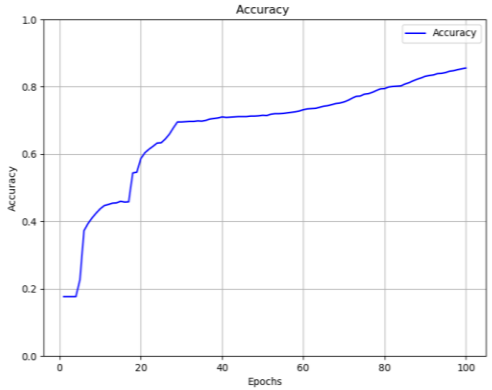
Hidden_Node수 : 10
Learning_rate = 0.01
Epochs = 100

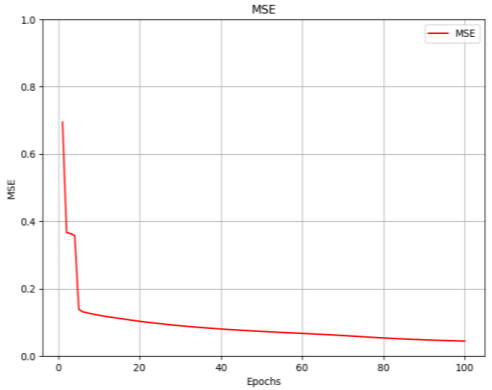
Chap.4 Two-Layer Neural Network – “Training” – Epochs변경





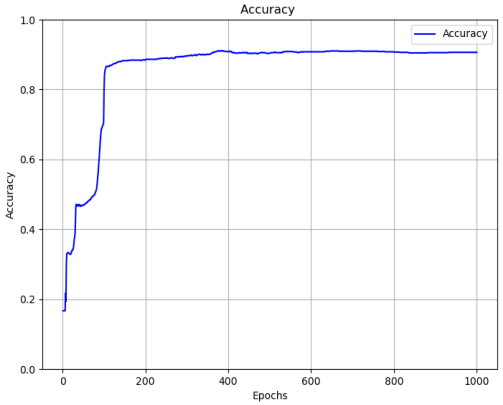
Hidden_Node수 : 10
Learning_rate = 0.01
Epochs = 10

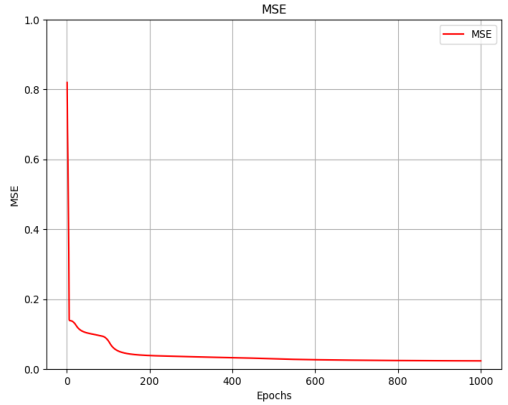




Hidden_Node수 : 10
Learning_rate = 0.01
Epochs = 100

Epochs수만 변경했을 때
일단 값이 매우 작으면 정확도가
낮게 나오거나 측정이 안 되는 것을
알 수 있고, 매우 큰 값을 주면 실행
속도가 매우 오래 걸리지만 값이 보다
정확히 나온다.





Hidden_Node수 : 10
Learning_rate = 0.01
Epochs = 1000

Chap.4 Two-Layer Neural Network – "Training,Test"- Confusion Matrix

```python
# confusion matrix 계산 함수
def compute_confusion_matrix(y_true, y_pred, num_classes):
    confusion_matrix = np.zeros((num_classes, num_classes))  # 초기화된 혼동 행렬
    for i in range(len(y_true)):
        row_index = int(y_pred[i]) - 1  # 예측값의 인덱스 계산
        col_index = int(y_true[i]) - 1  # 실제값의 인덱스 계산
        confusion_matrix[row_index, col_index] += 1  # 혼동 행렬 업데이트
    return confusion_matrix  # 혼동 행렬 반환

# confusion matrix 계산
confusion_matrix = compute_confusion_matrix(y_test, y_hat_test_index, output_size)  #
```

confusion_matrix - NumPy object array

	0	1	2	3	4	5
0	81	4	0	0	0	1
1	8	72	7	0	0	0
2	0	6	78	4	0	0
3	0	0	1	90	2	0
4	0	0	0	2	81	4
5	4	0	0	0	1	94

```python
# 학습
for epoch in range(epochs):
    for step in range(total_samples):
        A, b, b_with_dummy, B, y_hat = forward_propagation(x_train_with_dummy[step:step+1], v, w)  # 순전파
        wmse, vmse = backward_propagation(x_train_with_dummy[step:step+1], y_train_one_hot[step:step+1], A, b, b_with_dummy, B, y_hat, v, w)  # 역전파
        w -= learning_rate * wmse  # 출력층 가중치 업데이트
        v -= learning_rate * vmse  # 은닉층 가중치 업데이트


    A_train, b_train, b_with_dummy_train, B_train, y_hat_train = forward_propagation(x_train_with_dummy, v, w)  # 순전파
    predicted_labels = np.argmax(y_hat_train, axis=0) + 1  # 예측값의 클래스 인덱스
    accuracy = np.mean(predicted_labels == y_train.flatten())  # 학습 정확도 계산
    accuracy_list.append(accuracy)  # 학습 정확도 저장
    # 테스트 데이터에 대해 정확도 계산
    A_test, b_test, b_with_dummy_test, B_test, y_hat_test = forward_propagation(x_test_with_dummy, v, w)  # 순전파
    y_hat_test_index = np.argmax(y_hat_test, axis=0) + 1  # 예측값의 클래스 인덱스
    test_accuracy = np.mean(y_hat_test_index == y_test.flatten())  # 테스트 정확도 계산
```

코드는 이처럼 step에 대한 for문에 넣어주고 test는 순전파만 진행하여 계산해준다.
위 함수를 통해 Confusion Matrix를 구하고 LR = 0.05, Epoch = 300, Hidden Node = 10에서
결과는 이러하게 나온다.