Assignment2

202212080 강주현

1.1 다음 numpy 의 dot 의 정의를 이해하고, 예제를 만들어 설명하시오.

-> numpy.dot 함수는 두 배열의 점곱을 계산한다.

1 차원 배열(벡터)끼리는 벡터의 내적을, 2 차원 배열(행렬)끼리는 행렬 곱셈을 수행한다.

그리고 다차원 배열의 경우, 마지막 두 축을 제외한 모든 축을 따라 확장하여 계산한다.

- 1 차원 배열의 벡터의 내적의 예시

```python
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

result = np.dot(a, b)
print(result)  #결과 : 32

```

> 두 벡터의 내적 : 1 * 4 + 2 * 5 + 3 * 6 = 32 이어서 결과 32

- 2 차원 배열 행렬 곱셈

```python
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

result = np.dot(a, b)
print(result)

# 결과 :  [[19 22]
#         [43 50]]

```

> 행렬 곱셈을 수행한 결과 : [[19 22]
[43 50]]

- 다차원 배열의 점곱

```python
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([10, 20])

result = np.dot(a, b)
print(result)
# 결과:  [50 110]

```

> 다차원 배열의 점곱 결과: [1 * 10 + 2 * 20, 10 + 4 * 20] = [50,110] 이다.

1.2 다음 퀴즈의 난이도 3 단계의 문항 5 개이상을 해결하시오.

- exercise 50 : 2D 가우시안 배열 생성

```python
import numpy as np

x, y = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
d = np.sqrt(x*x + y*y)
sigma, mu = 1.0, 0.0
gaussian = np.exp(-((d - mu)**2 / (2.0 * sigma**2)))

print(gaussian)

```

// 가우시안 분포를 따르는 2D 배열을 생성한다.
x, y 는 −1 에서 1 까지의 값을 가지는 그리드 이고 d 는 각 점에서 원점까지의 거리이다.
gaussian 배열은 가우시안 부포를 계산하여 생성된다.

- exercise 60 : 16 x 16 배열에서 4 x 4 블록 합 구하기.

```python
import numpy as np

Z = np.ones((16,16))
k = 4
S = np.add.reduceat(
    np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0),
    np.arange(0, Z.shape[1], k), axis=1
)

print(S)

```

// reduceat 함수는 배열을 주어진 16 x 16 인덱스에서 4 x 4 로 축소한다.
4 x 4 블록단위로 배열을 축소해서 블록 합을 계산한다.

- exercise 70 : 주어진 배열에서 고정된 형태의 부분 배열을 추출하고 중심을 지정된 값으로 채우시오.

```python
import numpy as np

def extract_center(arr, shape, fill_value=0):
    pad_width = ((shape[0]//2, shape[0]//2), (shape[1]//2, shape[1]//2))
    arr_pad = np.pad(arr, pad_width, mode='constant', constant_values=fill_value)
    center = arr_pad.shape[0]//2, arr_pad.shape[1]//2
    return arr_pad[center[0]-shape[0]//2:center[0]+shape[0]//2+1,
                   center[1]-shape[1]//2:center[1]+shape[1]//2+1]

Z = np.random.randint(0, 10, (5, 5))
print(Z)
print(extract_center(Z, (3, 3)))
```

// extract_center 함수는 주어진 배열을 패딩하여 중심에서 지정된 형태의 부분 배열을 추출한다.
여기서 패딩은 ‘fill_value’로 채워진다.

□ exercise 80 : 배열의 두 행을 교환하시오.

```python
import numpy as np

A = np.arange(25).reshape(5, 5)
A[[0, 1]] = A[[1, 0]]

print(A)
```

//배열 A 의 첫 번째 행과 두 번째 행을 교환한다.
fancy indexing 을 사용하여 행을 교환한다.

□ exercise 90 : 큰 벡터 z 를 세가지 방법으로 세제곱 하시오.

```python
import numpy as np

Z = np.random.rand(1_000_000)

result1 = Z ** 3

result2 = np.power(Z, 3)

result3 = np.einsum('i,i,i->i', Z, Z, Z)

print(result1, result2, result3)
```

// 세 가지 다른 방법으로 벡터 z 의 각 요소를 세제곱한다.
result1 은 거듭제곱 연산자를 사용해서 세제곱하고
result2 는 np.power 함수를 사용하여 세제곱하고,
result3 는 einsum 함수를 사용하여 요소별 곱을 계산하여 세제곱한다.

1.3 A 가 정방행렬일 때,다음 선형방정식에서 해 x 를 구하는 numpy 코드(linearsol.py)를 작성하시오.
(main 에서는 코드에 대한 테스트도 포함)

▶ □□□□ □□ □□ □□□□ □□□□ □□ □□□ □□ =b 의 해 를 구 할 수 있 다 .

A (numpy.ndarray): 계수 행렬 (정방행렬)
b (numpy.ndarray): 상수 벡터
return (numpy.ndarray): 선 형 방 정 식 Ax=b 의 해 x

3 행에 solve_linear_equation(A,b) 함수를 통해 주어진 정방행렬 A 와 벡터 b 에 대해 Ax=b 의 해 x 를 구 한 다 .
np.linalg.inv(A) 를 사 용 하 여 행 렬 □ 의 역 행 렬 A^{-1} 을 계 산 한 다 .
np.dot(A_inv, b) 를 사 용 하 여 $A^{-1} * b$ 를 계 산 해 해 x 를 구 한 다 .

9 행 에 if __name__ == "__main__": 메 인 코 드 블 록
이 블 록 은 이 스 크 립 트 가 직 접 실 행 될 때 만 실 행 된 다 .

n = int(input("정방행렬 A 의 크기 n 을 입력하세요 (n x n): ")) 으로 행렬을 입력받는다.
n×n 행렬 A 의 요소를 행 단위로 입력받아 리스트로 저장한 후, 이를 numpy 배열로 변환한다.
20 행에서 벡터 b 의 요소를 입력받아 리스트로 저장한 후, 이를 numpy 배열로 변환한다.

입 력 받 은 행 렬 A 와 벡 터 b 를 사 용 하 여 해 x 를 구 하 고 출 력 한 다 .

np.linalg.solve(A, b) 를 사 용 하 여 예 상 결 과 를 계 산 한 다 .
assert np.allclose(x, expected_x) 를 사용하여 구한 해 x 와 예상 결과를 비교합니다.
- 일 치 하 지 않 을 경 우 "테 스 트 실 패 " 메 시 지 와 함 께 오 류 를 발 생 시 킨 다 .
-일치하는 경우 결과를 출력하고 “Success” 출력한다.

```python
import numpy as np

def solve_linear_equation(A, b):

    A_inv = np.linalg.inv(A)  # A의 역행렬
    x = np.dot(A_inv, b)  # A의 역행렬과 b를 곱하여 x를 구하기
    return x

if __name__ == "__main__":
    #(테스트) 행렬 A와 벡터 b 입력
    n = int(input("행렬 A 입력 (n x n): "))
    print(f"행렬 A의 요소를 입력:")
    A = []
    for i in range(n):
        row = list(map(float, input().split()))
        A.append(row)

    A = np.array(A)

    print(f"벡터 b요소 입력 :")
    b = list(map(float, input().split()))
    b = np.array(b)

    # 함수 호출, 결과 출력
    x = solve_linear_equation(A, b)
    print("X : ", x)

    # 테스트 결과 확인
    expected_x = np.linalg.solve(A, b)
    assert np.allclose(x, expected_x), f"테스트 실패(오류): 예상 {expected_x}, got {x}"
    print("Succees")
```

▶ A가 정방행렬이 아닌 일반 행렬일 때

solve_least_squares(A,b) 함수에서 최소 제곱 해 구한다.

11 행부터 행렬을 입력받는다
27 행에서 근사해를 구하고 결과를 출력한다.
scipy.linalg.lstsq 를 사용하여 예상 결과를 계산한다.
-assert np.allclose(x, expected_x)를 사용하여 구한 근사해 x 와 예상 결과를 비교한다.
테스트 통과하면 Success 를 출력하고, 실패하면 테스트 실패를 출력하고 예상결과를 출력한다.

```python
import numpy as np

def solve_least_squares(A, b):

    A_pinv = np.linalg.pinv(A)  # A의 유사역행렬(pseudoinverse)을 구하기
    x = np.dot(A_pinv, b)  # A의 유사역행렬과 b를 곱하여 근사해 x를 구하기
    return x

if __name__ == "__main__":
    # 행렬 A와 벡터 b 입력
    m = int(input("행렬 A의 행 개수 m을 입력: "))
    n = int(input("행렬 A의 열 개수 n을 입력: "))

    print(f"{m}x{n} 행렬 A의 요소를 행별로 입력:")
    A = []
    for i in range(m):
        row = list(map(float, input().split()))
        A.append(row)

    A = np.array(A)

    print(f"{m}차원 벡터 b의 요소를 입력:")
    b = list(map(float, input().split()))
    b = np.array(b)

    # 함수 호출, 결과 출력
    x = solve_least_squares(A, b)
    print("근사해 x :", x)

```

```python

    # 예상 결과 확인 (scipy를 사용한 비교 예제)
    from scipy.linalg import lstsq

    expected_x, _, _, _ = lstsq(A, b)  # lstsq는 최소제곱 해를 구하는 scipy 함수
    assert np.allclose(x, expected_x), f"테스트 실패 / 예상 결과 {expected_x}, 실제 결과 {x}"
    print("Success")

```

2.1 식 1 를 최소로 하는 w 와 b 를 구하기 위해∂J/∂w 와∂J/∂b 를 유도하시오.(행렬에 대한 미분공식을 이용하여 유도할 것)

$J = \sum_i (f(x_i) - y_i)^2 = \sum_i (w^T x_i + b - y_i)^2$

① w에 대한 편미분

$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w} \sum_i (w^T x_i + b - y_i)^2$

↓

$\frac{\partial J}{\partial w} = \sum_i 2(w^T x_i + b - y_i) x_i$

② b에 대한 편미분

$\frac{\partial J}{\partial b} = \frac{\partial}{\partial b} \sum_i (w^T x_i + b - y_i)^2$

↓

$\frac{\partial J}{\partial b} = \sum_i 2(w^T x_i + b - y_i)$

최종 유도식 유도 → 데이터벡터 활용

$X = [x_1, x_2 \cdots x_m]^T \quad y = [y_1, y_2 \cdots y_m]^T$

비용함수

$J = \sum_{i=1}^{m} (w^T x_i + b - y_i)^2$

→ 행렬형태로 표현

$J = (Xw + b1 - y)^T (Xw + b1 - y)$

① w에 대한 편미분

$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w} (Xw + b1 - y)^T (Xw + b1 - y)$

$\rightarrow \frac{\partial J}{\partial w} = 2X^T (Xw + b1 - y)$

② b에 대한 편미분

$\frac{\partial J}{\partial b} = \frac{\partial}{\partial b} (Xw + b_1 - y)^T (Xw + b1 - y)$

$\rightarrow \frac{\partial J}{\partial b} = 2 1^T (Xw + b1 - y)$

2.2 minibatch 단위의 Stochastic Gradient Descent (SGD) Method 의 알고리즘의 pseudo code 를 기술하시오.

▶ presude code

```python
w = np.random.randn(n)  # n is the number of features
b = 0
learning_rate = n
epochs = number_of_epochs
batch_size = m

for epoch in range(epochs):
    # Shuffle the data
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    y = y[indices]

    for i in range(0, X.shape[0], batch_size):
        X_mini = X[i:i + batch_size]
        y_mini = y[i:i + batch_size]

        predictions = np.dot(X_mini, w) + b

        gradient_w = (2 / batch_size) * np.dot(X_mini.T, (predictions - y_mini))
        gradient_b = (2 / batch_size) * np.sum(predictions - y_mini)

        w = w - learning_rate * gradient_w
        b = b - learning_rate * gradient_b

    cost = np.mean((np.dot(X, w) + b - y) ** 2)
    print(f'Epoch {epoch + 1}/{epochs}, Cost: {cost}')

# Final trained parameters
print("Trained weights:", w)
print("Trained bias:", b)
```

// 1. 모델 파라미터 w 와 b 를 임의의 값으로 초기화하고, 학습률 η를 설정합니다.
2. epoch 에서 데이터를 섞어 순서를 무작위로 바꾼다.
3. Mini-batch Gradient Descent 로 미니 배치를 생성하여 각 미니 배치에 대해 미니 배치 생성 , 예측계산, 그래디언트 계산을 수행한다.
4. 코스트 계산을 한다. 각 epoch 마다 비용함수 J 의 값을 계산하여 학습 진행 상황을 모니터링 가능하다.
5. 최종 파라미터를 출력한다./ w 와 b 를 출력한다.

2.3 Early stopping 방식은 무엇인지 조사하고, 위의 SGD 알고리즘에 Early stopping 추가한 pseudo code 를 기술하시오.

Early stopping 방식

▶ 머신 러닝 모델의 과적합을 방지하기 위한 정규화 기법이다. 모델이 검증 세트에서 더 이상 성능이 개선되지 않을 때 훈련을 멈추는 방식으로 작동한다. 이는 모델이 훈련 데이터의 노이즈에 맞춰 과적합되는 것을 방지하는데 도움이 된다.

▶ presude 코드

```python
import numpy as np

# 파라미터 초기화
w = np.random.randn(n)  # n=피처의 수
b = 0
learning_rate = n
epochs = number_of_epochs
batch_size = m  # 미니배치 크기
patience = patience_value  # 개선이 없는 에폭 수
min_delta = minimum_delta  # 개선 최소값

# Early stopping 변수
best_loss = float('inf')
epochs_no_improve = 0

# SGD with Early Stopping
for epoch in range(epochs):
    # 데이터 셔플링
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    y = y[indices]

    # 미니배치 경사 하강법
    for i in range(0, X.shape[0], batch_size):
        # 미니배치 생성
        X_mini = X[i:i + batch_size]
        y_mini = y[i:i + batch_size]
```

```python
        # 예측 계산
        predictions = np.dot(X_mini, w) + b

        # 그래디언트 계산
        gradient_w = (2 / batch_size) * np.dot(X_mini.T, (predictions - y_mini))
        gradient_b = (2 / batch_size) * np.sum(predictions - y_mini)

        # 파라미터 업데이트
        w = w - learning_rate * gradient_w
        b = b - learning_rate * gradient_b

    # 검증 손실 계산
    val_predictions = np.dot(X_val, w) + b
    val_loss = np.mean((val_predictions - y_val) ** 2)

    # Early stopping 체크
    if val_loss < best_loss - min_delta:
        best_loss = val_loss
        epochs_no_improve = 0
    else:
        epochs_no_improve += 1

    if epochs_no_improve >= patience:
        print(f'Early stopping on epoch {epoch+1}')
        break

    # 진행 상황 출력 (선택 사항)
    print(f'Epoch {epoch + 1}/{epochs}, Validation Loss: {val_loss}')
```

```python
# 최종 학습된 파라미터 출력
print("Trained weights:", w)
print("Trained bias:", b)
```

// 1. 모델 파라미터 w와 b를 임의의 값으로 초기화하고, 학습률과 Early stopping 관련 변수를 설정한다.

2. 각 epoch 에서 데이터를 셔플링하여 순서를 무작위로 바꾼다.
3. 각 미니배치에 대해 예측을 계산하고, 손실 함수의 그래디언트를 계산하여 파라미터를 업데이트한다.
4. 각 epoch 후 검증세트에 대한 손실을 계산한다.
5. 검증 손실이 최소 개선치 이상 개선되지 않으면 개선되지 않은 epoch 수를 증가시키고, 개선된 경우 초기화한다. 개선되지 않은 epoch 수가 설정한 patience 값을 초과하면 중지한다.
6. 최종 파라미터 w 와 b 를 출력한다.

2.4 지금까지 유도한 early stopping 을 사용한 minibatch-SGD 방법을 numpy 패키지를 이용하여 구현하시오 (linear
regression.py 코드 제출).
구현된 모듈을 테스트 하기 위해 다음과 같이 랜덤으로 생성된 데이터와 scikit 의 샘플데이터 각각에 대하여 테스트해보시오.

▶ 파일 제출함(gen_random_dataset.py / linear_regression.py)
1. generate_random_dataset 함수는 지정된 파라미터에 따라 Gaussian 분포를 기반으로 데이터를 생성하고, 이를 학습, 검증, 테스트 세트로 나누어 pickle 파일로 저장하여 랜덤데이터를 생성한다.
2. LinearRegressionSGD 클래스는 SGD 를 사용하여 선형 회귀 모델을 학습한다.. Early Stopping 기능이 포함되어 있어 검증 세트의 성능이 개선되지 않으면 학습을 중지한다.
3.test_random_dataset 함수는 생성된 랜덤 데이터를 불러와 모델을 학습하고, 학습 성능을 출력한다.
4. scikit-learn 샘플 데이터 테스트:
test_sklearn_diabetes 함수는 scikit-learn 의 당뇨병 데이터를 불러오고, 이를 학습 및 검증 세트로 나누어 모델을 학습하고 성능을 출력한다.

3.1
-각 데이터 샘플 xi 에 대한 Logistic Regression 의 예측값

$$o(x_i) = \text{Softmax}(Wx_i + b)$$

- Negative Log-Likelihood

$$J(W, b) = -\sum_{i=1}^{m} y_i^T \log(o(x_i))$$

-전체 데이터셋 D 에 대해 확장

$$J(W, b) = -\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ik} \log(o_k(x_i))$$

yik 는 yi의 k 번째 요소이며, $ok(xi)$는 o(xi)의 k 번째 요소이다.
최종적으로 가중치 W 와 편향 b 에 대한 그래디언트는

$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{i=1}^{m} (\mathbf{o}(\mathbf{x}_i) - \mathbf{y}_i)\, \mathbf{x}_i^T$$

$$\frac{\partial J}{\partial \mathbf{b}} = \sum_{i=1}^{m} (\mathbf{o}(\mathbf{x}_i) - \mathbf{y}_i)$$

이다.

3.2
▶ presude 코드

```python
import numpy as np
from sklearn.metrics import log_loss

class LogisticRegressionSGD:
    def __init__(self, learning_rate=0.01, epochs=1000, batch_size=32, patience=10, min_delta=1e-4):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.batch_size = batch_size
        self.patience = patience
        self.min_delta = min_delta

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def fit(self, X, y, X_val, y_val):
        self.W = np.random.randn(X.shape[1], y.shape[1])
        self.b = np.zeros((1, y.shape[1]))
        best_loss = float('inf')
        epochs_no_improve = 0

        for epoch in range(self.epochs):
            indices = np.arange(X.shape[0])
            np.random.shuffle(indices)
            X = X[indices]
            y = y[indices]

            for i in range(0, X.shape[0], self.batch_size):
                X_mini = X[i:i + self.batch_size]
                y_mini = y[i:i + self.batch_size]

                predictions = self.softmax(np.dot(X_mini, self.W) + self.b)
```

```
                error = predictions - y_mini

                gradient_W = np.dot(X_mini.T, error) / self.batch_size
                gradient_b = np.sum(error, axis=0, keepdims=True) / self.batch_size

                self.W -= self.learning_rate * gradient_W
                self.b -= self.learning_rate * gradient_b

            val_predictions = self.softmax(np.dot(X_val, self.W) + self.b)
            val_loss = log_loss(y_val, val_predictions)

            if val_loss < best_loss - self.min_delta:
                best_loss = val_loss
                epochs_no_improve = 0
            else:
                epochs_no_improve += 1

            if epochs_no_improve >= self.patience:
                print(f'Early stopping on epoch {epoch+1}')
                break

            print(f'Epoch {epoch + 1}/{self.epochs}, Validation Loss: {val_loss}')

    def predict(self, X):
        probabilities = self.softmax(np.dot(X, self.W) + self.b)
        return np.argmax(probabilities, axis=1)

# 데이터 로드 및 테스트
def test_logistic_regression(X, y):
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

    model = LogisticRegressionSGD(learning_rate=0.01, epochs=1000, batch_size=32, patience=10, min_delta=1e-4)
    model.fit(X_train, y_train, X_val, y_val)

    train_predictions = model.predict(X_train)
    val_predictions = model.predict(X_val)
    test_predictions = model.predict(X_test)

    print(f'Training Log Loss: {log_loss(y_train, model.softmax(np.dot(X_train, model.W) + model.b))}')
    print(f'Validation Log Loss: {log_loss(y_val, model.softmax(np.dot(X_val, model.W) + model.b))}')
    print(f'Test Log Loss: {log_loss(y_test, model.softmax(np.dot(X_test, model.W) + model.b))}')

    print(f'Training Accuracy: {np.mean(np.argmax(y_train, axis=1) == train_predictions)}')
    print(f'Validation Accuracy: {np.mean(np.argmax(y_val, axis=1) == val_predictions)}')
    print(f'Test Accuracy: {np.mean(np.argmax(y_test, axis=1) == test_predictions)}')

# 사용 예시
def test_mnist():
    from sklearn.datasets import fetch_openml
    mnist = fetch_openml('mnist_784')
    X = mnist.data / 255.0
    y = mnist.target.astype(int)
    y = np.eye(10)[y]  # 원-핫 인코딩

    test_logistic_regression(X, y)

test_mnist()
```

3.3

▶ 파일제출

1. 학습률, 최대 에폭 수, 미니배치 크기, patience, min_delta 등을 초기화한다.
2. 소프트맥스 함수로 예측 값을 소프트맥스 함수로 변환합니다.
3. 각 에폭마다 데이터를 셔플링하고 미니배치를 생성하여 SGD를 수행한다. 각 에폭 후 검증 손실을 계산하고, 개선이 없으면 patience를 증가시킨다. 개선이 없는 에폭 수가 설정된 patience 값을 초과하면 학습을 중지한다.
4. 소프트맥스 함수의 출력을 사용하여 예측을 수행한다.
5. scikit-learn의 MNIST 데이터를 사용하여 모델을 학습하고 성능을 출력한다.

4.1

1. 로지스틱 회귀 손실함수

$J = -y^T \log(o)$ //y 는 실제 레이블 벡터, o 는 예측된 확률 벡터

2.역전파 과정 단계

- 순전파 :입력 데이터를 각 층을 통해 전달하며 활성화 함수를 적용하여 출력 값을 계산한다.
- 손실 함수의 계산 :예측된 출력 o 와 실제 레이블 y 를 비교하여 손실 J 를 계산한다.
- 기울기 계산: 출력층에서부터 시작하여 각 가중치에 대한 손실 함수의 기울기를 계산한다.

■ 출력층 기울기 계산

$\frac{\partial J}{\partial U} = (o - y)h^T$

■ 은닉층 기울기 계산

$\frac{\partial J}{\partial W} = (\delta^l \cdot \sigma'(z^l))x^T$

- 가중치 업데이트

$U = U - \eta \frac{\partial J}{\partial U}$
$W = W - \eta \frac{\partial J}{\partial W}$

->역전파 알고리즘은 MLP 의 학습 과정에서 매우 중요한 역할을 한다. 각 가중치에 대한 손실 함수의 기울기를 계산하고, 이를 통해 가중치를 업데이트하여 모델의 성능을 향상시킨다. 이 과정은 출력층에서 시작하여 입력층으로 역방향으로 진행되며, 각 층의 활성화 함수와 손실 함수에 대한 기울기를 계산한다.

4.2

-출력층의 그래디언트

$\delta^o = o - y$

-은닉층의 그래디언트

$\delta^h = (U^T \delta^o) \odot \mathrm{ReLU}'(Wx+b)$

$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \le 0 \end{cases}$

-가중치와 편향의 기울기

$\frac{\partial J}{\partial U} = \delta^o h^T, \quad \frac{\partial J}{\partial d} = \delta^o$

$\frac{\partial J}{\partial W} = \delta^h x^T, \quad \frac{\partial J}{\partial b} = \delta^h$

4.3

▶ presude code

```python
import numpy as np
from sklearn.metrics import log_loss, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml

class MLP_SGD:
    def __init__(self, input_dim, hidden_dim, output_dim, learning_rate=0.01, epochs=100, batch_size=32, patience=10, min_delta=1e-4):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.batch_size = batch_size
        self.patience = patience
        self.min_delta = min_delta
        self.W = np.random.randn(input_dim, hidden_dim)
        self.b = np.zeros(hidden_dim)
        self.U = np.random.randn(hidden_dim, output_dim)
        self.d = np.zeros(output_dim)

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def relu(self, z):
        return np.maximum(0, z)

    def relu_derivative(self, z):
        return np.where(z > 0, 1, 0)

    def fit(self, X, y, X_val, y_val):
        best_loss = float('inf')
        epochs_no_improve = 0

        for epoch in range(self.epochs):
            indices = np.arange(X.shape[0])
            np.random.shuffle(indices)
            X = X[indices]
            y = y[indices]

            for i in range(0, X.shape[0], self.batch_size):
                X_mini = X[i:i + self.batch_size]
                y_mini = y[i:i + self.batch_size]

                # Forward pass
                h = self.relu(np.dot(X_mini, self.W) + self.b)
                o = self.softmax(np.dot(h, self.U) + self.d)

                # Backward pass
                delta_o = o - y_mini
                delta_h = np.dot(delta_o, self.U.T) * self.relu_derivative(np.dot(X_mini, self.W) + self.b)

                grad_U = np.dot(h.T, delta_o) / self.batch_size
                grad_d = np.sum(delta_o, axis=0) / self.batch_size
                grad_W = np.dot(X_mini.T, delta_h) / self.batch_size
                grad_b = np.sum(delta_h, axis=0) / self.batch_size

                # Update weights and biases
                self.U -= self.learning_rate * grad_U
                self.d -= self.learning_rate * grad_d
                self.W -= self.learning_rate * grad_W
                self.b -= self.learning_rate * grad_b

            val_predictions = self.predict_proba(X_val)
            val_loss = log_loss(y_val, val_predictions)
```

```
            if val_loss < best_loss - self.min_delta:
                best_loss = val_loss
                epochs_no_improve = 0
            else:
                epochs_no_improve += 1

            if epochs_no_improve >= self.patience:
                print(f'Early stopping on epoch {epoch+1}')
                break

            train_acc = accuracy_score(np.argmax(y, axis=1), self.predict(X))
            val_acc = accuracy_score(np.argmax(y_val, axis=1), self.predict(X_val))
            print(f'Epoch {epoch + 1}/{self.epochs}, Validation Loss: {val_loss}, Training Accuracy: {train_acc}, Validation Accuracy: {val_acc}')

    def predict_proba(self, X):
        h = self.relu(np.dot(X, self.W) + self.b)
        o = self.softmax(np.dot(h, self.U) + self.d)
        return o

    def predict(self, X):
        return np.argmax(self.predict_proba(X), axis=1)

# MNIST 데이터셋 테스트
def test_mnist():
    mnist = fetch_openml( name: 'mnist_784', as_frame=False)
    X = mnist.data / 255.0
    y = mnist.target.astype(int)
    y = np.eye(10)[y]  # 원-핫 인코딩

    X_train, X_temp, y_train, y_temp = train_test_split( *arrays: X, y, test_size=0.3, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split( *arrays: X_temp, y_temp, test_size=0.5, random_state=42)

    model = MLP_SGD(input_dim=784, hidden_dim=128, output_dim=10, learning_rate=0.01, epochs=100, batch_size=32, patience=10, min_delta=1e-4)
    model.fit(X_train, y_train, X_val, y_val)

    train_predictions = model.predict(X_train)
    val_predictions = model.predict(X_val)
    test_predictions = model.predict(X_test)

    print(f'Training Log Loss: {log_loss(y_train, model.predict_proba(X_train))}')
    print(f'Validation Log Loss: {log_loss(y_val, model.predict_proba(X_val))}')
    print(f'Test Log Loss: {log_loss(y_test, model.predict_proba(X_test))}')

    print(f'Training Accuracy: {accuracy_score(np.argmax(y_train, axis=1), train_predictions)}')
    print(f'Validation Accuracy: {accuracy_score(np.argmax(y_val, axis=1), val_predictions)}')
    print(f'Test Accuracy: {accuracy_score(np.argmax(y_test, axis=1), test_predictions)}')

# 사용 예시
test_mnist()
```

//1. 입력,은닉, 출력 차원 및 학습률, 에폭 수, 미니배치 크기, patience, min_delta 등을 초기화한다.
2. 활성화 함수와 그 도함수를 정의한다.
3. 각 에폭마다 데이터를 셔플링하고 미니배치를 생성하여 SGD 를 수행합니다. 각 에폭 후 검증 손실을 계산하고, 개선이 없으면 patience 를 증가시킨다. 개선이 없는 에폭 수가 설정된 patience 값을 초과하면 학습을 중지한다.
4. 소프트맥스 함수의 출력을 사용하여 에측을 수행한다.
5. skikit-learn 의 MINIST 데이터를 사용하여 모델을 학습하고 성능을 출력한다.

4.4

▶ 파일제출

MLP.py

-MLP 클래스는 은닉층 수를 포함한 MLP 모델을 정의한다.
-relu, relu_derivative, softmax 함수는 활성화 함수 및 그 도함수를 정의한다.
-forward 및 backward 함수는 순전파 및 역전파 과정을 구현한다.
-fit 함수는 SGD 방법을 사용하여 모델을 학습하고, 조기 종료를 지원한다.
-predict 함수는 입력 데이터에 대한 예측을 수행한다.

MLP_minist.py

-MNIST 데이터셋을 로드하고, 학습 및 테스트 데이터로 분할한다.
-MLP 모델을 생성하고, fit 함수를 사용하여 모델을 학습시킨다.
-학습된 모델을 사용하여 테스트 데이터에 대한 예측을 수행하고, 정확도를 출력한다.