Artificial Intelligence: Project 2

컴퓨터공학부 202246109 김기현

1.1. Numpy: dot, enisum

numpy.dot() 은 두 배열의 내적을 구하는 함수다.

1. 배열 a, b 가 1차원 배열인 경우: 벡터의 내적

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

# 두 벡터의 내적
print(np.dot(a, b))
```

✓ 0.0s

```
32
```

a · b = 1 * 4 + 2 * 5 + 3 * 6 = 32

2. 배열 a, b 가 2차원 배열인 경우: 행렬의 곱

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# 행렬의 곱
print(np.dot(A, B))
```

✓ 0.0s

```
[[19 22]
 [43 50]]
```

(A · B)11 = 1 * 5 + 2 * 7 = 19

(A · B)12 = 1 * 6 + 2 * 8 = 22

(A · B)21 = 3 * 5 + 4 * 7 = 43

(A · B)22 = 3 * 6 + 4 * 8 = 50

3. N차원 배열과 M차원 배열의 곱

```
a = np.array([[1, 2], [3, 4], [5, 6]])
b = np.array([[7, 8], [9, 10]])

print(np.dot(a, b))
```

```
[[ 25  28]
 [ 57  64]
 [ 89 100]]
```

(A · B)11 = 1 * 7 +2 * 9 = 25

(A · B)12 = 1 * 8 + 2 * 10 = 28

(A · B)21 = 3 * 7 + 4 * 9 = 57

(A · B)22 = 3 * 8 + 4 * 10 = 64

(A · B)31 = 5 * 7 + 6 * 9 = 89

(A · B)32 = 5 * 8 + 6 * 10 = 100

numpy.einsum() 은 특정 index의 집합에 대한 합연산을 간결하게 표시하는 아인슈타인 표기법을 사용하여 행렬, 벡터의 dot products, outer products, transpose, trace, multiplication등을 일관성있게 표시할 수 있다.

예를들어 np.einsum(“ij, jk -> ik”, A, B) 라고 하면 A(i, j)요소와 B(j, k) 요소를 곱하여

(i, k)위치에 더한다는 것을 의미이며, 결과적으로 A와 B의 곱 연산을 수행하라는 뜻이다.

1. 두 1차원 배열의 내적

```
# 두 1차원 배열의 내적

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print(np.dot(a, b))
print(np.einsum('i, i', a, b))
```

✓ 0.0s

```
32
32
```

‘i, i’ 는 각 벡터의 i번쨰 요소를 곱한 뒤 더한다는 것을 의미한다.

2. 두 2차원 배열의 곱 (행렬의 곱)

```
# 두 2차원 배열의 곱
A = np.array([[1,2],[3,4]])
B = np.array([[2,3],[4,5]])
print(np.dot(A, B))
print(np.einsum("ij,jk->ik", A, B))
```

✓ 0.0s

```
[[10 13]
 [22 29]]
[[10 13]
 [22 29]]
```

'Ij, jk -> ik'는 A(i, j)요소와 B(j, k) 요소를 곱하여 (i, k)위치에 더한다는 것을 의미한다.

3. 다차원 배열의 곱

```
# 다차원 배열의 곱
a = np.array([[1, 2], [3, 4], [5, 6]])
b = np.array([[7, 8], [9, 10]])

print(np.dot(a, b))
print(np.einsum('ij,jk -> ik', a, b))
```

✓ 0.0s

```
[[ 25  28]
 [ 57  64]
 [ 89 100]]
[[ 25  28]
 [ 57  64]
 [ 89 100]]
```

1.2 Numpy: quiz풀이

65. How to accumulate elements of a vector (X) to an array (F) based on an index list (I)? (★★★)

```
X = [1,2,3,4,5,6]
I = [7,8,9,10,11,12]
F = np.bincount(I,X)
print(F)
```

✓ 0.0s

```
[0. 0. 0. 0. 0. 0. 0. 1. 2. 3. 4. 5. 6.]
```

69. How to get the diagonal of a dot product? (★★★)

```
a = np.arange(1, 10).reshape(3, 3)
b = np.arange(10, 19).reshape(3, 3)
print(np.einsum("ij,ji->i", a, b))
```

✓ 0.0s

```
[ 84 216 366]
```

72. How to swap two rows of an array? (★★★)

```
#answer(72)
a = np.arange(9).reshape(3, 3)
print(a)
a[[0, 1]] = a[[1, 0]]
print(a)
```

✓ 0.0s

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[3 4 5]
 [0 1 2]
 [6 7 8]]
```

71. Consider an array of dimension (5,5,3), how to mulitply it by an array with dimensions (5,5)? (★★★)

```
a = np.ones((5, 5, 3))
b = 2 * np.ones((5, 5))
print(a * b[:, :, None])
```

✓ 0.0s

```
[[[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]]
```

97. Considering 2 vectors A & B, write the einsum equivalent of inner, outer, sum, and mul function (★★★)

```
A = np.array([1, 2, 3, 4, 5])
B = np.array([6, 7, 8, 9, 10])

#inner
print("inner: ",np.einsum('i,i', A, B))
#outer
print("outer: ", np.einsum('i,j -> ij', A, B))
#sum
print("sum: ", np.einsum('i->', A))
#mul
print("mul: ",np.einsum('i,i -> i', A, B))
```

✓ 0.0s Python

```
inner:  130
outer:  [[ 6  7  8  9 10]
 [12 14 16 18 20]
 [18 21 24 27 30]
 [24 28 32 36 40]
 [30 35 40 45 50]]
sum:  15
mul:  [ 6 14 24 36 50]
```

1.3 Numpy: 선형방정식 풀기

Linearsol.py

```python
import numpy as np

def linearSolve(A, b):

    A_inv = np.linalg.inv(A)
    x = np.dot(A_inv, b)
    return x

def main():

    A = np.array(np.random.randint(-10, 10, 9).reshape(3, 3))
    b = np.array(np.random.randint(-10, 10, 3).reshape(3, 1))
    print("A: ", A)
    print('b: ', b)

    x = linearSolve(A, b)
    print("x:", x)


if __name__ == "__main__":
    main()
```

```
A:  [[ 5 -6  2]
 [ 3 -5  9]
 [-5 -8  5]]
b:  [[-7]
 [ 2]
 [-3]]
선형 방정식의 해 x: [[-0.34004024]
 [ 1.22132797]
 [ 1.01408451]]
```

leastsquaresol.py

```python
import numpy as np

def leastsSquareSolve(A, b):

    pseudoInv = np.linalg.pinv(A)
    x = np.dot(pseudoInv, b)
    return x

def main():

    A = np.array(np.random.randint(-10, 10, 6).reshape(3, 2))
    b = np.array(np.random.randint(-10, 10, 3).reshape(3, 1))
    print("A: ", A)
    print('b: ', b)

    x = leastsSquareSolve(A, b)
    print("x:", x)


if __name__ == "__main__":
    main()
```

```
A:  [[-2 -5]
 [ 8  8]
 [ 9 -4]]
b:  [[-3]
 [ 2]
 [-8]]
x: [[-0.53827195]
 [ 0.79480318]]
```

2.1 Linear regression을 위한 식 유도

$$f(x) = w^T x + b$$

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

$$\frac{dJ}{dw} = \sum_i 2(w^T x_i + b - y_i)\frac{d(w^T x_i + b - y_i)}{dw} \qquad \frac{d(w^T x_i + b - y_i)}{dw} = x_i$$

$$= 2\sum_i (w^T x_i + b - y_i) x_i = 0$$

$$\frac{dJ}{db} = \sum_i 2(w^T x_i + b - y_i)\frac{d(w^T x_i + b - y_i)}{db} \qquad \frac{d(w^T x_i + b - y_i)}{db} = 1$$

$$= 2\sum_i (w^T x_i + b - y_i) = 0$$

$$\sum_i 2(w^T x_i + b - y_i)x_i = 0 \qquad \sum_i x_i w^T x_i + \sum_i x_i b - \sum_i x_i y_i = 0$$

$$X^T(Xw + b1 - y) = 0 \qquad X^T Xw + bX^T 1 = X^T y$$

$$\sum_i (w^T x_i + b x_i) = 0 \qquad 1^T(Xw + b1 - y) = 0 \qquad 1^T Xw + b1^T 1 = 1^T y$$

$$X = [x_1, \dots, x_m] \qquad y = [y_1 \cdots y_m]^T$$

$$A = X^T X \quad B = X^T 1 \quad C = 1^T 1 \quad d = X^T y \quad e = 1^T y$$

$$Aw + bB = d$$

$$B^T w + bC = e$$

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}\begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} d \\ e \end{bmatrix}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}^{-1}\begin{bmatrix} d \\ e \end{bmatrix}$$

2.2 Linear regression학습을 위한 Stochastic Gradient De- scent (SGD)Method

```
# 에포크에 걸쳐 반복
for epoch in range(num_epochs):
    shuffle(X, y)  # Shuffle the data

    # 데이터를 미니배치로 분할
    num_batches = len(X) // batch_size
    for batch in range(num_batches):
        # 미니배치 생성
        X_batch = X[batch * batch_size : (batch + 1) * batch_size]
        y_batch = y[batch * batch_size : (batch + 1) * batch_size]

        # gradients 계산
        grad_w = sum((dot(w, x_i) + b - y_i) * x_i for x_i, y_i in zip(X_batch, y_batch)) / len(X_batch)
        grad_b = sum(dot(w, x_i) + b - y_i for x_i, y_i in zip(X_batch, y_batch)) / len(X_batch)

        # 파라미터 업데이트
        w -= learning_rate * grad_w
        b -= learning_rate * grad_b

# 손실 계산 함수
def compute_loss(X, y, w, b):
    return sum((dot(w, X[i]) + b - y[i]) ** 2 for i in range(len(X))) / len(X)
```

2.3 Early stopping 추가

Early stopping 이란 과적합을 방지하기 위해 SGD와 같은 반복 학습 알고리즘에서 사용되는 정규화 기법이다.

Early stopping의 기본 개념은 훈련 중에 검증 set에서 모델의 성능을 모니터링하고

과적합되어 성능이 저하되는 경우 훈련 프로세스를 중지한다.

```
# Initialize validation set (X_val, y_val)
# Optionally split the data into training and validation sets

# 에포크에 걸쳐 반복
for epoch in range(num_epochs):
    shuffle(X, y)  # Shuffle the data

    # 데이터를 미니배치로 분할
    num_batches = len(X) // batch_size
    for batch in range(num_batches):
        # 미니배치 생성
        X_batch = X[batch * batch_size : (batch + 1) * batch_size]
        y_batch = y[batch * batch_size : (batch + 1) * batch_size]

        # gradients 계산
        grad_w = sum((dot(w, x_i) + b - y_i) * x_i for x_i, y_i in zip(X_batch, y_batch)) / len(X_batch)
        grad_b = sum(dot(w, x_i) + b - y_i for x_i, y_i in zip(X_batch, y_batch)) / len(X_batch)

        # 파라미터 업데이트
        w -= learning_rate * grad_w
        b -= learning_rate * grad_b

    # 각 에포크 후 유효성 손실 계산하기
    val_loss = compute_loss(X_val, y_val, w, b)

    # Early stopping 체크
    if val_loss < best_loss - tolerance:
        best_loss = val_loss
        best_w = w
        best_b = b
        epochs_without_improvement = 0
    else:
        epochs_without_improvement += 1
    # 성능 저하시 중지
    if epochs_without_improvement >= patience:
        print("Early stopping triggered.")
        w = best_w
        b = best_b
        break

# 손실 계산 함수
def compute_loss(X, y, w, b):
    return sum((dot(w, X[i]) + b - y[i]) ** 2 for i in range(len(X))) / len(X)
```

2.4 minibatch-SGD 구현

2.4.1 랜덤 데이터 생성기 구현: Gaussian분포에 기반

1. true 파라미터 값의 랜덤 할당, 데이터셋생성

```python
def generate_dataset(N, d, R=10, alpha=0.1):
    w = np.random.uniform(-R, R, d)
    b = np.random.uniform(-R, R)
    X = np.random.uniform(-R, R, (N, d))
    sigma = alpha * R
    y = X @ w + b + np.random.normal(0, sigma, N)
    return X, y, w, b
```

2. 데이터셋 분리

```python
def split_dataset(X, y, train_ratio=0.85, dev_ratio=0.05, test_ratio=0.10):
    total_samples = X.shape[0]
    indices = np.random.permutation(total_samples)

    # 각 set의 샘플 수 계산하기
    train_end = int(train_ratio * total_samples)
    dev_end = train_end + int(dev_ratio * total_samples)

    # train, dev, test set 분할
    train_indices = indices[:train_end]
    dev_indices = indices[train_end:dev_end]
    test_indices = indices[dev_end:]


    # 하위 집합 생성
    X_train, y_train = X[train_indices], y[train_indices]
    X_dev, y_dev = X[dev_indices], y[dev_indices]
    X_test, y_test = X[test_indices], y[test_indices]

    return (X_train, y_train), (X_dev, y_dev), (X_test, y_test)

def save_dataset(data, filename="myrandomdataset.pkl"):
    with open(filename, "wb") as f:
        pickle.dump(data, f)
```

3. 데이터셋 저장

총 데이터갯수는 $N = 1000, 10000, 100000$로 다양하게 설정하여, 다른 이름으로 저장하여 테스트 수행할 것.

```
#N = 1000
#N = 10000
N = 100000
```

```
def save_dataset(data, filename="myrandomdataset.pkl"):
    with open(filename, "wb") as f:
        pickle.dump(data, f)
```

- minibatch 크기 사용자 설정: m은 option으로 사용자가 설정하도록 하고, default로 10에서 100사이의 값을 택하여 사용하라.

```
def main(m, e):
    test_sgd_with_datasets(m, e)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='options: --m')
    parser = argparse.ArgumentParser(description='options: --e')
    parser.add_argument('--m', type=int, default=10, help='Set batch size')
    parser.add_argument('--e', type=int, default=100, help='Set epoch size')
    args = parser.parse_args()
    main(args.m, args.e)
```

- 매epoch마다 성능 출력: 주어진 데이터셋 전체 예제를 한번 학습할때 (1 epoch 시)마다 training, dev, test상에서 mean squared error을 출력한다. 랜덤 데이터인 경우에 한해서, 파라미터 w와 b와 true값과 학습된 값들간의 squared error도 함께 출력하시오.

```
y_train_pred = X_train.dot(w) + b
y_dev_pred = X_dev.dot(w) + b
y_test_pred = X_test.dot(w) + b

train_error = mean_squared_error(y_train, y_train_pred)
dev_error = mean_squared_error(y_dev, y_dev_pred)
test_error = mean_squared_error(y_test, y_test_pred)

mse_train.append(train_error)
mse_dev.append(dev_error)
mse_test.append(test_error)

print(f"Epoch {epoch+1}/{max_epochs} - Train MSE: {train_error}, Dev MSE: {dev_error}, Test MSE: {test_error}")
```

• Early stopping적용: early stopping을 적용하여, dev set상 성능 개선이 오랫동안 없는 경우 학습을 종료한다.

```
if dev_error < best_dev_error:
    best_dev_error = dev_error
    best_w, best_b = w.copy(), b
    no_improvement = 0
else:
    no_improvement += 1
    if no_improvement > 10:
        print(f"Early stopping after {epoch+1} epochs")
        break
```

• 최대 epoch수 설정: 최대 epoch수를 사용자가 설정할 수 있도록 하고, de- fault 값으로 100을 사용한다.

```
def main(m, e):
    test_sgd_with_datasets(m, e)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='options: --m')
    parser = argparse.ArgumentParser(description='options: --e')
    parser.add_argument('--m', type=int, default=10, help='Set batch size')
    parser.add_argument('--e', type=int, default=100, help='Set epoch size')
    args = parser.parse_args()
    main(args.m, args.e)
```

3.logistic regression

3.1 logistic regression 학습을 위한 식 유도

$O(X) = \text{Softmax}(WX+b)$ $\quad J = -y^T \log(O)$ $\quad y = [y_1 \cdots y_i \cdots y_K]^T$

$y_i = \mathbb{I}(i=K)$ $\quad O = \text{Softmax}(WX+b) = [O_1 \cdots O_i \cdots O_K]^T$

$J = -Y^T \log(O) = -\sum_{i=1}^{K} y_i \log(O_i)$ $\quad J = -\log(O_K)$

$J_i = -\log(O_{gi})$ $\quad J_{total} = -\sum_{i=1}^{m} \log(O_{y_i})$

$O_i = \text{Softmax}(WX_i+b)$ $\quad O_{ij} = \dfrac{e^{z_{ij}}}{\sum_{k=1}^{K} e^{z_{ik}}}$

$\dfrac{dO_{ij}}{dz_{ij}} = O_{ij}(1-O_{ij})$ $\quad \dfrac{dO_{ij}}{dz_{ik}} = -O_{ij}O_{ik}$

$J_i = -\log(O_{iy_i})$ $\quad \dfrac{dJ_i}{dz_{ij}} = O_{ij} - y_{ij}$

$\dfrac{dJ_{total}}{dW} = \sum_{i=1}^{m} \dfrac{dJ_i}{dW} = \sum_{i=1}^{m} X_i (O_i - y_i)^T$

$\dfrac{dJ_{total}}{db} = \sum_{i=1}^{m} \dfrac{dJ}{db} = \sum_{i=1}^{m} (O_i - y_i)$

Logistic regression학습을 위한 Stochastic Gradient De- scent Method 구현

```
X, y = datasets.fetch_openml('mnist_784', version=1, return_X_y=True)
```

- minibatch 크기 사용자 설정: m은 option으로 사용자가 설정하도록 하고, default로 10에서 500사이의 값을 적절히 택하여 사용하라.
- 최대 epoch수 설정: 최대 epoch수를 사용자가 설정할 수 있도록 하고, de- fault값으로 100을 사용한다.

```
def main(m, e):
    test_sgd_with_datasets(m, e)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='options: --m')
    parser = argparse.ArgumentParser(description='options: --e')
    parser.add_argument('--m', type=int, default=10, help='Set batch size')
    parser.add_argument('--e', type=int, default=100, help='Set epoch size')
    args = parser.parse_args()
    main(args.m, args.e)
```

```
train_loss = -np.mean(y_train * np.log(y_train_pred) + (1 - y_train) * np.log(1 - y_train_pred))
dev_loss = -np.mean(y_dev * np.log(y_dev_pred) + (1 - y_dev) * np.log(1 - y_dev_pred))
test_loss = -np.mean(y_test * np.log(y_test_pred) + (1 - y_test) * np.log(1 - y_test_pred))

train_losses.append(train_loss)
dev_losses.append(dev_loss)
test_losses.append(test_loss)

print(f"Epoch {epoch+1}/{max_epochs} - Train Loss: {train_loss}, Dev Loss: {dev_loss}, Test Loss: {test_loss}")

if dev_loss < best_dev_loss:
    best_dev_loss = dev_loss
    best_w, best_b = w.copy(), b
    no_improvement = 0
else:
    no_improvement += 1
    if no_improvement > 10:
        print(f"Early stopping after {epoch+1} epochs")
        break
```

- Early stopping적용: early stopping을 적용하여, dev set상 성능 개선이 오랫동안 없는 경우 학습을 종료한다.