파이썬 입문

한국폴리텍대학

2023.06.09

PyTorch – numpy

```
# 1차원 x 1차원
a = np.array([1, 3, 5])
b = np.array([4, 2, 0])
print(np.dot(a, b))  # 1*4 + 3*2 + 5*0 = 10
print(np.matmul(a, b))  # or a@b (python 3.5+)

# 2차원 x 2차원
a = np.array([[1, 3], [2, 4]])
b = np.array([[1, 0], [2, 1]])
print(np.dot(a, b))  # [[7,3], [10, 4]]
print(np.matmul(a, b))

# 2차원 x 1 차원
a = np.array([[1, 3], [2, 4]])
b = np.array([2, 1])
print(np.dot(a, b))  # [5, 8]
print(np.matmul(a, b))

# 1차원 x 스칼라
a = np.array([1, 3, 5])
b = 2
print(np.dot(a, b))  # [2, 6, 10]
print(np.matmul(a, b))  # error
```

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

```
[1 3   [1 0
 2 4]   2 1]
```

```
[1 3   [2
 2 4]   1]
```

numpy.dot : 내적

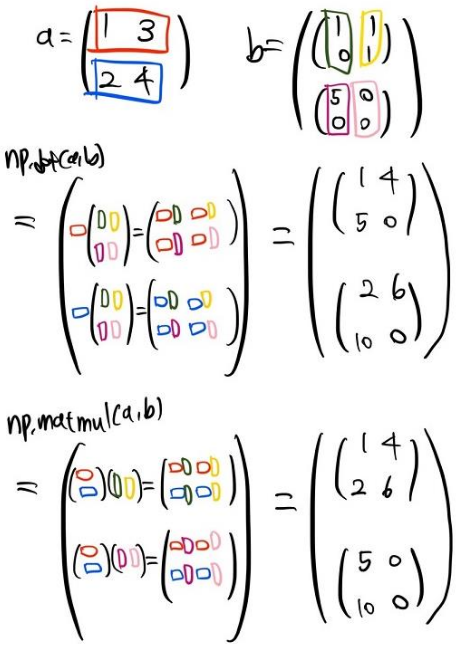
# dot product or inner product

numpy.matmul : 행렬곱 (@)

PyTorch – numpy

```
# 2차원 x 3차원
a = np.array([[1, 3], [2, 4]])
b = np.array([[[1, 1], [0, 1]], [[5, 0], [0, 0]]])
```

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



```
# n차원 x n차원
a = np.random.rand(9, 8, 5, 13)
b = np.random.rand(9, 8, 13, 3)
print(np.dot(a, b).shape)    #  (9, 8, 5, 9, 8, 3)
 print(np.matmul(a, b).shape)   # (9, 8, 5, 3)
```

(9, 8, 5, 13)

(9, 8, 13, 3)

(9, 8, 5, 13)

(9, 8, 13, 3)

PyTorch – numpy

브로드캐스팅 (broadcasting)

조건:
1. 요소가 1개
2. 행의 요소 수가 동일한 경우
3. 열의 요소 수가 동일한 경우

더하기, 빼기, 곱하기, 나누기

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 & 5 \\ 3 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} + \begin{bmatrix} 3 & 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 6 & 7 \\ 5 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} + \begin{bmatrix} 3 & 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 \\ 10 & 10 & 10 & 10 \end{bmatrix}$$

PyTorch – 선형회귀 (Linear Regression) 복습

dataset (데이터셋) :
예) 10,000개의 데이터 집합

epoch (에폭) :
예) 에폭 10 --> 10,000개의 데이터를 10회 학습

batch (mini-batch, 배치, 미니배치)
예) 1,000개씩 학습
데이터를 나눠서 학습. 데이터를 한꺼번에 학습하면 학습속도 및 효율이 낮음

iteration (이터레이션(반복))
예) 배치 학습 수. 1,000개 x 10 x 10
이터레이션 100회 (가중치(weight) 갱신(update) 100회)

learning rate (학습률)
예) 0 ~ 1 사이의 임의의 값. 0.001
학습속도를 조절할 수 있음. 큰 경우 예상치 못한 결과를 초래함.

PyTorch – 선형회귀 (Linear Regression) 복습

```
# y = 2x + 0  -- > y = wx + b : w = 2, b = 0

import torch    # pip install torch , or pip3.11 install torch

# train 데이터 생성
x_train = torch.FloatTensor([[1], [2], [3]])  # 학습데이터셋 (DataSet)
y_train = torch.FloatTensor([[2], [4], [6]])  # 정답 (Ground Truth)

w = torch.zeros(1, requires_grad=True) # 가중치(weight) 초기화
b = torch.zeros(1, requires_grad=True) # 편향(bias) 초기화

print(w)  # 출력물: tensor([0.], requires_grad=True)
print(b)  # 출력물: tensor([0.], requires_grad=True)

# 가설 세우기
hypothesis = x_train * w + b
print(hypothesis)
# model = torch.nn.Linear(1, 1)
# print(list(model.parameters()))

# Cost(Loss) function (MSE, Mean Squared Error)
cost = torch.mean((hypothesis - y_train) ** 2)
print(cost)  # 출력물: tensor(18.6667, grad_fn=<MeanBackward0>)

# 경사하강법
# Stochastic Gradient Descent 활용 - learning rate = 0.01
optimizer = torch.optim.SGD([w, b], lr=0.01)
```

```
epochs = 2000
for epoch in range(epochs + 1):
    # H(x)
    hypothesis = x_train * w + b
    # hypothesis = model(x_train) # or model.forward(x_train)

    # cost
    cost = torch.mean((y_train - hypothesis) ** 2)
    # cost = ((y_train - hypothesis) ** 2).mean()

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0:
        print(‘Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format(
                 epoch, epochs, w.item(), b.item(), cost.item()))

new_x = torch.Tensor([55])
new_y = new_x * w + b
# new_y = model(new_x)
print(new_y.item())   # number 요소가 1개인 tensor --> scalar값을 리턴
                      # [‘10’] --> error , [10, 20, 30]  --> error
```

PyTorch – 선형회귀 (Linear Regression) 복습

```
# gradient 초기화 (O)

print(f ' Epoch : {epoch+1:4d} ' )
print(f ' Step [1] : Gradient : {w.grad}, Weight : {w.item():.5f} ' )
optimizer.zero_grad()
print(f ' Step [2] : Gradient : {w.grad}, Weight : {w.item():.5f} ' )
cost.backward()
print(f ' Step [3] : Gradient : {w.grad}, Weight : {w.item():.5f} ' )
optimizer.step()
print(f ' Step [4] : Gradient : {w.grad}, Weight : {w.item():.5f}’)


# epoch 1
# cost = 18.6667  -->  ( ( 0 - 2 ) ** 2 + ( 0 - 4 ) ** 2 + ( 0 - 6 ) ** 2 ) / 3
# step [1] : Gradient : None , weight = 0.00000
# step [2] : Gradient : None  weight = 0.00000     --> zero_grad() : gradient(기울기)를 초기화
# step [3] : Gradient : -18.6667 , weight = 0.00000  --> backward() : 기울기 계산
# step [4] : Gradient : -18.6667 , weight = 0.18667  --> step() : 가중치(weight) 갱신(update)

# epoch 2
# cost = 14.7710  -->  ( ( 0.2667 - 2 ) ** 2 + ( 0.4533 - 4 ) ** 2 + ( 0.6400 - 6 ) ** 2 ) / 3
# step [1] : Gradient : -18.6667 , weight = 0.18667
# step [2] : Gradient : None  weight = 0.18667      --> zero_grad() : gradient(기울기)를 초기화
# step [3] : Gradient : -16.6044 , weight = 0.18667  --> backward() : 기울기 계산
# step [4] : Gradient : -16.6044 , weight = 0.35271  --> step() : 가중치(weight) 갱신(update)
```

PyTorch – 선형회귀 (Linear Regression) 복습

```
import torch
import torch.nn as nn
import torch.nn.functional as F

torch.manual_seed(1)

# 데이터
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])

class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(1, 1)  # input: x_train.shape[-1]
    def forward(self, x):
        return self.linear(x)

model = LinearRegressionModel()

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```
epochs = 2000
for epoch in range(epochs+1):
    # H(x) 계산
    prediction = model(x_train)

    # cost 계산
    cost = F.mse_loss(prediction, y_train)

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format( epoch, epochs, cost.item() ))
```