# 연구지도 및 연구윤리

## (2주차 - Ch1 정리)

경북대학교 전자공학부
2016113566 김남영

1. 기본 numpy 문법 복습

```python
import numpy as np

x = np.array([1, 2, 3]) # 행벡터
y = np.array([[1],[2],[3]]) # 열벡터
print(x)
print(y)
print(x.__class__) # 클래스 이름 표시
print(x.shape) # (3,)로 출력
print(y.shape) # (1,3)으로 출력
print(x.ndim) # 1로 출력 => 1차원(한 줄로 늘어서 있으므로)
print(y.ndim) # 2차원으로 본다

W = np.array([[1, 2, 3],[4, 5, 6]])
print(W)
print(W.shape) # (2,3)
print(W.ndim) # 2차원

W = np.array([[1,2,3], [4,5,6]])
X = np.array([[0,1,2], [3,4,5]])
print(W+X) # 원소별 연산
print(W*X) # 원소별 연산

A = np.array([[1,2], [3,4]])
print(A * 10) # 브로드캐스트되어 원소별 연산함

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

A = np.array([[1,2], [3,4]])
B = np.array([[5,6], [7,8]])
print(np.matmul(A, B)) # 행렬의 곱

#################

W1 = np.random.randn(2, 4)
b1 = np.random.randn(4)
x = np.random.randn(10, 2) #10개의 샘플 데이터를 이용
h = np.matmul(x, W1) + b1
# (10,2) 행렬과 (2, 4) 행렬이 곱해져 (10,4) 행렬이 만들어지고
# b1은 (10, 4)로 브로드캐스트 되어 더해진다.
```

2. Sigmoid 함수 구현 및 Matmul 함수를 통한 신경망 흐름 복습

```python
import numpy as np

def sigmoid(x):
    return 1/(1+np.exp(-x))

x = np.random.randn(10,2)
W1 = np.random.randn(2, 4)
b1 = np.random.randn(4)
W2 = np.random.randn(4, 3)
b2 = np.random.randn(3)

h = np.matmul(x, W1) + b1
a = sigmoid(h) # a는 (10,4) 행렬
s = np.matmul(a, W2) + b2
# (10, 4) 행렬과 (4, 3) 행렬의 곱으로 (10, 3) 행렬이 되고
# b2가 (10, 3)으로 브로드 캐스트되어 더해진다.

print(s)
# 즉 출력층은 (10, 3) 행렬의 형상이고
# 2개의 입력을 가진 10쌍의 입력이 3개의 출력을 가진 10쌍의 출력으로 변환된 것을 의

# 3개의 출력을 가지므로 3개로 분류가 가능하다는 의미
```

3. Sigmoid와 Affine을 계층으로 구현

```python
import numpy as np

# 계층 구현 규칙
# 모든 계층은 forward()와 backward() 메서드를 가진다.
# 모든 계층은 인스턴스 변수인 params와 grads를 가진다.

class Sigmoid: #sigmoid 계층
    def __init__(self):
        self.params, self.grads = [], []
        self.out = None

    def forward(self, x):
        out = 1/(1+np.exp(-x))
        self.out = out #인스턴스 변수로 저장해서 backward에서 쓰려고
        return out

    def backward(self, dout): #dout은 상류에서 하류로 흘러오는 값
        dx = dout * (1.0 - self.out) * self.out # 시그모이드를 미분하면 y(1-y)
        return dx

class Affine: # 완전 연결 계층
    def __init__(self, W, b):
        self.params = [W, b] #초기화하면 W, b로 이루어진 리스트가 생성
        self.grads = [np.zeros_like(W), np.zeros_like(b)]
        self.x = None

    def forward(self, x):
        W, b = self.params
        out = np.matmul(x, W) + b
        self.x = x
        return out # 입력을 받아서 앞서 배운대로 변환해서 전달

    def backward(self, dout):
        W, b = self.params
        dx = np.matmul(dout, W.T)
        dW = np.matmul(self.x.T, dout)
        db = np.sum(dout, axis=0) # b가 브로드캐스트 되므로 거꾸로 갈때는 sum

        self.grads[0][...] = dW
        self.grads[1][...] = db

        return dx
```

4. TwoLayerNet 구현

```python
class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size):
        I, H, O = input_size, hidden_size, output_size

        W1 = np.random.randn(I, H)
        b1 = np.random.randn(H)
        W2 = np.random.randn(H, O)
        b2 = np.random.randn(O)
        # 매개 변수를 생성

        self.layers = [
            Affine(W1, b1),
            Sigmoid(),
            Affine(W2, b2)
        ]
        # 생성된 매개변수로 신경망을 위한 계층을 생성한 것

        self.params = [] # 파라미터 저장을 위한 리스트가 생성되고
        for layer in self.layers:
            self.params += layer.params
            # layer.params는 sigmoid, affine 계층에 있는 params를 의미함
            # Affine.params, Sigmoid.params 겠지..
            # 리스트 값을 더하는게 아니라 결합하는 것임.
            # 따라서 모든 계층의 매개변수가 리스트형태로 결합되어 저장됨

    def predict(self, x):
        for layer in self.layers:
            x = layer.forward(x)
        return x


x = np.random.randn(10, 2)
model = TwoLayerNet(2, 4, 3)
s = model.predict(x)
print(s)
```

5. 분기에 대한 이해

```python
import numpy as np

D, N = 8, 7
x = np.random.randn(1, D)
y = np.repeat(x, N, axis=0) # x를 N개로 분기
dy = np.random.randn(N, D) # 기울기를 그냥 무작위로 설정한것
dx = np.sum(dy, axis=0, keepdims=True) # 역전파에서는 총 합이므로

print(dx)

D, N = 8, 7
x = np.random.randn(N, D)
y = np.sum(x, axis=0, keepdims=True)

dy = np.random.randn(1, D) # 기울기 무작위로 설정
dx = np.repeat(dy, N, axis=0)
```

6. Matmul 계층 구현(역전파까지)

```python
import numpy as np

class MatMul:
    def __init__(self, W):
        self.params = [W]
        self.grads = [np.zeros_like(W)]
        self.x = None

    def forward(self, x):
        W, = self.params
        out = np.matmul(x, W)
        self.x = x
        return out

    def backward(self, dout):
        W, = self.params
        dx = np.matmul(dout, W.T)
        dW = np.matmul(self.x.T, dout)
        self.grads[0][...] = dW # [...]하면 배열 덮어쓰기함(메모리고정)

        return dx
```

7. SGD 구현

```python
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr # lr은 학습률

    def update(self, params, grads):
        for i in range(len(params)): # 첫번째 차원의 행 수 반환
            params[i] -= self.lr * grads[i]
            # 미니배치된 입력 데이터마다 파라미터와 기울기가 존재하므로
```

8. Trainer 클래스를 이용한 신경망 학습

```python
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from common.optimizer import SGD
from common.trainer import Trainer
from dataset import spiral
from two_layer_net import TwoLayerNet


# 하이퍼파라미터 설정
max_epoch = 300
batch_size = 30
hidden_size = 10
learning_rate = 1.0

x, t = spiral.load_data()
model = TwoLayerNet(input_size=2, hidden_size=hidden_size, output_size=3)
optimizer = SGD(lr=learning_rate)

trainer = Trainer(model, optimizer) # 트레이너 클래스를 이용해서 학습
trainer.fit(x, t, max_epoch, batch_size, eval_interval=10)
trainer.plot()
```

9. TwoLayerNet 구현

```python
import sys
sys.path.append('..')  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import Affine, Sigmoid, SoftmaxWithLoss # 미리 다 추가해놓음

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size):
        I, H, O = input_size, hidden_size, output_size

        W1 = 0.01 * np.random.randn(I, H)
        b1 = np.zeros(H)
        W2 = 0.01 * np.random.randn(H, O)
        b2 = np.zeros(O)
        # 가중치, 편향 초기화

        self.layers = [
            Affine(W1, b1),
            Sigmoid(),
            Affine(W2, b2)
        ] # 초기화된 파라미터로 계층 생성
        self.loss_layer = SoftmaxWithLoss() # 손실함수를 계산하는 계층

        self.params, self.grads = [], [] # 계층마다 params와 grads를 모음
        for layer in self.layers:
            self.params += layer.params
            self.grads += layer.grads

    def predict(self, x):
        for layer in self.layers:
            x = layer.forward(x) # 각 계층에서 순전파해서 통과
        return x

    def forward(self, x, t): # 순전파하여 통과한 값으로 손실값을 계산
        score = self.predict(x)
        loss = self.loss_layer.forward(score, t)
        return loss

    def backward(self, dout=1):
        dout = self.loss_layer.backward(dout)
        for layer in reversed(self.layers):
            dout = layer.backward(dout)
        return dout
```

10. 8번에서 Trainer 클래스 사용 시 오류

```
  File "D:\coding\spyder
\ch3\two_layer_net.py", line 25, in __init__
    self.params += layer.params

AttributeError: 'Affine' object has no
attribute 'params'
```

## 고찰

1. 8번에서 Trainer 클래스 사용 시 Affine 계층에 params가 없다고 오류가 발생하는데, 아무리 확인해봐도 Affine 계층을 생성하면 params가 초기화 되게끔 코드가 짜여져 있었다. 무엇 때문에 오류가 발생했는지 다시 한 번 확인해봐야겠다.

2. 다시 한 번 신경망을 복습하다보니 까먹은 부분이 많았고, 아직도 이해되지 않는 코드도 많았다. 꼼꼼히 정리하는 시간을 가져야 할 것 같다.