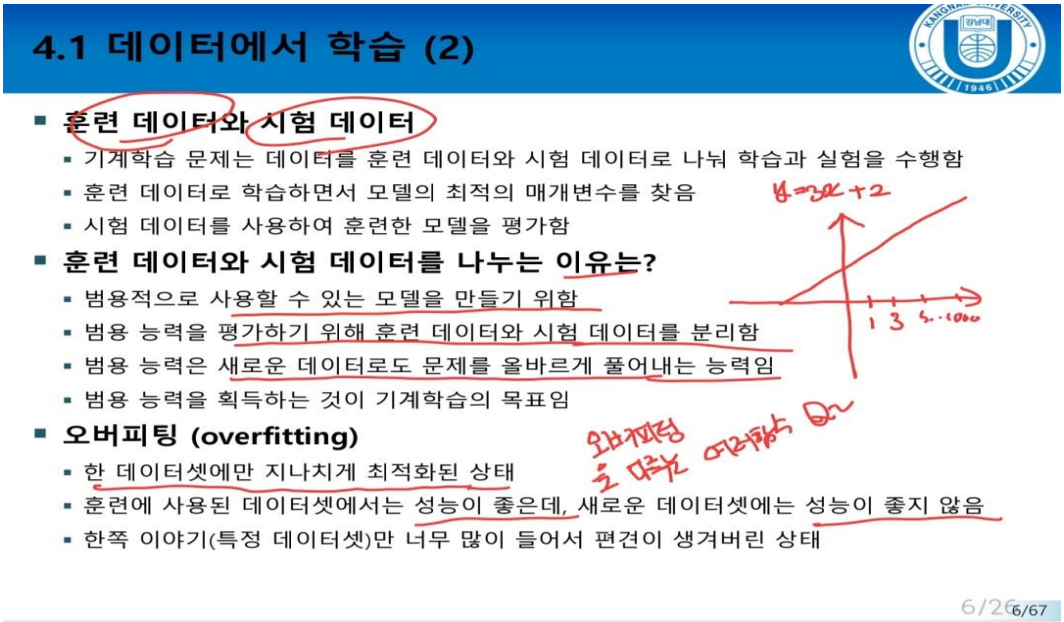
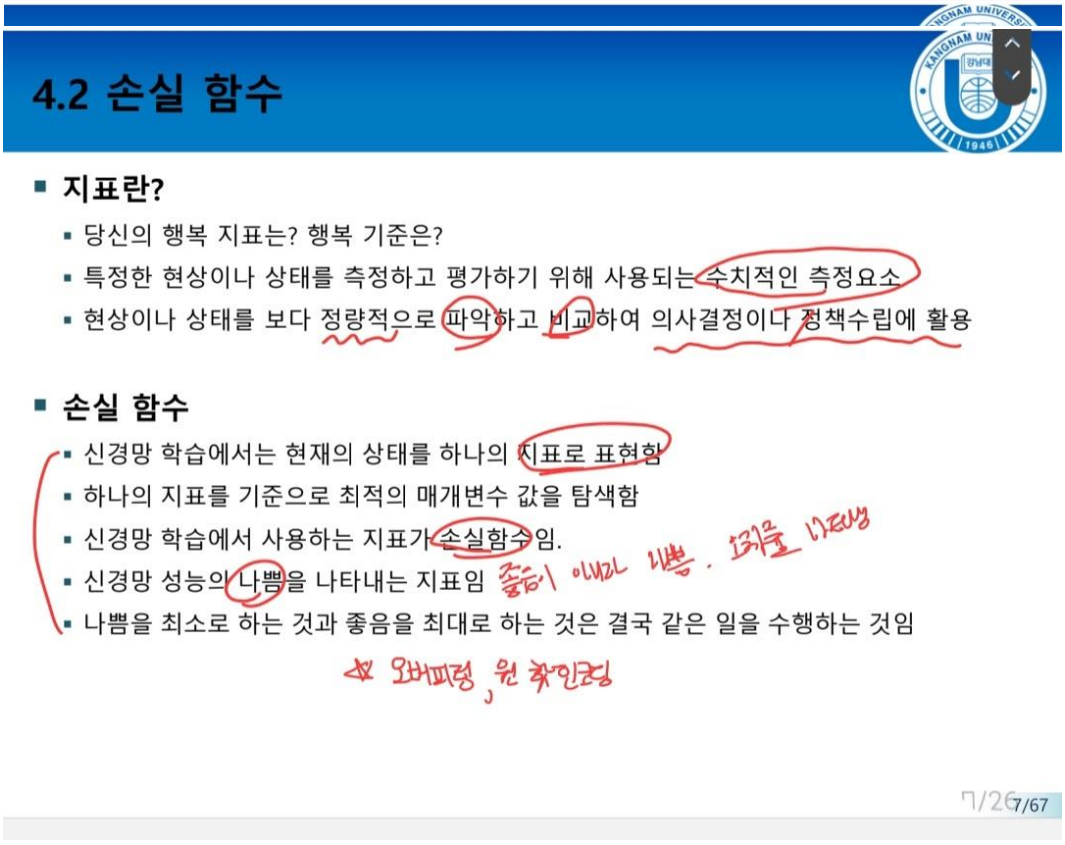
정리노트#2

소프트웨어응용학부 202004051 박용선, ICT 융합공학부 202204023 김보민

<신경망학습>

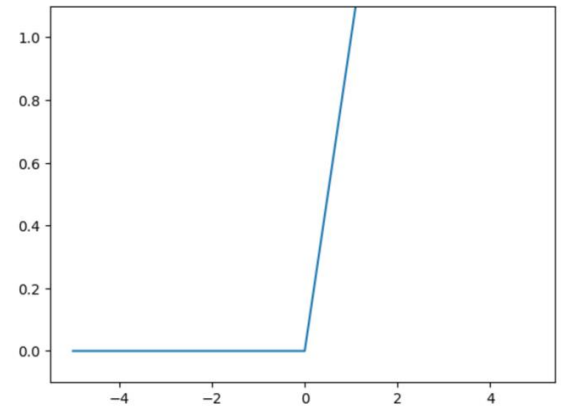




<실습>

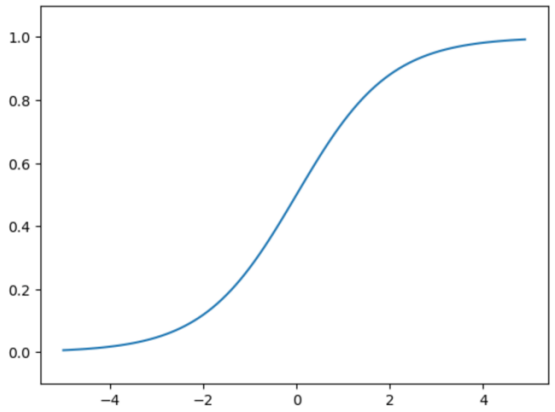
```
def relu(x):
    return np.maximum(0, x)
```

```
X = np.arange(-5.0, 5.0, 0.1)
Y = relu(X)
plt.plot(X,Y)
plt.ylim(-0.1, 1.1)
plt.show()
```




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

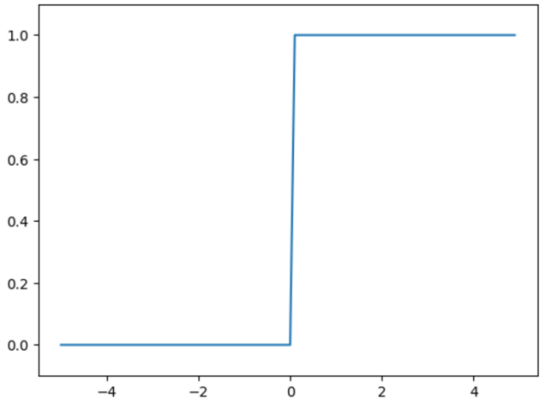
```
X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X,Y)
plt.ylim(-0.1, 1.1)
plt.show()
```




```
import numpy as np
import matplotlib.pylab as plt

def step_function(x):
    return np.array(x>0, dtype = np.int32)
```

```
X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X,Y)
plt.ylim(-0.1, 1.1)
plt.show()
```



```
import numpy as np
A = np.array([1,2,3,4])
print(A)
```

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

```
np.ndim(A)
```

```
1
```

```
A.shape
```

```
(4,)
```

```
A.shape[0]
```

```
4
```

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

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

```
np.ndim(B)
```

```
2
```

```
B.shape
```

```
(3, 2)
```

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

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

```
W.shape
```

```
(2, 3)
```

```
X = np.array([1.0, 0.5])
W1 = np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])
B1 = np.array([0.1,0.2,0.3])
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
(2, 3)
(2,)
(3,)
```

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

```
X = np.array([1.0, 0.5])
W1 = np.array([[0.1, 0.3, 0.5],[0.2,0.4,0.6]])
B1 = np.array([0.1,0.2,0.3])
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
(2, 3)
(2,)
(3,)
```

```
A1 = np.dot(X,W1) + B1
print(A1)
```

```
[0.3 0.7 1.1]
```

```
Z1 = sigmoid(A1)

print(A1)
print(Z1)
```

```
[0.3 0.7 1.1]
[0.57444252 0.66818777 0.75026011]
```

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

```
W2 = np.array([[0.1,0.4],[0.2,0.5],[0.3,0.6]])
B2 = np.array([0.1,0.2])

print(Z1.shape)
print(W2.shape)
print(B2.shape)
```

```
(3,)
(3, 2)
(2,)
```

```
A2 = np.dot(Z1, W2) +B2
Z2 = sigmoid(A2)
```

```
def identity_function(x):
    return x
```

```
W3 = np.array([[0.1,0.3], [0.2,0.5]])
B3 = np.array([0.1,0.2])

A3 = np.dot(Z2, W3) + B3
Y = identity_function(A3)

print(Y)
```

```
[0.31682708 0.77338016]
```

```
def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a
    return y
```

```
a = np.array([0.5,3.0,5.0])
y = softmax(a)
print(y)
print('sum = ' , np.sum(y))
```

```
[0.00968996 0.11804785 0.87226219]
sum =  1.0
```

[4]:
```
import numpy as np

def mean_squared_error(y, t) :
    return 0.5 * np.sum((y-t) ** 2)


t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

x = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

mean_squared_error(np.array(x), np.array(t))
```

[4]:
```
0.5975
```

9]:
```
import numpy as np

def cross_entropy_error(y, t) :
    delta = 1e-7
    return -np.sum(t*np.log(y + delta))


t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]

cross_entropy_error(np.array(x), np.array(t))
```

9]:
```
2.302584092994546
```

:
```
import numpy as np

def cross_entropy_error(y, t) :
    delta = 1e-7
    return -np.sum(t*np.log(y + delta))


t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

cross_entropy_error(np.array(x), np.array(t))
```

:
```
2.302584092994546
```

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
        load_mnist(normalize = True, one_hot_label = True)

print(x_train.shape)
print(t_train.shape)

np.random.choice(60000, 10)

train_size = x_train.shape[0]
batch_size = 10
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]

print(x_batch.shape)
print(t_batch.shape)


def numerical_diff(f, x) :
    h = 1e - 4
    return (f(x + h) - f(x - h ) / 2 * h)
```

```
(60000, 784)
(60000, 10)
(10, 784)
(10, 10)
```

[8]:
```
def numerical_diff(f, x) :
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)

def function_1 (x) :
    return 0.01 * x ** 2 + 0.1 * x

numerical_diff(function_1, 5)
```

[8]:
```
0.1999999999990898
```

[ ]:
```

```

:
```
def numerical_diff(f, x) :
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)

def function_1 (x) :
    return 0.01 * x ** 2 + 0.1 * x

numerical_diff(function_1, 10)
```

:
```
0.2999999999986347
```

:
```

```

```python
import numpy as np

def partial_diff(x):
    return x[0] ** 2 + x[1] ** 2

def numerical_gradient(f, x):
    h = 1e-4
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]

        x[idx] = float(tmp_val) + h
        fxh1 = f(x)

        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2 * h)
        x[idx] = tmp_val

    return grad

print(numerical_gradient(partial_diff, np.array([3.0, 4.0])))
print(numerical_gradient(partial_diff, np.array([0.0, 2.0])))
print(numerical_gradient(partial_diff, np.array([3.0, 0.0])))
```

```
[6. 8.]
[0. 4.]
[6. 0.]
```

```python
import numpy as np

def mean_squared_error(y, t) :
    return 0.5 * np.sum((y-t) ** 2)

t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]


mean_squared_error(np.array(y), np.array(t))
```

```
0.09750000000000003
```

Q1

용선: 기계 학습 분류 기법인 SVM, KNN가 궁금해

보민: 나도 잘 모르는 거라 각각 하나씩 알아보고 설명해주는 방식으로 공부해보자, 내가 SVM에 대해서 공부해볼게.

용선: 그래

보민: SVM은 Support Vector Machine의 약자로 기계학습에서 사용되는 분류 기법중 하나야. 이는 데이터를 분류하기 위해서 결정 경계를 찾는데 사용한대. 이 경계는 클래스 간의 간격을 최대화하도록 하기에 이를 통해 데이터를 분류할 때 오류를 최소화하기 위한 것이라고 해. 그리고 SVM은 다양한 커널 함수를 사용해서 데이터를 매핑할 수 있다는 특징을 가지고 있지.

용선: 그렇구나, KNN은 용어가 어려워서 이해하기 좀 어려웠는데 쉽게 말해 학습 단계가 존재하지 않고 예측 시점에서만 주어진 데이터의 이웃을 검색해서 예측하는 것을 의미해. 따라서 KNN은 학습 데이터 분포에 민감하고 데이터가 많은 경우에는 계산비용이 높아지게 되지. 이 학습 분류 기법은 적절한 k 값을 찾는게 중요하며 이 값이 모델의 성능에 큰 영향을 미친다고 알면 될 거 같아.

보민: 아 그러면 KNN은 학습 단계가 없기에 바로 사용할 수 있다는 점과 이웃 데이터에 의해 예측되기에 이상치에 덜 민감하다고 볼 수 있는 장점을 가졌네

용선: 네가 이해한 게 맞아. 단점도 정리해보자면 예측 시점에서 모든 학습 데이터 간의 거리를 계산해야 하기에 계산 비용이 높지. 그리고 적절한 이웃의 수 즉, k를 선택하기 어렵다는 단점이 있지.

보민: 아하 그럼 이제 SVM도 장단점을 정리해보자

용선: 음 내가 이해한 건 다양한 커널 수를 사용하면 다양한 종류의 데이터에 적용할 수 있지 않나 싶어. 잘 이해하지 못한 거 같아..

보민: 방금 설명한 것도 맞아 그치만 내가 조금 더 자세히 설명해줄게. SVM은 주어진 데이터를 고차원 공간으로 매핑해서 클래스를 분리하는 최적의 초평면을 찾는 것을 의미해. 그렇기 때문에 선형/비선형 문제에 모두 적용될 수 있지. 여기서 초평면이란 쉽게 말하면 다차원 공간에서 차원을 하나씩 낮춘 부분공간을 의미해. 2차원에서는 직선, 3차원에서는 평면이 두 부분으로 나뉘는 것처럼 말이지.

용선: 아 이제 좀 이해가 돼. 그럼 이거는 선형 문제를 해결할 수 있고 경계를 찾는게 중요한 역할을 하겠구나.

보민: 맞아. SVM은 결정 경계를 찾는 데 초점을 맞추고 있으면서 데이터의 분포에 따라 복잡한 모양의 결정 경계를 만들 수 있다고 알고 있으면 될 거 같아.

용선: 좋아. 같이 공부하고 서로 설명해주면서 이해하니까 이해가 더 잘 되는 느낌이야

보민: 나도. 열심히 해보자.

Q2

보민: 인공지능에서 미분이 어떤 역할을 할까 ?

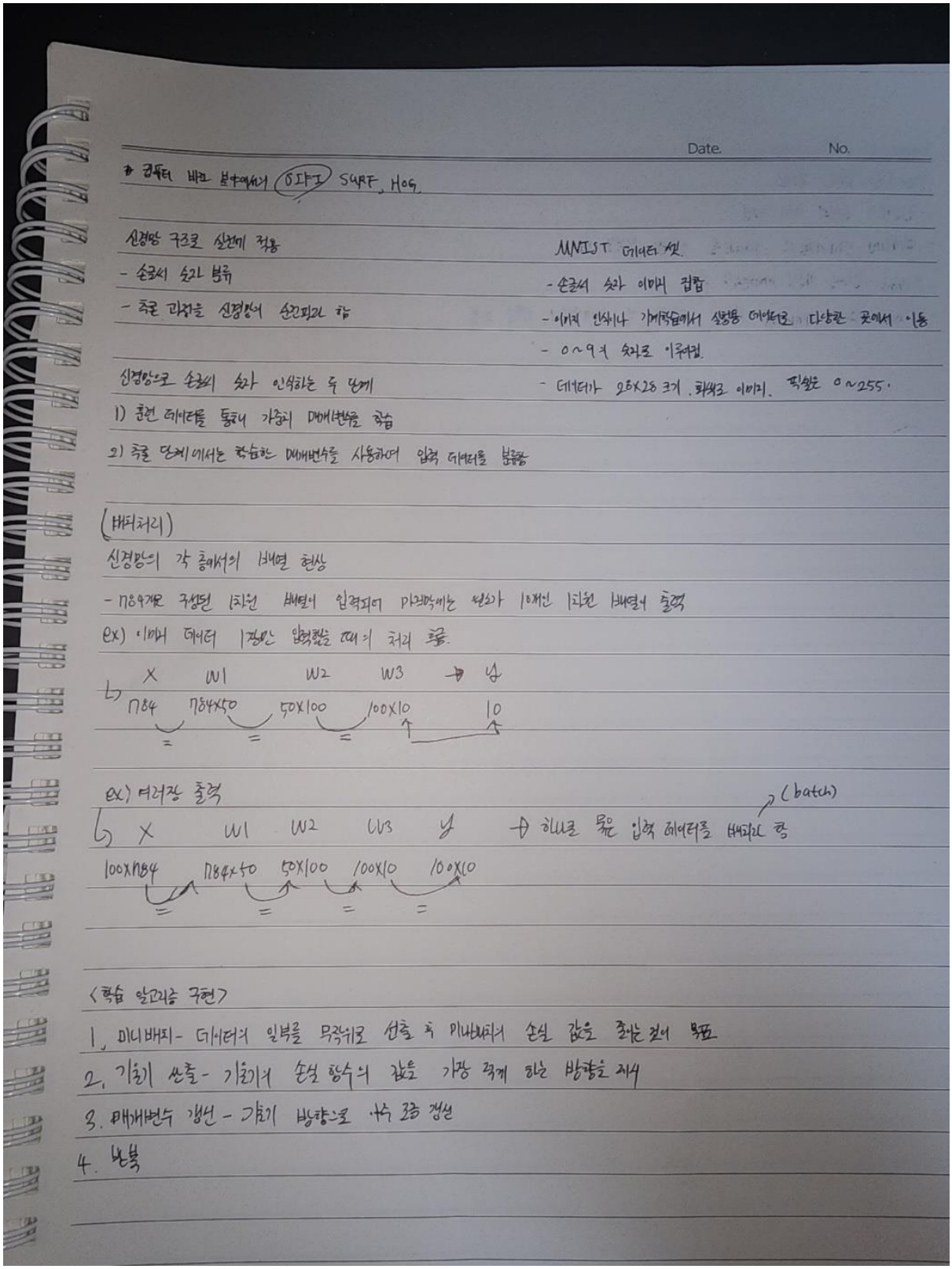
용선: 인공지능에서 미분은 주로 딥러닝 모델의 학습과 최적화에 사용돼. 미분은 함수의 변화율을 나타내니까 이를 통해서 손실 함수의 기울기를 계산하지. 그리고 경사 하강법이라는 것도 있는데 이는 손실 함수의 기울기를 사용해서 모델의 파라미터를 업데이트 하는 최적화하는 알고리즘을 의미해.

보민: 그러면 미분은 손실 함수의 기울기를 사용해서 모델의 가중치를 조정하고 더 나은 예측을 할 수 있도록 해주는 역할을 하나 ?

용선: 맞아. 잘 이해했어. 좀 더 자세히 설명하자면 딥러닝 모델은 여러 층의 뉴런으로 이루어져 있는데, 역전파 알고리즘을 사용해서 각 층의 가중치를 조정할 때 미분을 사용하는 거지. 역전파 알고리즘은 오차를 역방향으로 전파하면서 각 층의 가중치에 대한 미분을 계산함으로써 모델이 어디서 실수를 했는지 알고 그에 맞게 가중치를 조절할 수 있다는 걸 알 수 있지.

보민: 아.. 너무 이해가 잘 돼.. 인공지능에서 미분은 모델의 학습과 최적화에 되게 중요한 역할을 하는 거였구나..

<영상강의 노트정리>



Date No.

컴퓨터 비전 분야에서 이미지 SURF, HOG

신경망 구조로 실제에 적용
- 손글씨 숫자 분류
- 즉로 과정을 신경망이 순전파로 함

MNIST 데이터 셋
- 손글씨 숫자 이미지 집합
- 이미지 인식나 기계학습에서 실험용 데이터로 다양한 곳에서 이용
- 0~9까지 숫자로 이루어짐
- 데이터가 28X28 크기, 회색조 이미지, 픽셀은 0~255.

신경망으로 손글씨 숫자 인식하는 두 단계
1) 훈련 데이터를 통해 가중치 매개변수를 학습
2) 추론 단계에서는 학습한 매개변수를 사용하여 입력 데이터를 분류함

(배치처리)

신경망의 각 층에서의 배열 현상
- n차원 구성된 1차원 배열이 입력되어 m개의 뉴런이 1개인 1차원 배열이 출력

ex) 이미지 데이터 1장만 입력했을 때의 처리 흐름

X W1 W2 W3 → Y

784 784X50 50X100 100X10 10

ex) 여러장 출력 (batch)

X W1 W2 W3 Y → 하나로 묶은 입력 데이터를 배치라고 함

100X784 784X50 50X100 100X10 100X10

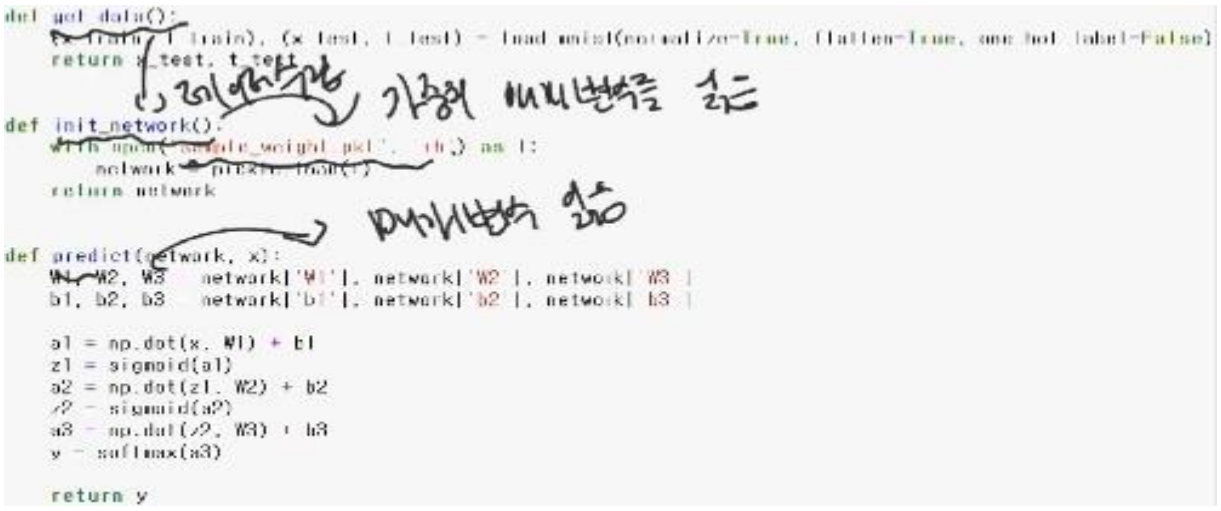
〈학습 알고리즘 구현〉

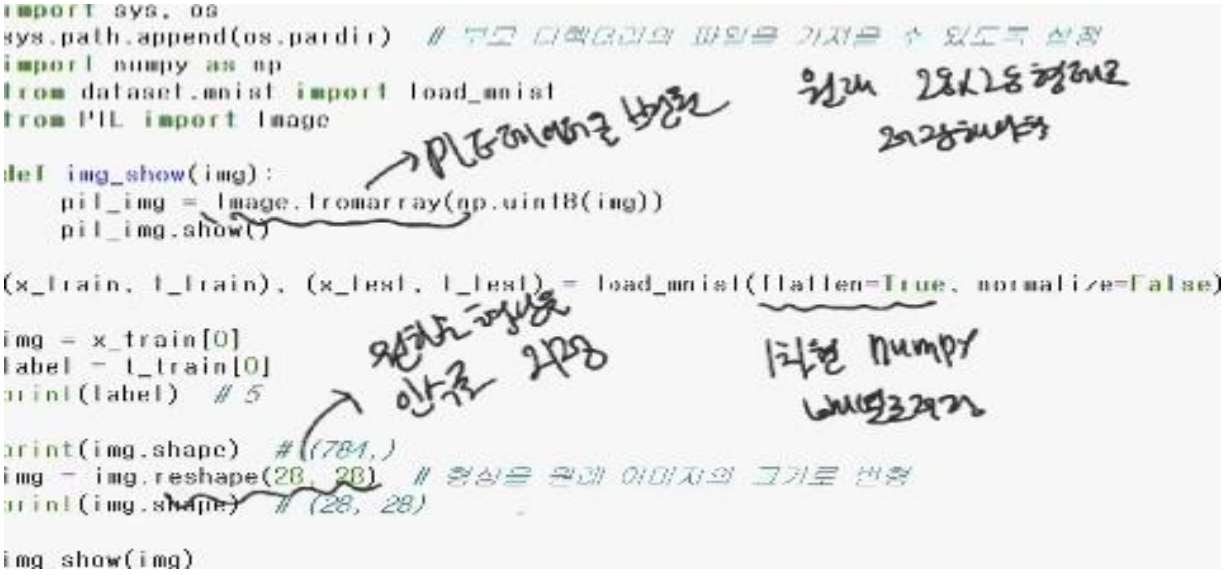
1. 미니배치 - 데이터의 일부를 무작위로 선출 후 미니배치의 손실 값을 줄이는 것이 목표
2. 기울기 산출 - 기울기가 손실 함수의 값을 가장 작게 하는 방향을 제시
3. 매개변수 갱신 - 기울기 방향으로 조금 갱신
4. 반복

Date. No.

Two LayerNet 클래스

- 신경망의 순전파 처리 구현
- 딕셔너리인 params와 grads를 인스턴스 변수로 갖음.
- params 변수에는 가중치 매개변수가 저장
- grads 변수에는 params 변수에 대응하는 각 매개변수의 기울기 저장





```python
import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=False, flatten=True, one_hot_label=False)

# 각 데이터의 형상 출력
print('x_train = ',x_train.shape, 't_train = ',t_train.shape)
print('x_test = ',x_test.shape, 't_test = ',t_test.shape)
```

```
x_train =  (60000, 784) t_train =  (60000,)
x_test =  (10000, 784) t_test =  (10000,)
```

Q3

용선: 에폭이 진행될수록 훈련 데이터와 테스트 데이터의 평가 정확도가 모두 좋아지는 이유가 뭔지 자세하게 알려줘.

보민: 우선 에폭에 대해서 이해해야 할 거 같아. 나는 에폭이 전체 데이터셋을 모델에 통과시켜 데이터를 사용해 모델을 학습시키는 횟수라고 이해했어. 그렇기에 에폭을 여러번 거치면 거칠수록 데이터를 여러번 학습시키므로써 모델은 데이터의 패턴을 학습하기에 예측 성능이 향상할 수 밖에 없게 되지.

용선: 아 그러면 에폭이 돌면 돌수록 데이터의 학습량이 증가하기에 더 좋은 모델이 만들어질 수 있는 거구나. 그렇다면 테스트 데이터의 평가 정확도가 좋아지는 이유는 무엇일까 ?

보민: 에폭이 돌수록 모델이 훈련 데이터 뿐만 아니라 다양한 데이터에서 유용한 패턴을 학습하고 일반화할 수 있기 때문에 테스트 데이터에서도 좋은 성능을 내게 되지. 에폭의 수에 따라 훈련 데이터와 테스트 데이터의 성능 향상은 비례관계라고 생각하면 될 거 같아.

용선: 아 그렇구나. 내가 에폭을 완벽히 이해하지 못하고 있었다는 걸 깨달았어 알려줘서 고마워.

Q4

보민: 신경망 학습에서 기울기가 잘 이해가 안돼..

용선: 음 우선 이론적으로 설명하자면 손실 함수의 파라미터에 대한 편미분 값들의 모음이야. 이 값은 경사 하강법과 같은 최적화 알고리즘에서 사용되어 모델의 파라미터를 업데이트 하는데 중요한 역할을 하지.

보민: 그러면 기울기는 어떻게 구할 수 있는데 ?

용선: 기울기는 손실 함수를 각 파라미터로 편미분해서 구할 수 있어. 각 파라미터에 대한 기울기는 해당 파라미터를 어느 방향으로 움직여야 손실을 줄일 수 있는 지를 나타내지. 그래서 경사 하강법은 기울기를 이용해서 손실을 줄이는 방향으로 모델의 파라미터를 조금씩 업데이트하는 특징을 가지고 있어.

보민: 다른 방법은 없어 ?

용선: 역전파 알고리즘을 사용하는 방법도 있어. 역전파 알고리즘은 출력층에서부터 입력층까지 역방향으로 진행하면서 각 층의 가중치와 편향에 대한 기울기를 계산해. 이때 체인룰을 이용하여 각 층의 기울기를 계산하지. 여기서 체인룰이란 합성 함수의 미분을 계산하는 방법이야.

보민: 아 그렇구나. 근데 손실 함수를 각 파라미터로 편미분해서 구하면 계산 비용이 많이 들 거 같아.. 그래도 수치적으로는 직관적이고 안정적일 수는 있지만.. 역전파 알고리즘을 사용하는 게 더 많이 쓰일 거 같다

용선: 맞아 체인룰을 사용해서 각 층의 미분을 계산하기 때문에 효율적으로 기울기를 구할 수 있어 보통 신경망 학습에서 가장 널리 사용되지

보민: 고마워. 덕분에 정리가 잘 됐어.