4장 학습 내용 소프트웨어 202284011 김연재

학습이란? 훈련 데이터로부터 가중치 매개변수의 최적값을 자동을 획득하는 것을 뜻함
한 데이터셋에만 지나치게 최적화된 오버피팅을 피해야함

손실 함수: 신경망 학습에선 현재의 상태를 하나의 지표로 표현함 하나의 지료를 기준으로 최적의 매개변수 값을 탐색함 신경망 성능의 나쁨을 나타내는 지표

In [ ]:
```
# 평균 제곱 오차
## 가장 많이 쓰이는 손실 함수

import numpy as np

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

In [ ]:
```
# t는 정답 테이블
# 한 원소만 1로 하고 나머지를 0으로 나타내는 표기법을 원-핫 인코딩
# 정답은 '2'
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# y는 신경망의 출력(신경망이 추정한 값), 소프트맥스 함수의 출력임
# '2'일 가능성이 가장 높다고 추정함
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))
```

Out[ ]:
```
0.09750000000000003
```

결과가 0.09750000000000003로 오차가 적게 나타남

In [ ]:
```
# 정답은 똑같이 2이지만
# '7'일 가능성이 가장 높다고 추정함
y = [0.1, 0.05, 0.0, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

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

Out[ ]:
```
0.6925
```

결과가 0.6925로 오차가 크게 나타남

In [ ]:
```
#교차 엔트로피 오차
def cross_entropy_error(y,t):
    delta = 1e-7
    return -np.sum(t*np.log(y+delta))

#아주 작은 delta값을 넣는 이유는 np.log() 함수에 0을 입력하면 마이너스 무한대가 나
```

In [ ]:
```
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(y), np.array(t))
```

Out[ ]:
```
0.510825457099338
```

In [ ]:
```
y = [0.1, 0.05, 0.0, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
cross_entropy_error(np.array(y), np.array(t))
```

Out[ ]:
```
16.11809565095832
```

기계학습 문제의 학습 방법 훈련 데이터에 대한 손실 함수의 값을 구하고, 그 값을 최대한 줄여주는 매개변수를 찾음 모든 훈련 데이터를 대상으로 손실 함수를 구해야 함 훈련 데이터가 100개 있으면 계산한 100개의 손실 함수 값들의 합을 지표로 삼음 합을 N으로 나눔으로써 평균 손실 함수를 구함.

미니배치 모든 데이터의 손실 함수 합을 구하려면 시간이 걸림 따라서 데이터의 일부를 추려 근사치로 이용함

In [ ]:
```
import sys, os
sys.path.append(os.pardir)

# load_mnist 함수는 훈련 데이터와 시험 데이터를 읽는다.
from dataset.mnist import load_mnist

(x_train, t_train), (x_text, t_text) = load_mnist(normalize=True, one_hot

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

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

In [ ]:
```
# 이 훈련 데이터에서 무작위로 10장만 빼내려면
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]

# 이 함수가 출력한 배열을 미니배치로 뽑아낼 데이터의 인덱스로 사용하면 된다.
np.random.choice(60000,10)
```

Out[ ]:
```
array([35865, 48445, 51494,  9272, 18743, 21386, 31819, 18440, 21601,
       25077])
```

(배치용) 교차 엔트로피 오차 구현하기

In [ ]:
```
def cross_entropy_error(y,t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(t*np.log(y+1e-7))/batch_size
```

왜 정확도가 아니라 손실 함수를 사용하는가? 자동으로 최적값을 얻기 위해선 미분을 사용해야 함 미분의 역할: 매개변수의 값을 서서히 갱신하는 과정을 반복 미분값이 음수면 매개변수를 양의 방향으로, 미분값이 양수면 음의 방향으로 변화시켜 손실 함수의 값을 줄임

정확도를 지표로 하면 매개변수 미분이 대부분 0이 됨 계단 함수를 활성화 함수로 사용하지 않는 이유와 같음.(신경망 학습이 잘 안됨)

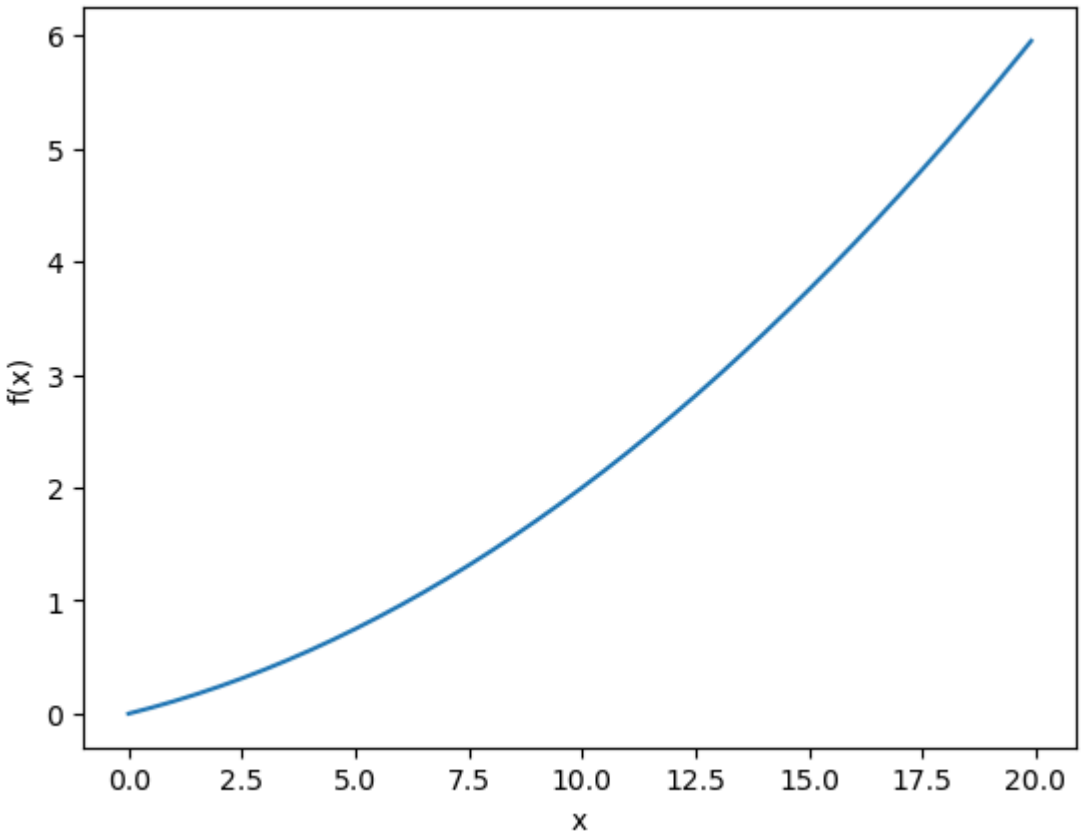
In [ ]:
```
# 경사법에서는 기울기 값을 기준으로 나아갈 방향을 정한다.
# 미분을 파이썬으로 구현

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

In [ ]:
```
def func_1(x):
    return 0.01*x**2 + 0.1*x

import matplotlib.pylab as plt

x = np.arange(0.0, 20.0, 0.1)
y = func_1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x,y)
plt.show()
```



In [ ]:
```
numerical_diff(func_1, 5)
```

Out[ ]:
```
0.1999999999990898
```

In [ ]:
```
numerical_diff(func_1, 10)
```

Out[ ]:
```
0.2999999999986347
```

In [ ]:
```
# 편미분
def partial_diff(x):
```

```
    return x[0]**2 + x[1]**2
```

편미분 - 여러 개의 변수로 이루어진 다변수 함수의 각 변수에 대한 미분을 계산하는 것

In [ ]:
```
def partial_diff_0(x0):
    return x[0]**2 + x[1]**2
```

편미분과 기울기. 기울기는 모든 변수의 편미분을 벡터로 정리한 것이다.

In [ ]:
```
def numerical_diff(f, x):
    h = 1e-4
    grad = np.zeros_like(x)

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

기울기가 의미하는 것은? 기울기는 함수의 가장 낮은 장소(최솟값)을 가리키는 것 같음
기울기는 각 지점에서 낮아지는 방향을 가르킴

신경망 학습에서 기울기 신경망은 최적의 매개변수(가중치와 편향)을 학습 시에 찾아야 함 최적이란 손실 함수가 최솟값이 될 때의 매개변수 값임

경사법 경사법은 현 위치에서 기울어진 방향으로 일정 거리만큼 이동 이동한 곳에서 기울기를 구하고, 그 기울어진 방향으로 나아가기를 반복하며 함수의 값을 줄여감

In [ ]:
```
#경사법(경사 하강법)
# f는 최적화 함수, init_x는 초깃값, lr은 학습률, step_num은 경사법에 따른 반복 횟
# 기울기에 학습률을 곱한 값으로 갱신하는 처리를 step_num번 반복한다.

import numpy as np

def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x

    for i in range(step_num):
        grad = numerical_diff(f, x)
        x -= lr*grad

    return x

init_x = np.array([-3.0, 4.0])
gradient_descent(partial_diff, init_x=init_x, lr=0.1, step_num=100)
```

Out[ ]:
```
array([-6.11110793e-10,  8.14814391e-10])
```

In [ ]:
```
#신경망으로 기울기를 구하는 코드
#softmax와 cross_entropy_error 함수, numerical_gradient 메서드를 사용
```

```python
#simpleNet 클래스는 가중치 매개변수를 하나만 가진 간단한 신경망이다.
#predict 메서드는 예측을 수행하고, loss 메서드는 손실 함수의 값을 구한다.
#numerical_gradient 메서드는 가중치 매개변수의 기울기를 구한다.

class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3)

    def predict(self, x):
        return np.dot(x, self.W)

    def loss(self, x, t):
        z = self.predict(x)
        y = softmax(z)
        loss = cross_entropy_error(y, t)

        return loss

net = simpleNet
x = np.array([0.6, 0.9])
p = net.predict(x)
np.argmax(p)
t = np.array([0, 0, 1])
net.loss(x, t)

def f(W):
    return net.loss(x, t)

dW = numerical_gradient(f, net.W)
```

질문 1) 신경망이 왜 중요한가? 신경망은 무엇에 사용되는가?

신경망은 인간이 일일이 개입하지 않고도 컴퓨터가 지능적인 결정을 내리는 데 도움이 될 수 있다. 신경망은 현재 컴퓨터 비전(이미지와 동영상을 인식하는 기능), 음성 인식, 검색 엔진의 추천 기능 등에 사용되고 있다.

질문 2) 현재 상용화된 기술들에 사용하는 신경망 학습 기법중에 가장 성능이 좋은것은 무엇인가?

각 분야에 따라서 다른 기술이 사용되기 때문에 가장 좋다는 기술 하나만을 선택하는 것은 어렵다. 최근 가장 많이 사용되며 높은 성능을 보이는 신경망 학습 기법은 다음과 같다.

- 심층 신경망 (DNNs): 딥러닝의 핵심이 되는 기법 중 하나로, 다층으로 쌓인 신경망 구조를 의미한다. 대규모 데이터셋과 충분한 연산 자원을 활용하여 학습된 심층 신경망은 이미지 인식, 음성 인식, 자연어 처리 등 다양한 분야에서 뛰어난 성능을 보인다.
- 컨볼루션 신경망 (CNNs): 이미지 처리에 특화된 신경망으로, 이미지 내의 공간적 구조를 고려하여 특징을 추출하고 분류하는 데 사용된다. 컴퓨터 비전 분야에서 주요 기술로 사용되며, 이미지 분류, 객체 검출, 분할 등의 작업에서 우수한 성능을 보인다.
- 순환 신경망 (RNNs): 순차적인 데이터 처리에 적합한 신경망으로, 이전의 정보를 현재 상태에 반영하여 시퀀스 데이터를 처리한다. 자연어 처리, 음성 인식, 시계열 예

측 등에 널리 사용되며, LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Unit)와 같은 변형이 성능 향상에 기여한다.

- 변이형 오토인코더 (Variational Autoencoders, VAEs): 생성 모델의 한 종류로, 데이터의 분포를 학습하여 새로운 데이터를 생성하는데 사용된다. 이미지 생성, 데이터 임베딩, 데이터 복원 등에 활용되며, 확률적인 요소를 고려하여 더욱 유연한 모델을 제공한다.
- 강화 학습 (Reinforcement Learning)
- 변환 모델 (Transformers)