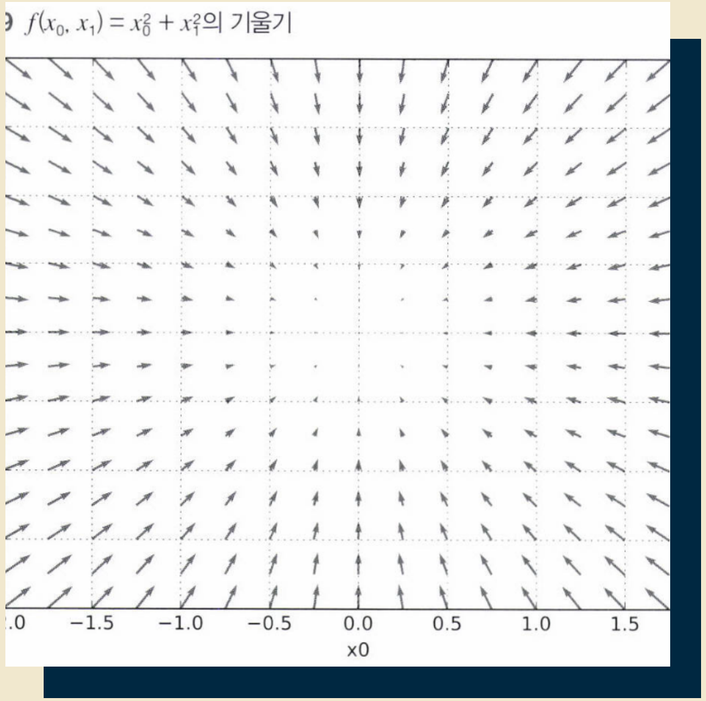
04.

기울기

경사법, 신경망에서의 기울기

Simple Presentation_04

기울기



모든 변수의 편미분을 벡터로 정리한 것을 기울기라고 한다
기울기의 결과에 마이너스를 붙인 벡터를 그려보면 왼쪽의 사진과 같다
기울기는 각 지점에서 낮아지는 방향을 가리킨다. 기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향이라고 볼 수 있다.

Simple Presentation_04

기울기

```
def numerical_gradient(f,x):
    h = 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열 형성

    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = tmp_val + h
        fxh1 = f(x)

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

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 값 복원
    return grad
print(numerical_gradient(function_2,np.array([3.0,4.0])))
print(numerical_gradient(function_2,np.array([0.0,2.0])))
print(numerical_gradient(function_2,np.array([3.0,0.0])))
```

f: 미분하려는 함수.
x: 기울기를 계산하려는 점(배열 형태로 입력).
h: 작은 값(미세한 변화량). 수치 미분의 오차를 줄이기 위해 사용.
grad: x 와 같은 형상을 가지며, 각 요소에서의 기울기를 저장할 배열.
for 루프를 통해 x 의 각 요소마다 기울기를 계산.
tmp_val: x[idx] 값을 저장해 두고, 미분 계산 후 복원.
fxh1: f(x + h) 값을 계산.
fxh2: f(x - h) 값을 계산.
중심 차분 방식(central difference method)으로 기울기 계산

Simple Presentation_04

경사법(경사 하강법)

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

기울기를 잘 이용해서 함수의 최소값을 찾으려는 것이 경사법이다.
경사법은 현 위치에서 기울어진 방향으로 일정 거리만큼 이동한다.
그 다음 이동한 곳에서도 마찬가지로 기울기를 구하고, 또 그 기울어진 방향으로 나아가기를 반복한다.
이렇게 해서 함수의 값을 점차 줄이는 것이 경사법.

η는 갱신하는 양을 나타낸다. 이를 신경망 학습에서는 학습률이라고 한다.
한 번의 학습으로 얼마만큼 학습해야 할지, 매개변수 값을 얼마나 갱신하느냐를 정하는 것이 학습률

신경망 학습에서는 보통 이 학습률 값을 변경하면서 올바르게 학습하고 있는지를 확인하면서 진행

Simple Presentation_07

경사 하강법 코드

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

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

f는 최적화하려는 함수, init_x는 초기값, lr는 learning rate, step_num은 경사법에 따른 반복 횟수
함수의 기울기는 numerical_gradient(f,x)로 구하고 그 기울기에 학습률을 곱한 값으로 갱신하는 처리를 step_num번 반복
학습률이 너무 크면 큰 값으로 발산하고 반대로 너무 작으면 거의 갱신되지 않은 채 끝나버리므로 학습률을 적절히 설정하는 것이 중요
학습률 같은 매개변수를 하이퍼파라미터라고 함
이는 가중치와 편향 같은 신경망의 매개변수와는 성질이 다른 매개변수
신경망의 가중치 매개변수는 훈련 데이터와 학습 알고리즘에 의해 자동으로 획득되는 매개변수인 반면,
학습률 같은 하이퍼파라미터는 사람이 직접 설정해야 하는 매개변수

Simple Presentation_07

신경망에서의 기울기

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

W : 가중치
L : 손실함수
dL/dW: 경사

dL/dW의 각 원소는 각각의 원소에 관한 편미분이다.
dL/dW의 형상이 W와 같다.

Simple Presentation_07

simpleNet 클래스

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient

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

    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
```

```
>>> def f(W):
...     return net.loss(x, t)
...
>>> dW = numerical_gradient(f, net.W)
>>> print(dW)
[[ 0.21924763  0.14356247 -0.36281009]
 [ 0.32887144  0.2153437  -0.54421514]]
```

x는 입력 데이터, t는 정답 레이블을 의미
predict(x) : 예측을 수행
loss(x,t) : 손실함수 값을 구함
numerical_gradient(f, x): 기울기 구함

Simple Presentation_07

학습 알고리즘 구현하기

1단계 – 미니배치

훈련 데이터 중 일부를 무작위로 가져옵니다. 이렇게 선별한 데이터를 미니배치라 하며, 그 미니배치의 손실 함수 값을 줄이는 것이 목표입니다.

2단계 – 기울기 산출

미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구합니다. 기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시합니다.

3단계 – 매개변수 갱신

가중치 매개변수를 기울기 방향으로 아주 조금 갱신합니다.

4단계 – 반복

1~3단계를 반복합니다.

Simple Presentation_07

학습 알고리즘 구현하기

```python
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)

train_loss_list = []

iters_num = 10000  # 경사법 반복 횟수
train_size = x_train.shape[0]
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1 에폭 당 반복 수(여기서는 600회)
iter_per_epoch = max(train_size / batch_size, 1)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
```

```python
for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print("train acc, test acc | " + str(train_acc) + ", " + str(t
```

Thank You

www.MIRIcompany.com
000.000.0000