Deep-Learning from Scratch

4장

최우진

2018/07/10

학습 목표 : 손실 함수의 결과값을 가장 작게 만드는 가중치 매개변수를 찾는 것.

학습 : 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것을 뜻함.

4.1 데이터에서 학습한다.

- **신경망의 특징: 데이터를 보고 학습할 수 있는 점**

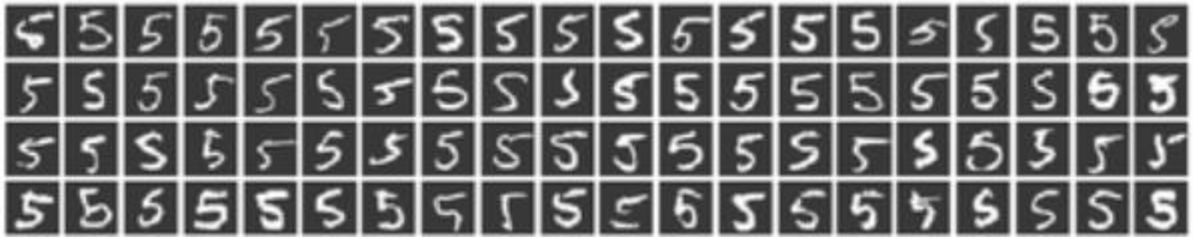
1) **데이터 주도 학습**

기계학습은 데이터에서 답을 찾고 패턴을 발견하고 이야기를 만드는 것이다.

그러므로 기계학습은 데이터가 생명이다.

4.1.1 데이터 주도 학습

Ex) 손 글씨 숫자 ‘5’



손 글씨 숫자 ‘5’에서 패턴을 찾기는 어렵다.

대신 이미지에서 특징을 추출하고 그 특징의 패턴을 기계학습 기술로 학습한다.

4.1.1 데이터 주도 학습

딥러닝 : 종단간 기계학습(사람의 개입없이 결과를 얻음)

4.1.2 훈련 데이터와 시험 데이터

- 기계학습 문제는 데이터를 훈련 데이터와 시험 데이터를 나눠서 학습과 실험을 수행한다.
- 훈련 데이터만 사용하여 학습을 통해 최적의 매개변수를 찾는다.
- 시험 데이터로 훈련한 모델의 실력을 평가한다.
- 오버 피팅: 한 데이터셋에만 지나치게 최적화된 상태

4.2 손실 함수

- 손실 함수 : 신경망 성능의 나쁨을 나타내는 지표, 즉 훈련 데이터를 얼마나 잘 처리하지 못하는지를 나타냄

- 종류

1) 평균 제곱 오차

2) 교차 엔트로피 오차

4.2.1 평균 제곱 오차

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

y_k = 신경망의 출력
t_k = 정답 레이블
k = 데이터의 차원 수

예시 1번과 예시 2번을 실행한 결과를 보면 예시 1번의 오차가 더 작으므로 평균 제곱 오차를 기준으로는 첫 번째 추정 결과가 정답에 더 가까울 것으로 판단할 수 있다.

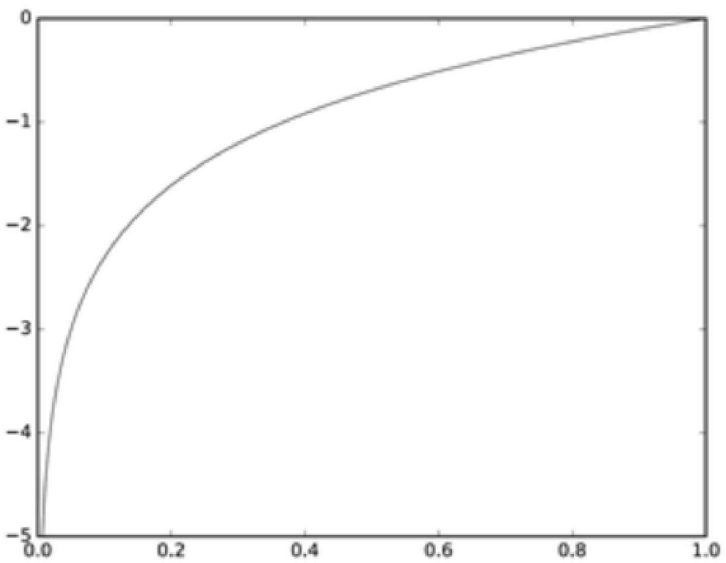
(소스)

```
import numpy as np

# 평균 제곱 오차
def mean_squard_error(y, t):
    return 0.5 * np.array((y - t)**2)

# 정답은 '2'
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 예1 : '2' 일 확률이 가장 높다고 추정함(0.6)
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
mean_squard_error(np.array(y),np.array(t))
# 0.09750000000000031

# 예2 : '7' 일 확률이 가장 높다고 추정함(0.6)
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
mean_squard_error(np.array(y),np.array(t))
# 0.5975000000000003
```

4.2.2 교차 엔트로피 오차

$$E = -\sum_k t_k \log y_k$$

y_k = 신경망의 출력
t_k = 정답 레이블
k = 데이터의 차원 수



자연로그 함수는 x=1 일 때 y=0이 되고, x가 0에 가까워질수록 y의 값은 점점 작아진다.
교차 엔트로피 오차 식 또한 정답일 때의 출력이 작아질수록 오차는 커진다.

4.2.2 교차 엔트로피 오차

(소스)

```python
import numpy as np

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

# 정답은 '2'
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 예1 : '2' 일 확률이 가장 높다고 추정함(0.6)
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))
# 0.510825457099338O2

# 예2 : '7' 일 확률이 가장 높다고 추정함(0.6)
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(y), np.array(t))
# 2.3025840929945458
```

예시 1번은 정답일 때의 출력이 0.6인 경우로,
교차 엔트로피 오차는 약 0.51 이다.
예시 2번은 정답일 때의 출력이 0.1인 경우로,
교차 엔트로피 오차는 약 2.30이다.
즉, 정답일 때의 출력이 큰 경우가
결과(오차)가 작다(정답일 가능성이 높다).

4.2.3 미니배치 학습

- 데이터의 양이 많은 경우, 모든 데이터를 대상으로 손실 함수의 합을 구하려면 시간이 오래 걸린다.
- 이런 경우 데이터 일부를 추려 전체의 '근사치'('미니배치')로 이용하여 일부만 골라 학습을 수행하는 것을 '미니배치 학습'이라고 한다.

4.2.3 미니배치 학습

MNIST의 데이터 셋을 읽어오는 코드

```
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)  # (60000, 784)
print(t_train.shape)  # (60000, 10)
```

데이터 셋에서 무작위로 10개의 데이터만 가져오는 코드

```
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]

np.random.choice(60000, 10)
# array([ 8013, 14666, 58210, 23832, 52091, 10153, 8107, 19410, 27260, 21411])
```

4.2.4 (배치용) 교차 엔트로피 오차 구현하기

데이터가 하나인 경우와 데이터가 배치로 묶여 입력된 경우
모두를 처리할 수 있도록 구현한 코드

```
def cross_entropy_error(y, t):
    # 데이터가 하나 일 경우(1차원)
    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)) / batch_size
```

y = 신경망의 출력
t = 정답 레이블
배치의 크기로 나눠 **정규화**하고 이미지 1장당
평균의 교차 엔트로피 오차를 계산합니다

원-핫 인코딩이 아닌 '2' 나 '7'등의 숫자 레이블로 주어졌을 때의 코드

```
def cross_entropy_error(y, t):
    # 데이터가 하나 일 경우(1차원)
    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[np.array(batch_size), t])) / batch_size
```

**원-핫 인코딩일 때 t가 0인 원소는
교차 엔트로피 오차도 0이므로, 그
계산은 무시해도 좋다는 것이 핵심**

4.2.5 왜 손실 함수를 설정하는가?

- 궁극적인 목적은 높은 '정확도'를 끌어내는 매개변수 값을 찾는 것이다.
- 정확도를 지표로 정하게 되면, 대부분의 장소에서 0이 되어 매개 변수가 더 이상 갱신이 안된다.
- 정확도 => 불연속적인 값 (ex. 계단 함수)
- 손실 함수 => 연속적인 값 (ex. 시그모이드 함수)

4.3 수치 미분
4.3.1 미분

- **미분 : ‘특정 순간’의 변화 량**

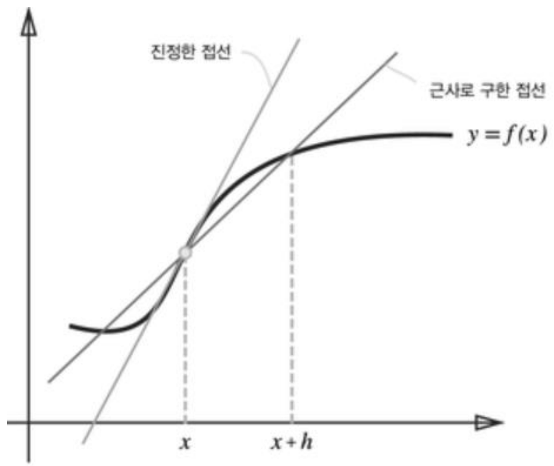
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

```
# 나쁜 구현 예
def numerical_diff(f, x):
    h = 10e-50
    return (f(x+h) - f(x)) / h
```

위 코드에서 개선 할 점

1. h에 작은 값을 대입하고 싶었기에 10e-50을 사용하였으나,
   이 방식은 **반올림 오차** 문제를 일으킨다.
2. h가 무한히 0으로 좁히는 것이 불가능해 진정한 미분과
   구현한 미분에 대해서 **차이**가 발생

4.3.1 미분




```python
# 개선된 미분 함수
def numberical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2 * h)
```

아주 작은 차분으로 미분을 구하는 것을 **수치 미분**이라고 한다.

진정한 접선과 근사로 구한 접선의 오차를 줄이기 위해, (x+h) 와 (x-h)일 때의 함수 f의 **차분**을 계산하는 방법을 사용한다.

4.3.2 수치 미분의 예

예 $y = 0.01x^2 + 0.1x$

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

```
numerical_diff(function_1, 5)
# 0.1999999999990898
numerical_diff(function_1, 10)
# 0.2999999999986347
```

(코드)

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

x = np.arange(0.0, 20.0, 0.1)  # 0 ~ 20 까지 0.1 간격의 배열 x를 만든다.
y = function_1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x,y)
plt.show()
```

df(x)/dx = 0.02x + 1 에 x = 5 와 x = 10 을 각각 대입 했을 때, 0.2 와 0.3이 나오므로 실제로 거의 같은 값이라고 해도 될 만큼 작은 오차이다.

4.3.3 편미분

- **변수가 여럿인 함수에 대한 미분**
- **식 :** $f(x_0, x_1) = x_0^2 + x_1^2$
- **코드 :**

```
def function_2(x):
    return x[0]**2 + x[1]**2 # return np.sum(x**2)
```

- **예 :**

```
# x0 = 3, x1 = 4일때, x0에 대한 편미분을 구하라.

def function_tmp1(x0):
    return x0 * x0 + 4.0 ** 2.0

numerical_diff(funcrion_tmp1, 3.0)
# 6.00000000000378
```

4.4 기울기

- **모든 변수의 편미분을 벡터로 정리한 것을 기울기라고 한다.**
- **소스 :**

```
def numerical_gradient(f, x):
    h = 1e-4  # 0.0001
    grad = np.zeros_like(x)  # x와 형상이 같은 배열을 생성

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

기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 줄이는 방향이다.

- **예) (3,4)에서의 기울기**

```
numerical_gradient(function_2, np.array([3.0, 4.0]))
# array([6., 8.])
```

4.4.1 경사법(경사 하강법)

- 1. 현 위치에서 기울어진 방향으로 일정 거리만큼 이동
- 2. 이동한 곳에서도 마찬가지로 기울기를 구함
- 3. 또 그 기울어진 방향으로 나아가는 일을 반복
- 이 방법을 통해 함수의 값을 점차 줄이는 것이 **경사법**이다.
- 수식 :

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

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

η (학습률)은 너무 크거나 작으면 올바르게 학습을 할 수 없다.

4.4.1 경사법(경사 하강법)

- **코드 :**

```python
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
```

함수 f는 최적화하려는 함수
init_x = 초깃값
lr = 학습률
step_num = 경사법에 따른 반복 횟수

- **예 :**

```python
# 경사법으로 f = x0**2 + x1**2의 최솟값을 구하라.

def function_2(x):
    return x[0]**2 + x[1]**2

init_x = np.array([-3.0, 4.0])
gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
# array([-6.11110793e-10,  8.14814391e-10])
```

4.4.2 신경망에서의 기울기

- **가중치 매개변수에 대한 손실 함수의 기울기**

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \end{pmatrix}$$

결과 : [[0.21924763 0.14356247 -0.36281009]
[0.32887144 0.2153437 -0.54421514]]

신경망을 예로 들어 실제로 기울기를 구하는 코드

```
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

x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])

net = simpleNet()

f = lambda w: net.loss(x, t)
dw = numerical_gradient(f, net.W)

print(dw)
```

4.5 학습 알고리즘 구현하기

전제

신경망에는 적응 가능한 가중치와 편향이 있고, 이 가중치와 편향을 훈련데이터에 적응하도록 조정하는 과정을 '학습'이라 합니다. 신경망 학습은 다음과 같이 4단계로 수행합니다.

1단계 – 미니배치

훈련 데이터 중 일부를 무작위로 가져옵니다. 이렇게 선별한 데이터를 미니배치라 하며, 그 미니배치의 손실함수 값을 줄이는 것을 목표로 한다.

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 ~ 3 단계를 반복합니다.

4장

감사합니다