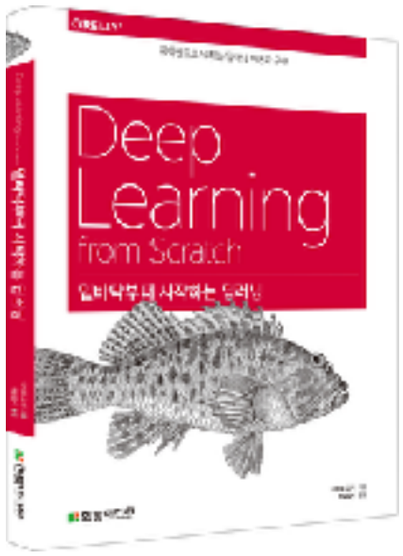

『밑바닥부터 시작하는 딥러닝』
(원서 : ゼロから作る Deep Learning)

Date:2018.01.27
Tim

목차

- 수치 미분
    - 미분: 미분은 ‘특정 순간’의 변화량을 뜻함.(미분의 기본 개념을 한 마디로 압축한다면 **변화율**이다.)
    - Ex: 마라톤
        - 10분에서 2km 달렸다.
        - 1분에 0.2km(평균 속도)
        - 미분은 ‘ 특정 순간’ 의 변화량을 뜻함.
            - 즉 직전 1분에서 달린 거리 or 직전 0.1초에 달린거리,... 식으로 갈수록 간격을 줄여 어느한 순간의 변화량 어느 한순간의 속도를 뜻함.

- 공식:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{(x + \Delta x) - x}$$

- f(x): x에 대한 미분(x에 대한 f(x)의 변화량)
  - x의 ‘작은 변화’가 함수 f(x)를 얼마나 변화 시켰는가 뜻함.

- **수치 미분:** 컴퓨터에서 무한대로 0에 가까운 변화량을 사용할수 없기때문에 수치미분 즉 근사로 구한 접선으로 사용함.
    - 아주 작은 차분(근사치)으로 미분하는 것을 수치 미분이라함.
    - 미분은 진정한 접선을 의미(교재 p123 참고)
- **반올림 오차:** 작은 값(가령 소수점 8자리 이하)이 생략되어 최종 계산 결과에 오차가 생기는 현상.
- **중심 차분(중앙 차분):** 오차를 줄이기 위해(x+h)와(x-h) 일 때의 함수 f의 차분을 x를 중심으로 그 전후의 차분을 계산하는 방법.
    - (x+h) 의 차분을 **전방 차분** 이라함

- 편미분(p126)
    - 정의: 변수가 여럿인 함수에 대한 미분을 편미분이라고 함.
    - 계산 방법: 여러 변수 중 목표 변수 하나에 초점을 맞추고 다른 변수는 값을 고정 하는 방식.
        - 목표 변수를 제외한 나머지를 특정 값에 고정하기 위해서 새로운 함수를 정의
        - 위 새로 정의한 함수에 대해 그동안 사용한 수치 미분 함수를 적용하여 편미분을 구함.

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



# 4.3.1 미분
# 나쁜 구현 예
def numerical_diff_bad(f, x):
    h = 10e-50
    return (f(x + h) - f(x)) / h
# h값이 너무 작아 반올림 오차를 일으킬 수 있음 10e-4정도가 적당하다고 알려짐
# 전방 차분에서는 차분이 0이 될 수 없어 오차가 발생
#  -> 오차를 줄이기 위해 중심 차분을 사용



def numerical_diff(f, x):
    h = 10e-4
    return (f(x + h) - f(x - h)) / (2 * h)
```

```python
# 4.3.2 수치 미분의 예
# y = 0.01x² + 0.1x
def function_1(x):
    return 0.01*x**2 + 0.1*x



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



# x = 5, 10일때 미분
print(numerical_diff(function_1, 5))   # 0.200000000000089
print(numerical_diff(function_1, 10))  # 0.29999999999996696
```

```python
# 접선의 함수를 구하는 함수
def tangent_line(f, x):
        d = numerical_diff(f, x)
        # print(d)
        y = f(x) - d*x
        return lambda t: d*t + y



tf = tangent_line(function_1, 5)
y2 = tf(x)
plt.plot(x, y2)
plt.show()
```

```python
# 4.3.3 편미분
# f(x0, x1) = x0² + x1²
def function_2(x):
    return x[0]**2 + x[1]**2
    # or return np.sum(x**2)



# x0 = 3, x1 = 4일 때, x0에 대한 편미분을 구하라.
def function_tmp1(x0):
    return x0**2 + 4.0**2.0


# x0 = 3, x1 = 4일 때, x1에 대한 편미분을 구하라.
def function_tmp2(x1):
    return 3.0**2.0 + x1 * x1



print(numerical_diff(function_tmp1, 3.0))  # 5.999999999998451
print(numerical_diff(function_tmp2, 4.0))  # 8.000000000000895
```

- 4.4 기울기(p127)
    - 모든 변수의 편미분을 벡터로 정리한 것을 기울기(gradient)라고 한다.

```python
# coding: utf-8
# cf.http://d.hatena.ne.jp/white_wheels/20100327/p3
import numpy as np
import matplotlib.pylab as plt
from mpl_toolkits.mplot3d import Axes3D


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

```python
def _numerical_gradient_no_batch(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] = float(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
```

- 기울기라는 게 의미하는 건 뭘까요?
    - 기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향(포인트)

- 4.4.1 경사법(경사 하강법)
  - 기계학습 문제 대부분 학습 단계 에서 최적의 매개 변수 찾기.
  - 최적의 매개 변수(가중치와 편향) 최적이란 손실 함수가 최솟값이 될 때의 매개변수 값
    - 매개 변수 공간이 광대하기 때문에 상황에서 기울기를 잘 이용해 함수의 최솟값(또는 가능한 한 작은 값)을 찾으려는 것이 경사법.

- Warning :
  - 함수가 극솟값, 최솟값 또 안장점(saddle point)이 되는 장소에서는 기울기가 0.
  - 극솟값은 국소적인 최솟값 즉 한정된 범위에서의 최솟값인 점.
  - 안장점은 어느 방향에서 보면 극댓값이고 다른방향에서 보면 극솟값이 되는 점.
  - 경사법은 기울기가 0인 장소를 찾지만 그것이 반드시 최솟값이라고 할수 없다.

- 경사법(gradient method)은 함수의 값을 점차 줄이는 방법이다. (p131)
    - 최솟값은 찾는 방법-> 경사 하강법(gradient descent method)
    - 최댓값을 찾는 방법-> 경사 상승법(gradient ascent method)
- 갱신하는 양-> 학습률(learning rate): 한 번의 학습으로 얼마만큼 학습해야 할지, 즉 매개변수 값을 얼마나 갱신하느냐를 정의.
- 학습률 같은 매개변수 & 사람이 직접 설정-> 하이퍼파라미터 (Hyper parameter, 초매개변수)

- 신경망에서의 기울기(p133)
  - 가중치 매개변수에 대한 손실 함수의 기울기.# coding: utf-8

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
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)
```

- 가중치 매개 변수에 대한 손실 함수의 기울기(p135)

- 파이썬에서는 새로운 함수를 정의하는 데”def f(x):...” 문법을 썼는데 이것을 람다(lambda) 기법 이라고 한다.

Reference

- http://www.modulabs.co.kr/DeepLabEDU_library/14261
- http://sungminoh.tistory.com/6
- http://www.iamroot.org/xe/index.php?mid=AI&document_srl=191307
- http://nbviewer.jupyter.org/github/ziwon/deep-learning-from-scratch/blob/master/ch04-about-gradient.ipynb

“ Q & A”

–Have a good day!!!