ゼロから作るDeep learning
4.3 数値微分
4.4　　　勾配

s1240094 Miyuka Nakamura

数値微分

微分…ある瞬間の変化の量

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

このまま実装すると、

hをできるだけ小さくしたい…h=10e-50にしてみる！

→丸め誤差によって良い計算結果が得られない

「誤差」を軽減するために…

- 改善ポイント
  - ポイント1：h=1e-4にする
  - ポイント2：関数fの差分（中心差分で求める）

```python
def numerical_diff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)
```

中心差分…xを中心として、その前後の差分を計算すること

偏微分

- 偏微分…複数の変数からなる関数の微分
- 例： $f(x_0, x_1) = x_0^2 + x_1^2$

数式で表すと、 $$\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}$$

勾配

- 勾配…全ての変数の偏微分をベクトルとしてまとめたもの

$$\left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right)$$

勾配の算出方法

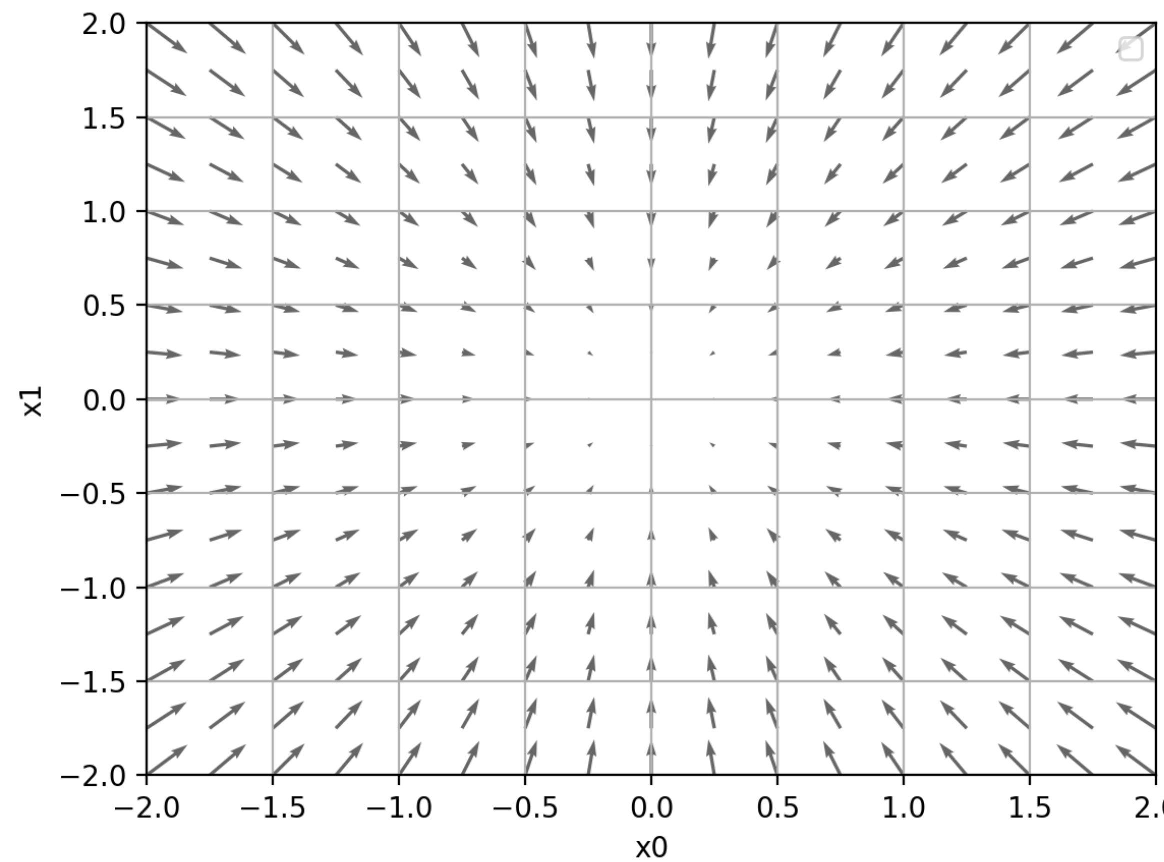
```python
def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

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

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

        x[idx] = tmp_val # 値を元に戻す

    return grad
```



勾配法

- 損失関数が最小値をとるときのパラメータを探すのは難しいので勾配をうまく利用して関数の最小値を探す（→勾配法）
- 勾配は関数の値を最も減らす方向であるが、それが必ずしも最小値だとは限らない

勾配法の進め方

現在の場所から勾配方向に一定の距離進む

移動した先でも同様に勾配を求め、進む

- これを繰り返すことで損失関数の値を徐々に減らす。
- このように最小値を探すことを勾配降下法、最大値を探すことを勾配上昇法という。

学習率

- 学習率…1回の学習でどれだけ学習すべきか、どれだけパラメータを更新するかを決める値

- 学習率のようなパラメータをハイパーパラメータと言い、人の手で値を適切に決める必要がある

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

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

ニューラルネットワークに対する勾配

- ニューラルネットワークに対する勾配とは、重みパラメータに関する損失関数の勾配である。

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

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

W: 重み
L: 損失関数