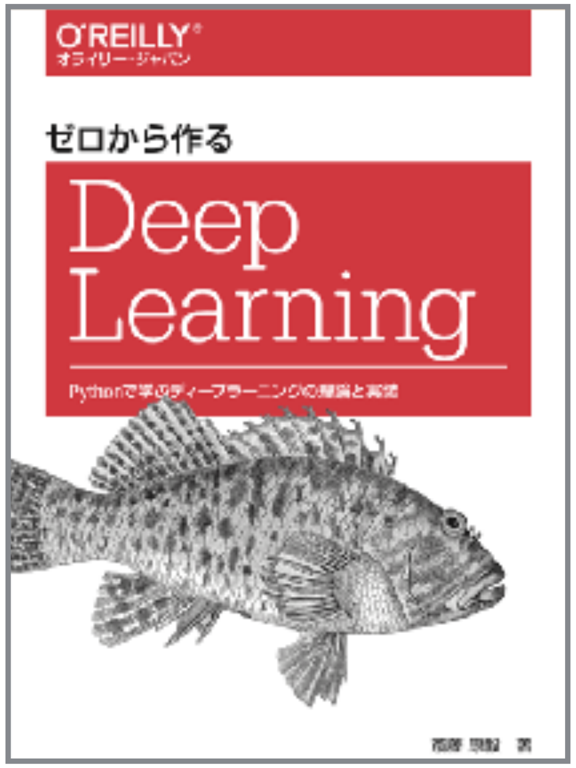
ゼロから始めるディープラーニング

~ 4章：ニューラルネットワークの学習 ~

情報理工学院 知能情報コース
Ishida Lab. M2 Tomohiro Oishi (大石 智博)

21th, May, 2018
教科書輪講

概要

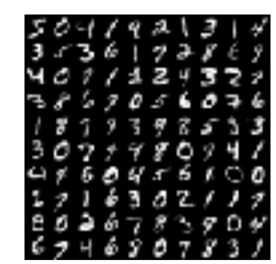


著者：斎藤 康毅

出版：オライリー・ジャパン

サポート：
https://github.com/oreilly-japan/deep-learning-from-scratch

- ニューラルネットワークの学習方法について説明
- MNISTの学習を行うニューラルネットワークを実装



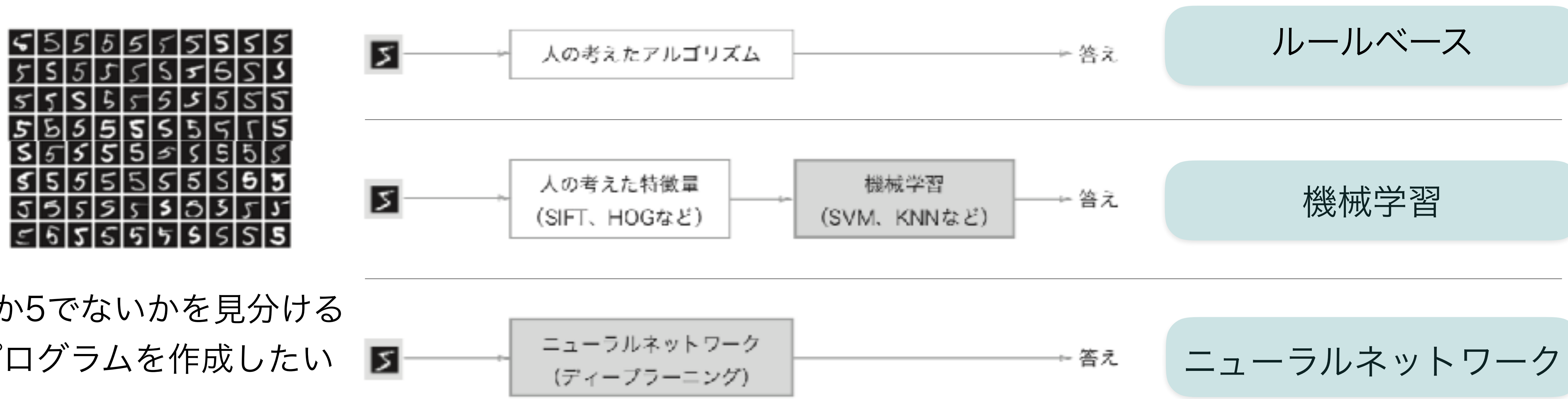
- 損失関数
- 勾配法

- NNの重みパラメータを自動最適化

目次

目次

ニューラルネットワークの特徴



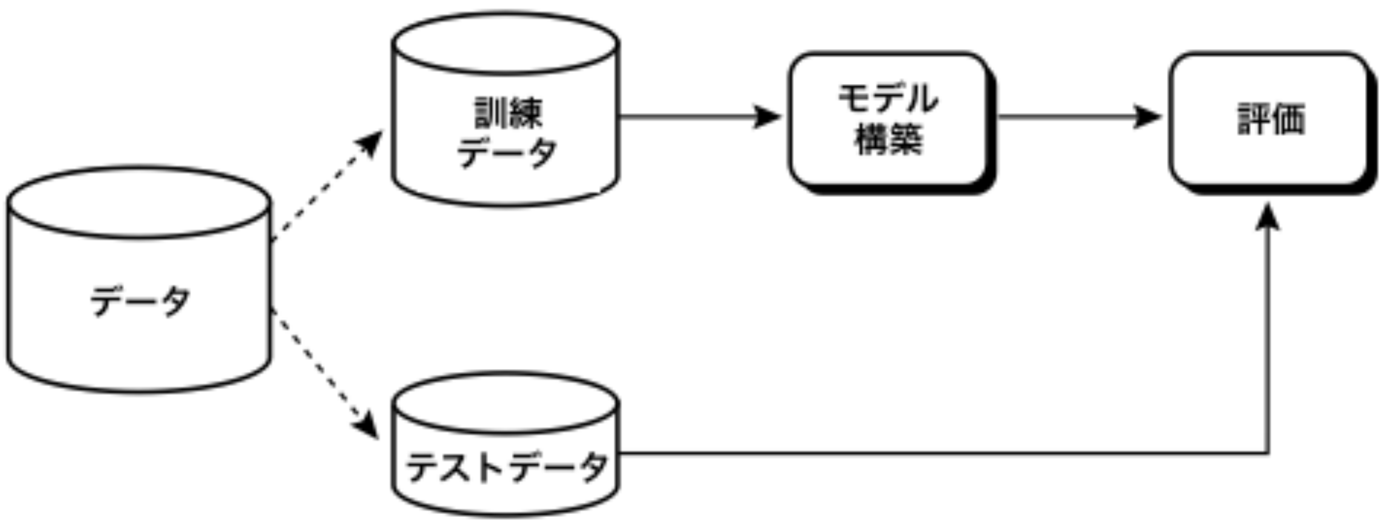
5か5でないかを見分ける
プログラムを作成したい

・ニューラルネットワークは，特徴量の設計が不要
  - 特徴量：入力データから本質的なデータを的確に抽出できるよう設計された変換器

・人の手を介在せず，生データからend to endで学習することができる
  - すべての問題を同じ流れで解くことができる

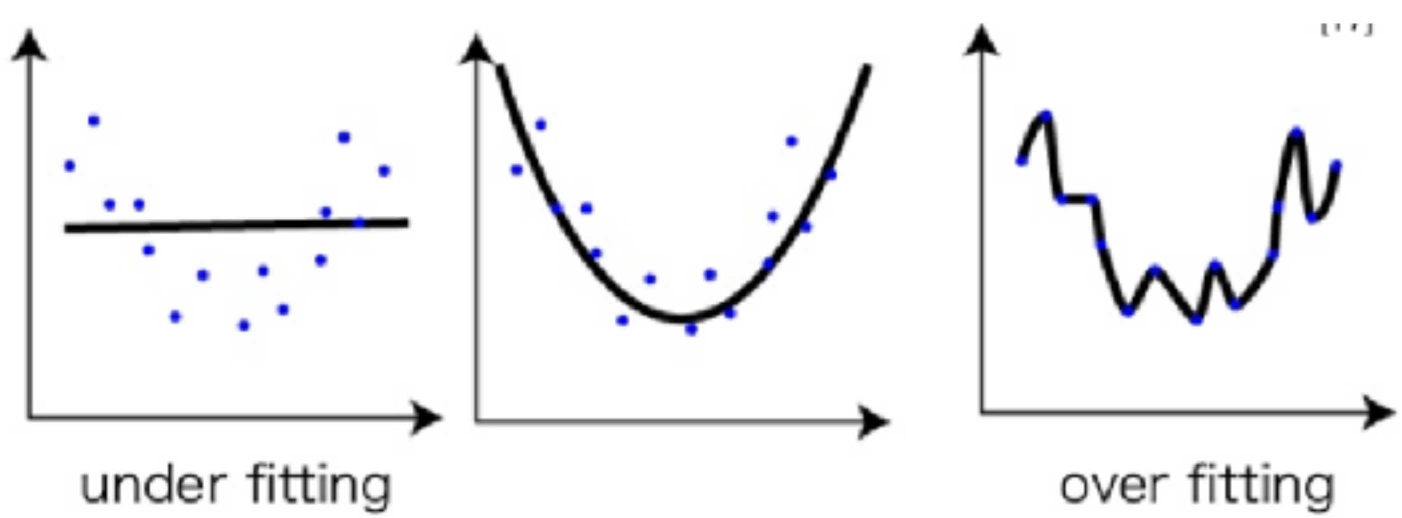
訓練データとテストデータ

- 機械学習では，データを訓練データ(教師データ)とテストデータに分割
  - モデルの汎用的な能力（汎化能力）を評価するため



訓練データ	：80%
テストデータ	：20%

- 特定のデータセットに過度に適応した状態を過学習(over fitting)という



目次

損失関数 (loss function)

NNの重みパラメータを最適化するために損失関数という「指標」を導入

損失関数：

性能の悪さを測る指標．この値を最小化する重みパラメータの探索をおこなう．

ex)

・　二乗和誤差

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

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

・　交差エントロピー誤差

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

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

二乗和誤差 (mean squared error)

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

y_k ：ニューラルネットワークの出力

t_k ：正解ラベルとなるインデックスのみ1で他は0のone-hot表現

教師データ：t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] に対して,

NNの出力 ：y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]

y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] で出力をそれぞれ比較.

※tは正解ラベルのみ1としたone-hot表現.

※yはソフトマックス関数の出力で，確率に対応する.

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

# 教師データ：「2」
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 例 1:「2」の確率が最も高い場合(0.6)
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
print(mean_squared_error(np.array(y), np.array(t)))

# 例 2:「7」の確率が最も高い場合(0.6)
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
print(mean_squared_error(np.array(y), np.array(t)))
```

```
0.0975
0.5975
```

教師データと適合してる方が,
二乗和誤差の値が小さい.

交差エントロピー誤差 (cross entropy error)

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

y_k ：ニューラルネットワークの出力

t_k ：正解ラベルとなるインデックスのみ1で他は0のone-hot表現

つまりこの式は，正解ラベルが1に対応するNNの出力の自然対数を計算するだけ

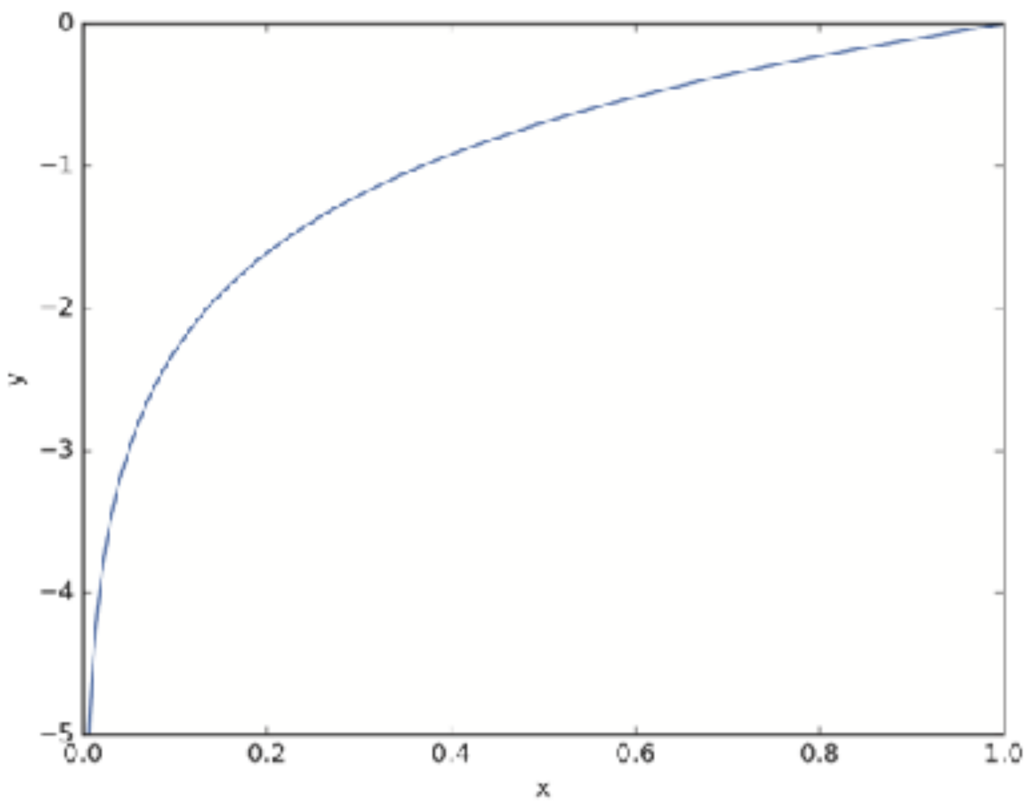


fig. y = log(x)のグラフ

ex1)
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] の時
E = -log0.6 = 0.51

ex2)
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] の時
E = -log0.1 = 2.30

```python
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta))
```

・deltaを足すことでlog0 = -infを防止

ミニバッチ学習

・先程は１つのデータの損失関数を求めていた
・訓練データ全ての損失関数の和に拡張したい

$$E = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

・Nで割ることで正規化
・平均の損失関数を求める　　※N　：データ数

この時，全ての訓練データを学習すると時間がかかる

ランダムに選ばれた一部のデータを全体の近似として学習させる（ミニバッチ学習）

MNISTの全60,000件の訓練データから10件をバッチとして取得してみよう！

交差エントロピー誤差の実装

- 正解ラベルに対してニューラルネットワークの出力が得られれば, 交差エントロピー誤差を計算できる.

- 教師データtがone-hot表現の時 （例）t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

In [83]:
```
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
```

- 教師データtがラベルとして与えられた時 （例）t = 2

In [84]:
```
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(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

なぜ損失関数を設定するのか

Question

「指標」に認識精度を用いればいいのでは？？なぜ損失関数？？

Answer

- 学習は損失関数が小さくなるように重みパラメータを更新する作業
- 損失関数を小さくするために損失関数の傾き，つまり微分を使う
- 認識精度を指標にすると殆どの場所で微分値が0になりパラメータの更新ができない
- 損失関数なら微分値を求めることができるため，損失関数を用いる

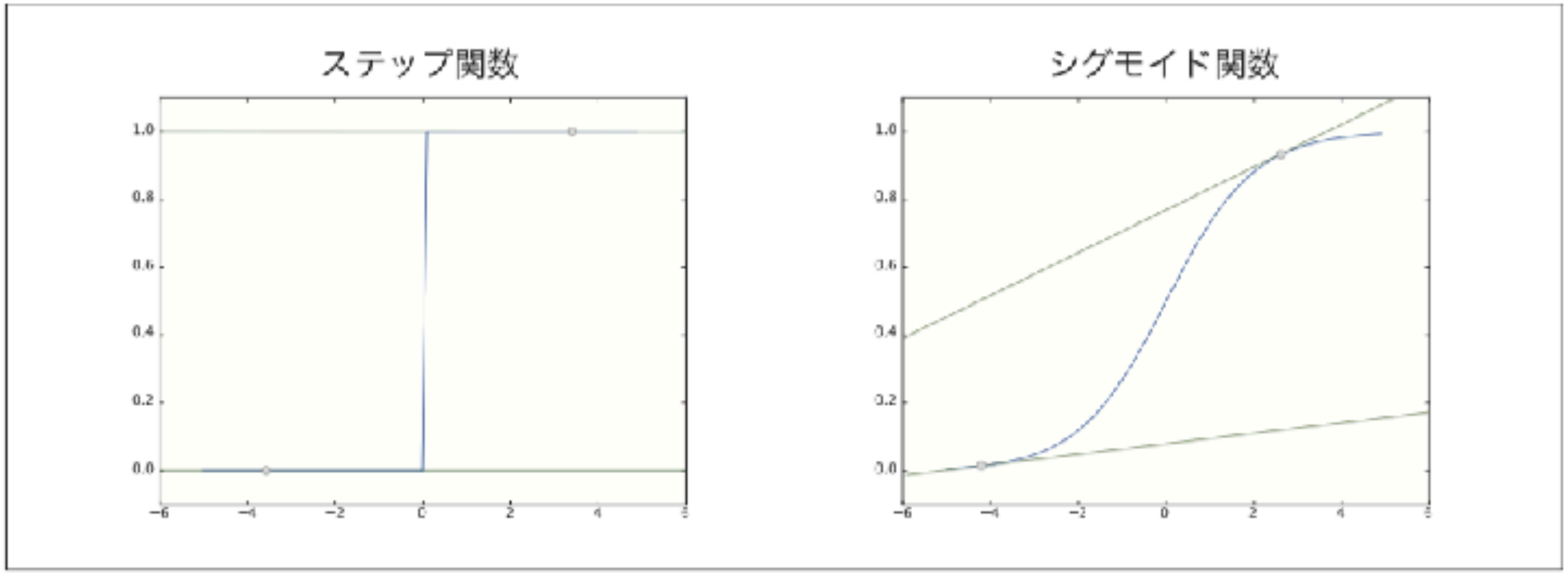


図 4-4　ステップ関数とシグモイド関数：ステップ関数はほとんどの場所で傾きは 0 であるのに対して、シグモイド関数の傾き（接線）は 0 にならない

目次

数値微分

微分とは「ある瞬間」の変化の量をあらわしたもの

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

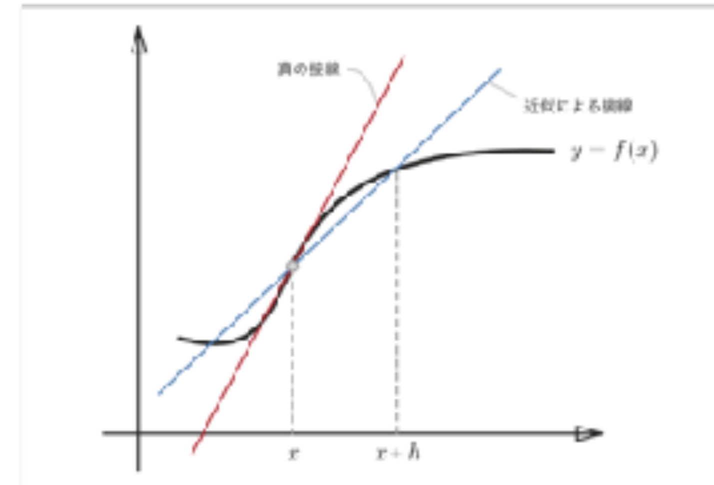


図4-5 真の微分（真の接線）と数値微分（近似による接線）の値は異なる

プログラムで実装してみる

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

▶

- 丸め誤差が問題
  hが0.0に丸め込まれてしまう
- 近似解が問題

改善案

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

- (h = 1e-4) を用いた
- 中心差分を用いた
  (x + h) と (x - h) の差分

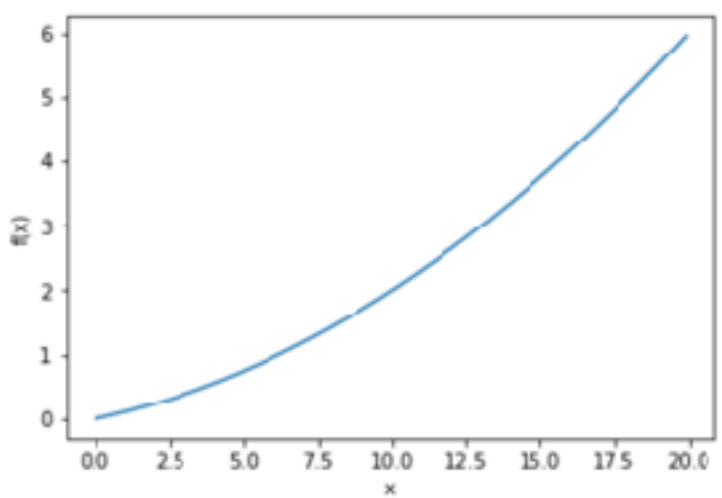
ここで行っているように、微小な差分によって微分を求めることを数値微分(numerical differentiation)と言う

exercise) 数値微分で簡単な関数を微分

$$y = 0.01x^2 + 0.1x \quad \blacktriangleright \quad \frac{df(x)}{dx} = 0.02x + 0.1$$

In [97]:
```
def function_1(x):
        return 0.01*x**2 + 0.1*x
```

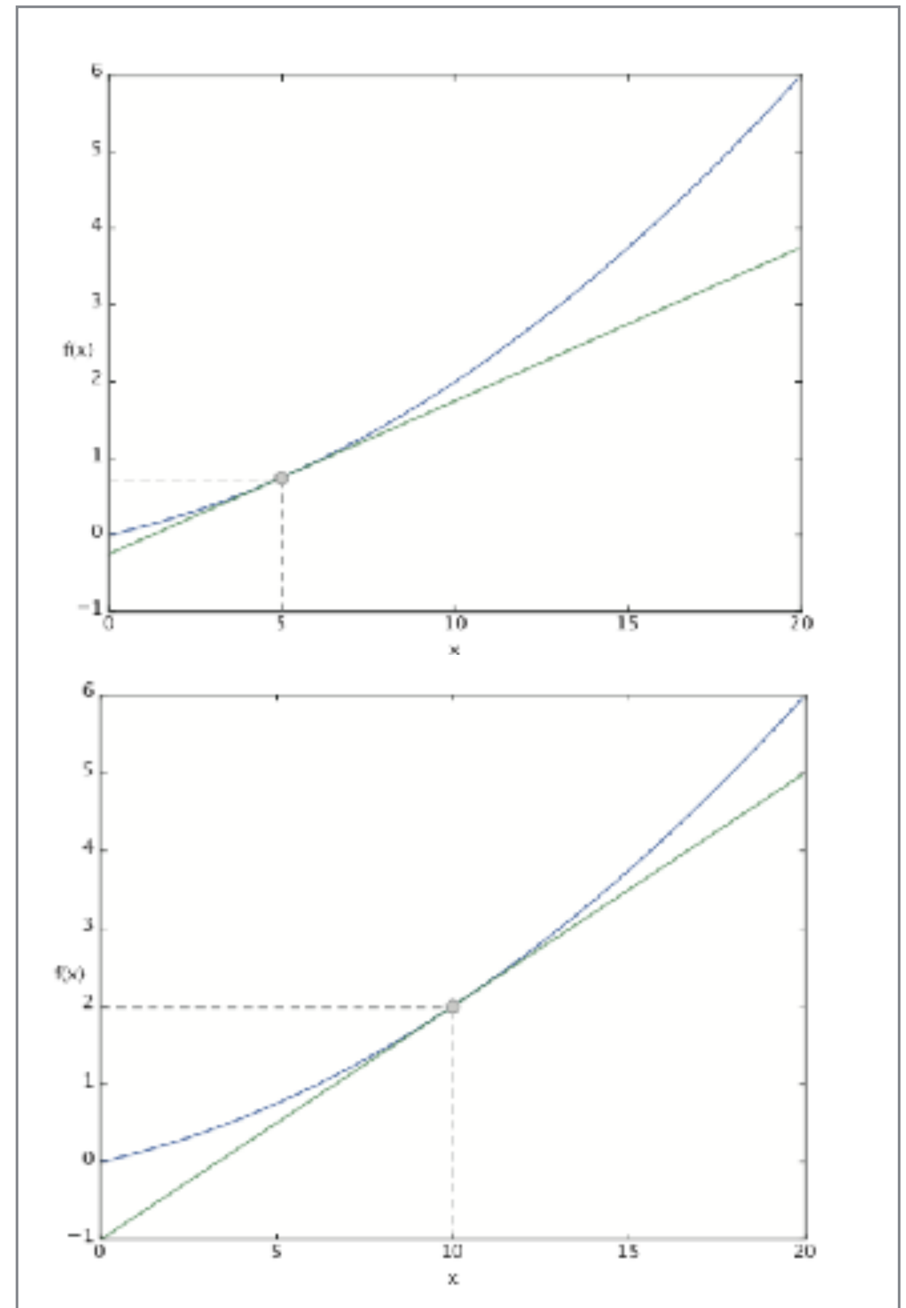
In [99]:
```
import numpy as np
import matplotlib.pylab as plt
x = np.arange(0.0, 20.0, 0.1) # 0 から 20 まで、0.1 刻みの x 配列
y = function_1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y)
plt.show()
```



In [104]:
```
numerical_diff(function_1, 5)
```
Out[104]: 0.1999999999990898

In [105]:
```
numerical_diff(function_1, 10)
```
Out[105]: 0.2999999999986347

数値微分で求めた x = 5, x = 10 での接線



偏微分

偏微分：
複数の変数からなる関数の微分のこと

次の関数を考える

$$f(x_0, x_1) = x_0^2 + x_1^2$$

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

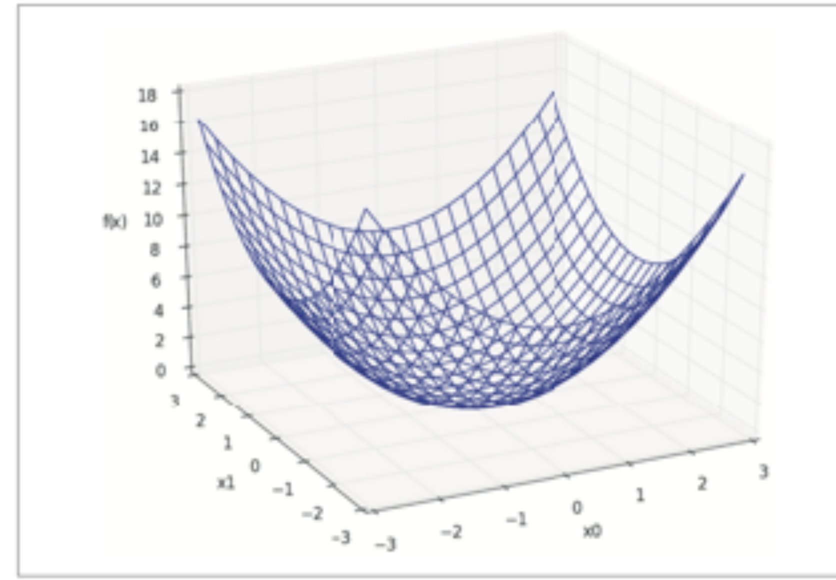


図4-8 $f(x_0, x_1) = x_0^2 + x_1^2$ のグラフ

(exercise)

問 1：$x_0 = 3$、$x_1 = 4$ のときの x_0 に対する偏微分 $\frac{\partial f}{\partial x_0}$ を求めよ。

```
>>> def function_tmp1(x0):
...     return x0*x0 + 4.0**2.0
...
>>> numerical_diff(function_tmp1, 3.0)
6.00000000000378
```

問 2：$x_0 = 3$、$x_1 = 4$ のときの x_1 に対する偏微分 $\frac{\partial f}{\partial x_1}$ を求めよ。

```
>>> def function_tmp2(x1):
...     return 3.0**2.0 + x1*x1
...
>>> numerical_diff(function_tmp2, 4.0)
7.999999999999119
```

(Point)
複数ある変数の中でターゲットの変数を
ひとつに絞り、他の変数はある値に固定