

# 機械学習入門

---

その2

福岡工業大学短期大学部  
情報メディア学科  
東 昭太朗



# 今回の要点

- 機械学習において欠かせない数学の基礎(続き)
  - 微分、偏微分、非線形関数etc...
- 教師あり学習のアプローチ
  - 活性化関数、解析解と数値解、統計学的要素
  - 回帰モデル
  - 分類モデル

# 曲面で表す

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

xn = 10
wn = 10

x = np.linspace(-5, 5, xn)
w = np.linspace(-5, 5, wn)

def f_2(x, w):
    return (x-w) * x * (x-2)


y = np.zeros((xn, wn))

for i0 in range(xn):
    for i1 in range(wn):
        y[i0, i1] = f_2(x[i0], w[i1])

#plt.plot(x, y)
from mpl_toolkits.mplot3d import Axes3D

Axes3D.plot_surface(x, w, y)

plt.show()
```



```
$ python plot3dsur.py
Traceback (most recent call last):
  File "plot3dsur.py", line 26, in <module>
    Axes3D.plot_surface(x, w, y)
TypeError: plot_surface() missing 1 required
positional argument: 'Z'
```

きちんと引数を3つ与えているのに  
エラーが返される



[Stackoverflowのとあるスレッド](#)にて同じ現象が発生している人を発見

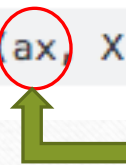
My problem is that even though `Z` is 2D array, `.plot_surface()` refuses to recognize it as such.

As you said, it seems that the right way to do this is via axis object ( `ax` in the code below):

```
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z)
```

Alternatively, you can still call it using `Axes3D` , but then you need to pass it the axis object `ax` , like so:

```
Axes3D.plot_surface(ax, X, Y, Z)
```



**axis**オブジェクトを生成しておく  
必要があるらしい

今まで触れていなかったmatplotlibの描画構造について調べた



```
import matplotlib.pyplot as plt  
plt.figure()  
plt.show()
```

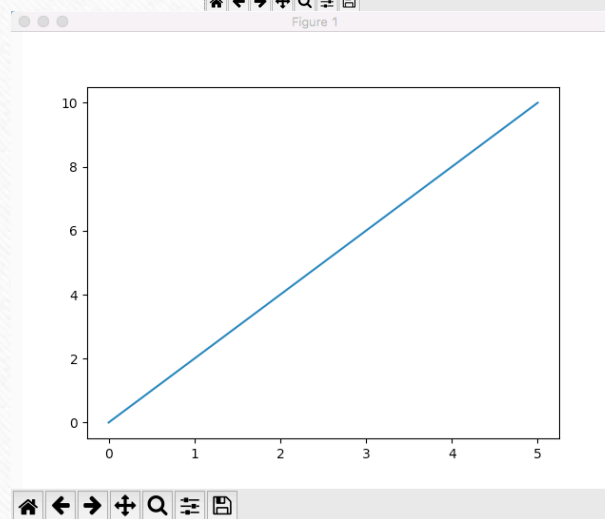
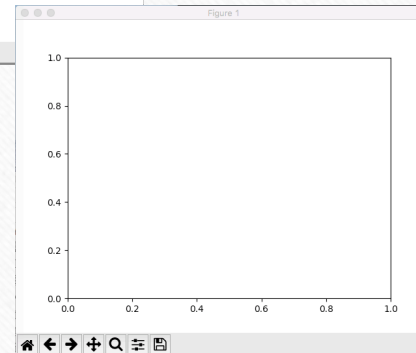
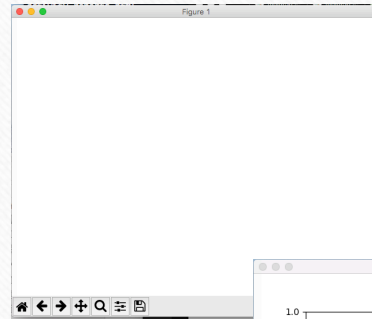
```
plt.figure()
```

```
plt.subplot(1,1,1)
```

```
plt.show()
```

ここでaxesオブジェクト  
が返される

```
import numpy as np  
import matplotlib.pyplot as plt  
  
x = np.linspace(0, 5)  
  
plt.figure()  
plt.subplot(1,1,1)  
  
plt.plot(x, 2*x)  
  
plt.show()
```



描画領域を生成



座標領域を生成



グラフを生成

## 3Dプロットに再挑戦



# 曲面で表す

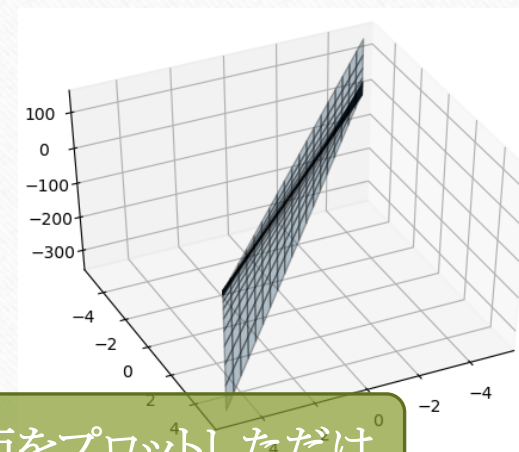
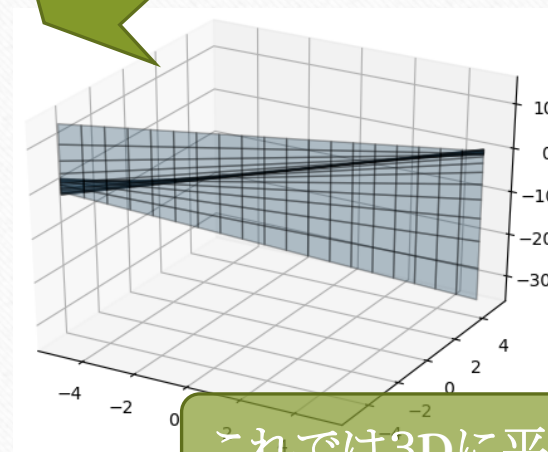
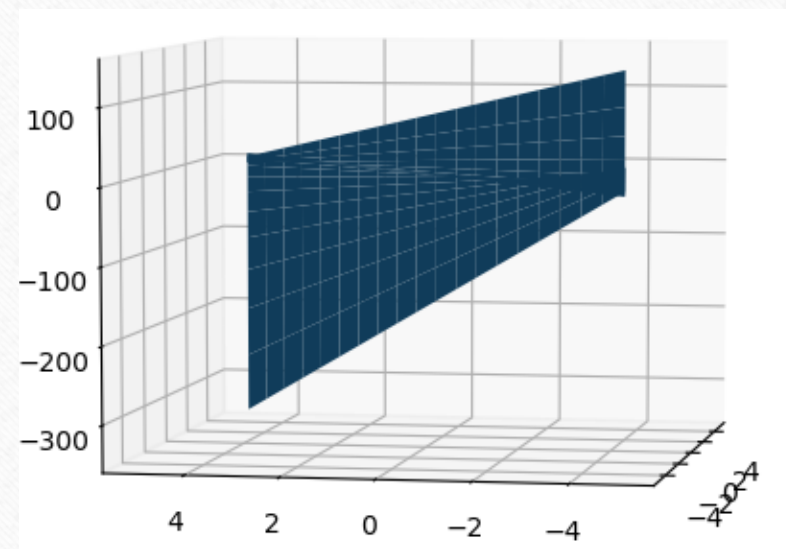
```
#plt.plot(x, y)
from mpl_toolkits.mplot3d import Axes3D

plt.figure()
ax = plt.subplot(1, 1, 1, projection='3d')
Axes3D.plot_surface(ax, x, w, y)
# ax.plot_surface(x, w, y) でも良い

plt.show()
```

```
plt.figure()
ax = plt.subplot(1, 1, 1, projection='3d')
Axes3D.plot_surface(ax, x, w, y, alpha=0.3,
edgecolor='black')

plt.show()
```



これでは3Dに平面をプロットしただけ

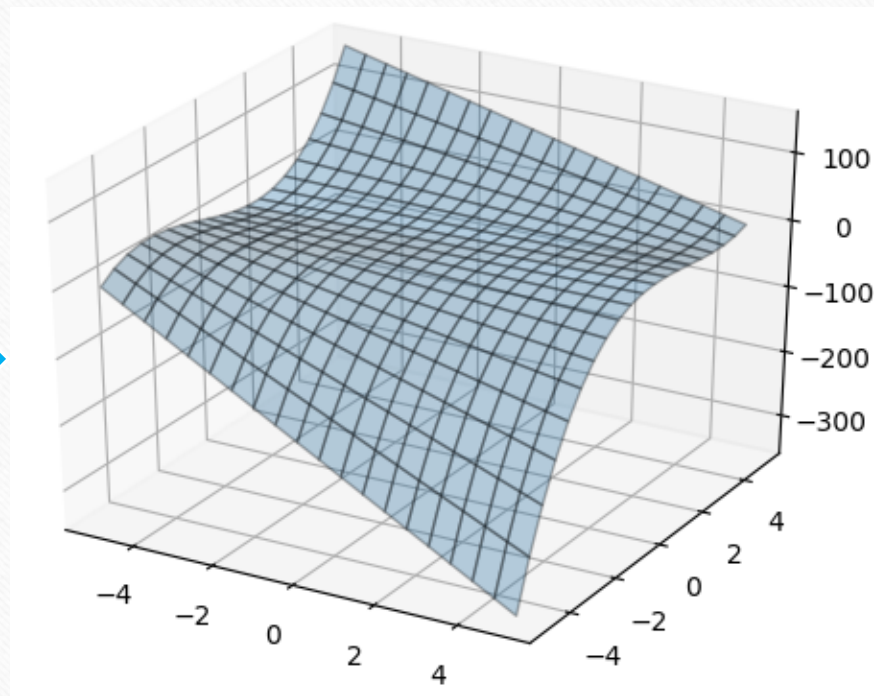
余計わかりづらくなっているようだ...



## 曲面で表す

これでは3Dに平面をプロットしただけ

```
#plt.plot(x, y)
from mpl_toolkits.mplot3d import Axes3D
xx, ww = np.meshgrid(x, w)
ax = plt.subplot(1, 1, 1, projection='3d')
Axes3D.plot_surface(ax, xx, ww, y, alpha=0.3,
                    edgecolor='black')
plt.show()
```



成功！！

## 機械学習に必要な数学の知識 ～基礎編～



# 何を使うのか？

- 微分
  - 常微分
  - 偏微分
- 線形代数
  - ベクトル
  - 行列
- 指数・対数
  - ネイピア数
- 活性化関数として利用される関数
  - シグモイド関数
  - ソフトマックス関数
  - ガウス関数

## 線形と非線形

- 簡単に言えば線形→直線、非線形→曲線
- 厳密には直線でも線形に分類できない場合もあるし、曲線で線形に分類できる場合もある



# 線形と非線形

## 線形

- $x$ の変化量に対する $y$ の変化量(つまり、変化の割合)が一定な関数？

- 定義:  $f(x + y) = f(x) + f(y)$

$$f(kx) = kf(x)$$

を満たす関数

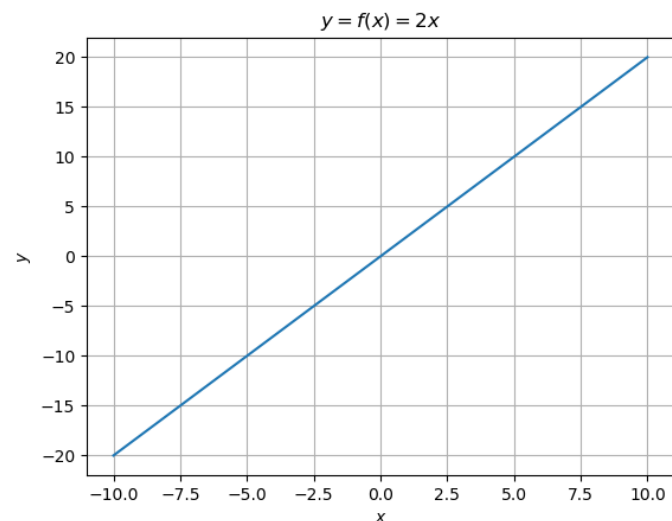
## 非線形

- 線形の定義に当てはまらないもの全て

# 線形の例

$$f(x) = 2x$$

$$\begin{aligned} f(x+y) &= 2(x+y) \\ &= 2x + 2y \\ &= f(x) + f(y) \end{aligned} \quad \begin{aligned} f(kx) &= 2(kx) \\ &= k(2x) \\ &= kf(x) \end{aligned}$$



他にも...

- ベクトルの内積
- シグマ
- 微分・積分

などが線形に分類できる例としてある



線形ではないという事は...

- $x$ の変化量に対する $y$ の変化量が一定でない(2次以上の)関数
  - ゆえに、変化の割合を求めたい需要が現れるのは必然



微分の出番

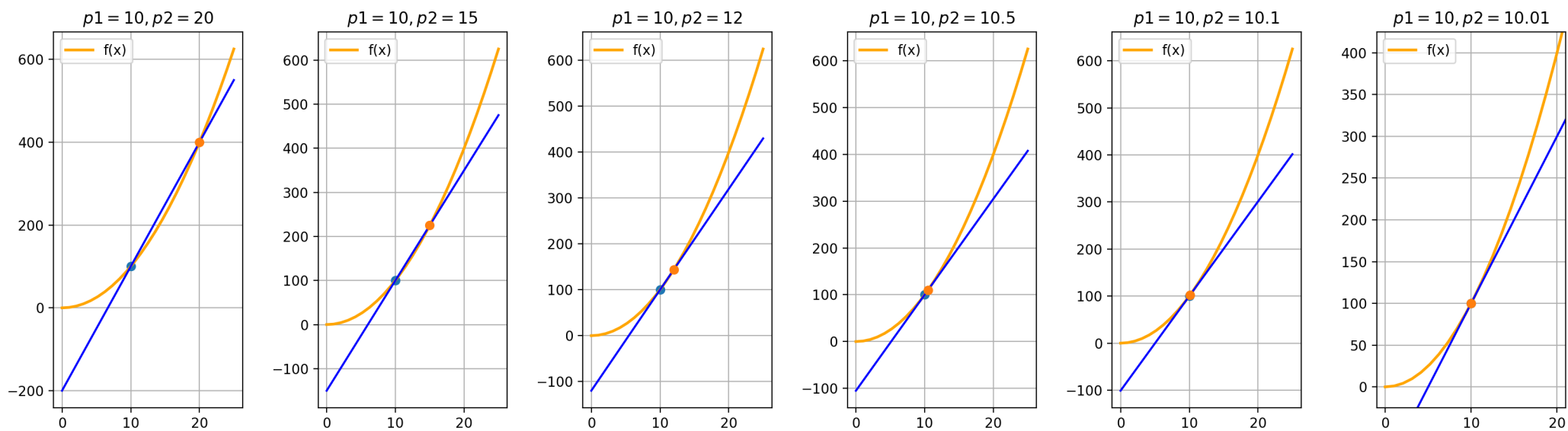
## 微分とは？

- 変化前と変化後の2点を取り、その変化の割合( $x$ の変化量/ $y$ の変化量)を求めることに着想を得ている
- その2点間を**限りなく近づけた**時の値を求めることを**微分する**という
- 任意の点 $x$ における微分係数を求める関数が導関数となる

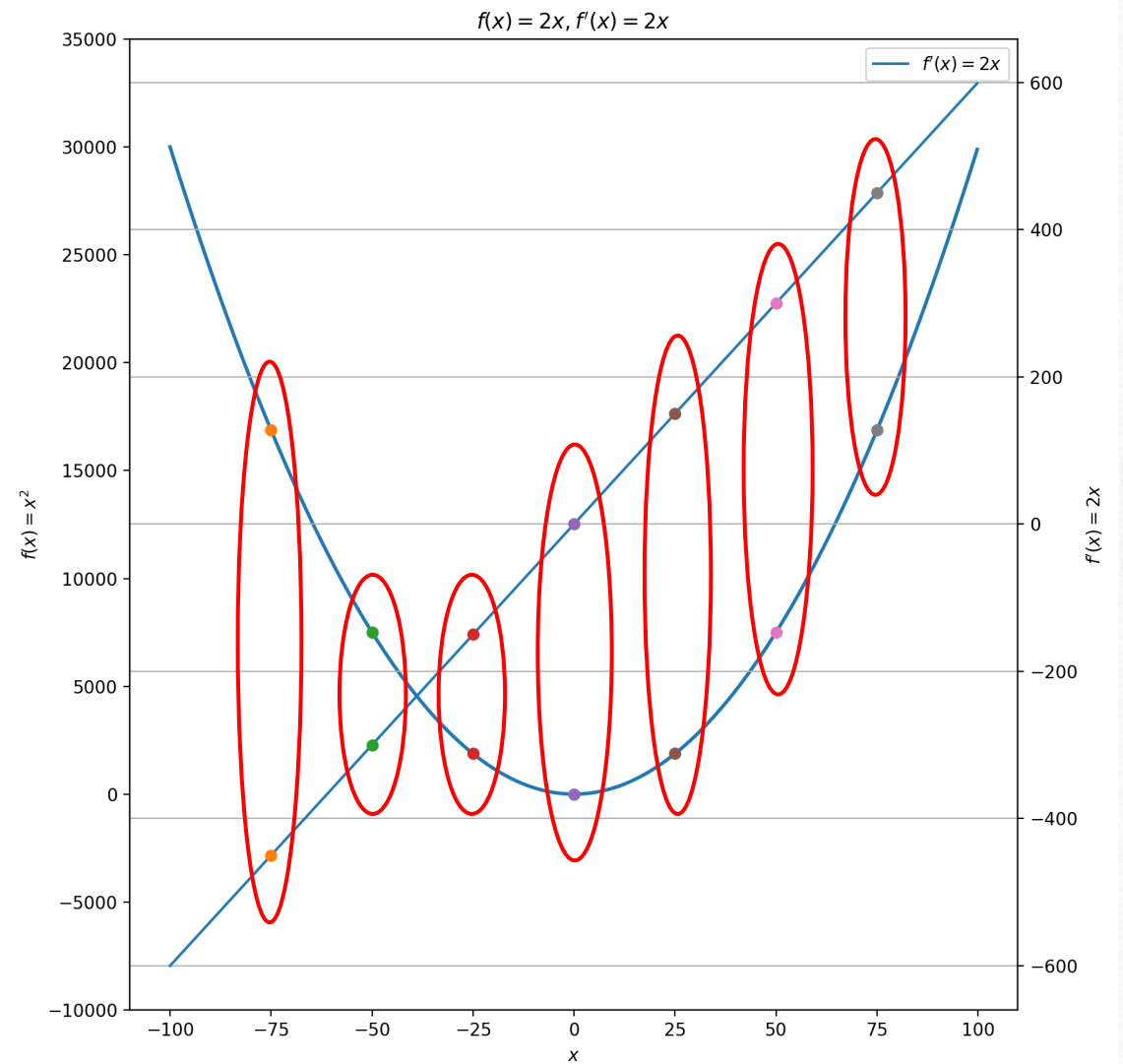
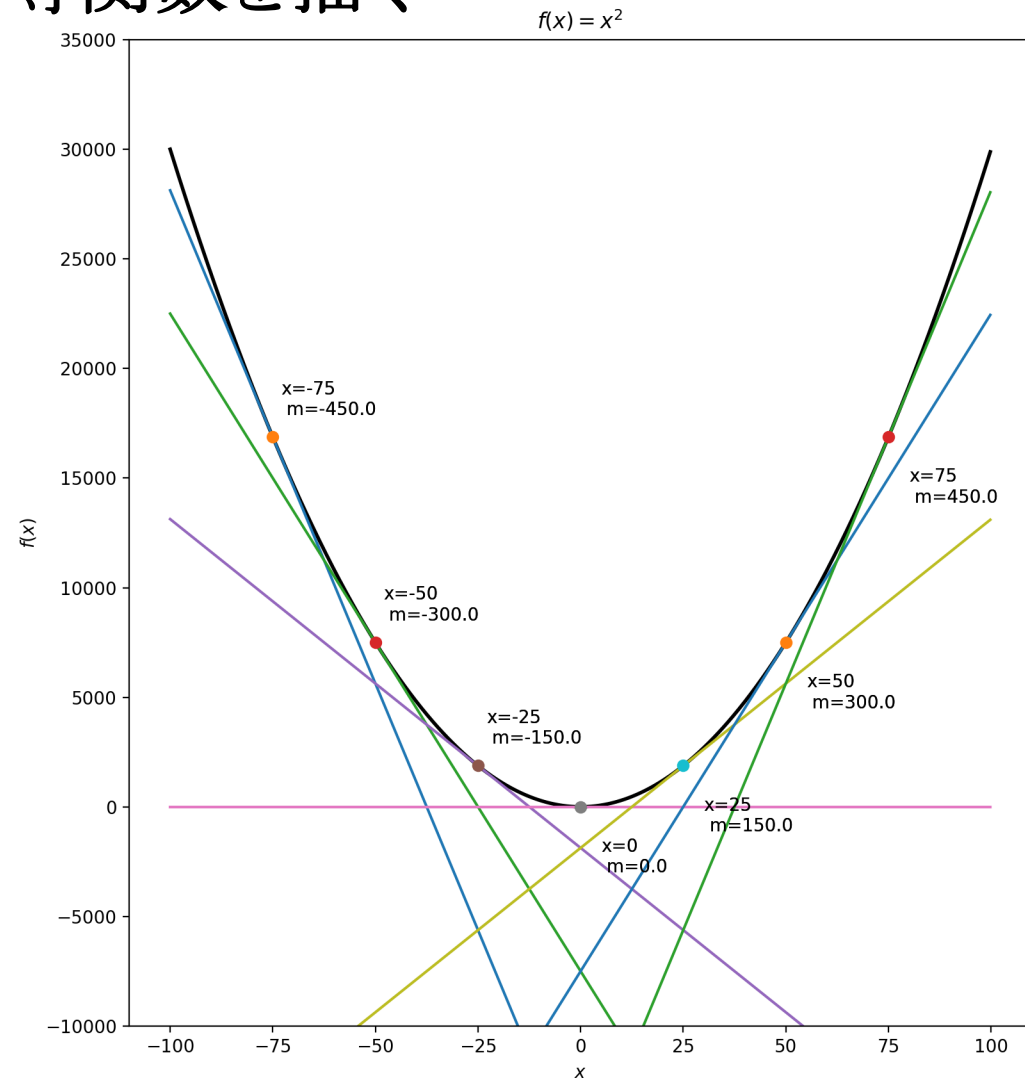


2点間の距離(差)を限りなく0に近づけていく = 極限

$$\lim_{h \rightarrow 0}$$



# 導関数を描く





# 活性化関数

- 出力された結果が、データの利用目的において扱いやすい値となるよう調整するために導入されるもの

NNでは最後の出力層にのみ組み込まれる

# シグモイド関数

- 出力された結果が、データの利用目的において扱いやすい値となるよう調整するために導入されるもの
- 一定の区間

NNでは最後の出力層にのみ組み込まれる



# ソフトマックス関数

- 出力された結果が、データの利用目的において扱いやすい値となるよう調整するために導入されるもの

NNでは最後の出力層にのみ組み込まれる

# ガウス関数

- 統計学の世界では正規分布を表現するものとして有名である
- グラフの中央が山になっているのが特徴

NNでは最後の出力層にのみ組み込まれる



## 教師あり学習のアプローチ

# 教師あり学習

- 人間が学校などで誰から教わるスタイルをそのまま再現したメソッド
- 具体的には以下の2ステップに分かれる
  1. 教師データをもとに**モデル**を生成する段階
  2. 生成したモデルを利用して予測値を導出する段階

教師データ: データと正解値のセットのこと



1次元

直線モデル

2次元

面モデル

多次元

D次元線形回帰モデル

多次元

線形基底関数モデル

非線形のものを線形の世界に持ち込む方法の一つ

# 直線モデル

- なんの変哲もない一次関数のモデル
- 要素(変数)が1つの時はこれが使える



# 面モデル

- 要素(変数)が2つの時はこれを使う

# D次元線形回帰モデル

- なんの変哲もない一次関数のモデル
- 要素(変数)が1つの時はこれが使える



# 線形基底関数モデル

- 入力データをそのまま計算に利用するのではなく、1つのデータを複数の(基底)函数を用いて導出することで、式を線形の形に保ったまま非線形函数を表現することができる
- 活性化関数が出力データを整えるものであるなら、基底関数は入力データを一定の規則に整えるものであると言える
- 基底関数は人間があらかじめ設定する必要がある  
→何を扱うかは数学的なセンスと腕の見せ所

# リファレンス

- あたらしい機械学習の教科書
- プログラミング言語Pythonの紹介
- NumPy と matplotlib を利用した可視化の方法