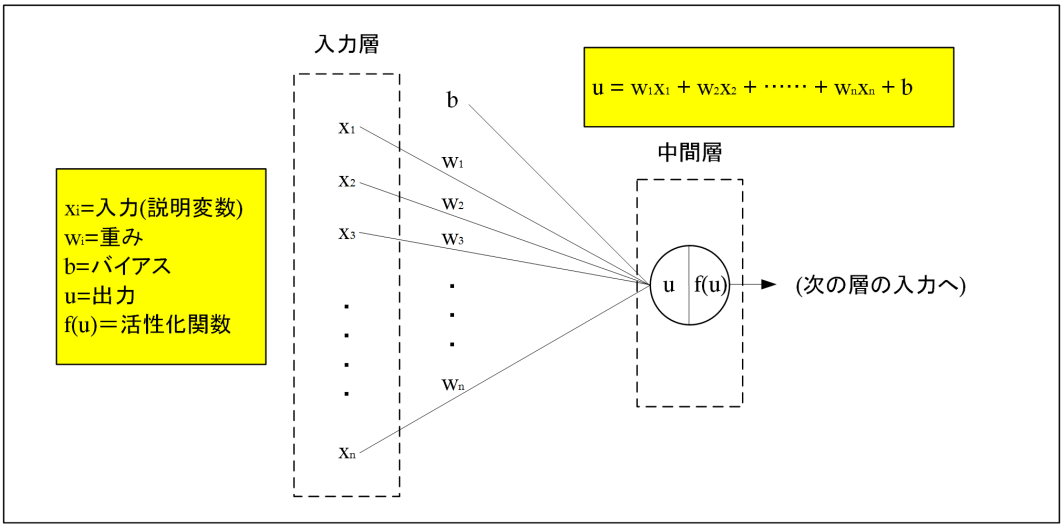
1　入力層～中間層

・入力層：説明変数を入力する層
・中間層：入力層から入力されたデータを線形結合する層
　(さらに活性化関数の出力を伴う (場合がほとんど (恒等写像を省く)))
・入力層と中間層の結合にて、重み (**w**) が積算、バイアス (*b*) が加算、される
・要するに以下図の様な結合となる
　(無論、実際の DNN はこれら構成を何層も結合した構成 (出力を次の入力へ結合する))



・コード (線形結合)

```
#入力層～中間層(n=10)
x = np.array([ 1, 2,  3,  4,  5,  6,  7,  8,  9, 10])
w = np.array([-1, 1, -1,  1, -1,  1, -1,  1,  0,  1])
b = 1
u = np.dot(w.T,x) + b

print(u)  #15
```

✓ 0.6s

```
15
```

2 活性化関数

・ニューラルネットワークにおいて、次の層への出力の大きさを決める非線形関数
・入力値の値によって、次の層への信号の ON/OFF や強弱を定める働きを持つ
・中間層の出力、最終段の出力、それぞれに適した活性化関数が利用される
・後述の誤差逆伝搬法における勾配消失問題の対策 (例：中間層における ReLU 関数)
・中間層の活性化関数は現在ほぼ ReLU 関数が利用される
・出力層の活性化関数は用途に応じて選択する

回帰：　　恒等写像
二値分類：　シグモイド関数
多クラス分類：ソフトマックス関数

・ステップ関数

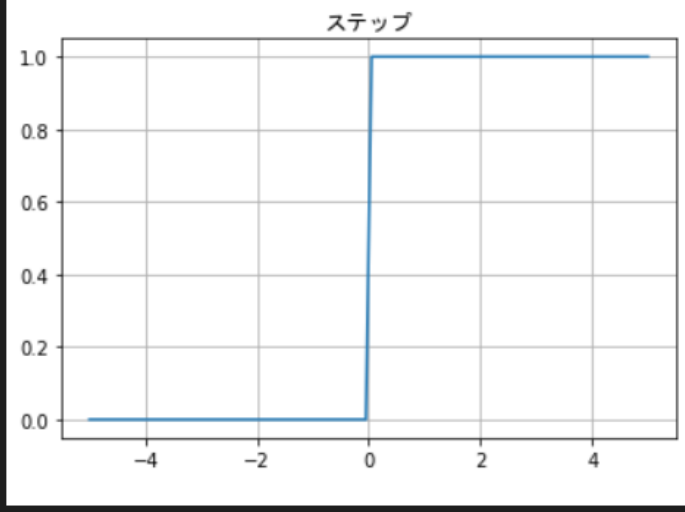
```
#活性化関数(ステップ関数)
def stepFunc(x):
    ret = np.zeros(len(x))
    for i in range(len(x)): ret[i] = 1 if x[i] >= 0 else 0
    return ret

x = np.linspace(-5,5,100)

# pltに要素を設定する
plt.title(label='ステップ', fontname="MS Gothic")
plt.grid()
plt.plot(x, stepFunc(x))
```

[103] ✓ 0.2s

```
[<matplotlib.lines.Line2D at 0x1721db71b20>]
```



閾値(0)を境に発火する、出力は0または1のみ

この場合線形分離可能なものしか学習できない

・シグモイド関数

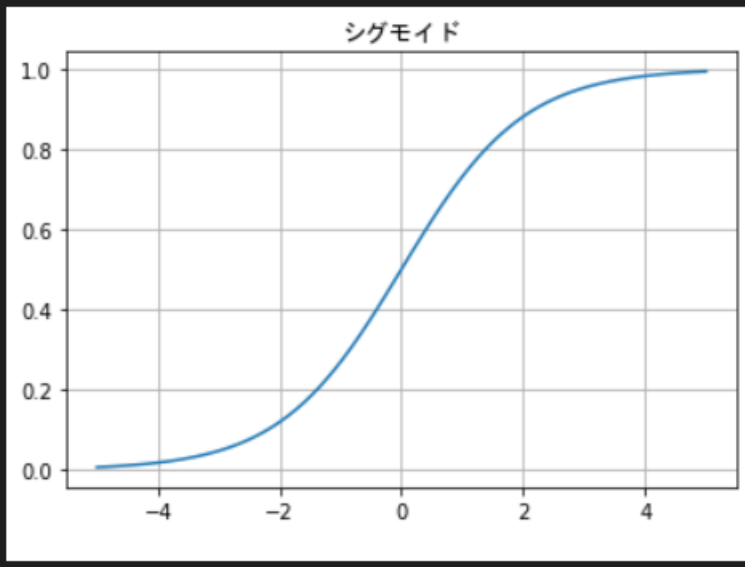
```
#活性化関数(シグモイド関数)
def sigmoidFunc(x):
    return 1/(1 + np.exp(-1 * x))

x = np.linspace(-5,5,100)

# pltに要素を設定する
plt.title(label='シグモイド', fontname="MS Gothic")
plt.grid()
plt.plot(x, sigmoidFunc(x))
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x2a27e8d5760>]
```



0～1の状態を表現できる、ニューラルネットワーク普及のきっかけとなった

ただし1付近の出力変化が少なく、勾配消失問題を引き起こすことがある

二値分類の出力層に使われる(そのまま確率として見える)

・ReLU 関数

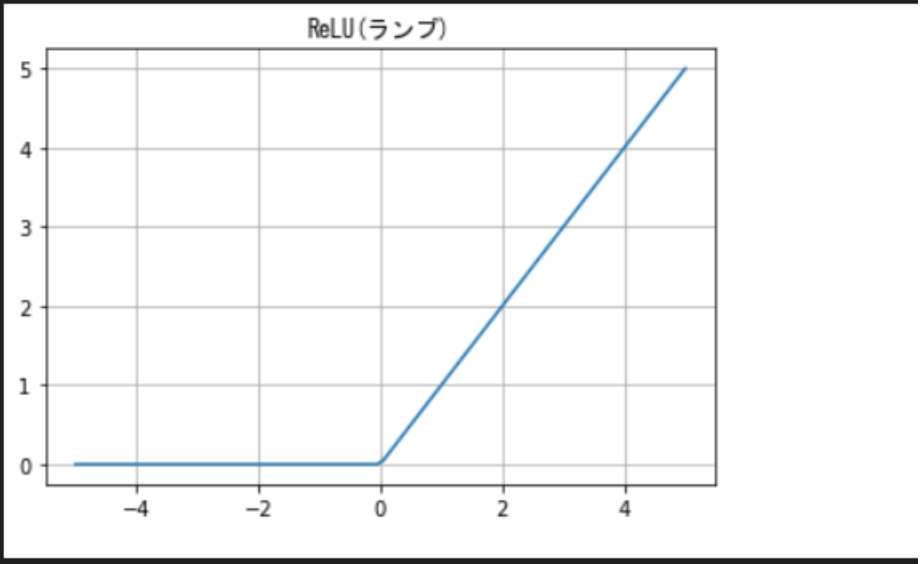
```
#活性化関数(ReLU(ランプ)関数)
def reluFunc(x):
    return np.maximum(0, x)

x = np.linspace(-5,5,100)

# pltに要素を設定する
plt.title(label='ReLU(ランプ)', fontname="MS Gothic")
plt.grid()
plt.plot(x, reluFunc(x))
```

✓ 0.2s

```
[<matplotlib.lines.Line2D at 0x1721dc2c310>]
```



現在最も利用される活性化関数とのこと

勾配消失問題の回避、スパース化(0以下の)に貢献

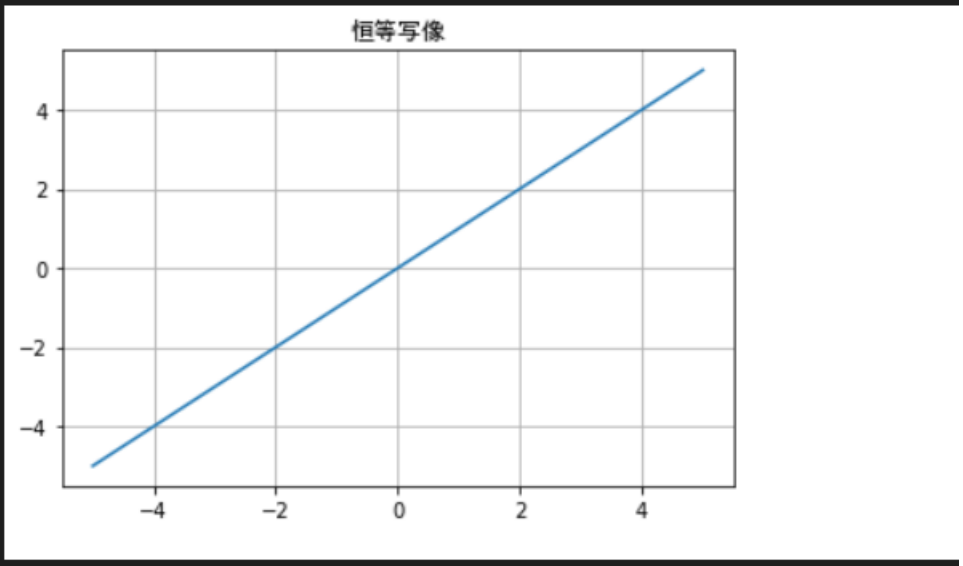
・恒等写像

```
#活性化関数(恒等写像)
def reluFunc(x):
    return x

x = np.linspace(-5,5,100)

# pltに要素を設定する
plt.title(label='恒等写像', fontname="MS Gothic")
plt.grid()
plt.plot(x, reluFunc(x))
```

```
[<matplotlib.lines.Line2D at 0x1721dc8a670>]
```



入力がそのまま出力される

・ソフトマックス

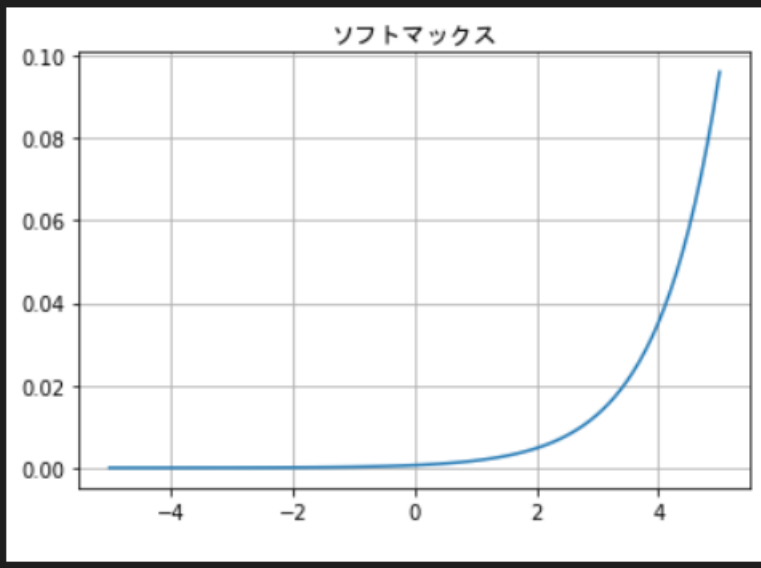
```python
#活性化関数(ソフトマックス)
def softmaxFunc(x):
    u = np.sum(np.exp(x))
    return np.exp(x)/u

x = np.linspace(-5,5,100)

# pltに要素を設定する
plt.title(label='ソフトマックス', fontname="MS Gothic")
plt.grid()
plt.plot(x, softmaxFunc(x))
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x2a27e9eec10>]
```



多値分類モデルの出力層に使われる(全出力の合計が100%を示すので適合確率がわかる)

3　出力層

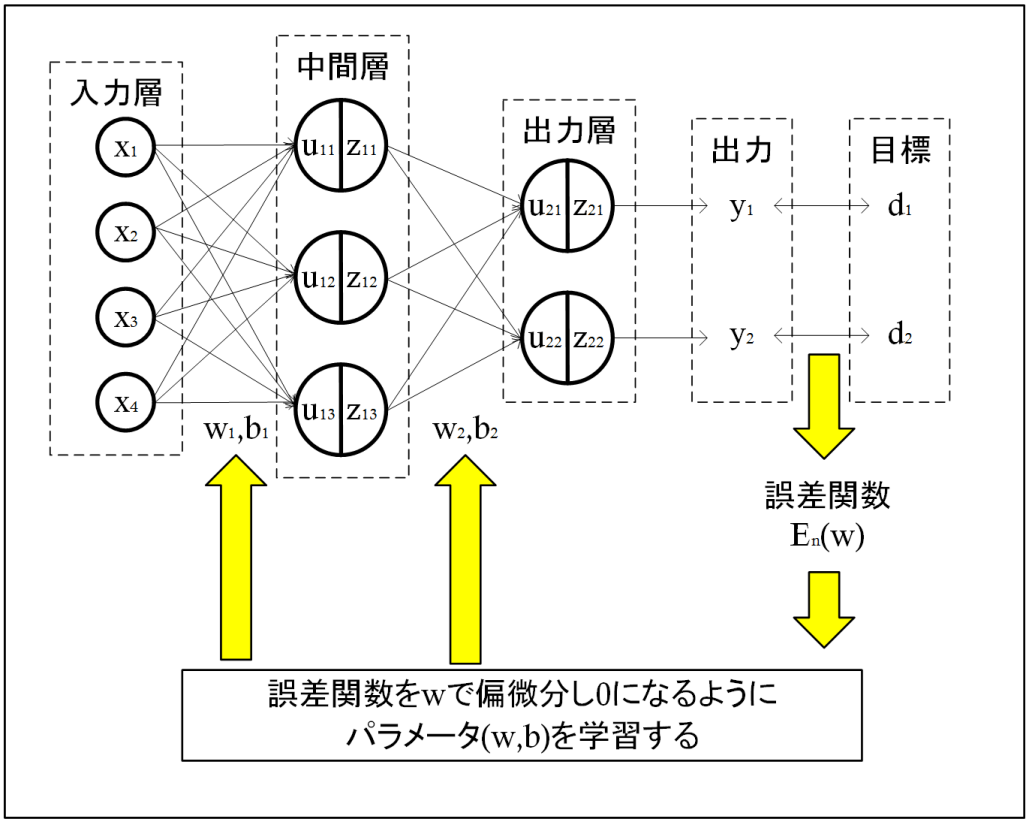
・分類問題の場合は確率を出力するとともに学習の為の誤差を得る必要がある (誤差関数を定義)
・回帰問題の場合は出力予測値とともに学習の為の誤差を得る必要がある (誤差関数を定義)

回帰誤差関数：　　二乗誤差

$$E_n(w) = \frac{1}{2}\sum_{i=1}^{l}(y_i - d_i)^2$$

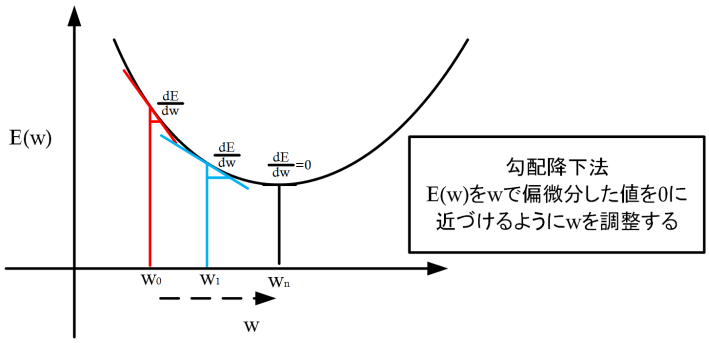
分類誤差関数：　　交差エントロピー

$$E_n(w) = -\sum_{i=1}^{l} d_i \log y_i$$



4　勾配降下法

・誤差関数 $E(w)$ を最小化するパラメータ w を探す方法として勾配降下法を利用する



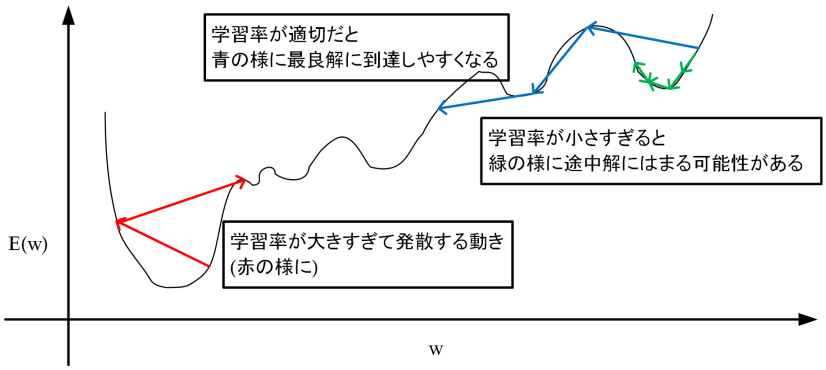
・以下の様に誤差関数を w で偏微分し、w を補正 (=学習) し、更新する

$$w^{(t+1)} = w^{(t)} - \epsilon \nabla E$$

(ϵ は学習率 (ハイパーパラメータ))

・学習率を小さくしすぎると収束しにくくなる (逆に大きすぎると学習が発散する)

・DNN のモデルは多次元式になるので極値 (偏微分値が 0 になる点 (底)) が複数存在する



・とは言え、単純に学習率を選定するだけでは学習の収束は難しい

・確率的勾配降下法 (SGD)：学習データをランダムに抽出して誤差を得る

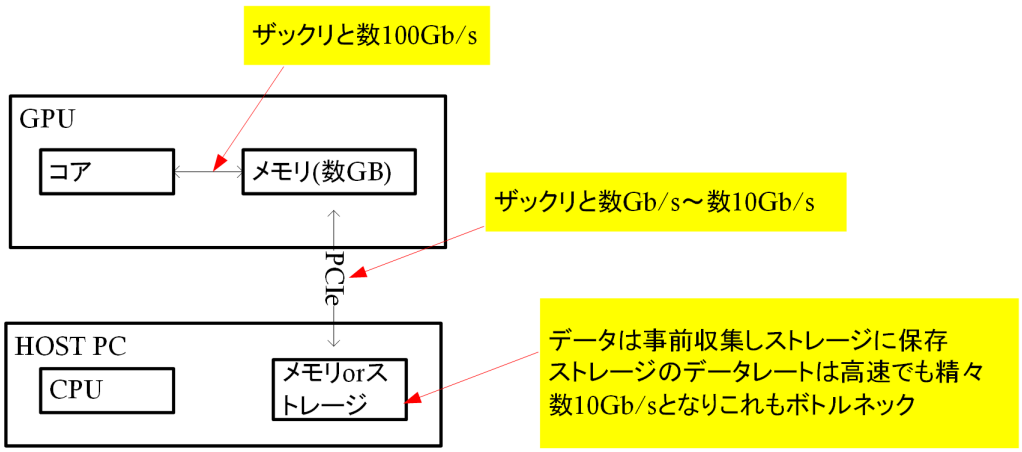
・ミニバッチ勾配降下法：学習データをランダムに分割した集合 (ミニバッチ) を切り替え学習

・いずれもいわゆるディザの様な効果と理解する

・また、勾配降下法の学習率決定アルゴリズムが幾つか紹介される (day2 にて)

・ミニバッチ勾配降下法は確率的勾配降下法の思想を取り入れている
・確率的勾配降下法はオンライン学習でもある (その時に出現したデータを学習)
・ミニバッチ勾配降下法は「準備済データをランダムに分割」である

・ミニバッチ勾配降下法はメモリ節約の手法でもある
・DNN の計算は並列計算可能 →GUP の様な並列計算に特化した計算機を利用
・GPU へデータを送るための I/F は一般的に PCIe である

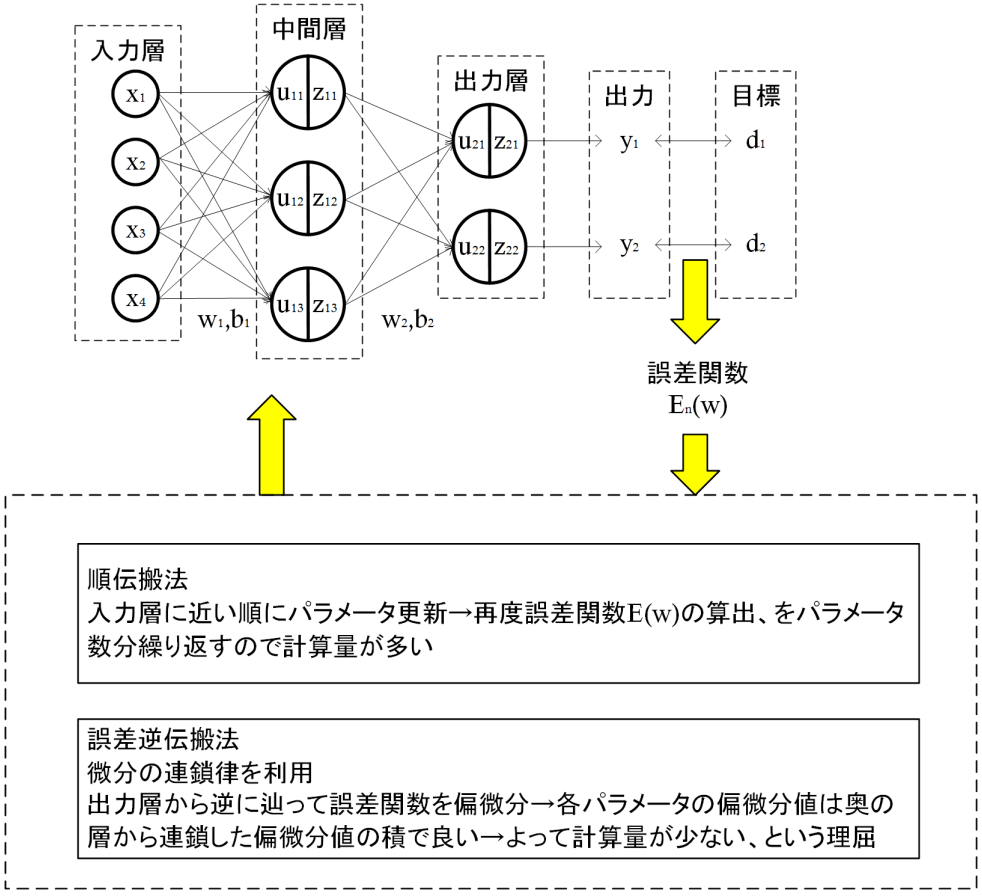


・PCIe の速度は GPU 等に近いメモリの速度より数 10 倍遅い
・実質的に「GPU 直下のメモリに乗るだけのデータしか学習に使えない」とされる
・例えば RTX-3080 の様な GPU のメモリサイズは数 GB 程度、これを超過するデータ群は扱えない
・HOST 装置のメモリを利用するにしても PCIe のボトルネックは避けられない
・これは GPU に限らず TPU 等の計算機でも同様 (のはず)

・ミニバッチ勾配降下法は一度に学習するサイズを分割 (=計算機のメモリに乗る量で学習する)

5　誤差逆伝搬法

・パラメータについてそれぞれ偏微分するが、順伝搬法だと計算量が多い
・順伝搬法：１つのパラメータ更新 → 全体誤差再計算、の流れをパラメータ分繰り返す
・誤差逆伝搬法を用いて計算量を減らす
・誤差逆伝搬法は微分の連鎖律 (の性質) を利用する



6 実習 (day1 と day2)

・本レポートと同じレポジトリにそれぞれ課題を実装したコードを置く

1_1_forward_propagation.ipynb
　コード上に記載の「やってみよう」を課題と解釈して改造実施した

1_2_back_propagation.ipynb
　学習率の選定がされてない (かもしれない) と考えて
　学習実施前後の誤差を比較した実装に変更

1_3_stochastic_gradient_descent.ipynb
　コード上に記載の「やってみよう」を課題と解釈して改造実施した

1_4_1_mnist_sample.ipynb
　コード上に記載の「やってみよう」を課題と解釈して改造実施した