

## 内容

1	Section1_入力層~中間層 .....	2
1. 1	プロローグ .....	2
1. 1. 1	プロローグ 1 __識別モデルと生成モデル .....	2
1. 1. 2	プロローグ 2 __識別器の開発アプローチ .....	3
1. 1. 3	プロローグ 3 __識別器における生成モデル／識別モデル .....	3
1. 1. 4	プロローグ 4 __万能近似定理と深さ .....	4
1. 2	ニューラルネットワークの全体 .....	5
1. 3	入力層~中間層 .....	8
1. 4	その他 .....	10
1. 4. 1	ディープラーニングの開発環境 .....	10
1. 4. 2	入力層の計算 .....	11
1. 4. 3	過学習 .....	12
1. 4. 4	データ集合の拡張 .....	13
1. 4. 5	CNN で扱えるデータの種類 .....	14
1. 4. 6	特徴量の転移 .....	14
2	Section2_活性化関数 .....	15
3	Section3_出力層 .....	17
3. 1	誤差関数 .....	17
3. 2	活性化関数 .....	20
4	Section4_勾配降下法 .....	22
4. 1	勾配降下法 .....	22
4. 2	確率的勾配降下法 .....	25
4. 3	ミニバッチ勾配降下法 .....	25
4. 4	誤差勾配の計算 .....	27
5	Section5_誤差逆伝播法 .....	30
5. 1	誤差勾配の計算 .....	30

## 1 Section1\_入力層~中間層

### 1. 1 プロローグ

#### 1. 1. 1 プロローグ 1 \_\_識別モデルと生成モデル

2つにモデルを分けることができる。

##### 1) 識別モデル

画像を入力すると、犬か猫を識別する。

##### 2) 生成モデル

##### 3) 機械学習（深層学習）モデルを入力・出力の目的で分類したもの

	識別モデル	生成モデル
目的	データを目的のクラスに分類する	特定のクラスのデータを生成する
例	イヌと猫の画像のデータを識別する. $P(C_k x)$	イヌらしい画像を生成する $P(x C_k)$
計算結果	条件“あるデータ $x$ ”が与えられたという条件の基で、クラス $C_k$ である確率	条件“あるクラス $y$ に属する”という条件の基でのデータ $C_k$ の分布

##### 4) 機械学習・深層学習モデルのそれぞれで識別・生成を行う手法が存在する。

	識別モデル	生成モデル
具体的なモデル	決定木、ロジスティック回帰、サポートベクターマシン (SVM)、ニューラルネットワーク	隠れマルコフモデル、ベイジアンネットワーク、変分オートエンコーダ (VAE)、敵対的生成ネットワーク (GAN)
特徴	高次元→低次元 実用な学習データ：少	低次元→高次元 必要な学習データ：多い
応用例	画像認識	画像の超解像 テキストの生成

## 1. 1. 2 プロログ 2 \_\_識別器の開発アプローチ

1) 識別器の開発には3つのアプローチが存在する。生成も識別に応用できる。

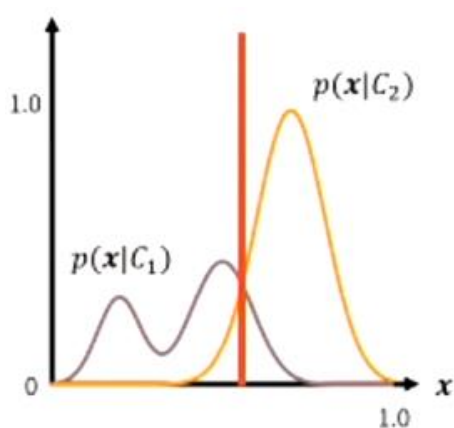
	生成モデル	識別モデル	識別関数
識別の計算	$P(x C_k) \cdot p(C_k)$ を推定ベイズの定理より $P(C_k x)=p(x C_k) \cdot p(C_k)/p(x)$ ただし $p(x)=\sum p(x C_k) \cdot p(C_k)$	$P(C_k x)$ を推定 決定理論に基づき識別結果を得る ※閾値に基づく決定など	入力値 $x$ を直接クラスに写像(変換)する関数 $f(x)$ を推定する
モデル化の対象	<ul style="list-style-type: none"> <li>各クラスの生起確率</li> <li>データのクラス条件付き密度</li> </ul>	<ul style="list-style-type: none"> <li>データがクラスに属する確率</li> </ul>	<ul style="list-style-type: none"> <li>データの属するクラスの情報のみ</li> <li>※確率はけいさんされない</li> </ul>
特徴	<ul style="list-style-type: none"> <li>データを自動で生成できる確率的な識別</li> </ul>	<ul style="list-style-type: none"> <li>確率的な識別</li> </ul>	<ul style="list-style-type: none"> <li>学習量が少ない</li> <li>決定的な識別</li> </ul>
コスト	大	中	小

パターン認識と機械学習

識別器の開発アプローチは生成モデル、識別モデル、識別関数がある。

## 1. 1. 3 プロログ 3 \_\_識別器における生成モデル／識別モデル

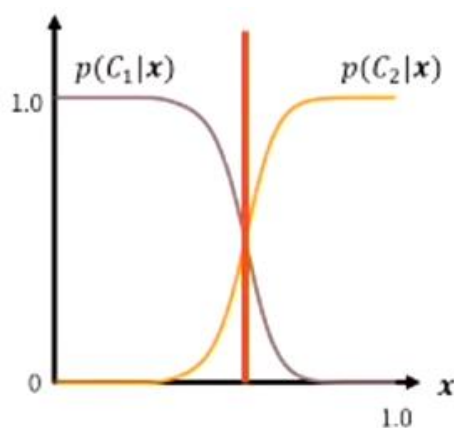
1) データの分布は分類結果より複雑なことがある。生成モデルはこれを推定する。



データのクラス条件付き密度

生成モデルが学習する内容

灰色の線が2つの山があるを学習する

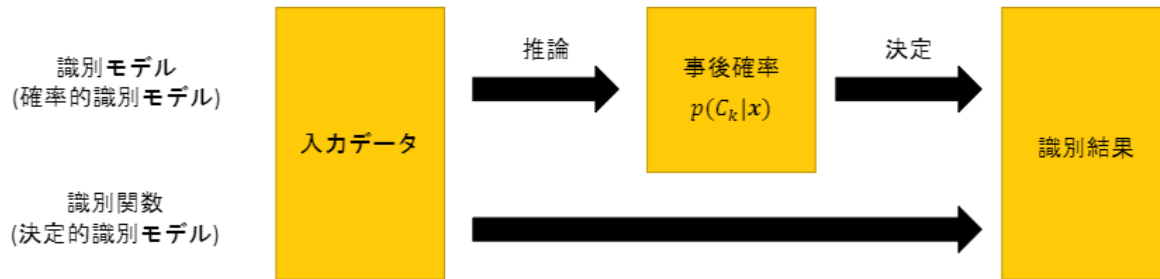


データがクラスに属する確率

識別モデルが学習する内容

- データのクラス条件付き密度は分類結果より複雑→生成モデルはこの結果を推定する。計算量が多い
- 単に分類結果を得るのであれば、直接データクラスにある確率を求める。→分類モデルのアプローチ

2) 識別モデルは識別結果の確率が得られる。識別関数では、識別結果のみ



①. 識別モデルによる識別 (確率的識別モデル)

- ・推論：入力データを基に事後確率を求める。
- ・決定：事後確率を基に識別結果を得る。

→間違いの程度を測ることができる

→推論結果の取り扱いを決められる (破棄など)

②. 識別関数による識別 (決定的識別モデル)

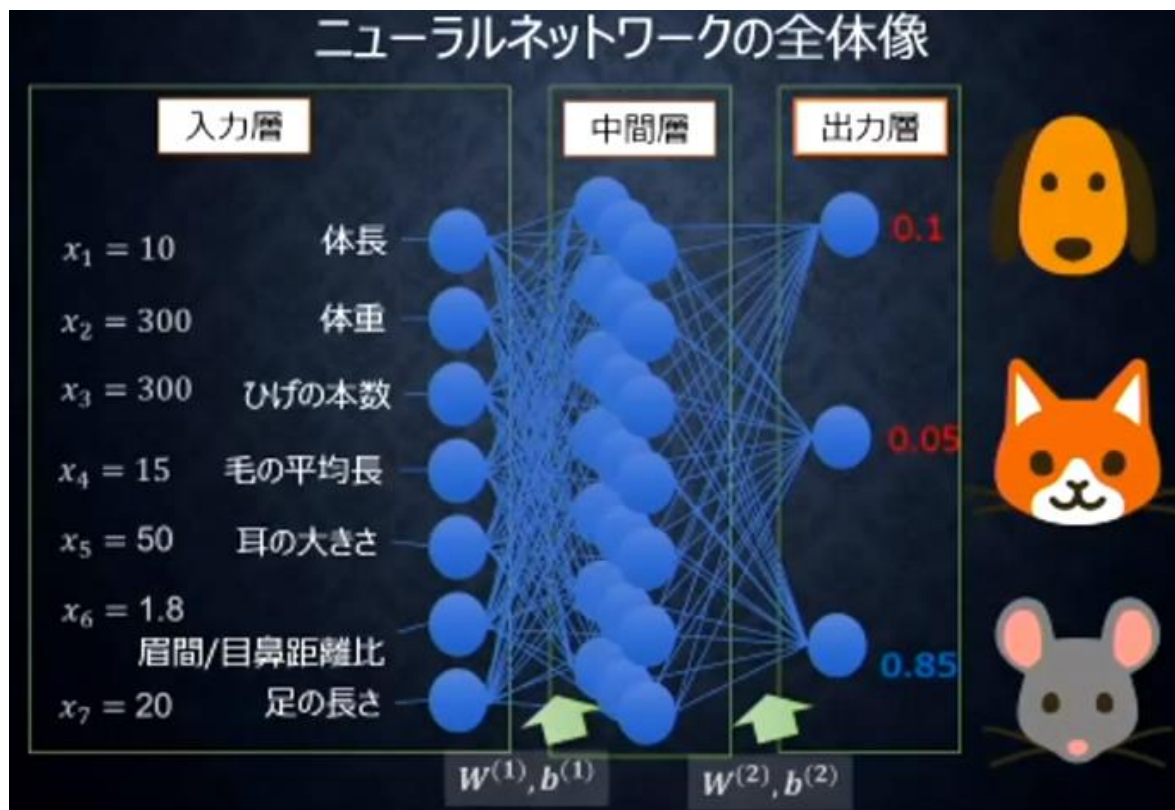
- ・入力データから識別結果を一気に得る。

→間違いの程度を測ることができない。

#### 1. 1. 4 プロローグ 4 \_\_ 万能近似定理と深さ

- ・ノーフリーランチ定理
- ・ニューラルネットワークが優れているというモチベーション的な話

## 1. 2 ニューラルネットワークの全体



入力データは数字の集まりで、出力はイヌ、ネコ、鼠の確率を出す

・  $w$ 、 $b$  を学習していく

### 1) 確認テスト

ディープラーニングは、結局何をやろうとしているか2行以内で述べよ

また次の中のどの値の最適化が最終目的化全て選べ

①入力値 $[X]$ 、②出力値 $[Y]$ 、③重み $[W]$ 、④バイアス $[b]$ 、⑤総入力、⑥中間層入力 $[z]$ 、⑦学習率 $[\rho]$

解) 明示的なプログラムの代わりに、多数の中間層を持つニューラルネットワークを用いて、入力値から目的とする出力値に変換する学習モデルを構築すること

## 2) ニューラルネットワークの全体像

### 【事前に用意する情報】

入力:  $\mathbf{x}_n = [x_{n1} \dots x_{ni}]$

訓練データ:  $\mathbf{d}_n = [d_{n1} \dots d_{ni}]$

### 【多層ネットワークのパラメータ】

$$\mathbf{w}^{(l)} \begin{cases} \text{重み: } \mathbf{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & \dots & w_{li}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{j1}^{(l)} & \dots & w_{jl}^{(l)} \end{bmatrix} \\ \text{バイアス: } \mathbf{b}^{(l)} = [b_1^{(l)} \dots b_l^{(l)}] \end{cases}$$

活性化関数:  $\mathbf{f}^{(l)}(\mathbf{u}^{(l)}) = [f^{(l)}(u_1^{(l)}) \dots f^{(l)}(u_l^{(l)})]$

中間層出力:  $\mathbf{z}^{(l)} = [z_1^{(l)} \dots z_k^{(l)}] = \mathbf{f}^{(l)}(\mathbf{u}^{(l)})$

総入力:  $\mathbf{u}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$

出力:  $\mathbf{y}^{(l)} = [y_{n1}^{(l)} \dots y_{nk}^{(l)}] = \mathbf{z}^{(l)}$

誤差関数:  $E_n(\mathbf{w})$

入力層ノードのインデックス:  $i (= 1 \dots I)$

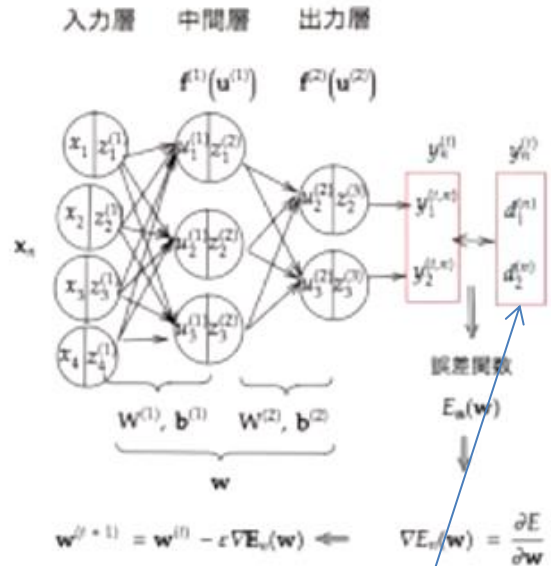
中間層ノードのインデックス:  $j (= 1 \dots J)$

出力層ノードのインデックス:  $k (= 1 \dots K)$

層のインデックス:  $l (= 1 \dots L)$

訓練データのインデックス:  $n (= 1 \dots N)$

試行回数のインデックス:  $t (= 1 \dots T)$



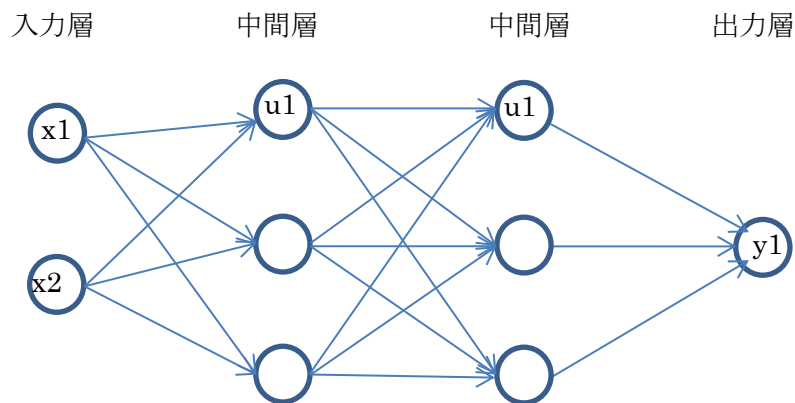
正解値

## 3) テスト

入力: 2 ノード 1 層

中間層: 3 ノード 2 層

出力層: 1 ノード 1 層



#### 4) ニューラルネットワークの問題の種類

##### ①. 回帰

###### ●予想結果

- ・売り上げ予想
- ・株価予想

###### ●ランキング

- ・競馬順位予想
- ・人気順位予想

##### ②. 分類

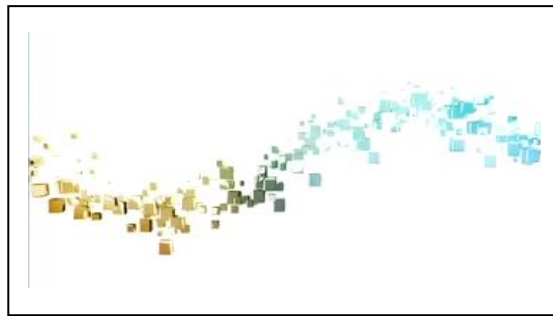
- ・猫写真の判別
- ・手書き文字の認識
- ・花の種類分類

#### 5) 回帰の問題

連続する実数値をとる関数の近似

〔回帰分析〕

- ①. 線形回帰
- ②. 回帰木
- ③. ランダムフォレスト
- ④. ニューラルネットワーク (NN)



#### 6) 分類の問題

性別（男あるいは女）や動物の種類など離散的な結果を予想する。

〔分類分析〕

- ①. ベイズ分類
- ②. ロジスティック回帰
- ③. 決定木
- ④. ランダムフォレスト
- ⑤. ニューラルネットワーク (NN)



#### 7) 深層学習の応用例

##### ①. 自動売買（トレード）

ある時点での数字の集まりを入力にして、次の時点での予想を行う。

##### ②. チャットボット

文章を入力として、文章を出力する。

##### ③. 翻訳

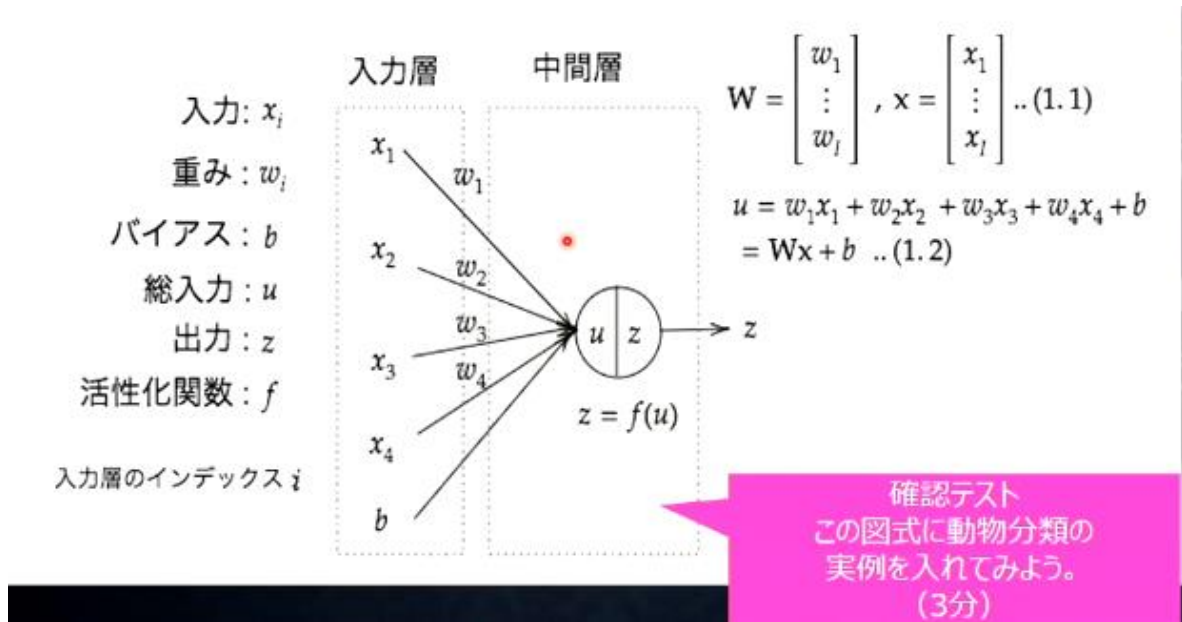
文章を数字に置き換え、変換して文字を出力する

##### ④. 音声解釈

音声データは数字になる（周波数）

##### ⑤. 囲碁、将棋 AI

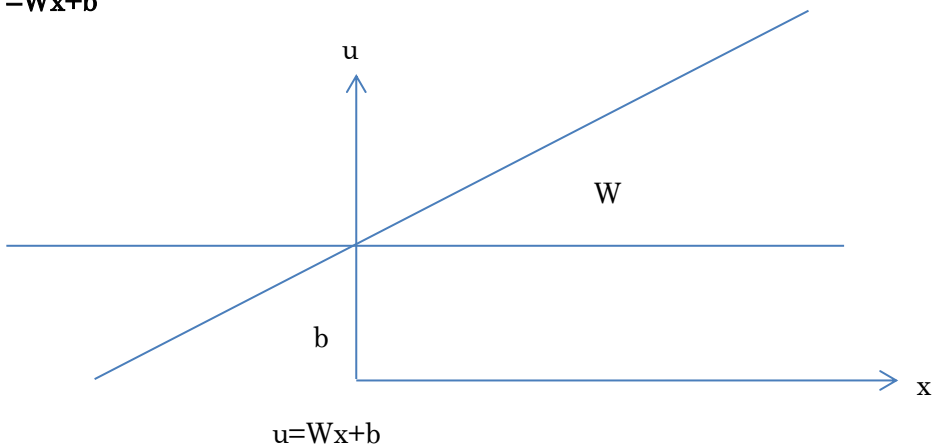
### 1. 3 入力層～中間層



$$W = \begin{bmatrix} w_1 & x_1 \\ w_2 & x_2 \\ w_3 & x_3 \\ w_4 & x_4 \end{bmatrix}, x = x_2 \dots (1,1)$$

$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

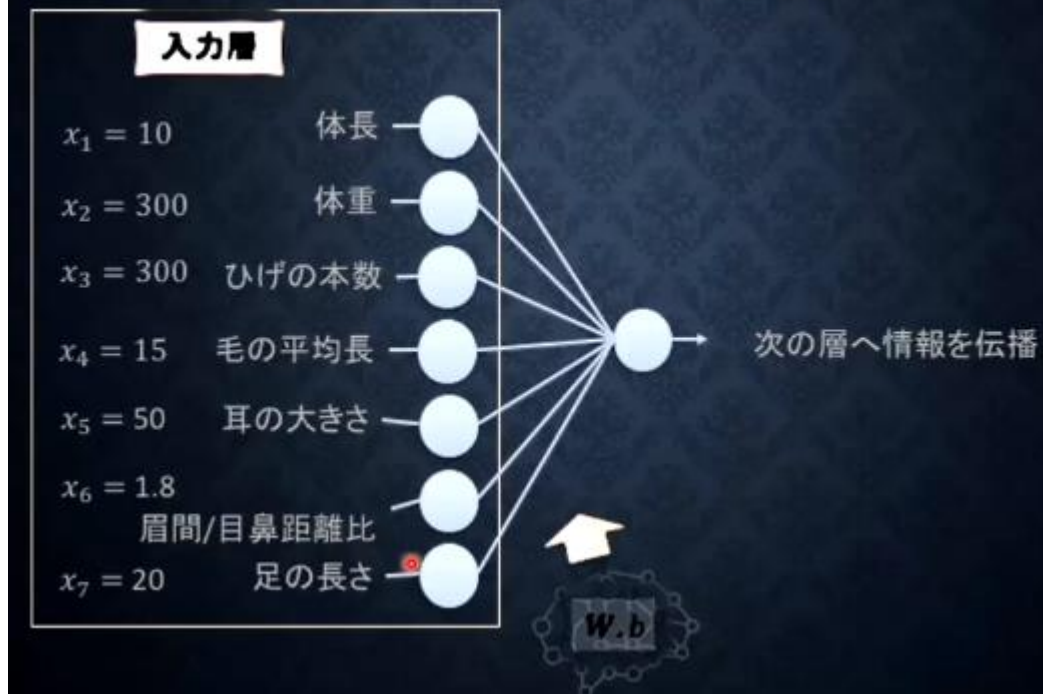
$$= Wx + b$$



動物問題の実例



## 動物種類分類ネットワーク



$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$= Wx + b$$

Python で記述

$$u = \text{np.dot}(x, W) + b$$

## 1. 4 その他

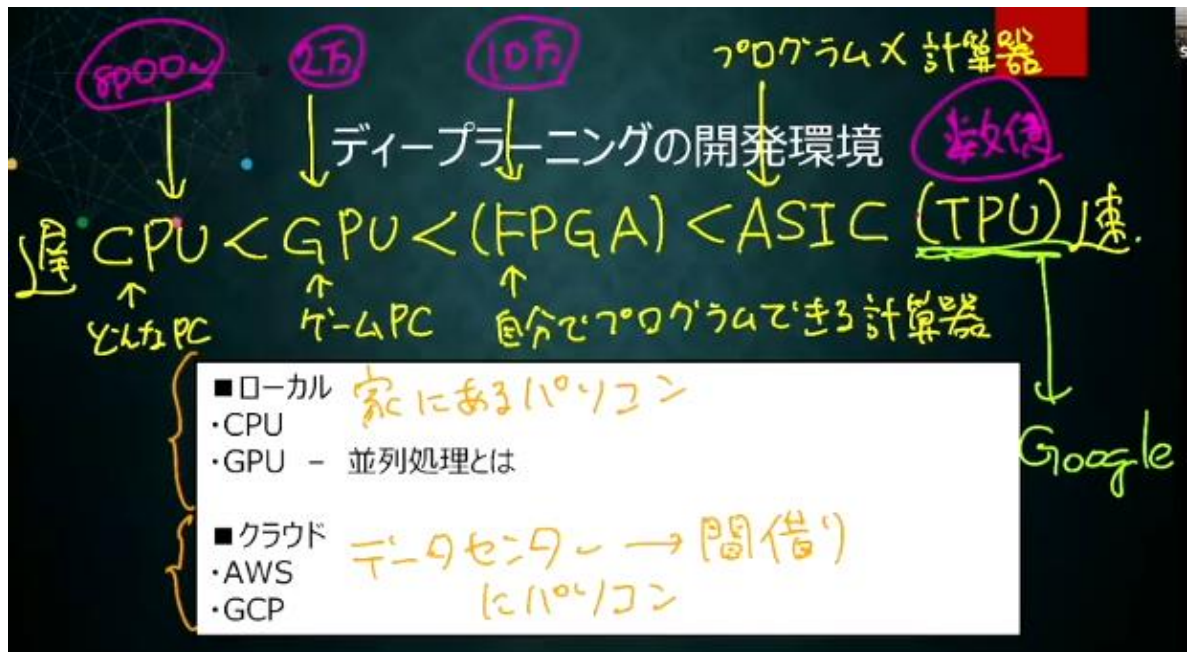
### 1. 4. 1 ディープラーニングの開発環境

#### 1) ローカル

- CPU
- GPU

#### 2) クラウド

- AWS
- GPU



## 1. 4. 2 入力層の計算

### 1) 入力として取り得るデータ

- ①. 連続する実数
- ②. 確率
- ③. フラグ値 (クラス)

• 入力としてとり得るデータ

- 連続する実数
- 確率
- フラグ値

↓

[0, 0, 1] ← one-hot ラベル  
犬 猫 ネ

### 2) 入力として取るべきでないデータ

- ①. 欠損値が多いデータ
- ②. 誤差の大きいデータ
- ③. 出力そのもの、出力を加工した情報
- ④. 連続性のないデータ (背番号とか)
- ⑤. 無意味な数が割り当てられているデータ → one hot ラベル

end2end : 入力から出力まで

• 入力層として取るべきでないデータ

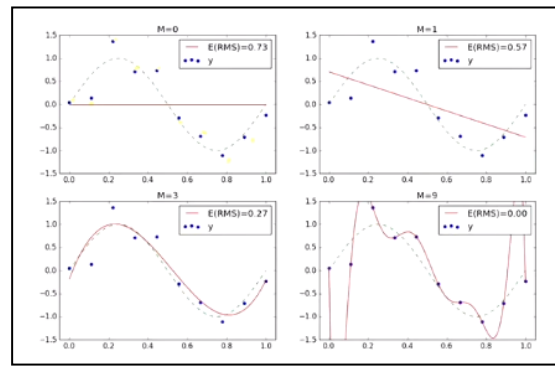
- ★ 欠損値が多いデータ
- 誤差の大きいデータ
- 出力そのもの、出力を加工した情報
- 連続性の無いデータ (背番号とか)
- 無意味な数が割り当てられているデータ
  - 悪い例 Yes:1, No:0, どちらでもない: -1, 無回答:-1
  - 良い例 Yes:1, No:-1, どちらでもない: 0, 無回答:なし

生 → 変換 → 出

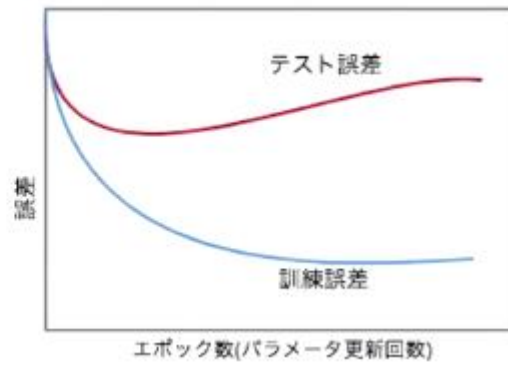
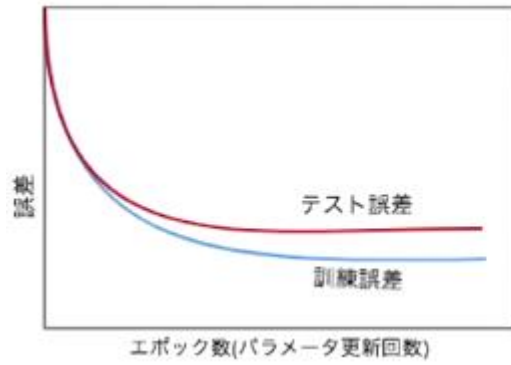
100 → [100, 200, 300, 250 ... 300]

N個 ↓ [ (X) X X 280, X X ]X

### 1. 4. 3 過学習



巨大な NN



過学習

## 1. 4. 4 データ集合の拡張

### Data Augmentation

学習データが不足しているときに、人工的にデータを作り増やす方法

#### 1) 分類タスク

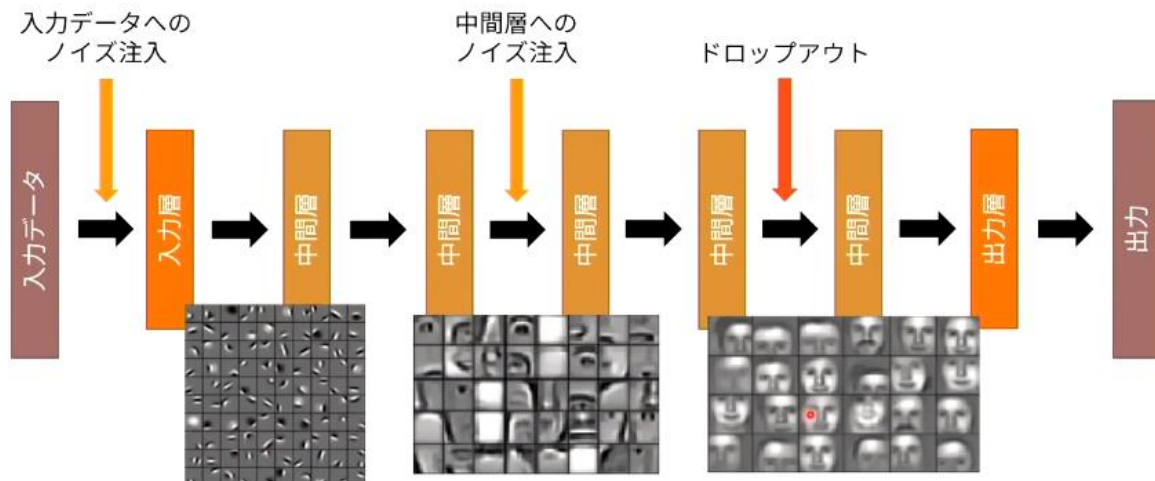
#### 2) その他のタスク

密度推定はデータを増やしてはだめ

#### 3) 画像でのデータ拡張手法は様々なものがある。

- ①. 輝度
- ②. ノイズ
- ③.

#### 4) 中間層へのノイズ注入で様々な抽象化レベルでのデータ拡張が可能



#### 6) データ拡張の用途

##### ①. データ拡張の効果と性能評価

- ・データ拡張を行うとしばしば劇的に汎化性能が向上する。
- ・ランダムなデータ拡張を行うときは学習データが毎度異なるため再現性に注意

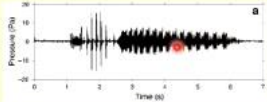
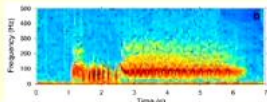




##### ②. データ拡張とモデルの捉え方

- ・一般的に適用可能なデータ拡張（ノイズ付加など）はモデルの一部として捉える（ドロップアウトなど）
- ・特定の作業に対してのみ適用可能なデータ拡張（クロップなど）は入力データの事前加工として捉える。

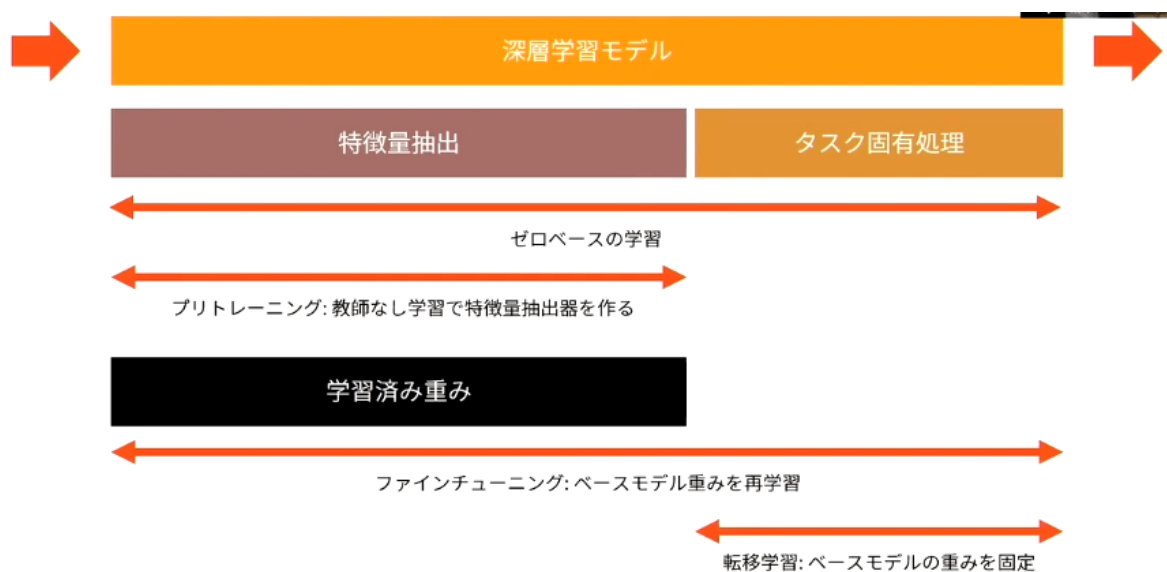
例：不用品検知の画像識別モデルに製品の一部だけが拡大された画像は入力されない。

## 1. 4. 5 CNNで扱えるデータの種類

CNNでは次元間でつながりのあるデータを扱える。

	1次元	2次元	3次元
単一チャンネル	音声 [時刻, 強度] 	フーリエ変換した音声 [時刻, 周波数, 強度] 	CTスキャン画像 [x, y, z, 強度] 
複数チャンネル	アニメのスケルトン [時刻, (腕の値, 膝の値 …)] 	カラー画像 [x, y, (R, G, B)] 	動画 [時刻, x, y, (R, G, B)] 

## 1. 4. 6 特徴量の転移



### 1) ファインチューニング

学習済みの重みの部分も学習させる

### 2) 転移学習

学習済みの部分は学習せず、やりたい部分だけ学習する。

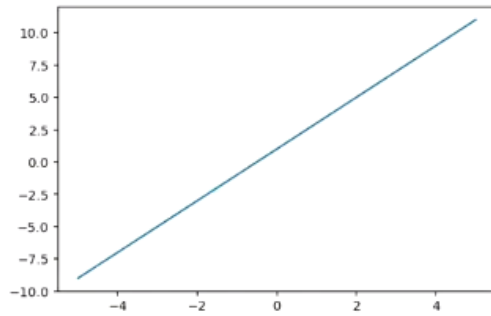
### 3) プリトレーニング

教師なし学習で特徴量抽出器を作る。

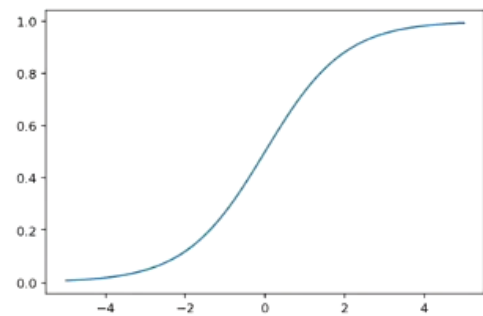
## 2 Section2\_活性化関数

- ・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の関数
- ・入力の値によって、次の層への信号の ON/OFF や強弱を定める働きをもつ。

### 1) 線形と非線形の違い



線形な関数



非線形な関数

線形な関数は

- ・加法性 :  $f(x+y)=f(x)+f(y)$
- ・斉次性 :  $f(kx)=kf(x)$

を満たす

非線形な関数は加法性、斉次性を満たさない。

### ●ニューラルネットワークの特徴

線形な関数で出力した値を非線形な関数に入力して出力すること。

### 2) 中間層の活性化関数

- ・ReLU 関数
- ・シグモイド（ロジスティック）関数
- ・ステップ関数

### 3) 出力層の活性化関数

- ・ソフトマックス関数
- ・恒等写像
- ・シグモイド（ロジスティック）関数

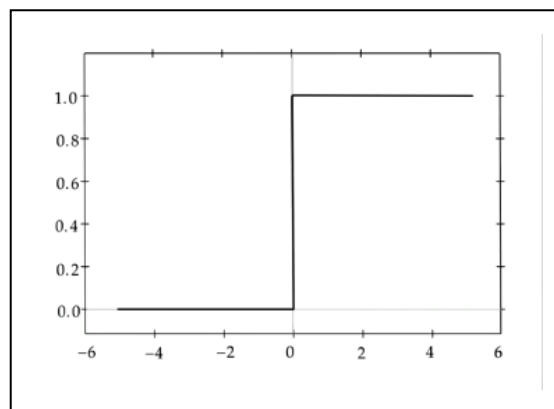
### 4) ステップ関数

#### ①. サンプルコード

```
def step_function(x):  
    if x > 0:  
        return 1  
    else  
        return 0
```

#### ②. 数式

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$



閾値を超えたら発火する関数であり、出力は常に 1 か 0。

パーセプトロン（ニューラルネットワークの前進）で利用された関数



課題：0 - 1 間の値を表現できず、線形分離可能なものしか学習できなかった。

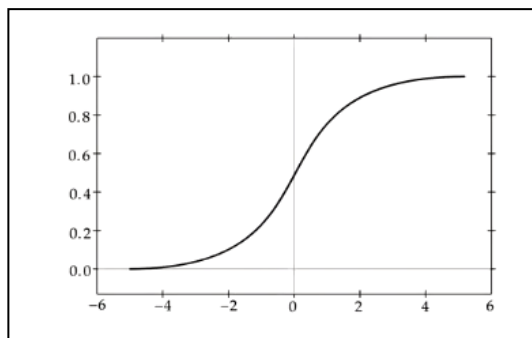
## 5) シグモイド関数

### ①. サンプルコード

```
def sigmoid(x):  
    return 1/(1+np.exp(-x))
```

### ②. 数式

$$f(x)=1/(1+\exp(-x))$$



0 から 1 の間を緩やかに変化する関数で、ステップ関数では ON/OFF しかない状態に対して、信号の強弱を伝えられるようになり、予想ニューラルネットワークの普及のきっかけになった。

課題：大きな値では出力の変化が微小なため、勾配消失問題を引き起こすことが有った。

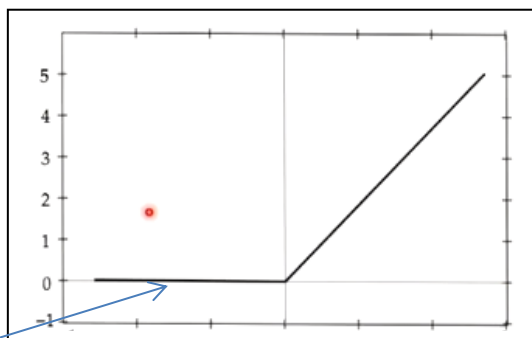
## 6) ReLU 関数

### ①. サンプルコード

```
def relu(x):  
    return np.maximum(0,x)
```

### ②. 関数

$$f(x)= \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$



今最も使われている活性化関数

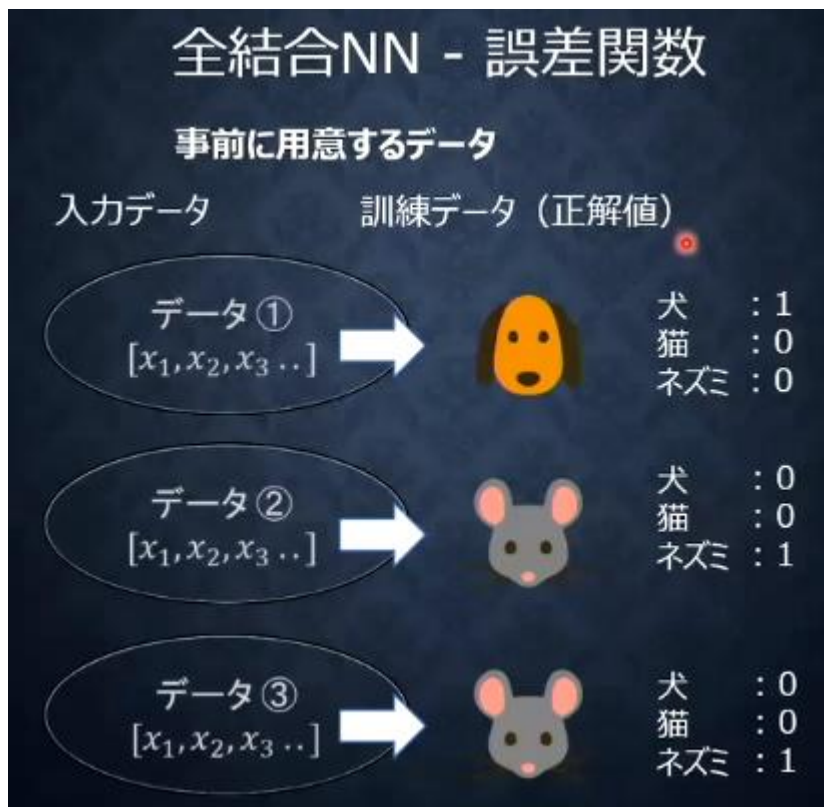
勾配消失問題の回避とスパース化に貢献することでよい成果をもたらしている。



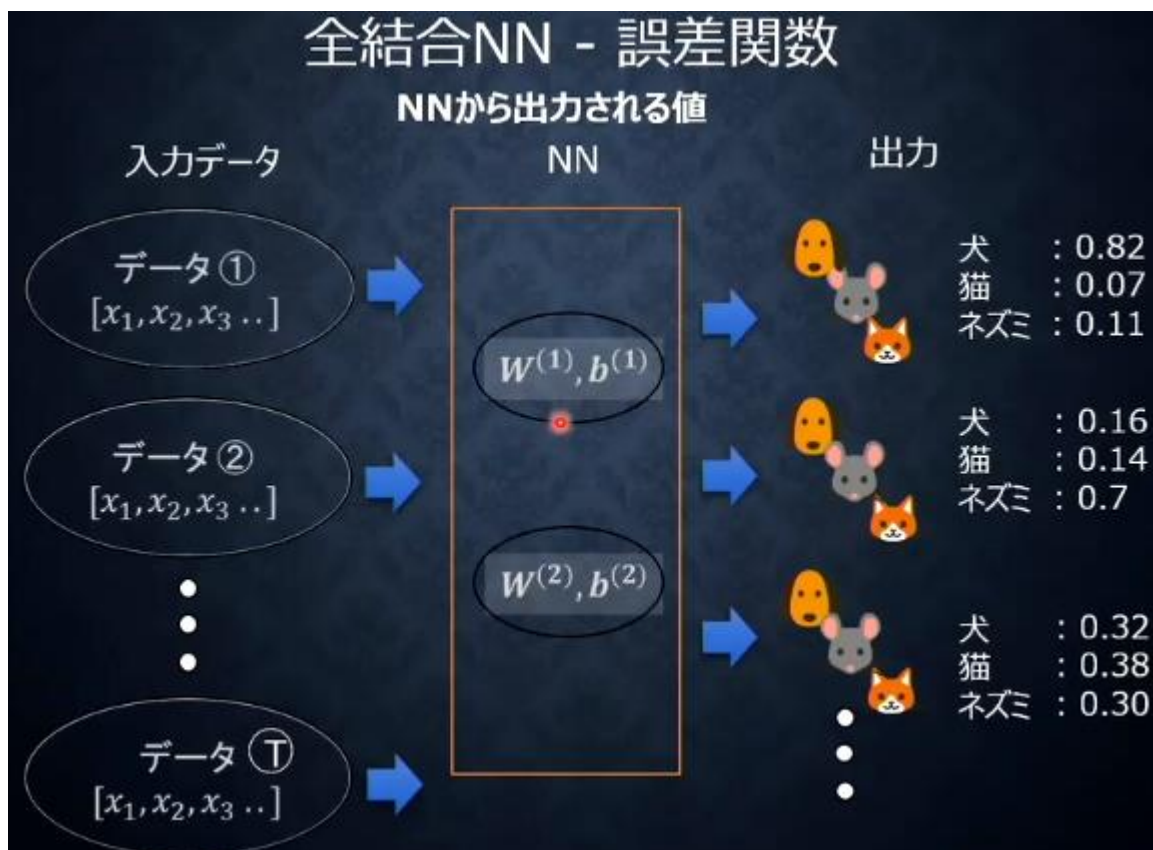
### 3 Section3\_出力層

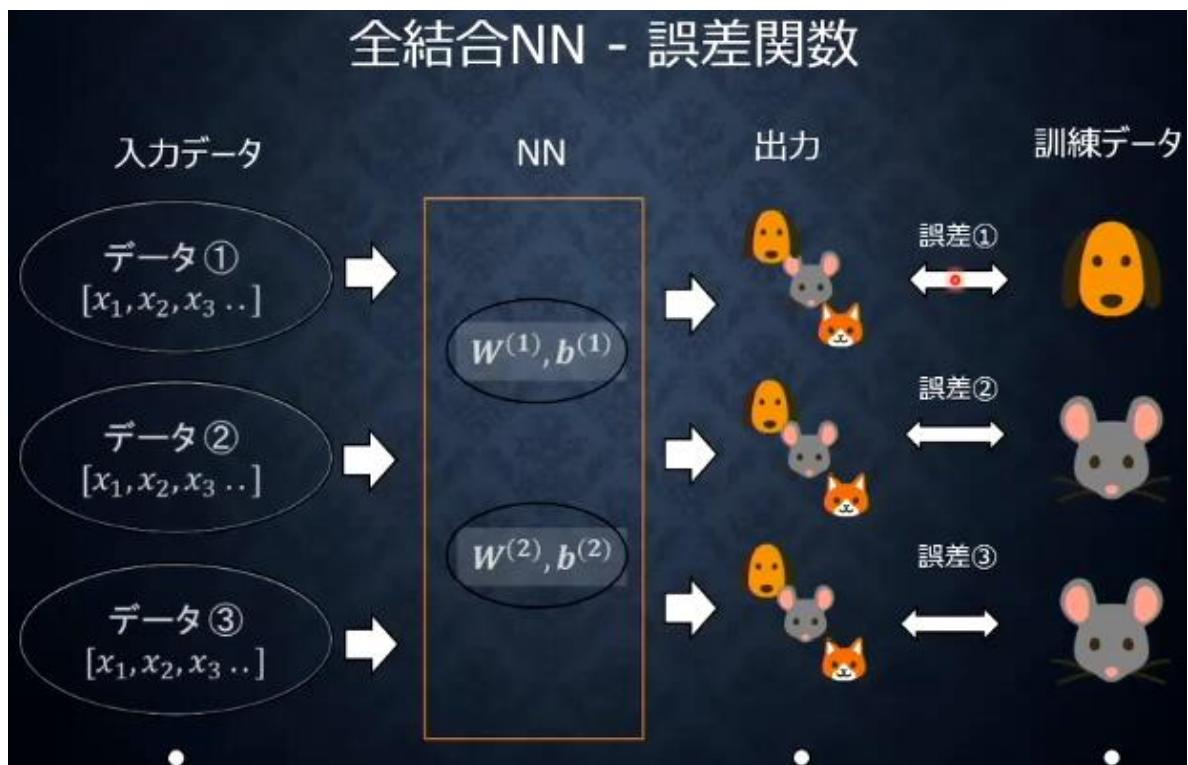
#### 3. 1 誤差関数

1) 訓練データ (入力データと正解値) : 事前に用意する



2) 実際の訓練方法





結果と実際のデータと比較する。

どのくらい合っていたかが誤差関数という

## 全結合NN - 誤差関数

誤差の計算 誤差関数 = 二乗誤差の場合

出力	訓練データ	誤差
犬 : 0.82 猫 : 0.07 ネズミ : 0.11	犬 : 1 猫 : 0 ネズミ : 0	$\frac{1}{2} ((1 - 0.82)^2 + (0 - 0.07)^2 + (0 - 0.11)^2)$ $= 0.01871$
犬 : 0.16 猫 : 0.14 ネズミ : 0.70	犬 : 0 猫 : 0 ネズミ : 1	$\frac{1}{2} ((0 - 0.16)^2 + (0 - 0.14)^2 + (1 - 0.70)^2)$ $= 0.0676$
犬 : 0.02 猫 : 0.38 ネズミ : 0.60	犬 : 0 猫 : 0 ネズミ : 1	$\frac{1}{2} ((0 - 0.02)^2 + (0 - 0.38)^2 + (1 - 0.60)^2)$ $= 0.1524$
⋮	⋮	⋮

$$E_n(\mathbf{w}) = \frac{1}{2} \sum_{j=1}^I (y_j - d_j)^2 = \frac{1}{2} \|\mathbf{y} - \mathbf{d}\|^2$$

誤差関数＝二乗誤差

数値が小さいと正解に近い、大きいと正解に遠い

### 3) 問題

①. なぜ引き算ではなく二乗するか

引き算のままだと合計が0になってしまう。

解)

引き算を行うだけでは、各ラベルの誤差で正負両方が発生し、実際の誤差を正しく表すのに都合が悪い。  
2乗してそれぞれのラベルでの誤差を正の値にするようにする。

②. 誤差関数の  $1/2$  は何故つけるか

微分した時二乗の2が出るため、 $1/2$ で相殺(1)にするため

解) 実際にネットワークを学習する時に行う誤差逆伝播の計算で、誤差関数の微分を用いるが、その際の計算式を簡単にするため。

4) 数式

$$E_n(\mathbf{w}) = \frac{1}{2} \sum_{j=1}^J (y_j - d_j)^2 = \frac{1}{2} \|\mathbf{y} - \mathbf{d}\|^2$$

5) サンプルコード

`loss=function.mean_squared_error(d,y)`————>回帰に用いられる

※本来ならば分類問題の場合誤差関数にクロスエントロピー誤差を用いるので

`loss=cross_entropy_error(d,y)`

となる。

### 3. 2 活性化関数

#### 1) 出力層と中間層の違い

「値の強弱」

- ・ 中間層：閾値の前後で信号の強弱を調整
- ・ 出力層：信号の大きさ（比率）はそのまま変換

「確率出力」

- ・ 分類問題の場合、出力層の出力は0～1の範囲に限定し、総和を1とする必要がある。

→出力層と中間層で利用される活性化関数が異なる。

#### 2) 出力層の種類

	回帰	二値分類	多クラス分類
活性化関数	恒等写像（何もしない） $f(u)=u$	シグモイド関数 $f(u)=1/(1+\exp(-u))$	ソフトマックス関数 $f(i,u)=e^i / \sum e^u$
誤差関数	二乗誤差	交差エントロピー	

#### ●訓練データサンプル当たりの誤差

二乗誤差  $E_n(w) = 1/2 \sum_{i=1}^I (y_n - d_n)^2$

交差エントロピー  $E_n(w) = \sum_{i=1}^I d_i \log y_i$

#### ●学習サイクル当たりの誤差

$$E(w) = \sum_{n=1}^N E_n$$

#### 3) シグモイド関数

##### ①. 数式

$$f(u) = 1/(1+e^{-u})$$

##### ②. サンプルコード

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

#### 4) ソフトマックス関数

##### ①. 数式

$$f(i,u) = e^i / \sum_{k=1}^K e^u$$

```
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x,axis=0)
        y = np.exp(x) / np.sum(np.exp(x),axis=0)
        return y.T
    x = x - np.max(x) #オーバフロー対策
    return np.exp(x) / np.sum(np.exp(x))
```

3クラス以上の時に全部を足したら1になるように出力する。

## 5) 誤差関数

### ①数式

$$E_n(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^I (y_n - d_n)^2 \quad \text{.. 二乗誤差}$$

### ②サンプルコード

```
def mean_squared_error(d,y):  
    return np.mean(np.squared(d-y))/2  
  
d:  
y:ニューラルネットワークで出力した値
```

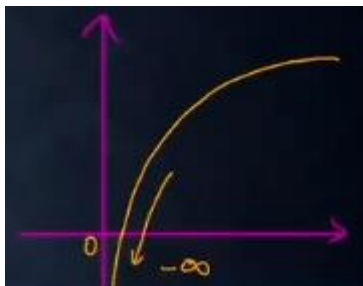
## 6) 交差エントロピー

### ①数式

$$E_n(\mathbf{w}) = - \sum_{i=1}^I d_i \log y_i \quad \text{.. 交差エントロピー}$$

### ②. サンプルコード

```
def cross_entropy_error(x):  
    if x.ndim == 1:  
        d=d.reshape(1,d.size)  
        y=y.reshape(1,y.size)  
        #教師データが one-hot-vector の時は、正解ラベルのインデックスに変換  
    if d.size == y.size:  
        d=d.argmax(axis=1)  
    batch_size=y.shape[0]  
    return -np.sum(np.log(y[np.arange(batch_size),d]+1e-7))/batch_size
```



1e-7 :  $-\infty$ にならないように

## 4 Section4\_勾配降下法

### 4. 1 勾配降下法

勾配降下法には3つある。

- ①. 勾配降下法
- ②. 確率的勾配降下法
- ③. ミニバッチ勾配降下法

#### 1) 深層学習の目的

学習を通して誤差を最小にするネットワークを作成すること。

→誤差  $E(w)$  を最小化するパラメータ  $w, b$  を発見すること。

#### 2) 勾配降下法を利用してパラメータを最適化

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E \quad \nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[ \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

$\varepsilon$  : 学習率

#### ①. 数式

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_n$$

#### ②. サンプルコード

```
network[key] -= learning_rate*grad[key]
```

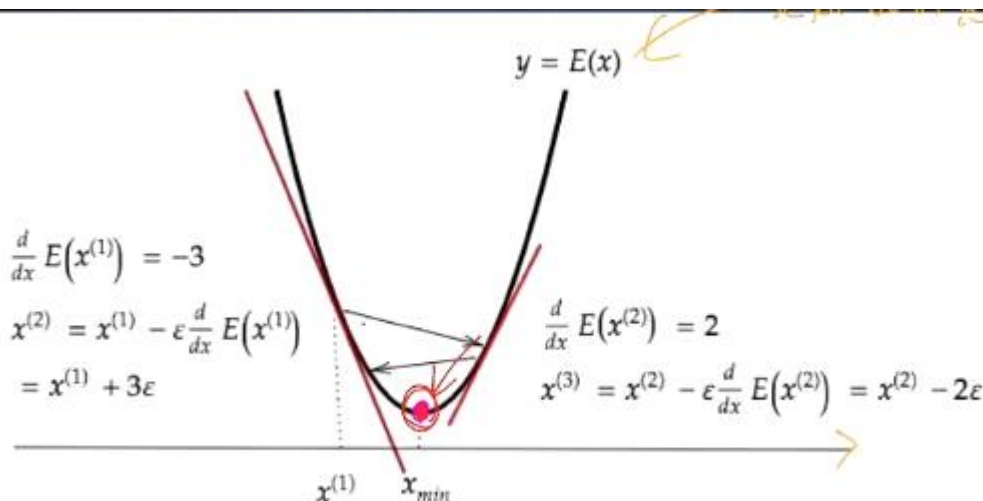
#### ①. 数式

$$\nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[ \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

#### ②. サンプルコード

```
grad=backword(x,d,z,y)
```

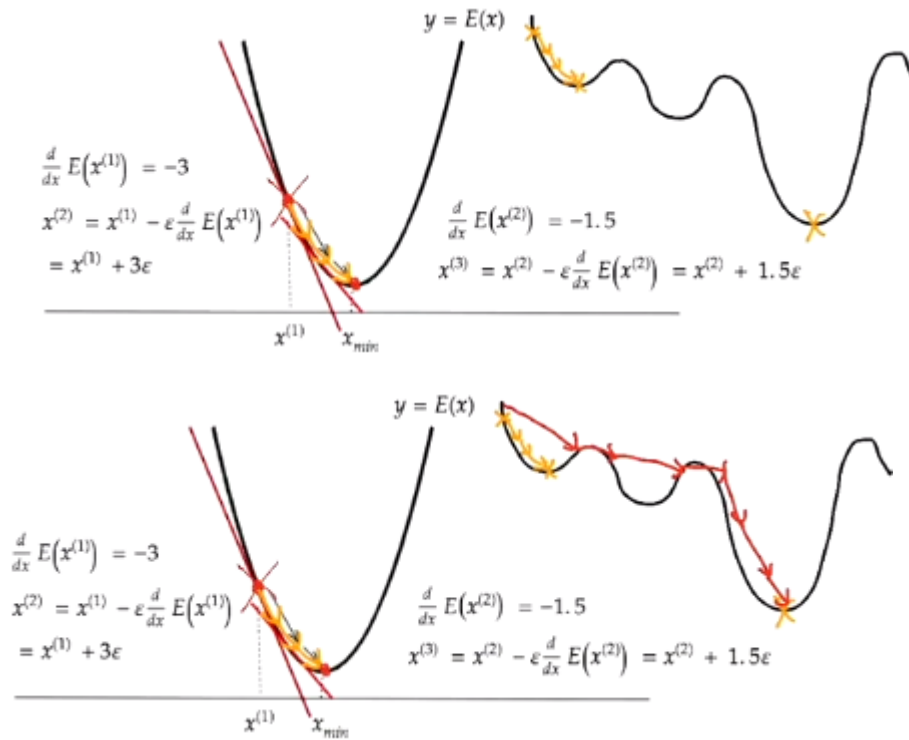
前回考えたことから、間違えた部分を生かして、次の計算をする。



3) 学習率が大きい場合は、発散してします

4) 学習率が小さい場合は収束に時間がかかる。

そこがたくさんある場合は、局所解を取ってしまう。

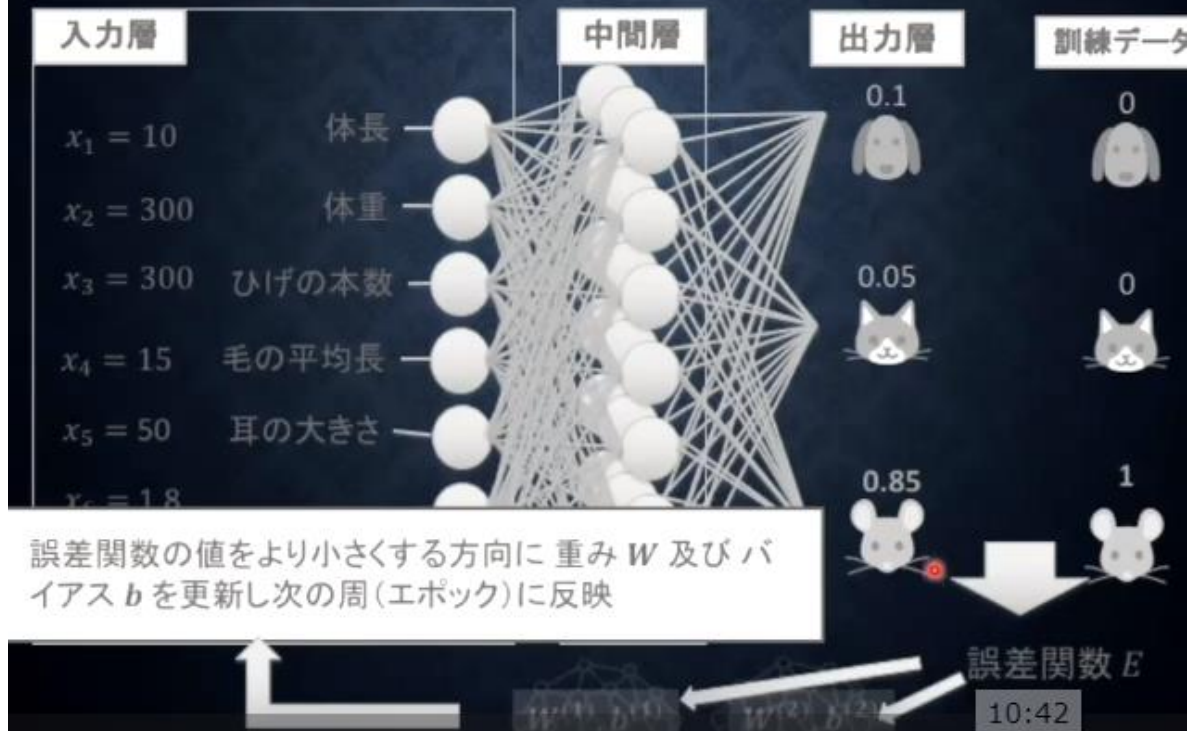


5) 学習率の更新方法

- ①. モーメンタム
- ②. AdaGrad
- ③. Adadelta
- ④. Adam 良く使われる



# 全結合NN - 勾配降下法



1回のサイクルをエポックという。



## 4. 2 確率的勾配降下法

【確率的勾配降下法】	【勾配降下法】
$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_n$	$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E$

ランダムに抽出したサンプルの誤差

全サンプルの平均誤差

### 1) 確率勾配法のメリット

- ①. データが冗長な場合の計算コストの軽減
- ②. 望まない局所極小解に収束するリスクの軽減
- ③. オンライン学習ができる。

### 2) オンライン学習：リアルタイムに学習させる

解) 学習データが入ってくるたびに都度パラメータ ( $\mathbf{w}, \mathbf{b}$ ) を更新し、学習を進めていく方法。一方バッチ学習では、一度に全ての学習データを使ってパラメータ更新を行う。

## 4. 3 ミニバッチ勾配降下法

オンライン学習を参考に容量を少なく処理する。(小分けにして行う)

ミニバッチ勾配降下法	
【ミニバッチ勾配降下法】	【確率的勾配降下法】
$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_t$	$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_n$
$E_t = \frac{1}{N_t} \sum_{n \in D_t} E_n$	
$N_t =  D_t $	

ランダムに分割したデータの集合 (ミニバッチ)

ランダムに抽出したサンプルの誤差

$D_t$  に属するサンプルの平均誤差

ミニバッチ数で誤差を割り平均化する。

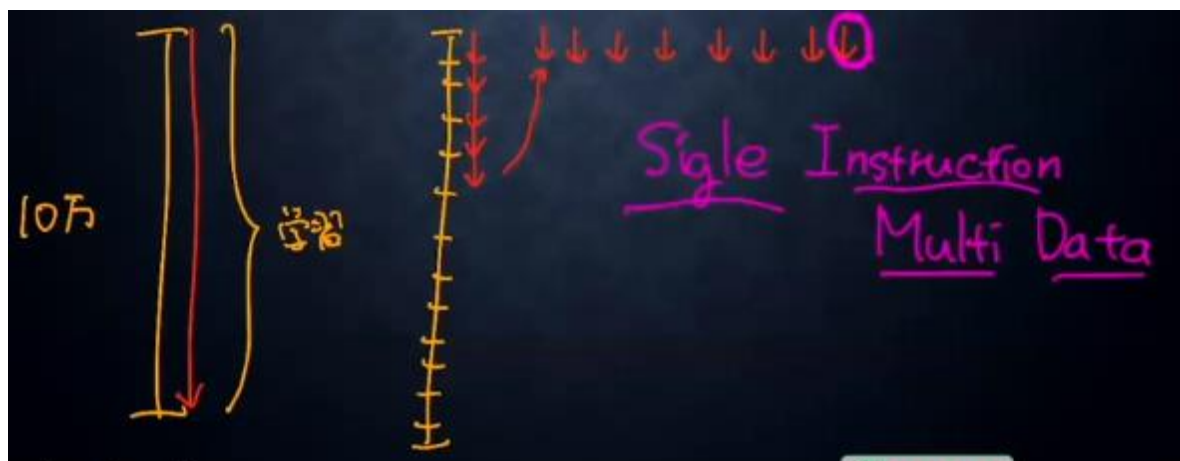
### 1) ミニバッチ勾配降下法のメリット

- ・ 確率勾配降下法のメリットを損なわず、計算機の計算資源を有効に利用できる

→CPU を使用したスレッド並列化や GPU を利用した SIMD 並列化

バッチごとに並列に処理できる

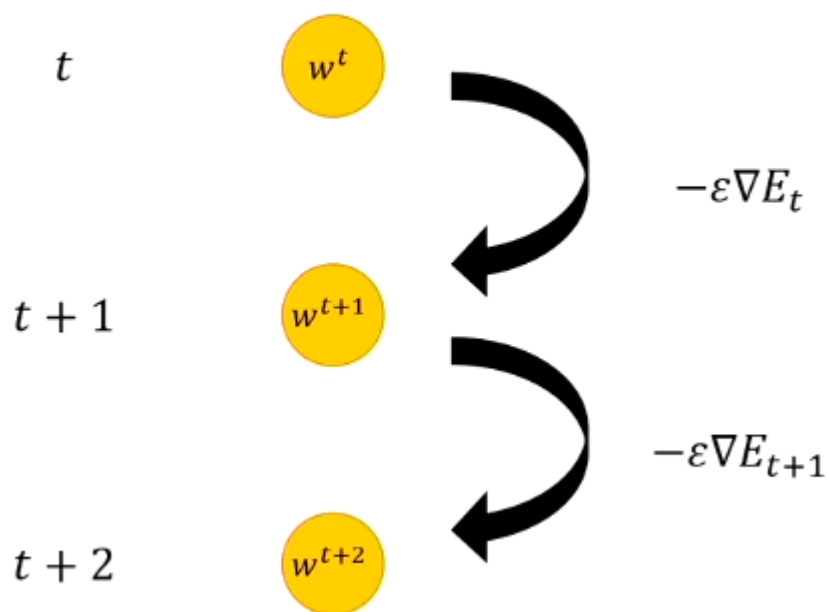
SIMD : 並列実行



$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_t$$

エポック

重み



#### 4. 4 誤差勾配の計算

$$\nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[ \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

どうやって導き出すか

重みは、3層であれば3か所ある。

[数値微分]

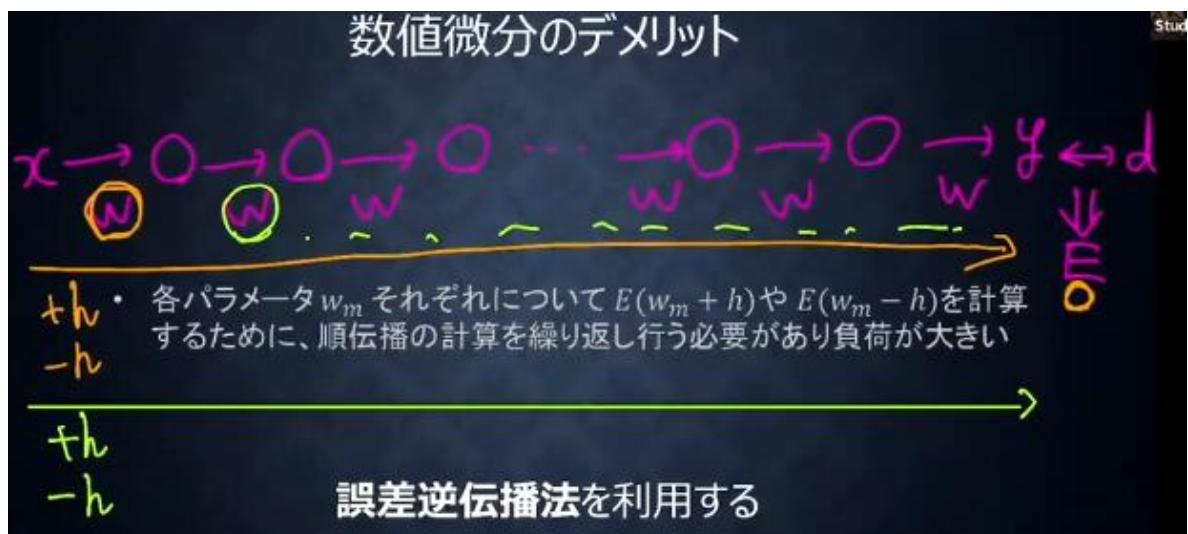
$$\frac{\partial E}{\partial w_m} \approx \frac{E(w_m + h) - E(w_m - h)}{2h}$$

部分を使って重みを更新する

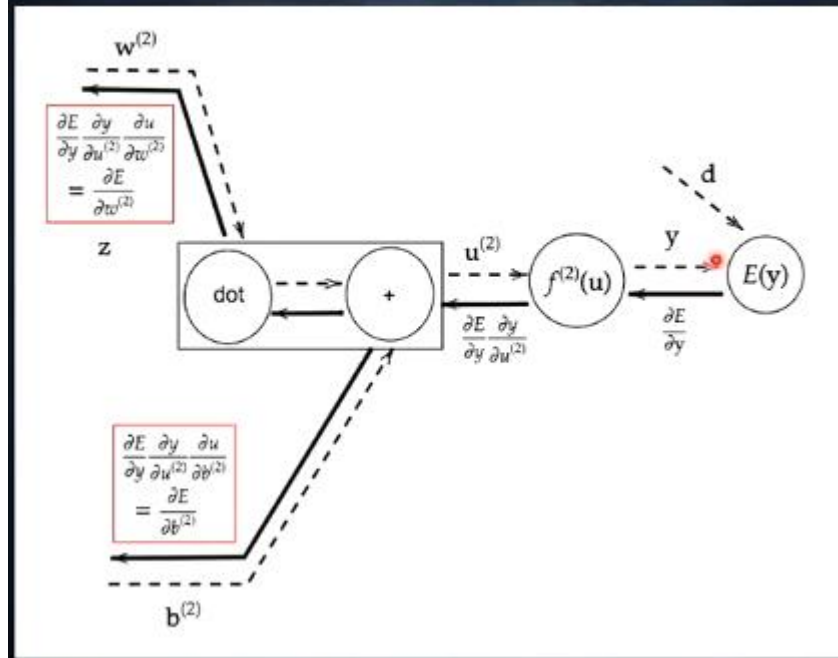
プログラムで微小な数値を生成し、疑似的に微分を計算する一般的な手法

m番目のwを微小変化させた状態で誤差Eを全てのwについて計算する。

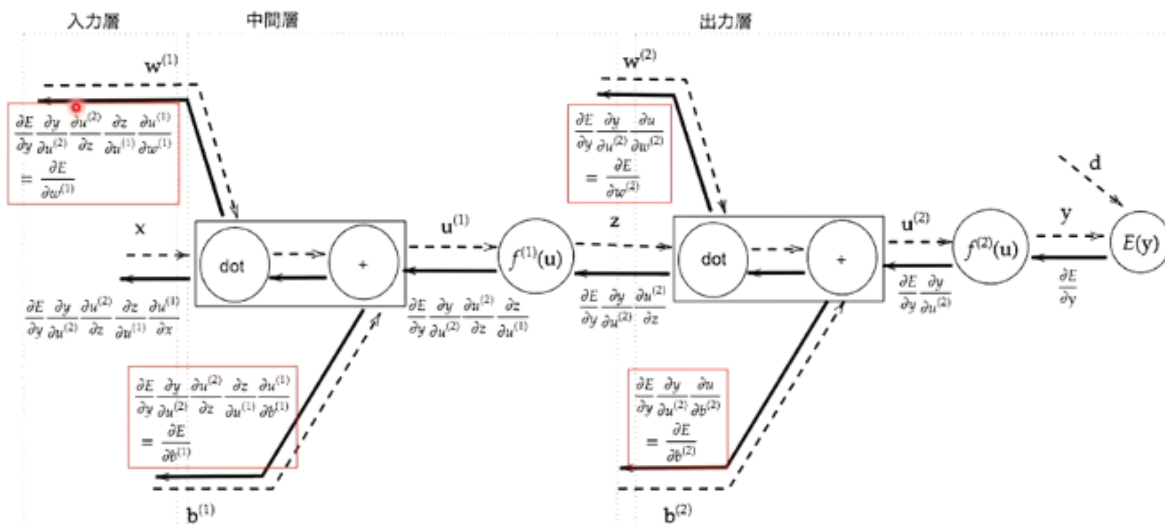
この方法は計算量が多くなってしまいます。



# 誤差勾配の計算 – 誤差逆伝播法



計算結果（＝誤差）から微分を逆算することで、不要な再帰的計算を避けて微分を算出できる。



## 1) 連鎖律

$$E(y) = \frac{1}{2} \sum_{j=1}^L (y_j - d_j)^2 = \frac{1}{2} \|y - d\|^2 : \text{誤差関数} = \text{二乗誤差関数}$$

$$y = u^{(L)} : \text{出力層の活性化関数} = \text{恒等写像}$$

$$u^{(l)} = w^{(l)} z^{(l-1)} + b^{(l)} : \text{総入力} の \text{計算}$$

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial w_{ji}^{(2)}}$$

1      2      3

①. E を y で微分する。

$$\frac{\partial E(y)}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} \|y - d\|^2 = y - d$$

② y(u) を u で微分

$$\frac{\partial y(u)}{\partial u} = \frac{\partial u}{\partial u} = 1$$

$$\frac{\partial u(w)}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} (w^{(l)} z^{(l-1)} + b^{(l)}) = \frac{\partial}{\partial w_{ji}} \left( \begin{bmatrix} w_{11}z_1 + \dots + w_{1i}z_i + \dots + w_{1l}z_l \\ \vdots \\ w_{j1}z_1 + \dots + w_{ji}z_i + \dots + w_{jl}z_l \\ \vdots \\ w_{l1}z_1 + \dots + w_{li}z_i + \dots + w_{ll}z_l \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \\ \vdots \\ b_l \end{bmatrix} \right) = \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix}$$

最終的に z だけになる。

結果)

$$\frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial w_{ji}^{(2)}} = (y - d) \cdot \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix} = (y_j - d_j) z_i$$

問題

1)  $\delta E / \delta y$

`delta2=functions.d_mean_squared_error(d, y)`

2)  $(\delta E / \delta y)(\delta y / \delta u)$

`delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)`

3)  $(\delta E / \delta y)(\delta y / \delta u)(\delta u / \delta w)$

# W1 の勾配

`grad['W1'] = np.dot(x.T, delta1)`

5 Section5\_誤差逆伝播法

5. 1 誤差勾配の計算

[誤差逆伝播法]

算出された誤差を、出力層側から順に微分し、前の層前の層へと伝播。

最小限の計算で各パラメータでの微分値を解析的に計算する方法。

