

ディープラーニング の発展と生成AI

津田塾大学
学芸学部 情報科学科
新田善久
nitta@tsuda.ac.jp
<https://nw.tsuda.ac.jp>

本日の内容

- 機械学習とAI
- ディープラーニング
- 生成AI
- 大規模言語モデル (ChatGPT)

機械学習とは (Machine Learning)

☞ 機械学習とはAIの分野のひとつであり、「データを扱う正しい処理方法をコンピュータ自身がデータから学習する」技術のことです

wikipediaより

経験からの学習により自動で改善するコンピュータアルゴリズム

「経験からの学習」→データ
(訓練データ、サンプルデータ)

「自動で改善」→間違いを減らす

人工知能とは (AI, Artificial Intelligence)

wikipediaより

言語の理解や推論、問題解決などの知的行動を人間に代わってコンピュータに行わせる技術



人間にしかできなかつた知的な行為を、どのような手段とどのようなデータを準備すれば、それを機械的に実行できるか

AI ≒ 機械学習

機械学習は高校数学で理解できます

微分  数II

関数 $f(x)$ の導関数は $f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$

合成関数の微分  数III

$y = f(u)$, $u = g(x)$ がともに微分可能なとき、

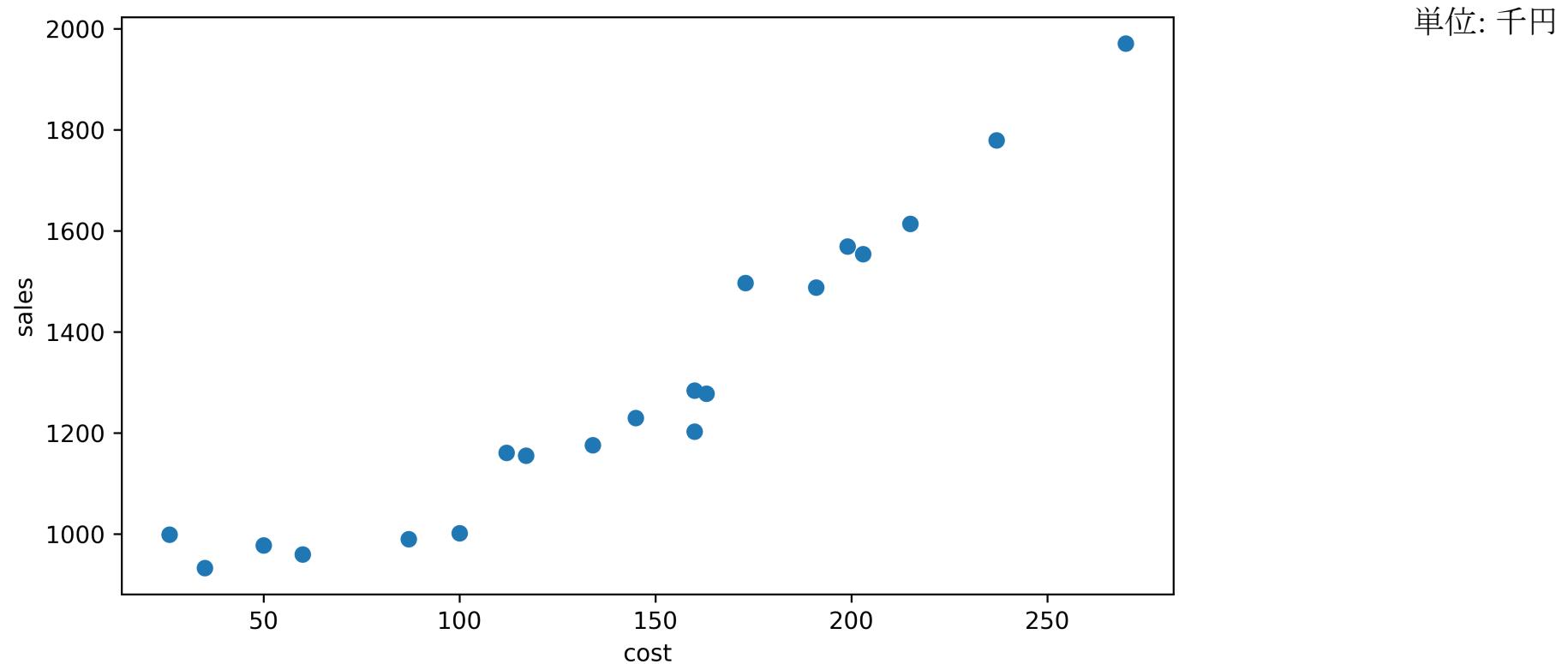
$$y = f(g(x)) \text{ も微分可能で } \frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

和の記号 Σ  数B

数列 $\{a_n\}$ の初項から第 n までの和 $a_1 + a_2 + \dots + a_n$ を $\sum_{k=1}^n a_k$ と書く

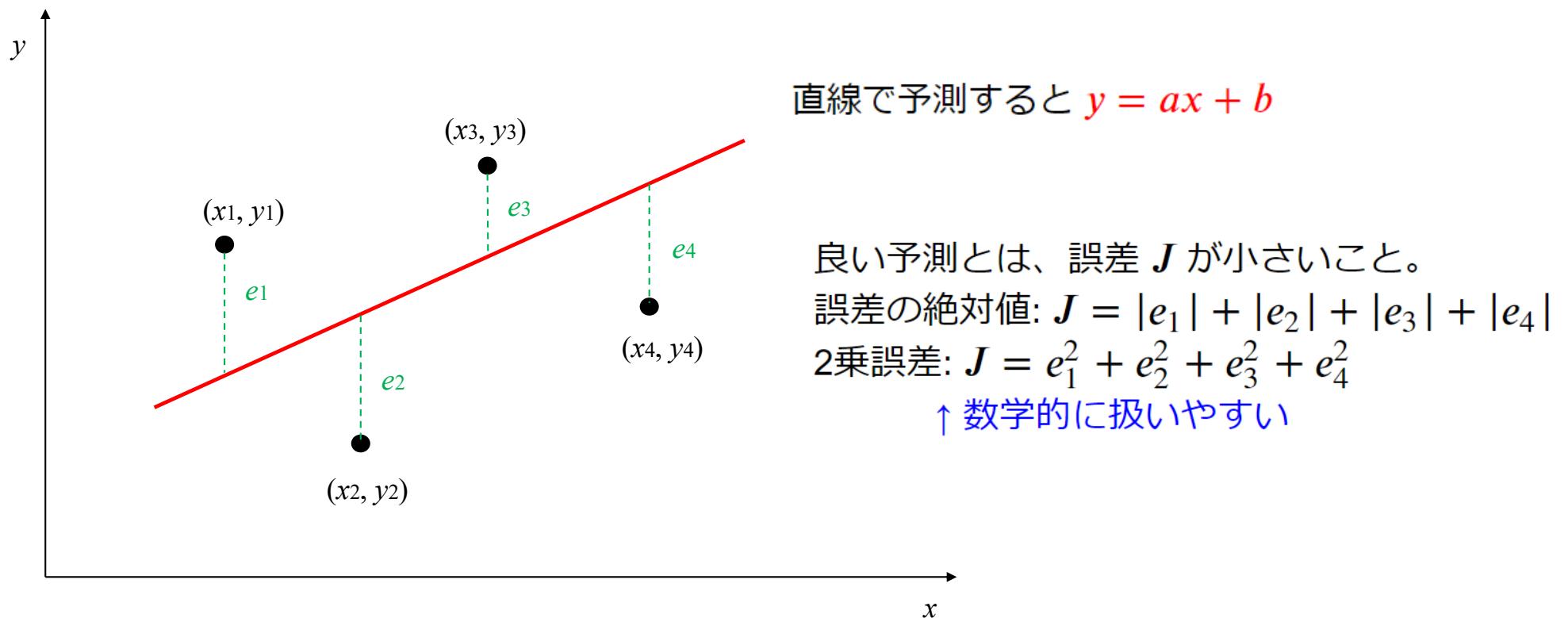
【例題1】ネット広告費と売上のデータから、広告費によって売り上げを予測したい。

cost	173	191	134	100	117	112	35	87	203	50	26	163	270	160	160	60	199
sales	1497	1488	1176	1002	1155	1161	933	990	1554	978	999	1278	1971	1203	1284	960	1569



【機械学習による解法1】直線で近似してみる

☞ 線形回帰 Linear Regression と言います



【機械学習による解法1】直線で近似してみる

誤差 J : 本来は、データの個数の影響を受けないように J をデータ数4で割りますが

$$\begin{aligned} J &= e_1^2 + e_2^2 + e_3^2 + e_4^2 \\ &= (y_1 - (ax_1 + b))^2 + (y_2 - (ax_2 + b))^2 + (y_3 - (ax_3 + b))^2 + (y_4 - (ax_4 + b))^2 \end{aligned}$$

誤差 J に対する a の責任 → a で微分すればわかる 数IIIっぽく d で書きます

$$\begin{aligned} \frac{d}{da} J &= (-2x_1)(y_1 - (ax_1 + b)) + (-2x_2)(y_2 - (ax_2 + b)) + (-2x_3)(y_3 - (ax_3 + b)) + (-2x_4)(y_4 - (ax_4 + b)) \\ &= -2(x_1(y_1 - (ax_1 + b)) + x_2(y_2 - (ax_2 + b)) + x_3(y_3 - (ax_3 + b)) + x_4(y_4 - (ax_4 + b))) \\ &= -2 \sum_{k=1}^4 x_k(y_k - (ax_k + b)) \end{aligned}$$

誤差 J に対する b の責任 → b で微分すればわかる

$$\begin{aligned} \frac{d}{db} J &= (-2)(y_1 - (ax_1 + b)) + (-2)(y_2 - (ax_2 + b)) + (-2)(y_3 - (ax_3 + b)) + (-2)(y_4 - (ax_4 + b)) \\ &= -2((y_1 - (ax_1 + b)) + (y_2 - (ax_2 + b)) + (y_3 - (ax_3 + b)) + (y_4 - (ax_4 + b))) \\ &= -2 \sum_{k=1}^4 (y_k - (ax_k + b)) \end{aligned}$$

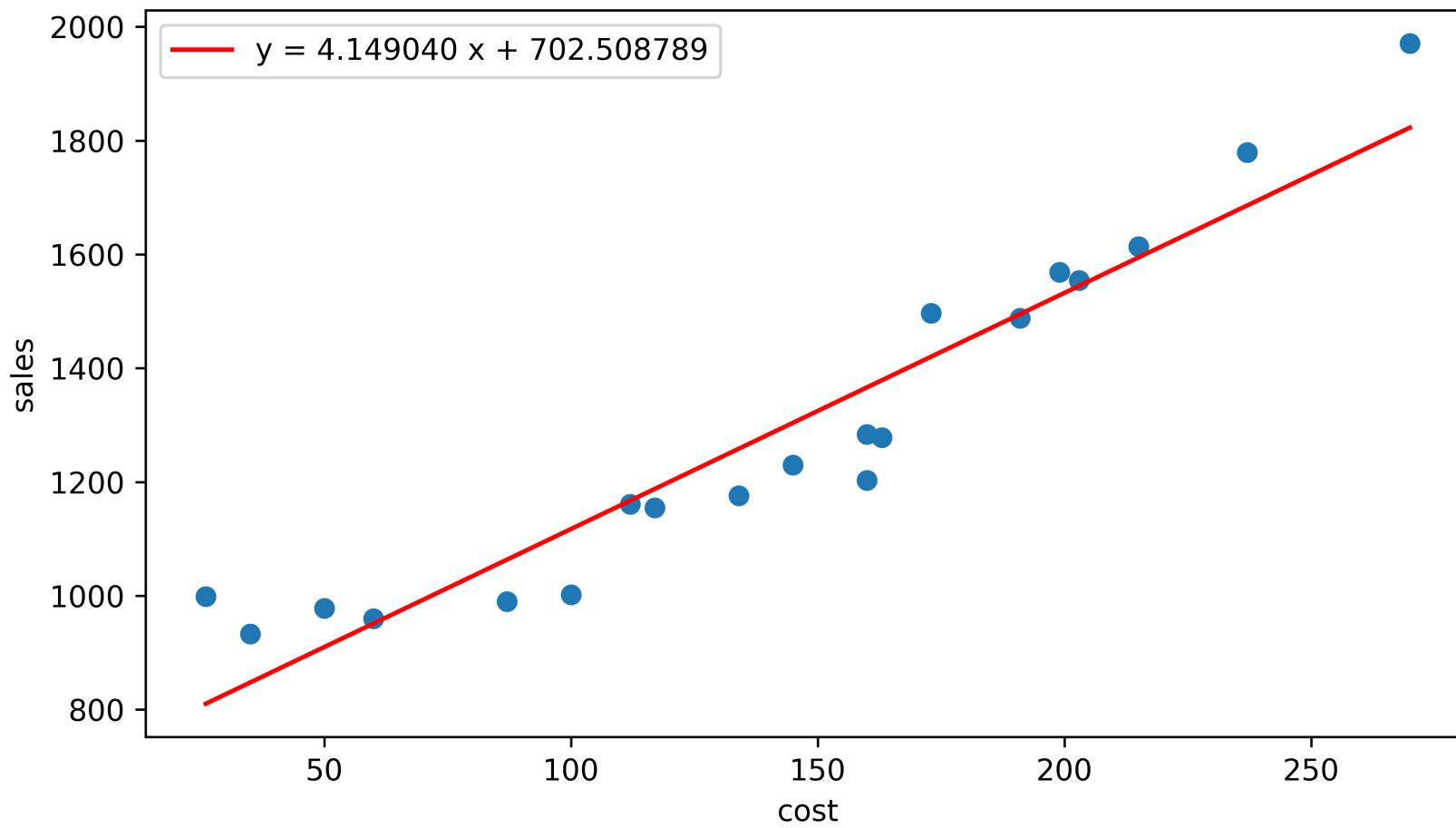
【機械学習による解法1】直線で近似してみる

適当な a, b の初期値から始めて、
繰り返し a, b を次のように変更すると
誤差 J が小さくなると期待できる

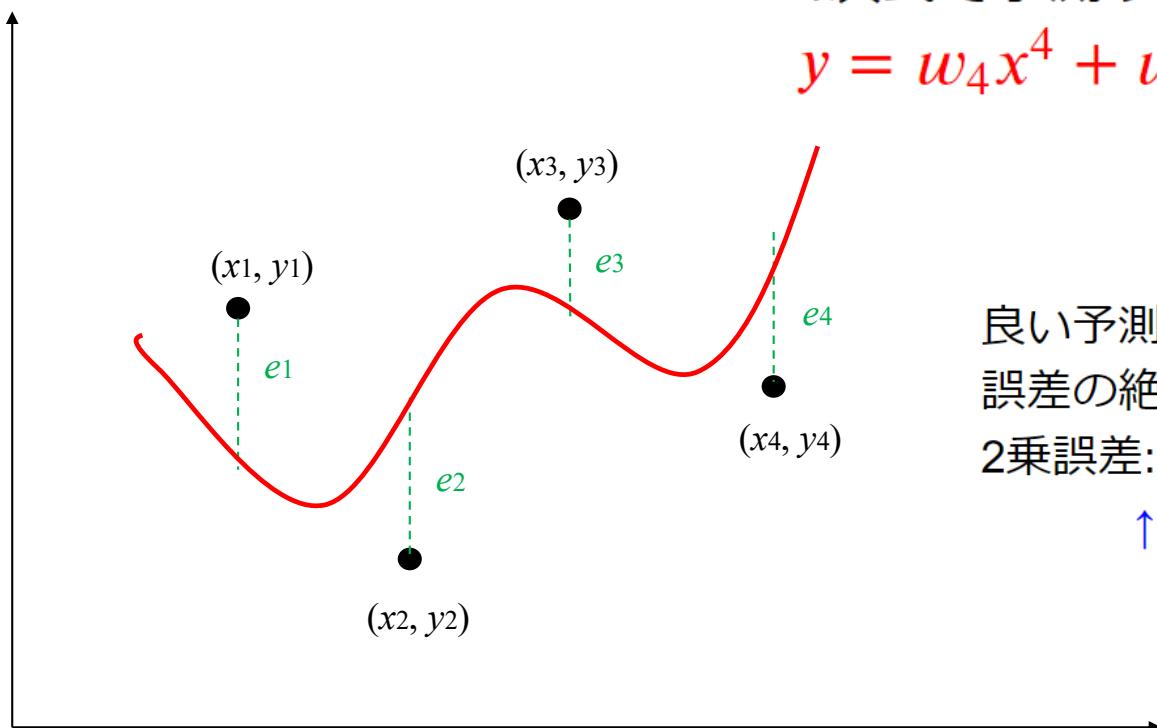
$$a := a - \text{小さい定数} \times \frac{d}{da} J$$

$$b := b - \text{小さい定数} \times \frac{d}{db} J$$

【機械学習による解法1】直線で近似してみる



【機械学習による解法2】4次式で近似してみる



4次式で予測すると

$$y = w_4x^4 + w_3x^3 + w_2x^2 + w_1x + b$$

良い予測とは、誤差 J が小さいこと。

誤差の絶対値: $J = |e_1| + |e_2| + |e_3| + |e_4|$

2乗誤差: $J = e_1^2 + e_2^2 + e_3^2 + e_4^2$

↑ 数学的に扱いやすい

【機械学習による解法2】4次式で近似してみる

誤差 J : 本来は、データの個数の影響を受けないように J をデータ数 4 で割りますが

$$\begin{aligned} J &= e_1^2 + e_2^2 + e_3^2 + e_4^2 \\ &= (y_1 - (w_4x_1^4 + w_3x_1^3 + w_2x_1^2 + w_1x_1 + b))^2 + \dots \\ &= \sum_{k=1}^4 (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))^2 \end{aligned}$$

誤差 J に対する各パラメータの責任 → そのパラメータで微分すればわかる 数 III っぽく d で書きます

$$\frac{d}{dw_4} J = -2 \sum_{k=1}^4 x_k^4 (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))$$

$$\frac{d}{dw_3} J = -2 \sum_{k=1}^4 x_k^3 (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))$$

$$\frac{d}{dw_2} J = -2 \sum_{k=1}^4 x_k^2 (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))$$

$$\frac{d}{dw_1} J = -2 \sum_{k=1}^4 x_k (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))$$

$$\frac{d}{db} J = -2 \sum_{k=1}^4 (y_k - (w_4x_k^4 + w_3x_k^3 + w_2x_k^2 + w_1x_k + b))$$

【機械学習による解法2】4次式で近似してみる

適当な w_4, w_3, w_2, w_1, b の初期値から始めて、

繰り返し次のように変更すると

誤差 J が小さくなると期待できる

$$w_4 := w_4 - \text{小さい定数} \times \frac{d}{dw_4} J$$

$$w_3 := w_3 - \text{小さい定数} \times \frac{d}{dw_3} J$$

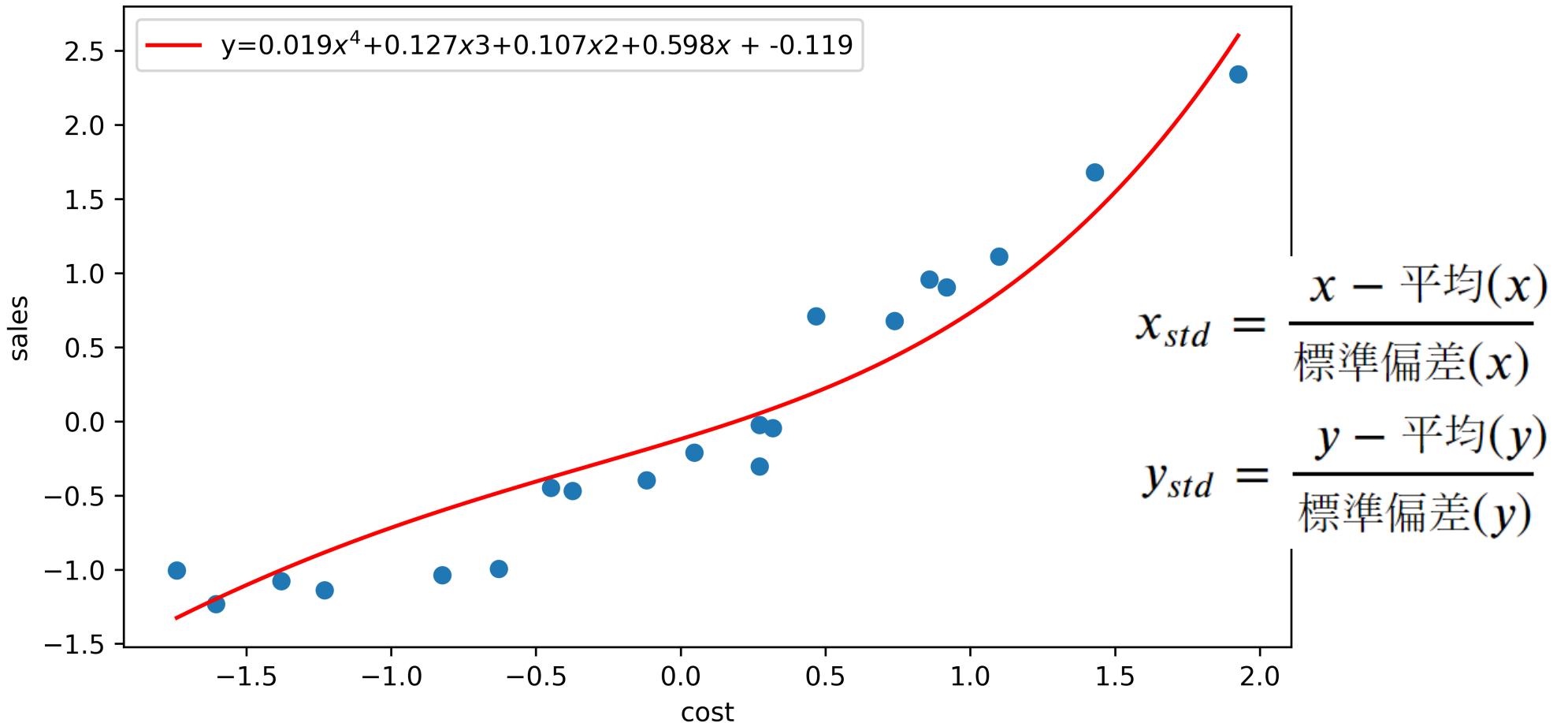
$$w_2 := w_2 - \text{小さい定数} \times \frac{d}{dw_2} J$$

$$w_1 := w_1 - \text{小さい定数} \times \frac{d}{dw_1} J$$

$$b := b - \text{小さい定数} \times \frac{d}{db} J$$

【機械学習による解法2】4次式で近似してみる

☞ 数値は標準化してから解いています

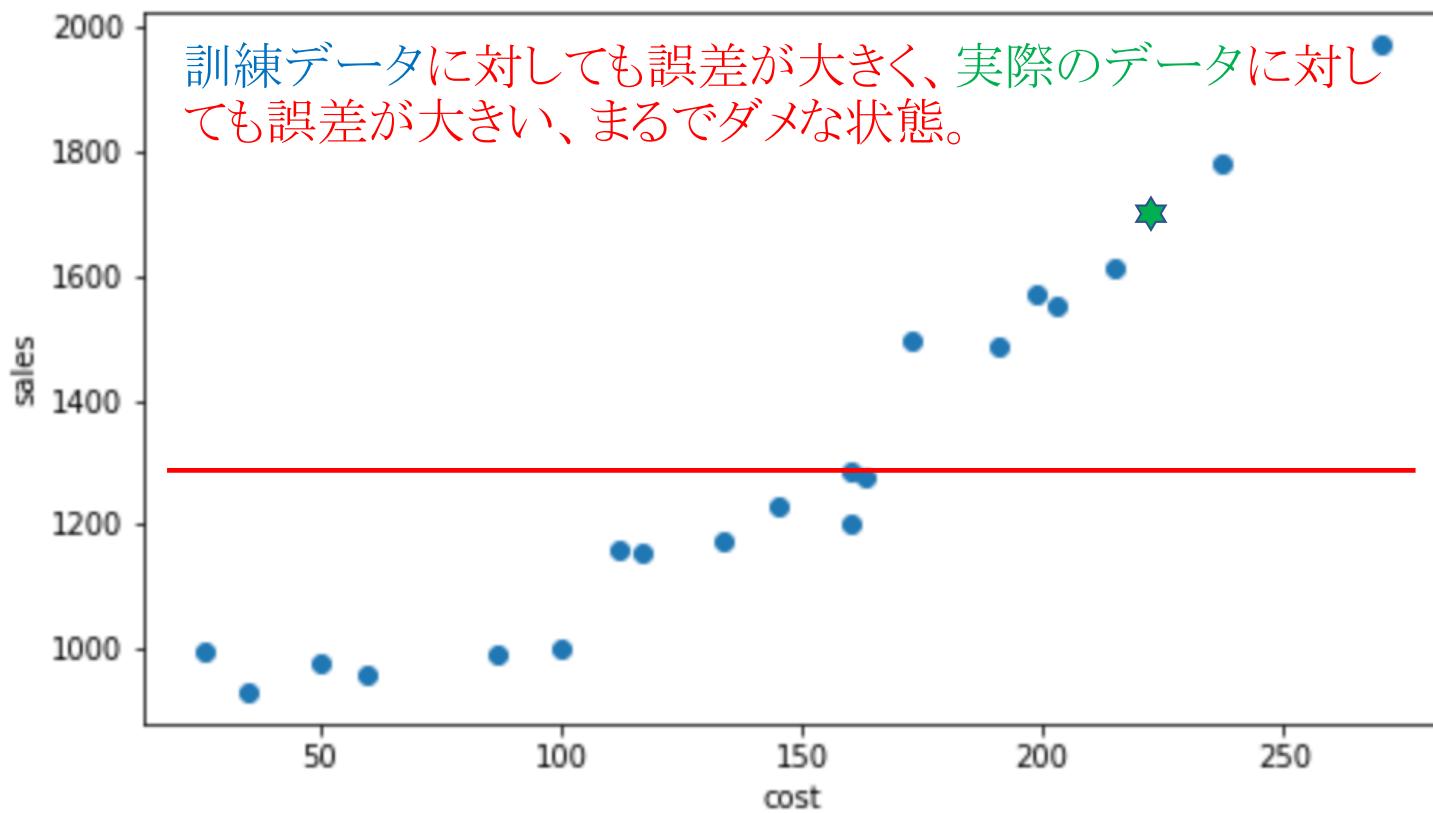


【考えてみよう】

【例題1】を近似するにはどんな式が適切なのだろうか？

単純すぎる式 → たとえば定数 → underfitting (過少適合、過少学習)

× 驄目

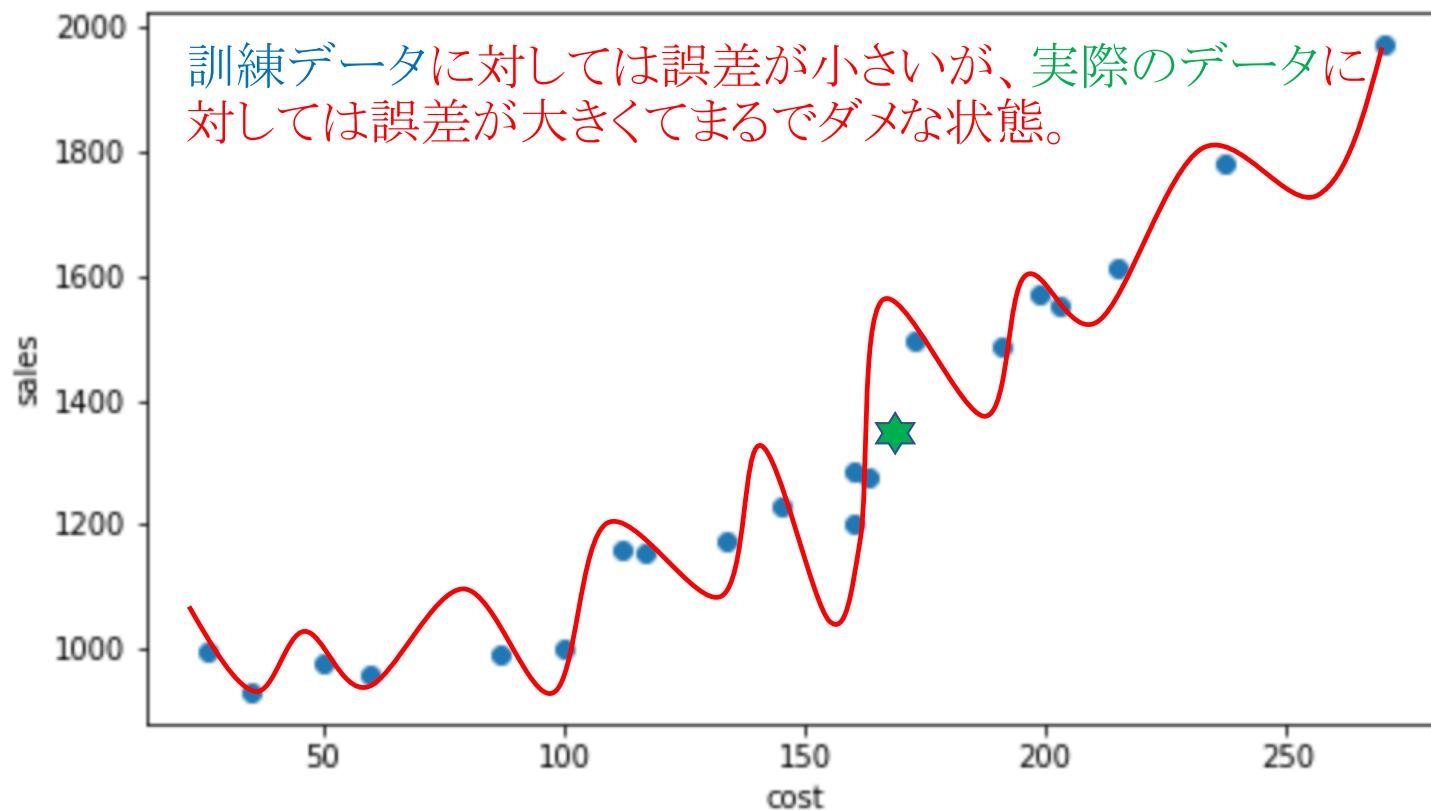


訓練データに対しても誤差が大きく、実際のデータに対しても誤差が大きい、まるでダメな状態。

【考えてみよう】 【例題1】を近似するにはどんな式が適切なのだろうか？

複雑すぎる式 → overfitting (過学習、過適合)

× 駄目



【新たな問題】 【例題1】を近似するにはどんな式が適切なのだろうか？

単純すぎる式 → 訓練データにそもそも適合できない (underfitting, 過少適合、過少学習)

複雑すぎる式 → 訓練データだけに適合してしまう (overfitting, 過学習、過適合)

× 驄目

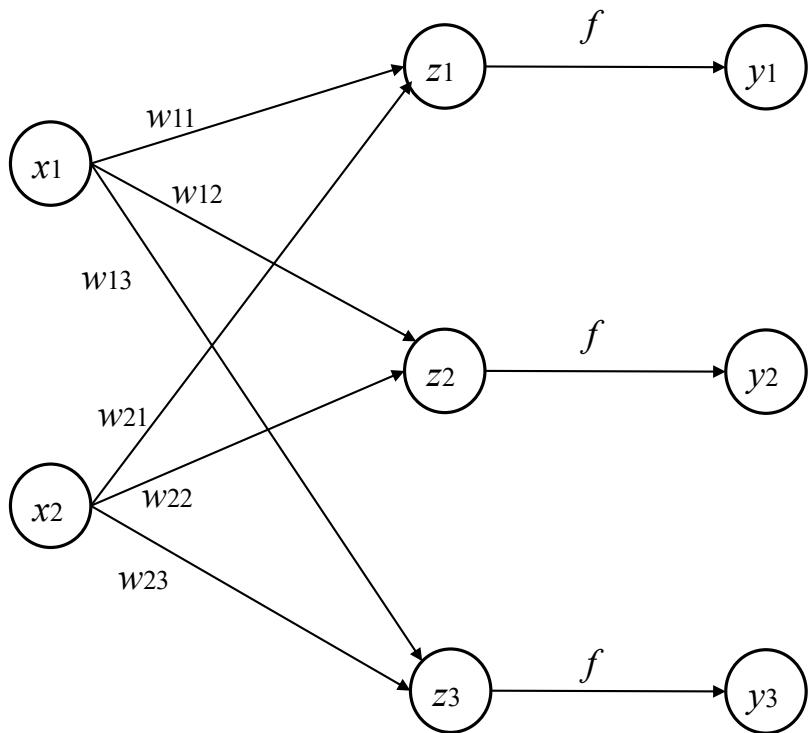


問題に応じた適切な程度に複雑な式 ○ 必要



コンピュータが勝手に見つけてくれたらいいいな

ニューラル・ネットワーク・モデル (Neural Network Model)



$$z_1 = w_{11}x_1 + w_{21}x_2$$

$$z_2 = w_{12}x_1 + w_{22}x_2$$

$$z_3 = w_{13}x_1 + w_{23}x_2$$

$$y_1 = f(z_1)$$

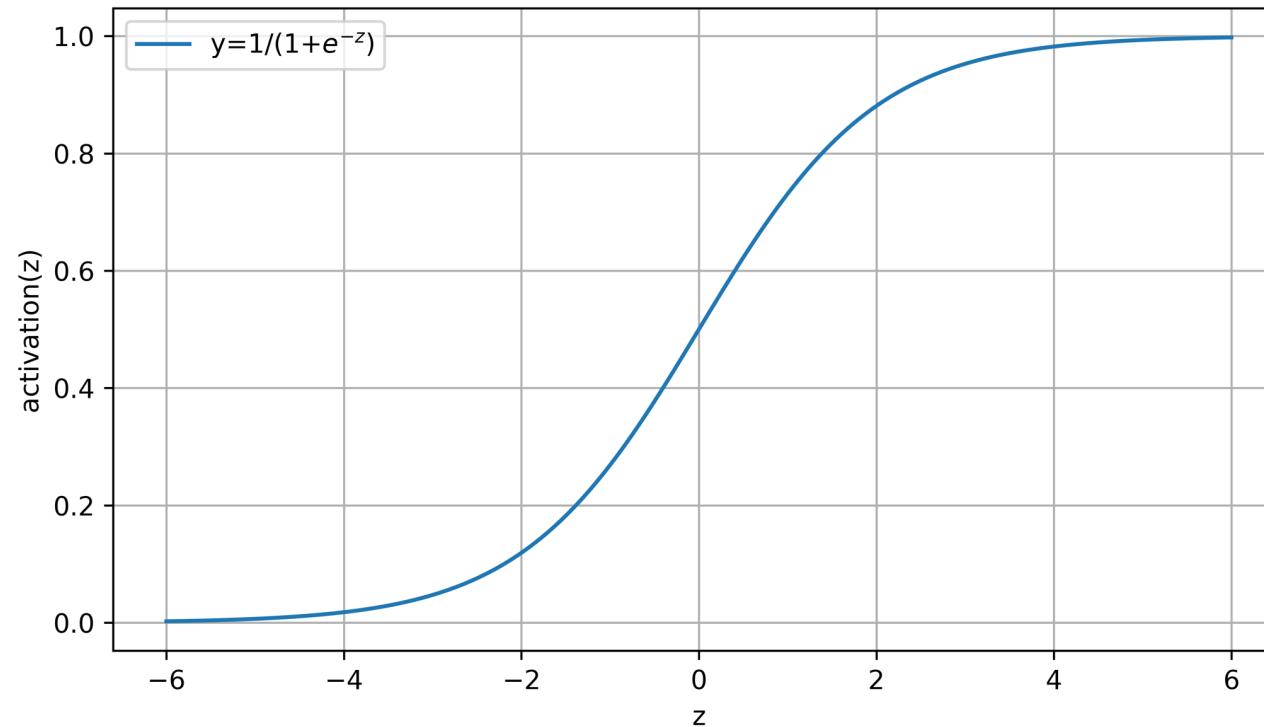
$$y_2 = f(z_2)$$

$$y_3 = f(z_3)$$

$f(z)$: 活性化関数
(activation function)

いろいろなアクティベーション関数: $y = f(z)$ 非線形関数

sigmoid: $y = \frac{1}{1 + e^{-z}}$



sigmoid関数: $a = \frac{1}{1 + e^{-z}}$

$$\frac{\partial a}{\partial z} = -\frac{-e^{-z}}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

また

$$(1 - a)a = \left(1 - \frac{1}{1 + e^{-z}}\right) \frac{1}{1 + e^{-z}} = \frac{1 + e^{-z} - 1}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

であるから

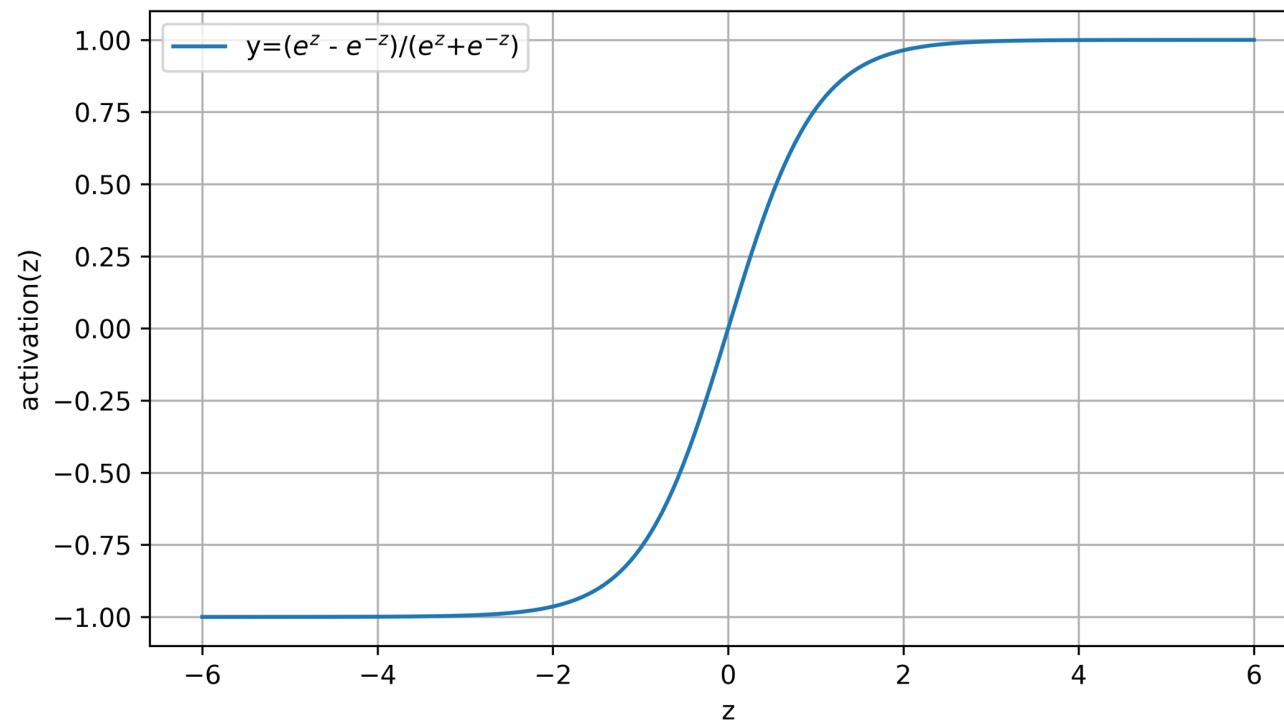
$$\frac{\partial a}{\partial z} = (1 - a)a$$

がいえる。したがって、

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \frac{\partial L}{\partial z} \cdot (1 - a)a$$

いろいろなアクティベーション関数: $y = f(z)$ 非線形関数

$$\tanh: y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



tanh関数: $a = \frac{e^z - e^{-z}}{e^z + e^{-z}} = (e^z - e^{-z})(e^z + e^{-z})^{-1}$

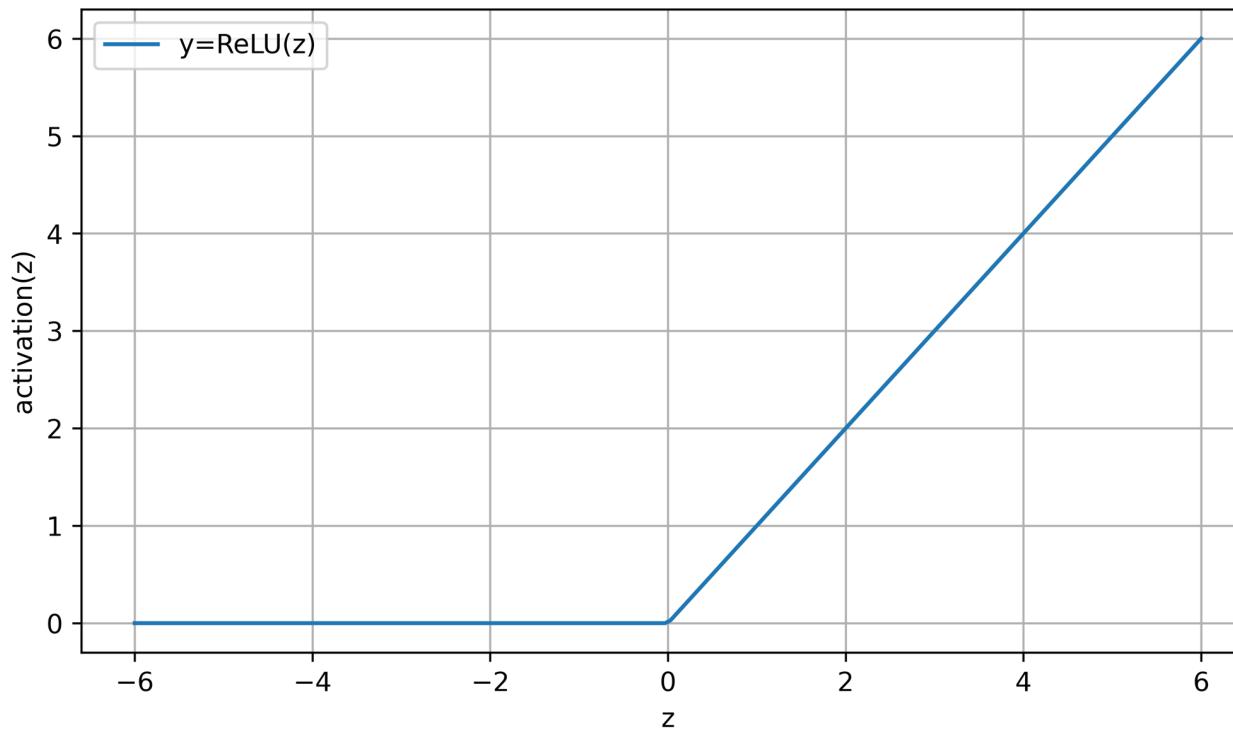
$$\begin{aligned}\frac{\partial a}{\partial z} &= (e^z + e^{-z})(e^z + e^{-z})^{-1} - (e^z - e^{-z})(e^z - e^{-z})(e^z + e^{-z})^{-2} \\ &= 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 \\ &= 1 - a^2\end{aligned}$$

なので

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \frac{\partial L}{\partial z} \cdot (1 - a^2)$$

いろいろなアクティベーション関数: $y = f(z)$ 非線形関数

$$\text{ReLU: } y = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



ReLU 関数 (Rectified Linear Unit)

$$a = \begin{cases} z & (z > 0) \\ 0 & (z \leq 0) \end{cases}$$

なので

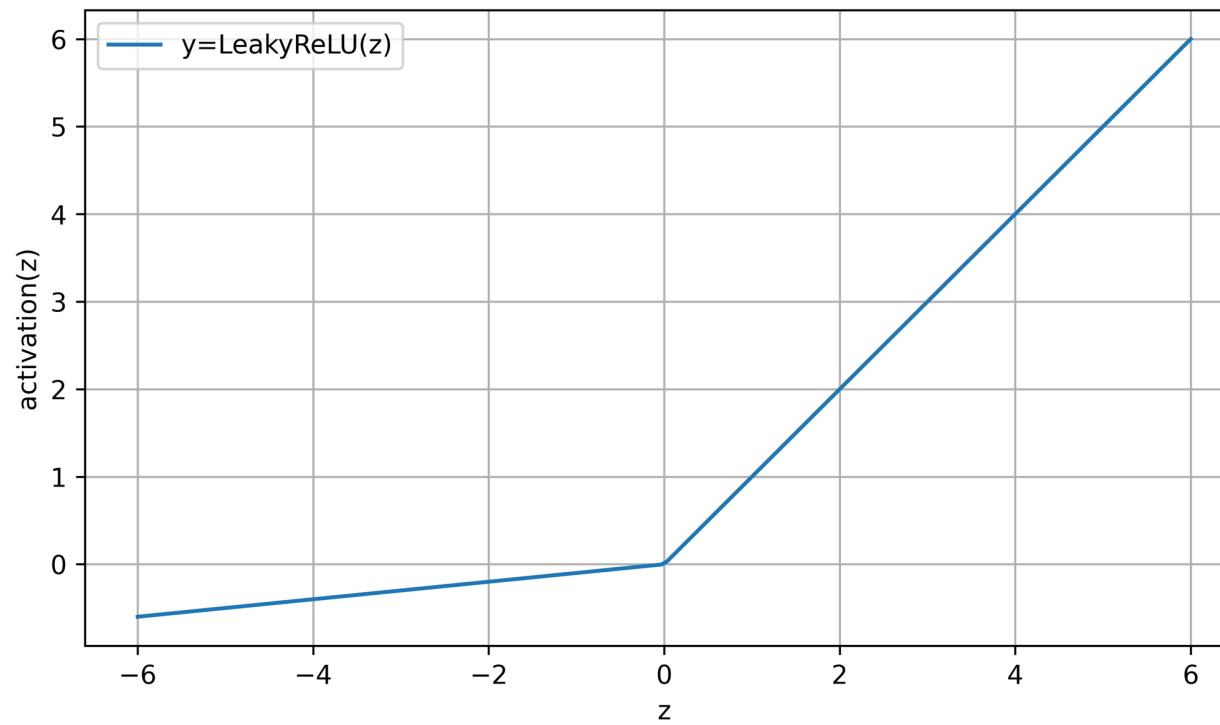
$$\frac{\partial a}{\partial z} = \begin{cases} 1 & (z > 0) \\ 0 & (z \leq 0) \end{cases}$$

したがって

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \begin{cases} \frac{\partial L}{\partial a} & (z > 0) \\ 0 & (z \leq 0) \end{cases}$$

いろいろなアクティベーション関数: $y = f(z)$ 非線形関数

LeakyReLU: $y = \begin{cases} x & \text{if } x > 0 \\ x \times 0.1 & \text{otherwise} \end{cases}$



leaky ReLU 関数 (leaky Rectified Linear Unit)

ReLUでは、training中に一部のneuronが0だけを出力するようになり、実質的に死んでしまう dying ReLU とよばれる問題が起きることがある。ReLU関数では、入力が負ならば gradient が0なのでneuron が復活する見込みはない。そこで改良されたのが leaky ReLU 関数である。

$$a = \begin{cases} z & (z > 0) \\ \alpha z & (z \leq 0) \end{cases} \quad \alpha \simeq 0.01$$

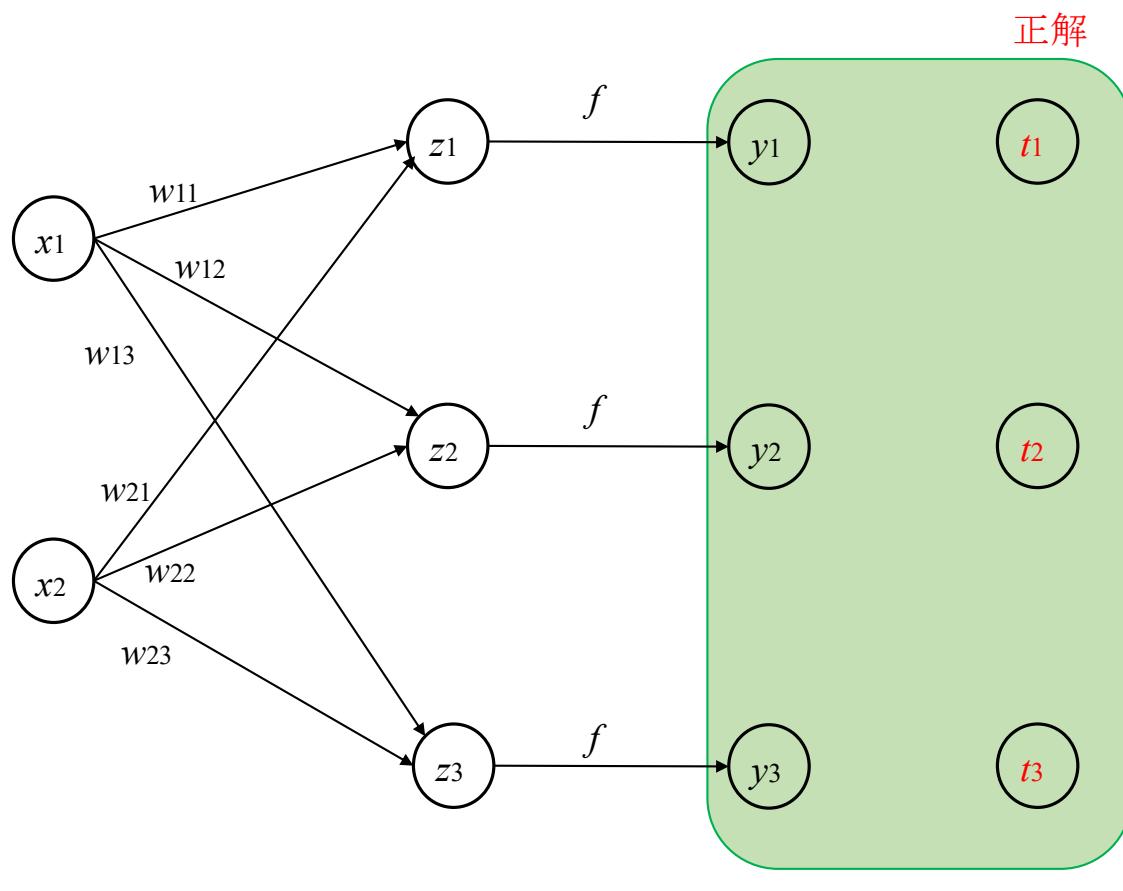
なので

$$\frac{\partial a}{\partial z} = \begin{cases} 1 & (z > 0) \\ \alpha & (z \leq 0) \end{cases}$$

したがって

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \begin{cases} \frac{\partial L}{\partial a} & (z > 0) \\ \alpha \frac{\partial L}{\partial a} & (z \leq 0) \end{cases}$$

ニューラル・ネットワーク・モデル (Neural Network Model)



$$z_1 = w_{11}x_1 + w_{21}x_2$$

$$z_2 = w_{12}x_1 + w_{22}x_2$$

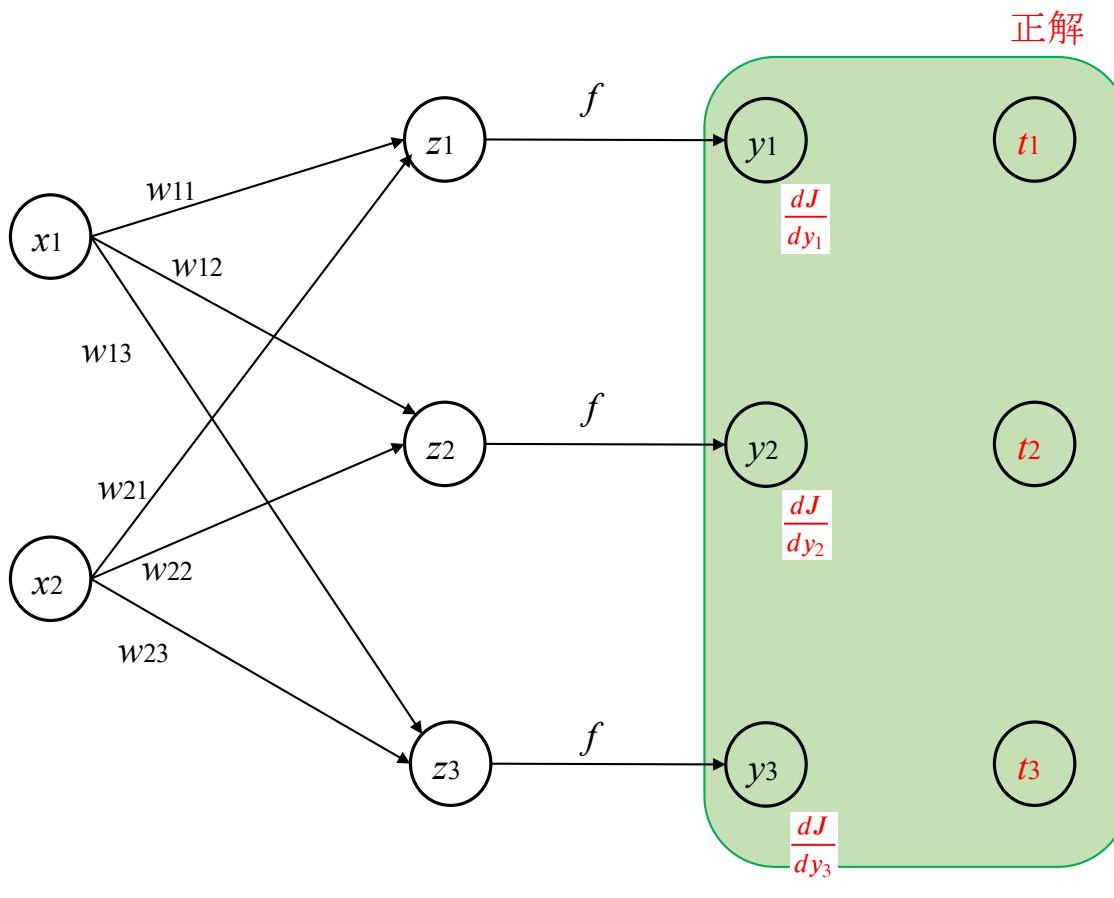
$$z_3 = w_{13}x_1 + w_{23}x_2$$

$$y_1 = f(z_1)$$

$$y_2 = f(z_2)$$

$$y_3 = f(z_3)$$

ニューラル・ネットワーク・モデル (Neural Network Model)



誤差Jを計算する

$$z_1 = w_{11}x_1 + w_{21}x_2$$

$$z_2 = w_{12}x_1 + w_{22}x_2$$

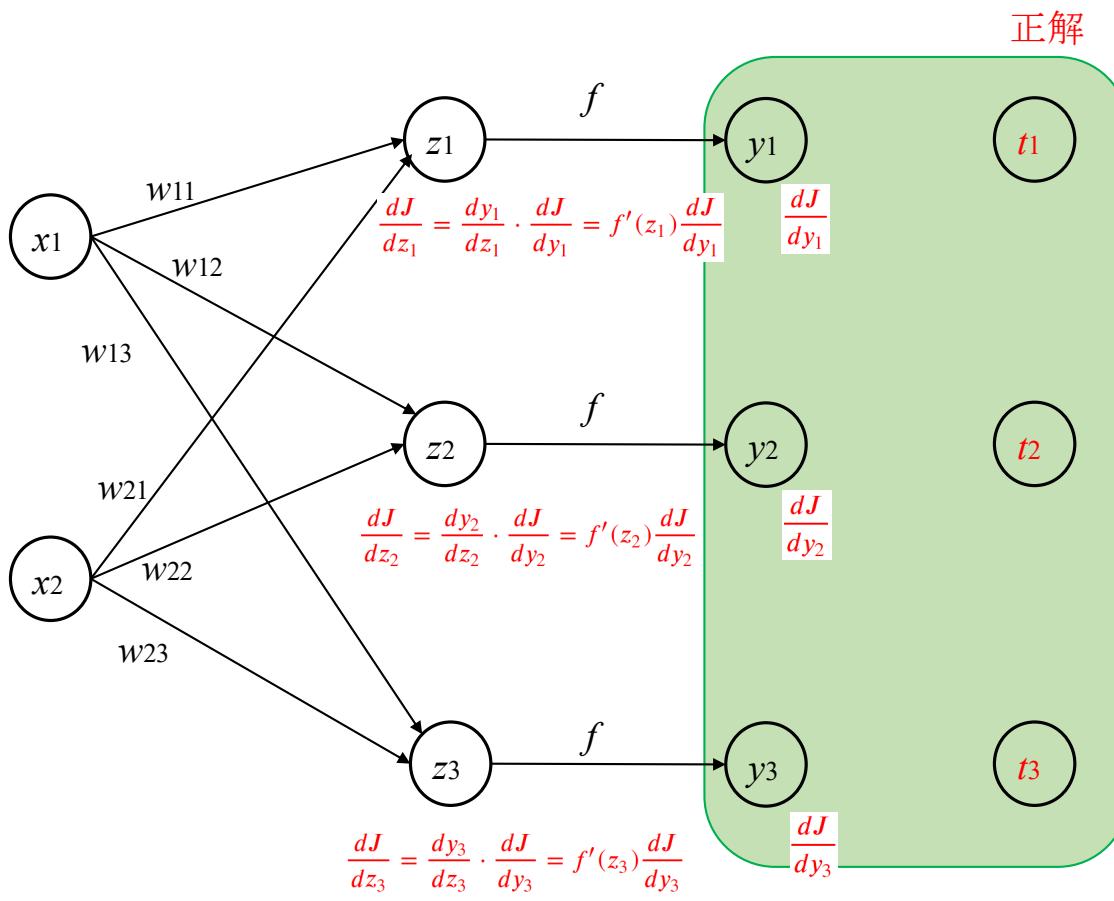
$$z_3 = w_{13}x_1 + w_{23}x_2$$

$$y_1 = f(z_1)$$

$$y_2 = f(z_2)$$

$$y_3 = f(z_3)$$

ニューラル・ネットワーク・モデル (Neural Network Model)



誤差Jを計算する

$$z_1 = w_{11}x_1 + w_{12}x_2$$

$$z_2 = w_{12}x_1 + w_{22}x_2$$

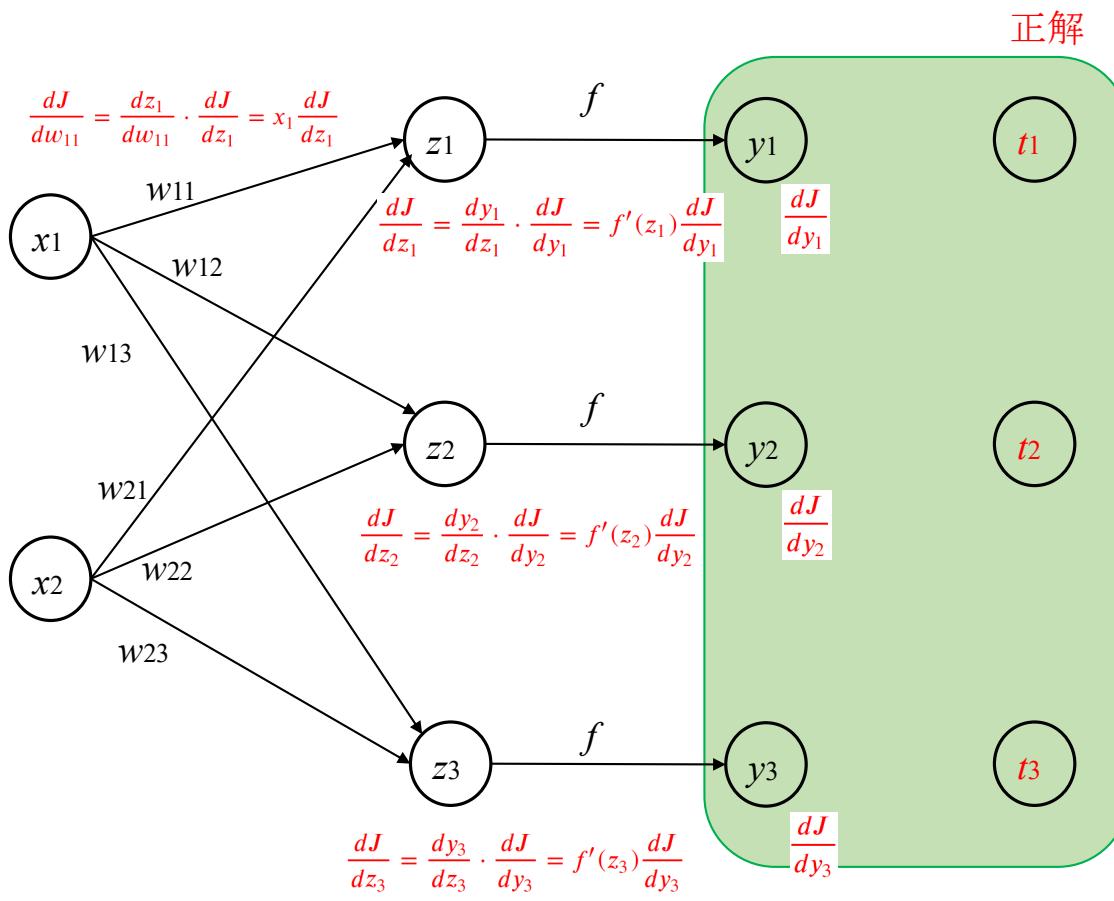
$$z_3 = w_{13}x_1 + w_{23}x_2$$

$$y_1 = f(z_1)$$

$$y_2 = f(z_2)$$

$$y_3 = f(z_3)$$

ニューラル・ネットワーク・モデル (Neural Network Model)



誤差Jを計算する

$$z_1 = w_{11}x_1 + w_{21}x_2$$

$$z_2 = w_{12}x_1 + w_{22}x_2$$

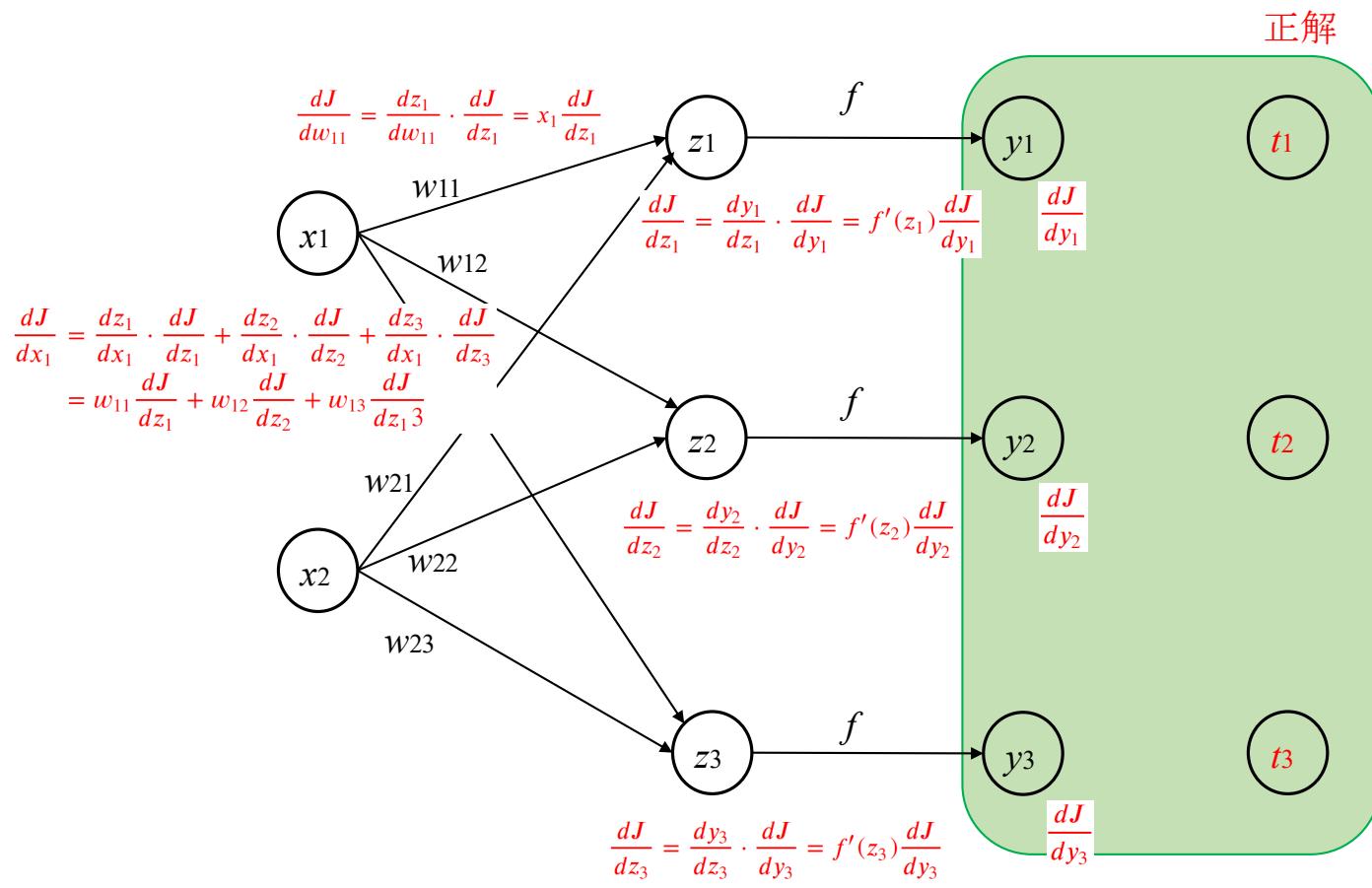
$$z_3 = w_{13}x_1 + w_{23}x_2$$

$$y_1 = f(z_1)$$

$$y_2 = f(z_2)$$

$$y_3 = f(z_3)$$

ニューラル・ネットワーク・モデル (Neural Network Model)



正解

誤差Jを計算する

$$z_1 = w_{11}x_1 + w_{12}x_2$$

$$z_2 = w_{13}x_1 + w_{21}x_2 + w_{22}x_2$$

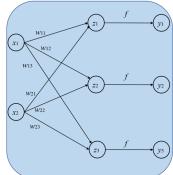
$$z_3 = w_{23}x_1 + w_{31}z_1 + w_{32}z_2$$

$$y_1 = f(z_1)$$

$$y_2 = f(z_2)$$

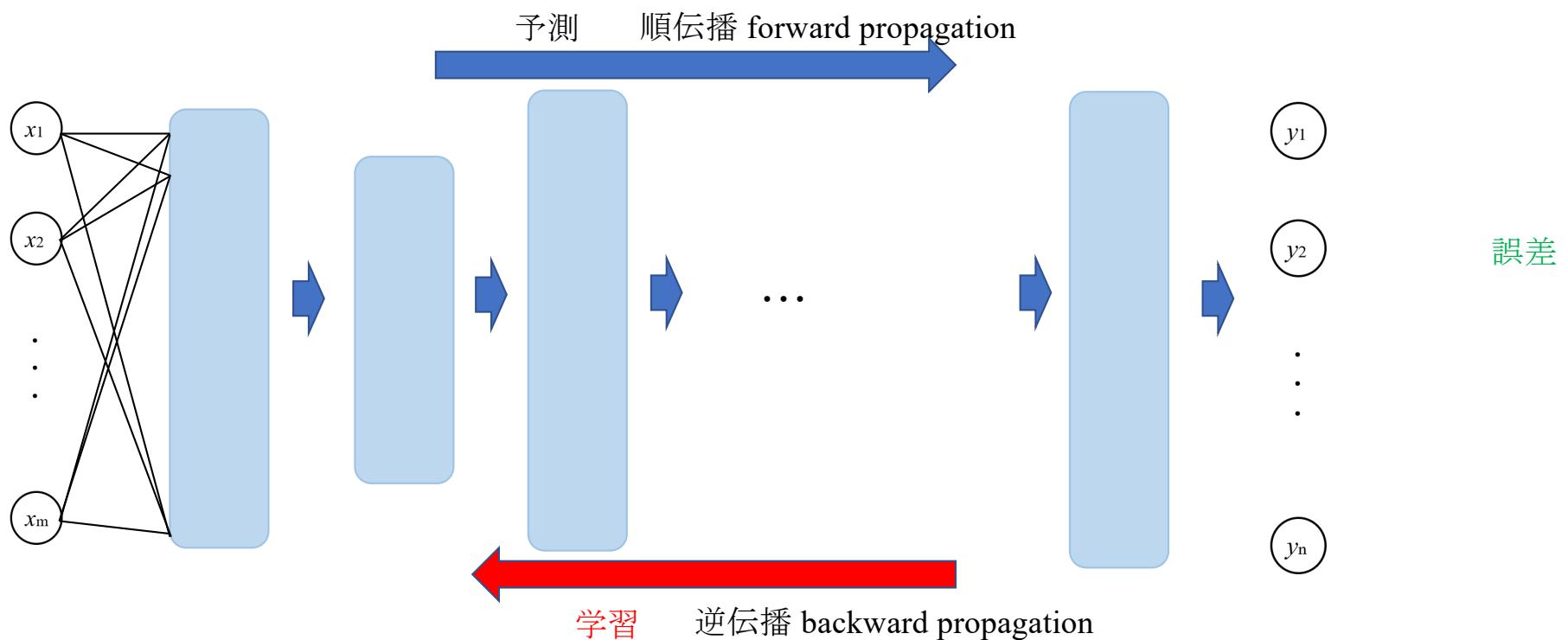
$$y_3 = f(z_3)$$

多層ニューラル・ネットワーク・モデル (Neural Network Model)



1層

ニューラルネットワークの層を何段も深くすることで、非常に複雑な関数を模倣することができる。



Deep Learningがどうして流行ったか？

- ニューラルネットワークの理論は1943年に提唱されていて、1957年にはパーセプトロンが開発されている。
- 逆伝播の論文は1986年 by Rumelhart, Hinton, Williams
- 何度か流行と冬の時代を繰り返す
- 3層のネットワーク(入力28x28画像→500→500→2000→10種類のラベル)が学習できたのは2006年(Hinton, 事前学習による)。
- コンピュータの発達(CPUの速度向上、メモリの増大)
- GPU コンピューティングの発展
- 2012年の物体認識コンテストILSVRC ImageNet Large Scale Visual Recognition Challenge)でぶっちぎりの優勝
- 学習方式の改善: Stochastic GD, RMSprop, Adam, ...
- モデルの改善: 疊み込み、リカレント(時間軸方向), Dropout, ...
- ツールの充実: TensorFlow, PyTorch, ...

【例題2】Fashion-MNISTのデータを画像分類してみよう

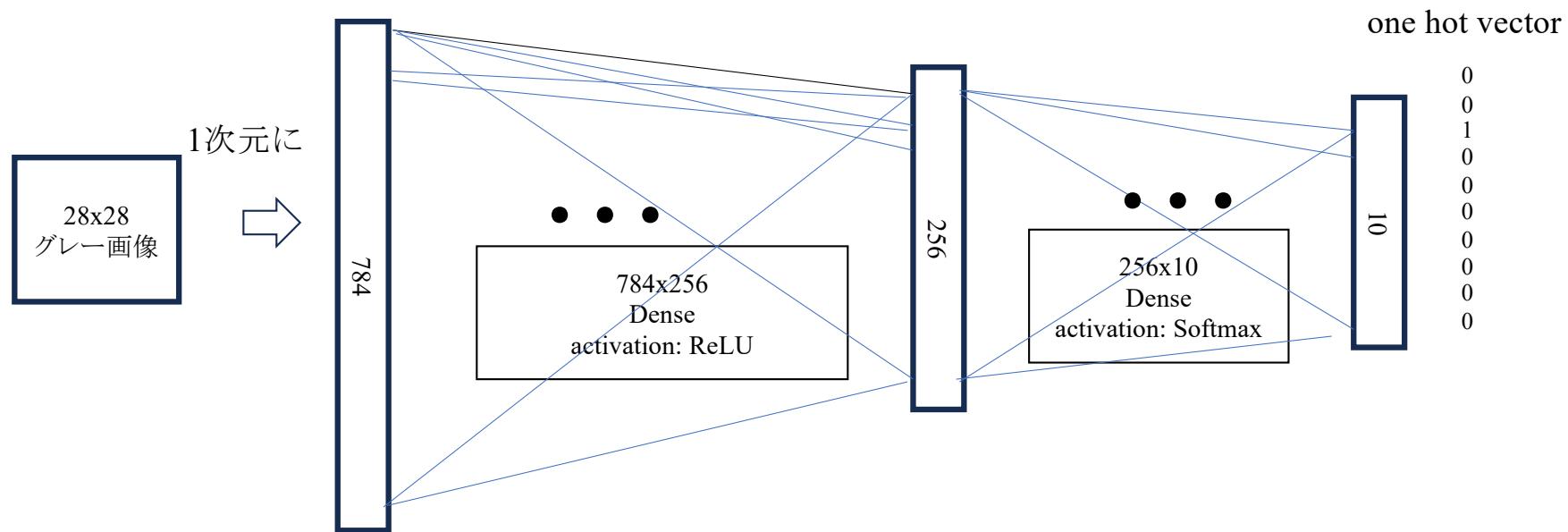
- 10種類のファッションアイテムのモノクロ画像
- 画像の解像度は28x28

ファッションアイテム	ラベル
Tシャツ／トップス	0
ズボン	1
プルオーバー	2
ドレス	3
コート	4
サンダル	5
シャツ	6
スニーカー	7
バッグ	8
アンクルブーツ	9



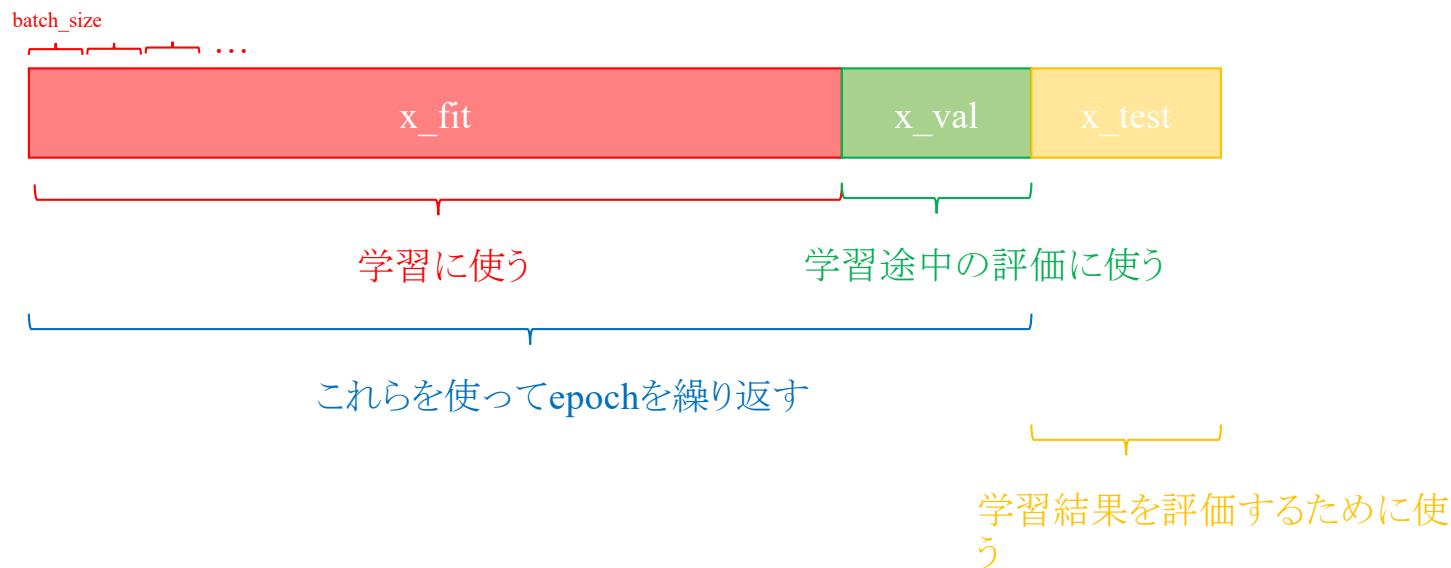
画像分類

- $28 \times 28 = 768$ 個の入力データから2層のニューラルネットワークで10種類に分類する。



【例題2】Fashion-MNISTのデータを画像分類してみよう

- ・訓練データは60000個 ➡ これを使って繰り返し学習する
 - ・学習には80%の48000個使う
 - ・検証用には20%の12000個使う
- ・テストデータは10000個 ➡ 最後にこれで学習結果を確認する



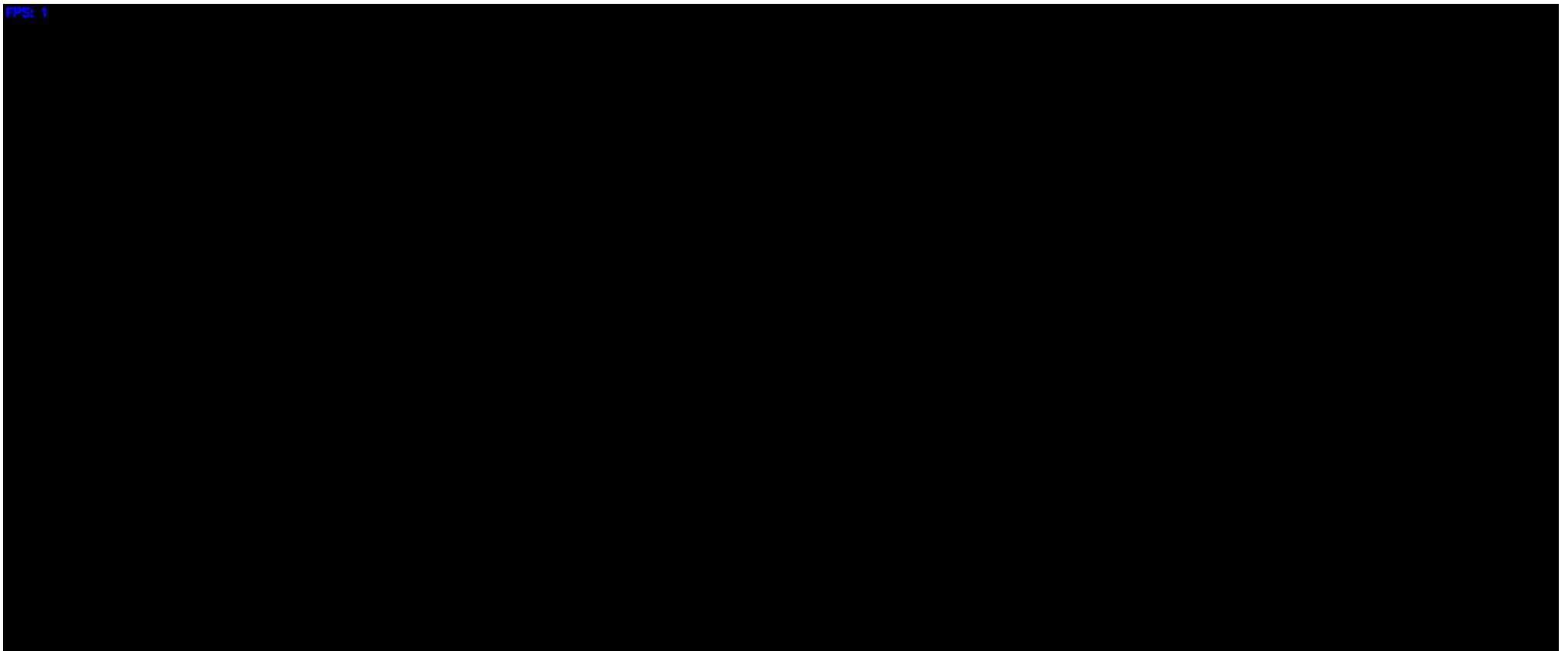
【例題2】Fashion-MNISTのデータを画像分類してみよう

- 実際に機械学習の様子をデモします。
- TensorFlow + Keras を使います。
- Google Colab <https://colab.research.google.com>
- 手順は以下のURLを参考にしてください。
 - https://nw.tsuda.ac.jp/lec/TensorFlow/book4/html/tf2_book4_ch04_03.html

【例題3】YOLO-3

- You Only Look Once
 - 一度に複数の物体検出を行う
 - 検出された物体の場所も返す
- サンプル画像
- デモ  リアルタイムのWeb Camの画像をYOLO-3で処理してみる

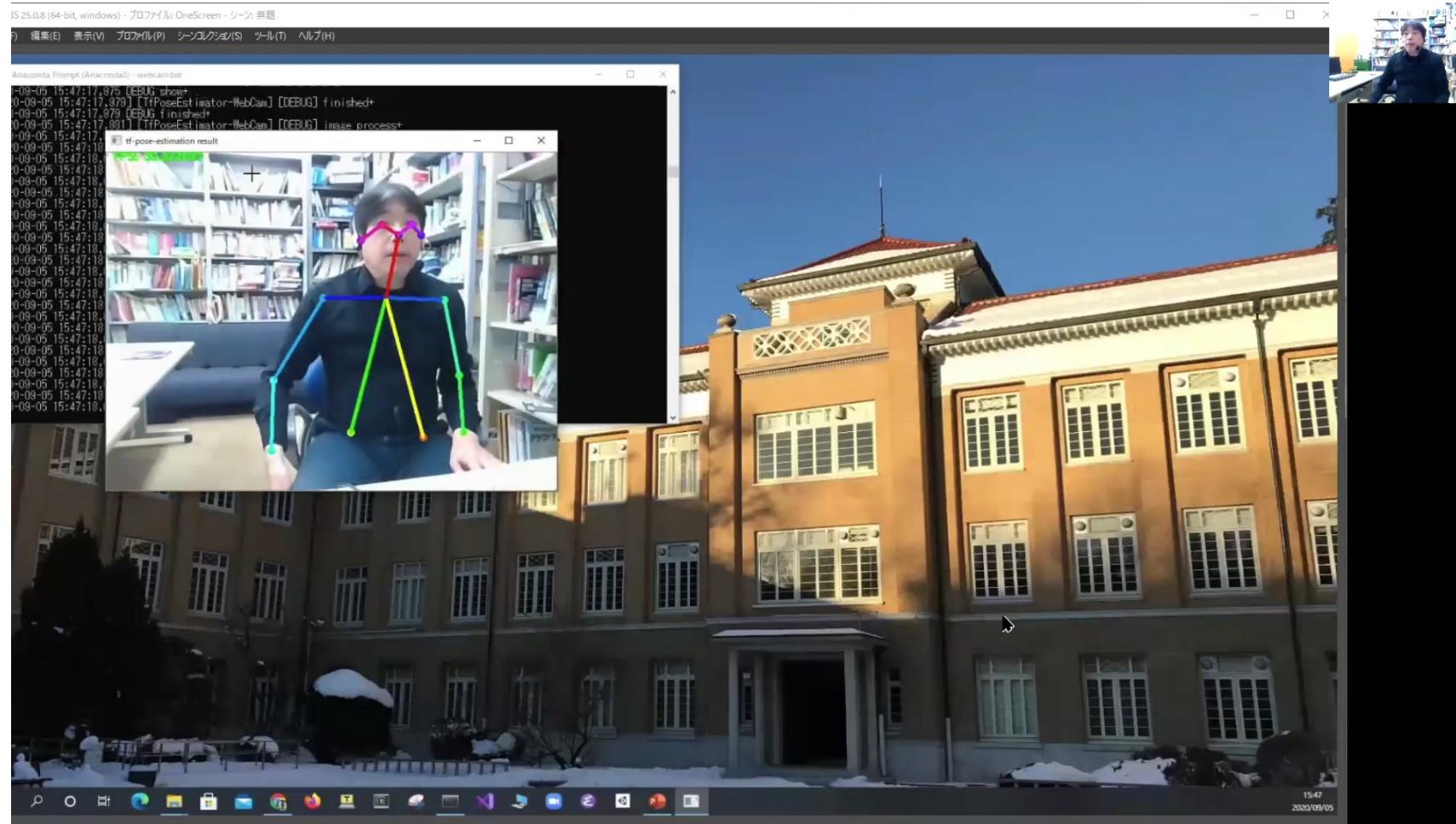
【例題3】YOLO-3



【例題4】OpenPose

- RGB画像から人間の骨格を検出する
- 何人でも、隠れた部分があってもOK
- CMU が開発
- デモ  リアルタイムのWeb Camの画像をOpenPoseで処理してみる

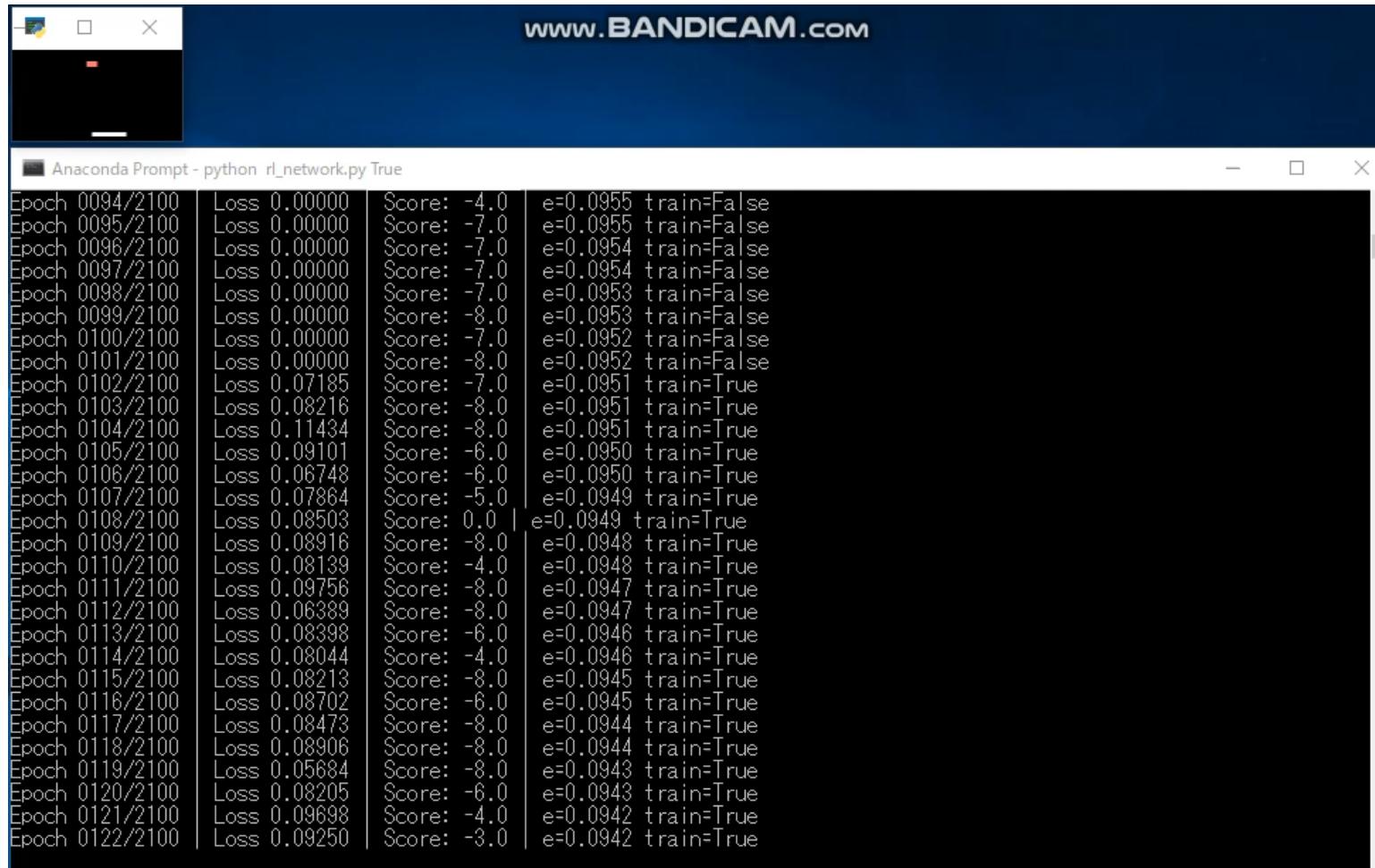
【例題4】OpenPose



【例題5】強化学習 (Reinforced Training)

- 正しい行動には報酬を与え、間違った行動にはペナルティ(負の)報酬を与える。
- ボールをキャッチするゲームを強化学習によって、訓練する

【例題5】強化学習 (Reinforced Training) ボールキャッチ epoch 0

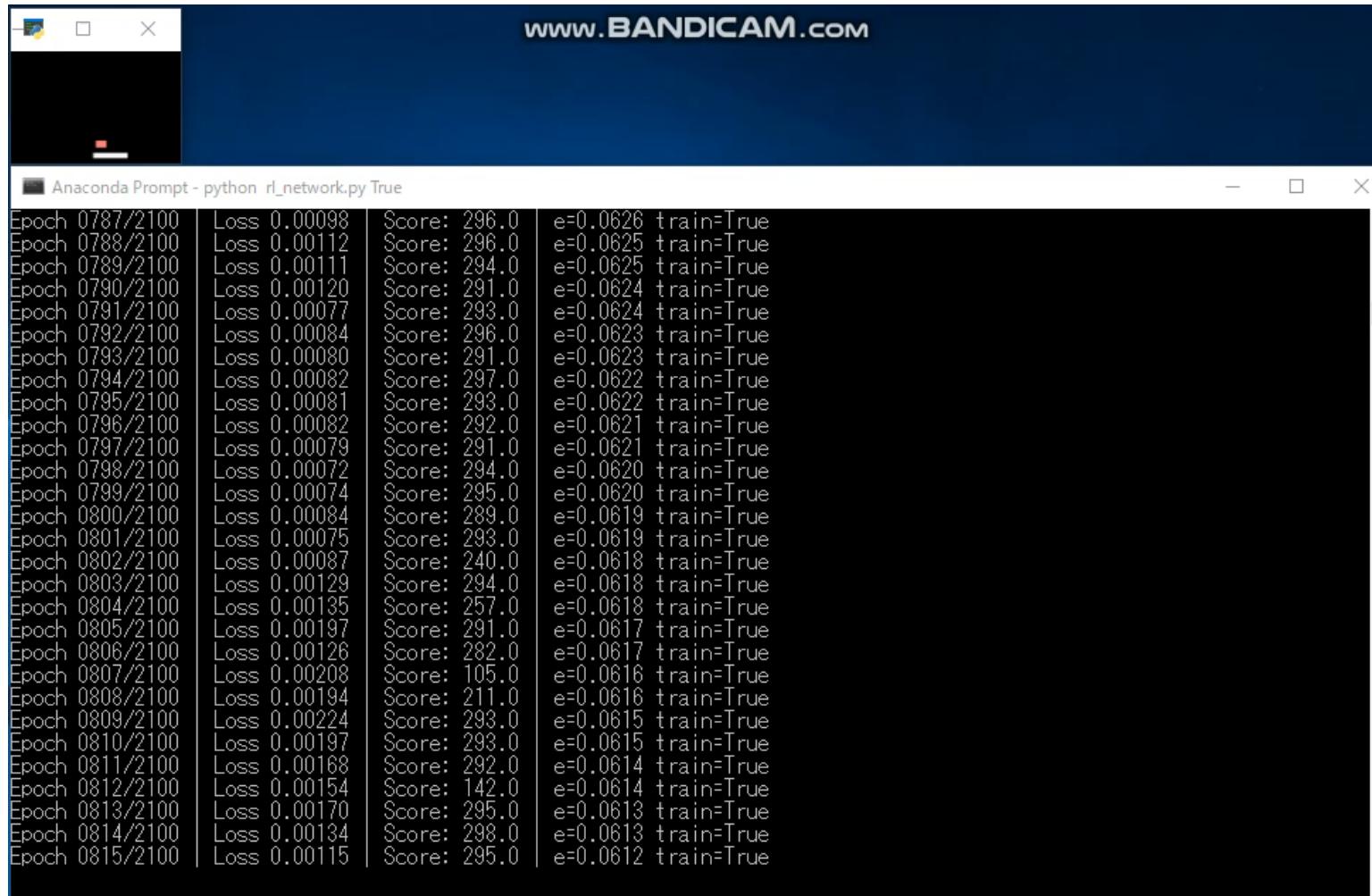


```
Epoch 0094/2100 | Loss 0.00000 | Score: -4.0 | e=0.0955 train=False
Epoch 0095/2100 | Loss 0.00000 | Score: -7.0 | e=0.0955 train=False
Epoch 0096/2100 | Loss 0.00000 | Score: -7.0 | e=0.0954 train=False
Epoch 0097/2100 | Loss 0.00000 | Score: -7.0 | e=0.0954 train=False
Epoch 0098/2100 | Loss 0.00000 | Score: -7.0 | e=0.0953 train=False
Epoch 0099/2100 | Loss 0.00000 | Score: -8.0 | e=0.0953 train=False
Epoch 0100/2100 | Loss 0.00000 | Score: -7.0 | e=0.0952 train=False
Epoch 0101/2100 | Loss 0.00000 | Score: -8.0 | e=0.0952 train=False
Epoch 0102/2100 | Loss 0.07185 | Score: -7.0 | e=0.0951 train=True
Epoch 0103/2100 | Loss 0.08216 | Score: -8.0 | e=0.0951 train=True
Epoch 0104/2100 | Loss 0.11434 | Score: -8.0 | e=0.0951 train=True
Epoch 0105/2100 | Loss 0.09101 | Score: -6.0 | e=0.0950 train=True
Epoch 0106/2100 | Loss 0.06748 | Score: -6.0 | e=0.0950 train=True
Epoch 0107/2100 | Loss 0.07864 | Score: -5.0 | e=0.0949 train=True
Epoch 0108/2100 | Loss 0.08503 | Score: 0.0 | e=0.0949 train=True
Epoch 0109/2100 | Loss 0.08916 | Score: -8.0 | e=0.0948 train=True
Epoch 0110/2100 | Loss 0.08139 | Score: -4.0 | e=0.0948 train=True
Epoch 0111/2100 | Loss 0.09756 | Score: -8.0 | e=0.0947 train=True
Epoch 0112/2100 | Loss 0.06389 | Score: -8.0 | e=0.0947 train=True
Epoch 0113/2100 | Loss 0.08398 | Score: -6.0 | e=0.0946 train=True
Epoch 0114/2100 | Loss 0.08044 | Score: -4.0 | e=0.0946 train=True
Epoch 0115/2100 | Loss 0.08213 | Score: -8.0 | e=0.0945 train=True
Epoch 0116/2100 | Loss 0.08702 | Score: -6.0 | e=0.0945 train=True
Epoch 0117/2100 | Loss 0.08473 | Score: -8.0 | e=0.0944 train=True
Epoch 0118/2100 | Loss 0.08906 | Score: -8.0 | e=0.0944 train=True
Epoch 0119/2100 | Loss 0.05684 | Score: -8.0 | e=0.0943 train=True
Epoch 0120/2100 | Loss 0.08205 | Score: -6.0 | e=0.0943 train=True
Epoch 0121/2100 | Loss 0.09698 | Score: -4.0 | e=0.0942 train=True
Epoch 0122/2100 | Loss 0.09250 | Score: -3.0 | e=0.0942 train=True
```

【例題5】強化学習 (Reinforced Training) ボールキャッチ epoch 11

```
Epoch 0462/2100 | Loss 0.04560 | Score: -3.0 | e=0.0780 train=True
Epoch 0463/2100 | Loss 0.04559 | Score: -5.0 | e=0.0780 train=True
Epoch 0464/2100 | Loss 0.04307 | Score: -2.0 | e=0.0779 train=True
Epoch 0465/2100 | Loss 0.03918 | Score: -3.0 | e=0.0779 train=True
Epoch 0466/2100 | Loss 0.04562 | Score: 2.0 | e=0.0778 train=True
Epoch 0467/2100 | Loss 0.04256 | Score: -5.0 | e=0.0778 train=True
Epoch 0468/2100 | Loss 0.03915 | Score: -7.0 | e=0.0777 train=True
Epoch 0469/2100 | Loss 0.03495 | Score: -6.0 | e=0.0777 train=True
Epoch 0470/2100 | Loss 0.04254 | Score: -2.0 | e=0.0776 train=True
Epoch 0471/2100 | Loss 0.04531 | Score: -3.0 | e=0.0776 train=True
Epoch 0472/2100 | Loss 0.04532 | Score: 0.0 | e=0.0775 train=True
Epoch 0473/2100 | Loss 0.04692 | Score: -2.0 | e=0.0775 train=True
Epoch 0474/2100 | Loss 0.03806 | Score: -4.0 | e=0.0775 train=True
Epoch 0475/2100 | Loss 0.04462 | Score: -3.0 | e=0.0774 train=True
Epoch 0476/2100 | Loss 0.04181 | Score: 5.0 | e=0.0774 train=True
Epoch 0477/2100 | Loss 0.04143 | Score: -6.0 | e=0.0773 train=True
Epoch 0478/2100 | Loss 0.04744 | Score: -5.0 | e=0.0773 train=True
Epoch 0479/2100 | Loss 0.04062 | Score: -3.0 | e=0.0772 train=True
Epoch 0480/2100 | Loss 0.04909 | Score: -5.0 | e=0.0772 train=True
Epoch 0481/2100 | Loss 0.04607 | Score: -2.0 | e=0.0771 train=True
Epoch 0482/2100 | Loss 0.05527 | Score: -7.0 | e=0.0771 train=True
Epoch 0483/2100 | Loss 0.04291 | Score: 1.0 | e=0.0770 train=True
Epoch 0484/2100 | Loss 0.05211 | Score: -7.0 | e=0.0770 train=True
Epoch 0485/2100 | Loss 0.04598 | Score: -2.0 | e=0.0769 train=True
Epoch 0486/2100 | Loss 0.03882 | Score: -5.0 | e=0.0769 train=True
Epoch 0487/2100 | Loss 0.04568 | Score: -4.0 | e=0.0768 train=True
Epoch 0488/2100 | Loss 0.04681 | Score: -3.0 | e=0.0768 train=True
Epoch 0489/2100 | Loss 0.04280 | Score: -6.0 | e=0.0767 train=True
Epoch 0490/2100 | Loss 0.04133 | Score: 1.0 | e=0.0767 train=True
```

【例題5】強化学習 (Reinforced Training) ボールキャッチ epoch 18



The screenshot shows a terminal window titled "Anaconda Prompt - python rl_network.py True". The window displays a log of training data for epoch 18, consisting of 2100 entries. Each entry contains four pieces of information: Epoch number, Loss value, Score value, and a parameter 'e' followed by its value. The 'train=True' part of the log entry is present in the original image but has been removed in the processed version.

Epoch	Loss	Score	e
0787/2100	0.00098	296.0	0.0626 train=True
0788/2100	0.00112	296.0	0.0625 train=True
0789/2100	0.00111	294.0	0.0625 train=True
0790/2100	0.00120	291.0	0.0624 train=True
0791/2100	0.00077	293.0	0.0624 train=True
0792/2100	0.00084	296.0	0.0623 train=True
0793/2100	0.00080	291.0	0.0623 train=True
0794/2100	0.00082	297.0	0.0622 train=True
0795/2100	0.00081	293.0	0.0622 train=True
0796/2100	0.00082	292.0	0.0621 train=True
0797/2100	0.00079	291.0	0.0621 train=True
0798/2100	0.00072	294.0	0.0620 train=True
0799/2100	0.00074	295.0	0.0620 train=True
0800/2100	0.00084	289.0	0.0619 train=True
0801/2100	0.00075	293.0	0.0619 train=True
0802/2100	0.00087	240.0	0.0618 train=True
0803/2100	0.00129	294.0	0.0618 train=True
0804/2100	0.00135	257.0	0.0618 train=True
0805/2100	0.00197	291.0	0.0617 train=True
0806/2100	0.00126	282.0	0.0617 train=True
0807/2100	0.00208	105.0	0.0616 train=True
0808/2100	0.00194	211.0	0.0616 train=True
0809/2100	0.00224	293.0	0.0615 train=True
0810/2100	0.00197	293.0	0.0615 train=True
0811/2100	0.00168	292.0	0.0614 train=True
0812/2100	0.00154	142.0	0.0614 train=True
0813/2100	0.00170	295.0	0.0613 train=True
0814/2100	0.00134	298.0	0.0613 train=True
0815/2100	0.00115	295.0	0.0612 train=True

【例題5】強化学習 (Reinforced Training) ボールキャッチ 学習終了



まとめ

- ・機械学習、AI、ディープラーニングについて簡単に説明しました。
- ・これらの領域は最近ではデータサイエンスと呼ばれていて、現在も急速に発展しています。社会における重要性も飛躍的に増大しています。
- ・GAN (Generative Adversarial Network), Q-Learning, ...
- ・基礎から理解すれば、基本となる技術は決して難しくはないことが分かっていただけたと思います。
- ・情報科学はいつも発展し続け、領域をどんどん広げていく学問です。
- ・情報科学の面白さを、少しでも理解していただけたら、うれしく思います。