aa-5. ニューラルネットワークの基本構造と画像分類

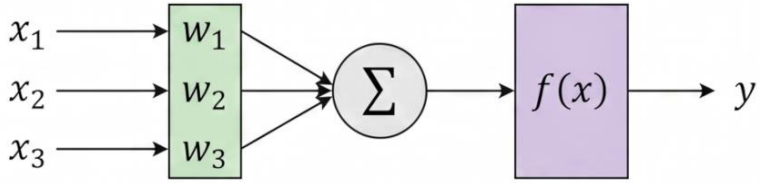
（人工知能）

金子邦彦

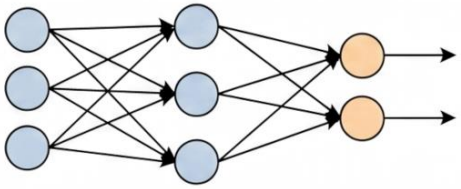
内容

1. ネットワークの基礎

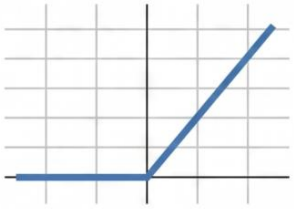
入力データ　重み　合計（和）　活性化関数　出力データ



入力層　中間層　出力層



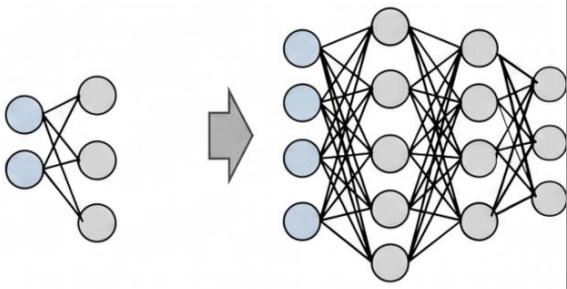
2. 活性化関数と深層学習



非線形変換

浅い
ネットワーク

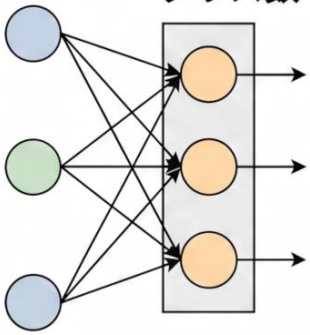
深いネットワーク
（ディープラーニング）



複雑なパターン認識が可能

3. 分類問題とソフトマックス関数

出力層のニューロン数
＝ クラス数



クラス1
(Tシャツ)

クラス2
(スニーカー)

クラス3
(バッグ)

ソフトマックス関数
（確率変換）

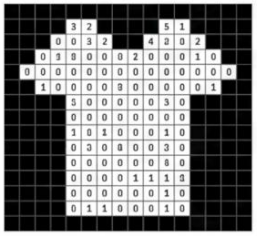
Tシャツ: 85%

スニーカー: 10%

バッグ: 5%

4. 画像データとFashion MNIST

画素（ピクセル）の並び



Fashion MNISTデータセット

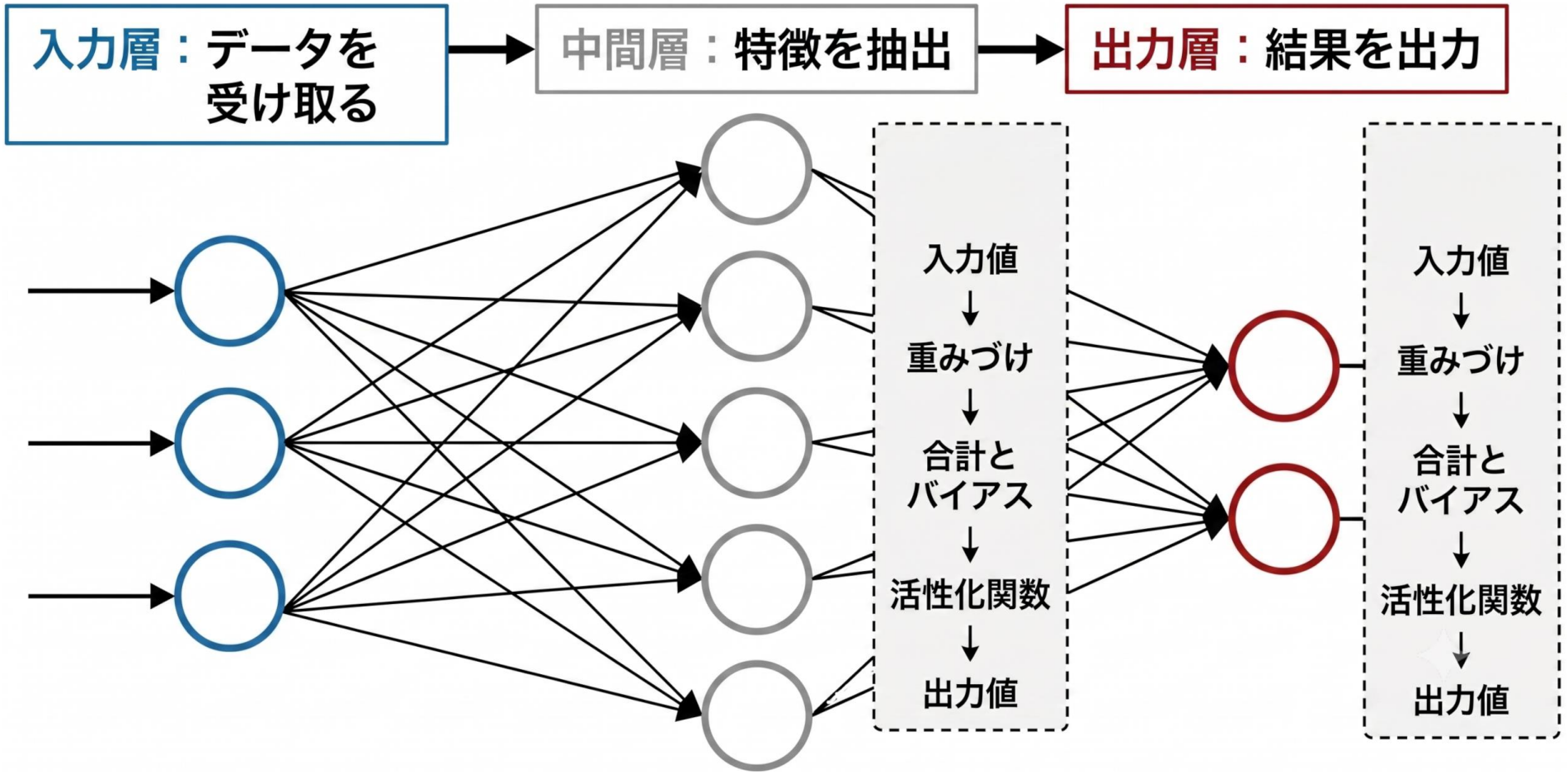
Tシャツ　スニーカー　バッグ　サンダル

- 画像サイズ: 28×28
- クラス数: 10クラス

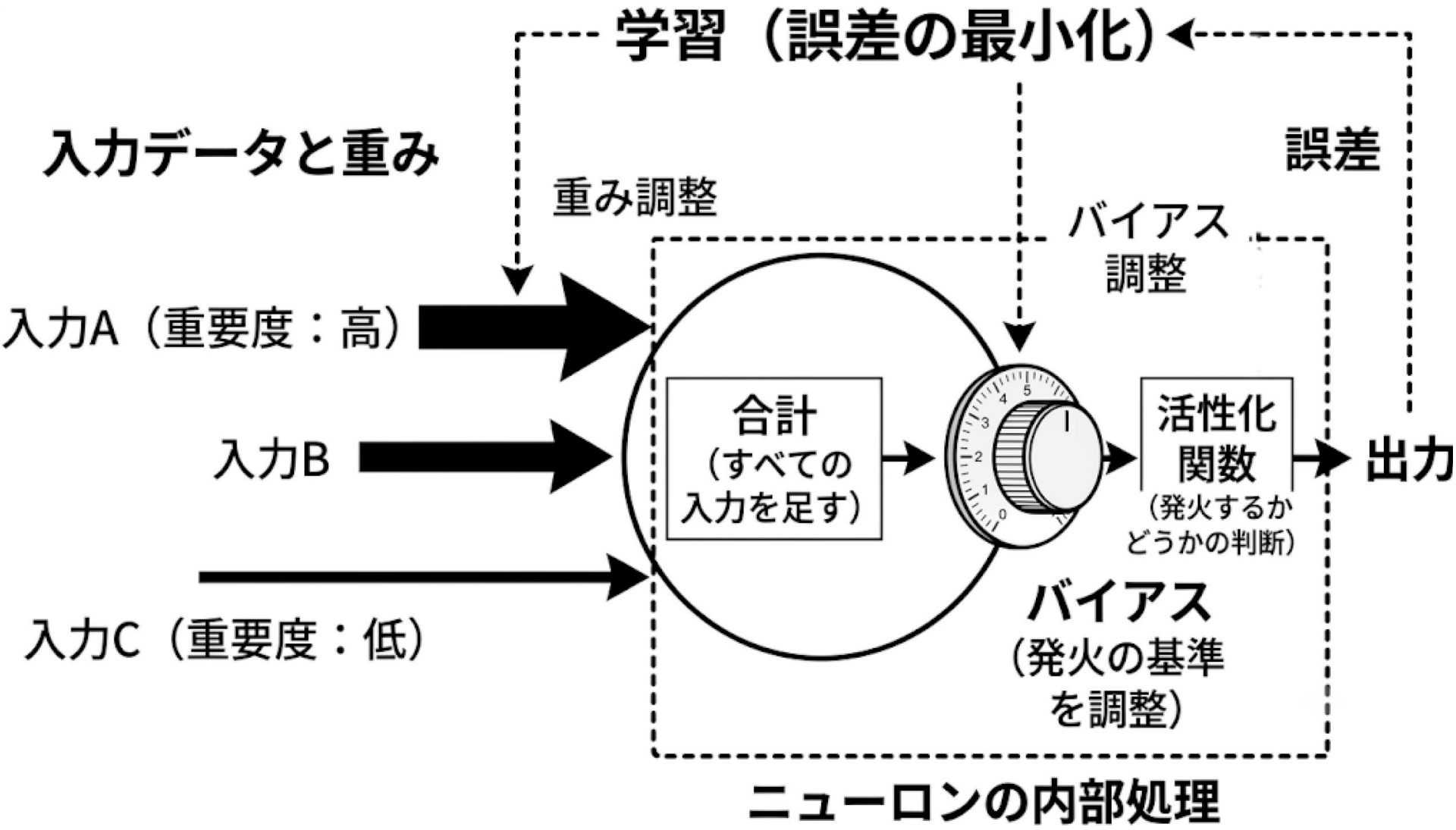
5-1. ニューラルネットワークの基本構造

ニューラルネットワークの構成と仕組み

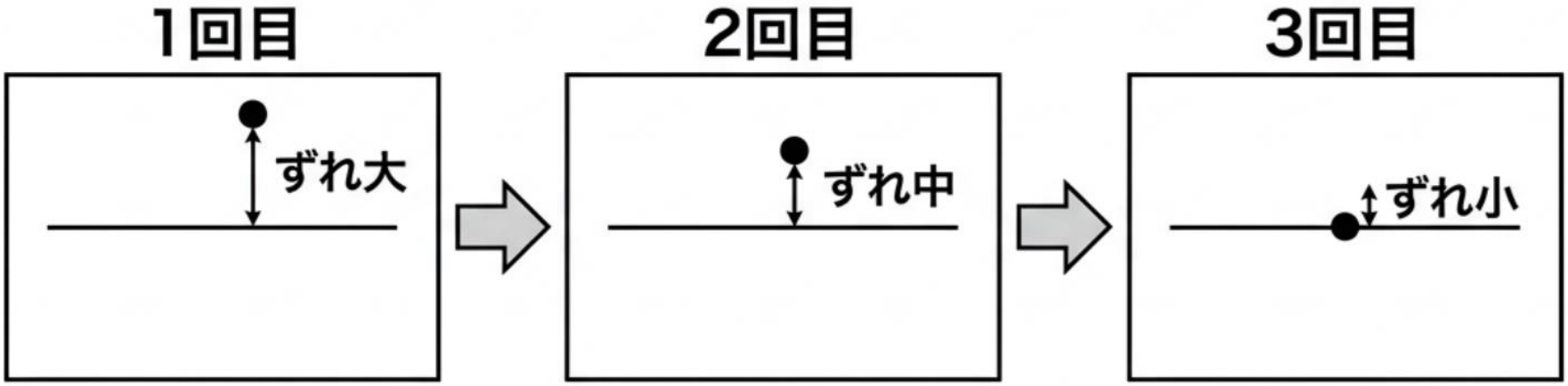
脳神経回路を模したネットワーク



ニューロンの処理と学習



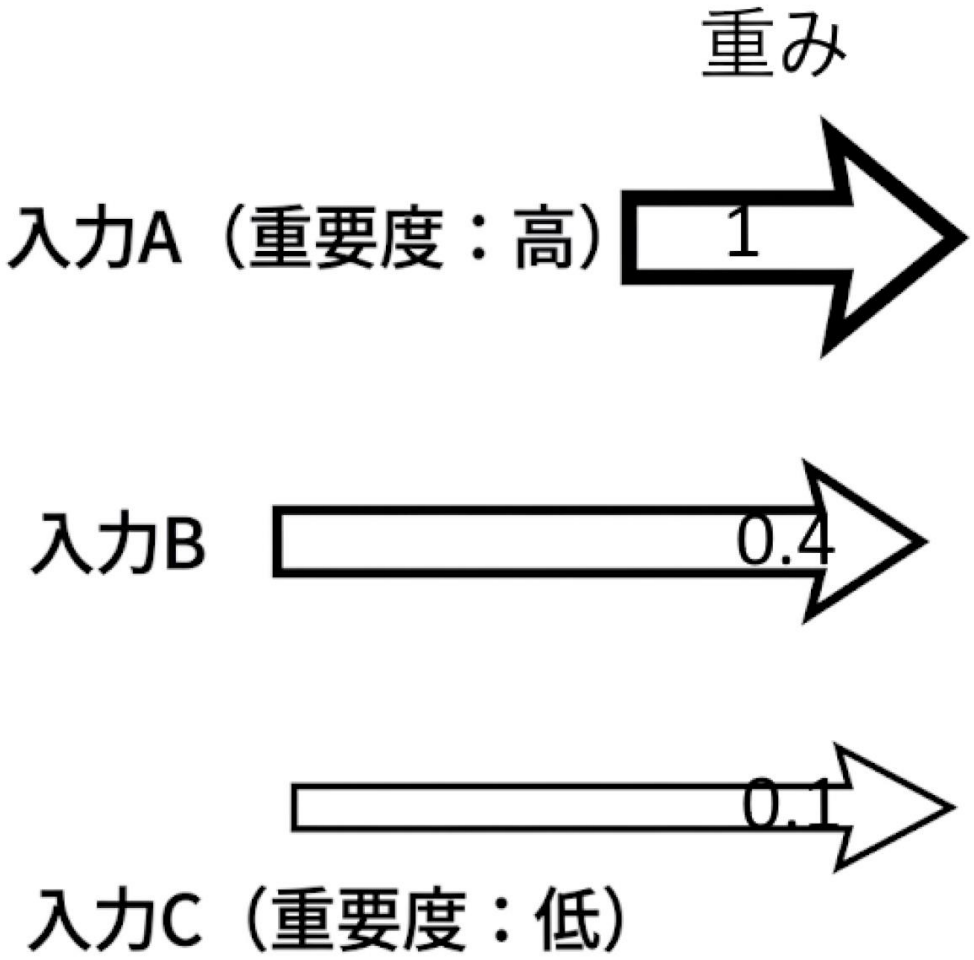
学習（誤差の最小化）



誤差の確認 ⇒ 少しずつ修正　　**繰り返す**

一度で正解にたどり着くのでなく、徐々に近づける

① 重みづけ



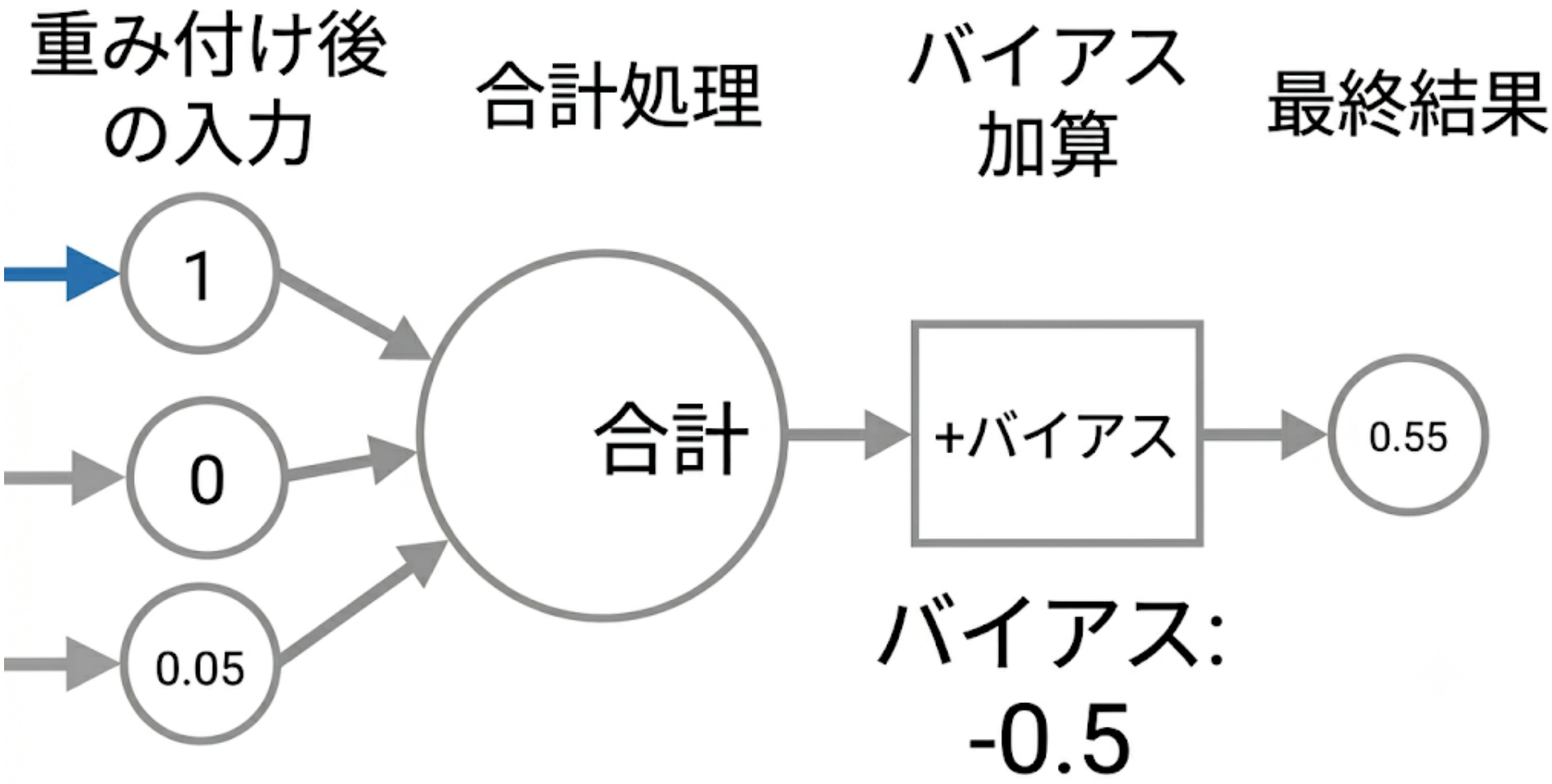
入力A		重み		重みづけ後
1.0	×	1	=	1.0

入力B		重み		重みづけ後
0.0	×	0.4	=	0.0

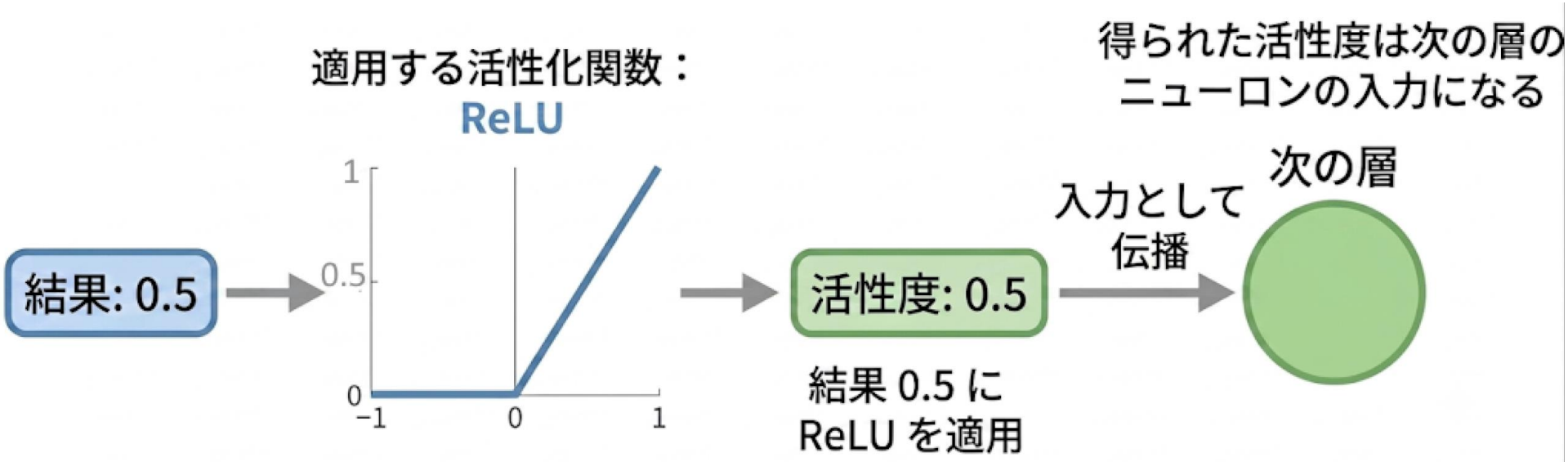
入力C		重み		重みづけ後
0.5	×	0.1	=	0.05

② 合計とバイアス

重みづけされた値の合計にバイアスを加える

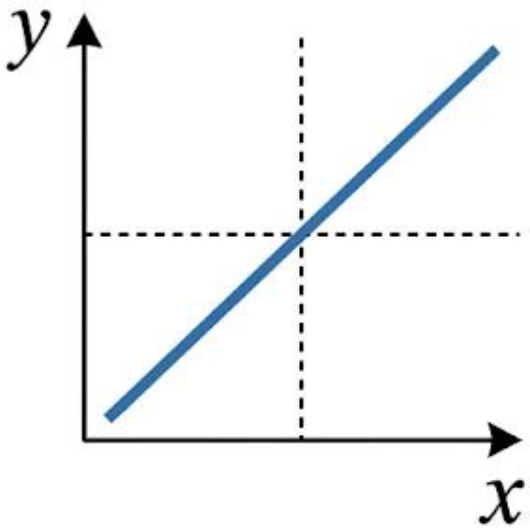


③活性化関数の適用


得られた活性度は次の層の
ニューロンの入力になる
適用する活性化関数：
ReLU
結果: 0.5
1
0.5
0
−1
0
1
活性度: 0.5
結果 0.5 に
ReLU を適用
入力として
伝播
次の層

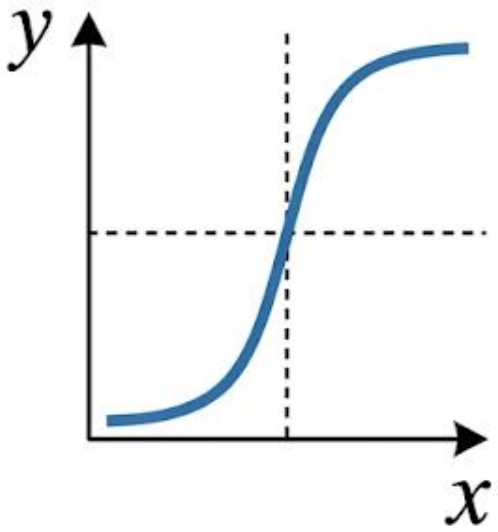
活性化関数

活性化関数は「非線形」の変換を行う



【活性化関数が線形】の場合

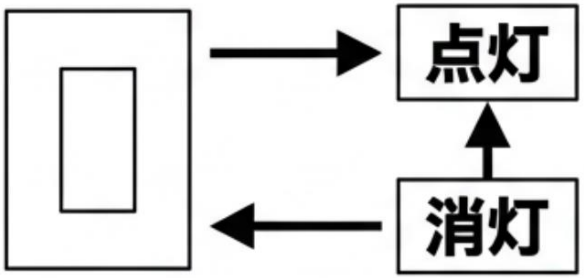
層をいくつ重ねても、全体として1つの線形変換。複雑なパターンを表現できない。



【活性化関数が非線形】の場合

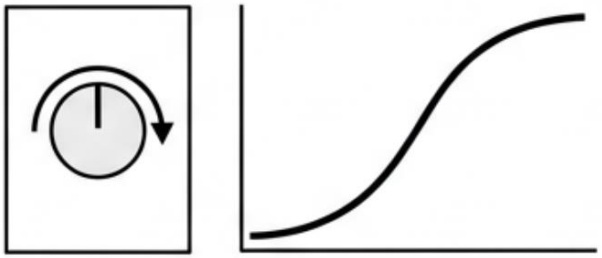
非線形な関数により、層を重ねるほどネットワークの表現力が高まる。複雑な特徴を抽出。

日常生活での非線形の例



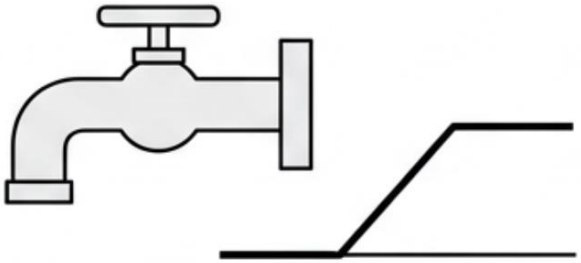
部屋の照明スイッチ（ステップ関数）

一定の力でスイッチを押すと
「点灯／消灯」が切り替わる



調光ライト（シグモイド関数）

つまみを回すと暗→明へ
なめらかに変化する



蛇口（ReLU関数）

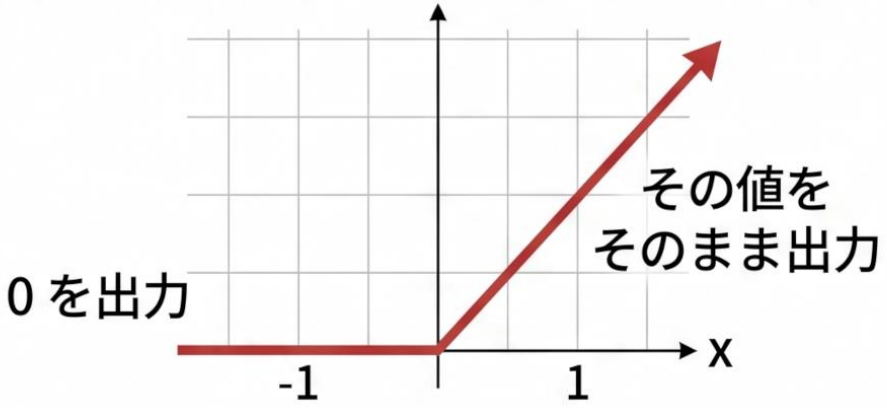
あるところまでは水が出ないが、
それを超えると勢いよく流れ出す

共通点：入力に対して、そのまま比例して反応するわけではない

活性化関数の種類

ReLU（整流線形ユニット）

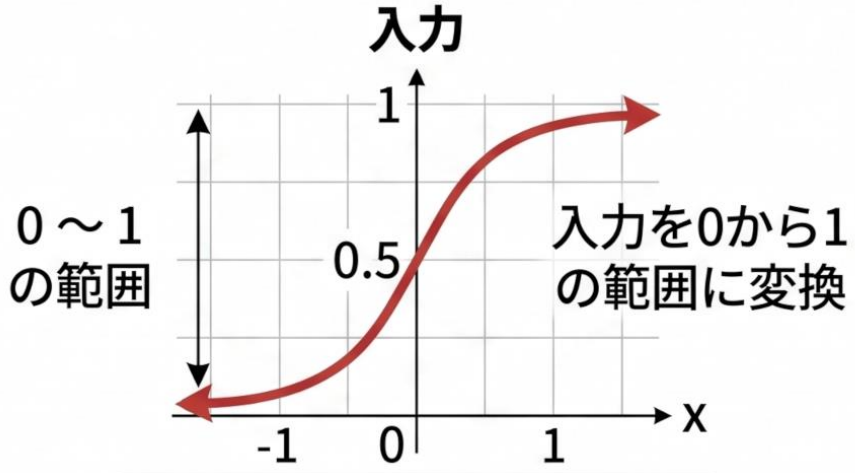
入力 ≤ 0 の場合　　入力 > 0 の場合



数式: $f(x) = \max(0, x)$

深層学習で広く使用される

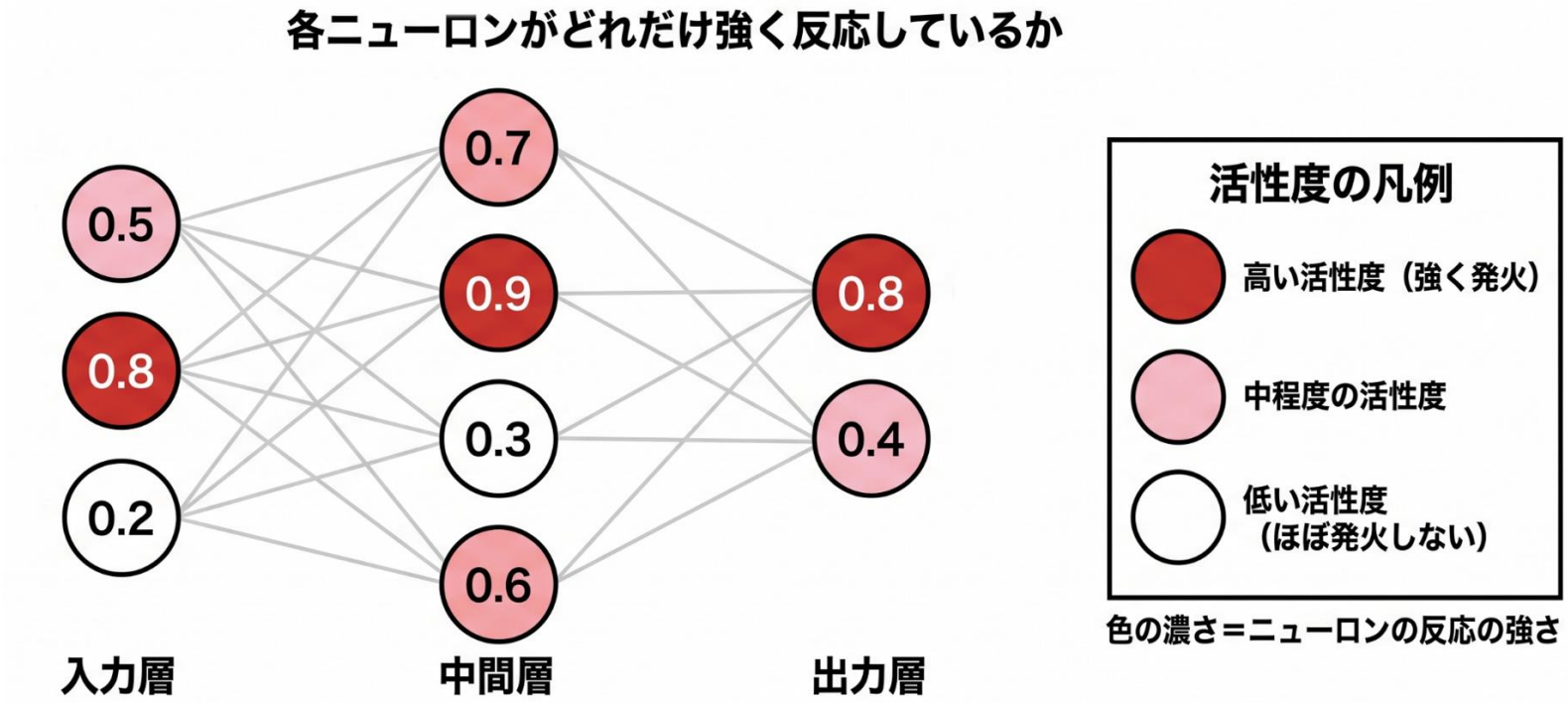
シグモイド



数式: $f(x) = \dfrac{1}{1 + e^{-x}}$

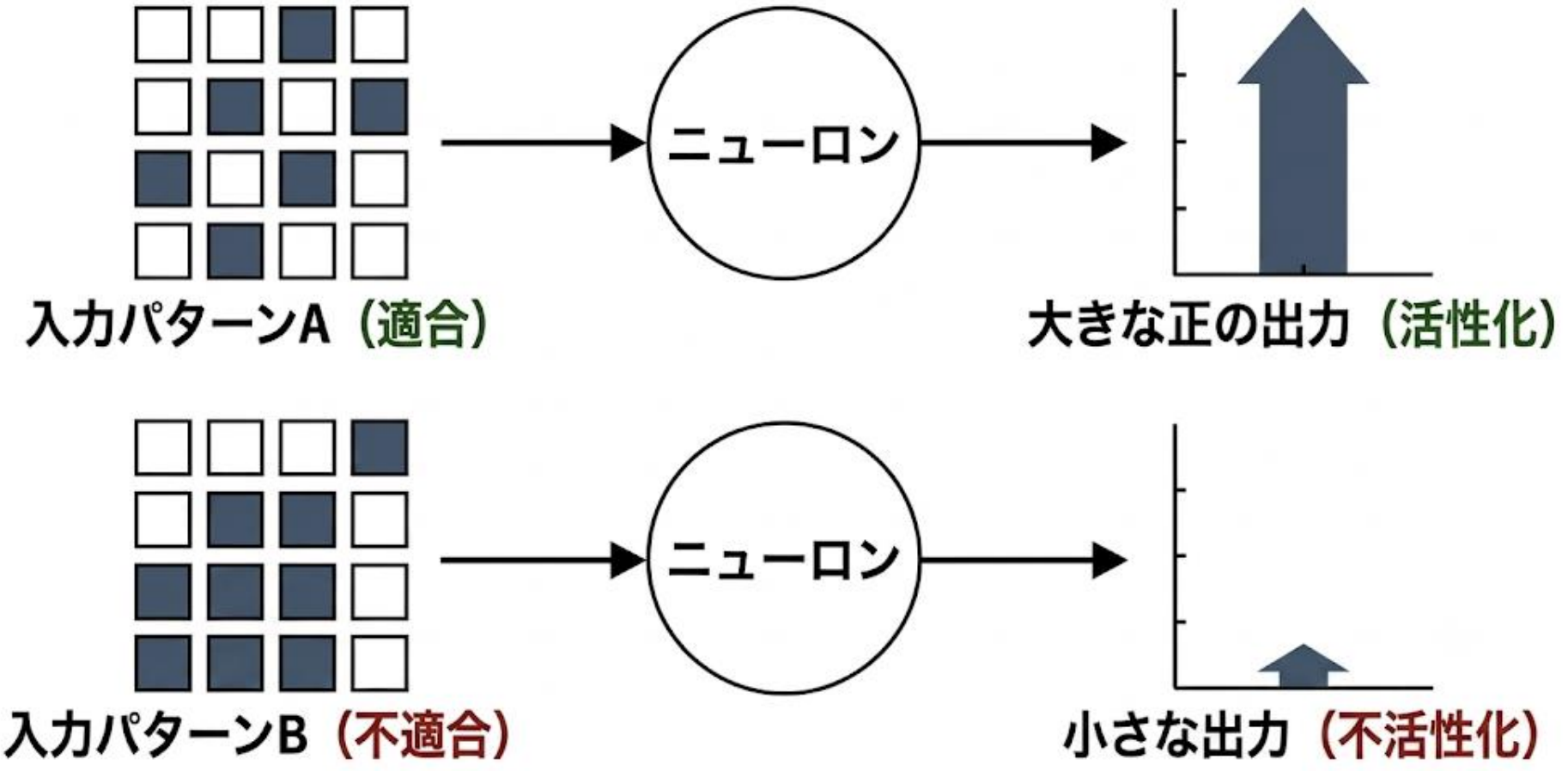
ニューロンの活性度

- 活性化関数を通して算出された各ニューロンの出力値である。
- 値が大きいほどそのニューロンは活性化している（発火）ことを示す。

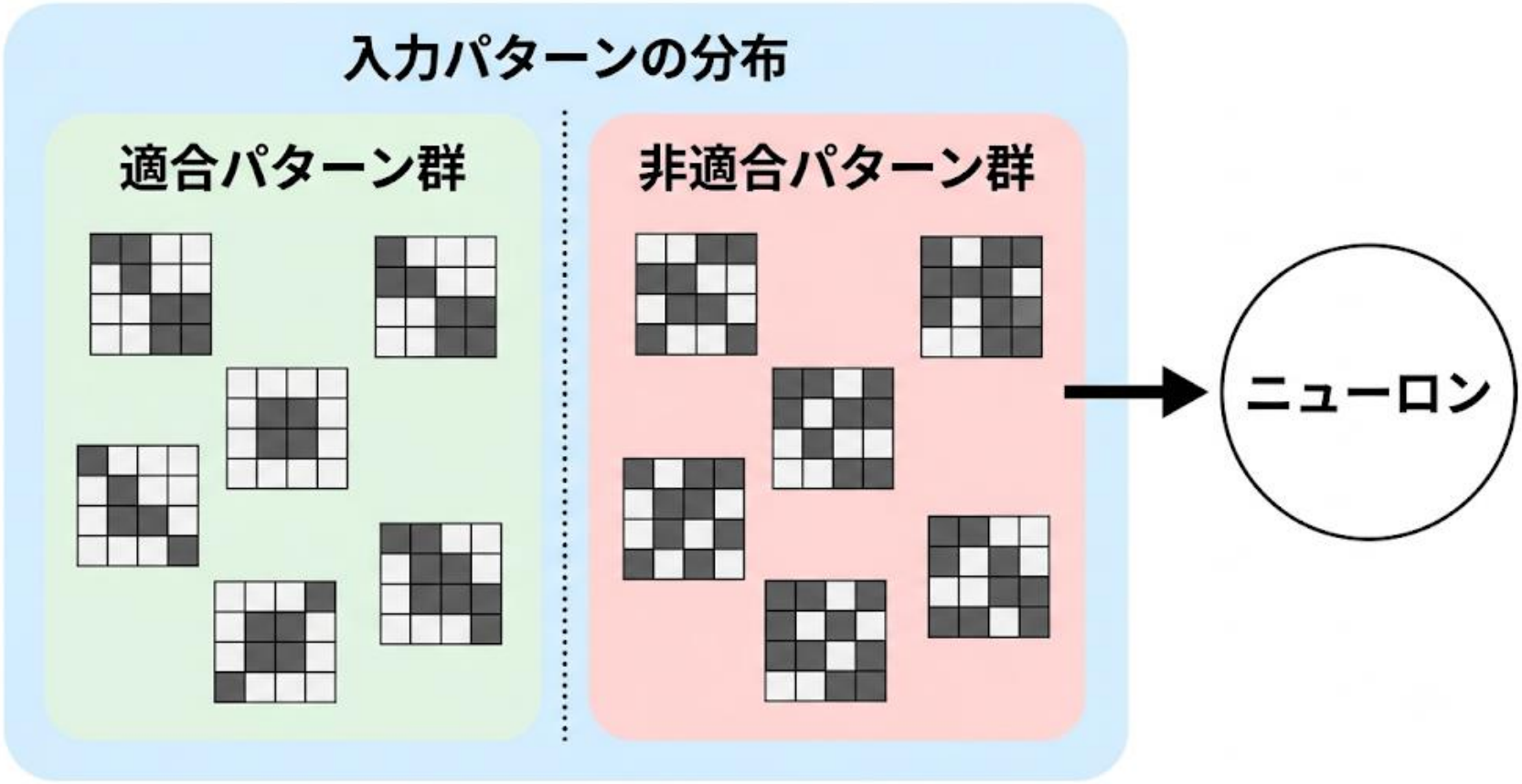


ニューロンの特定パターンでの活性化

ニューロンは特定のパターンに応じて活性化する



１個のニューロンは、パターンを識別し、適合（活性化）と不適合（非活性化）の二つに分ける働きを持つ

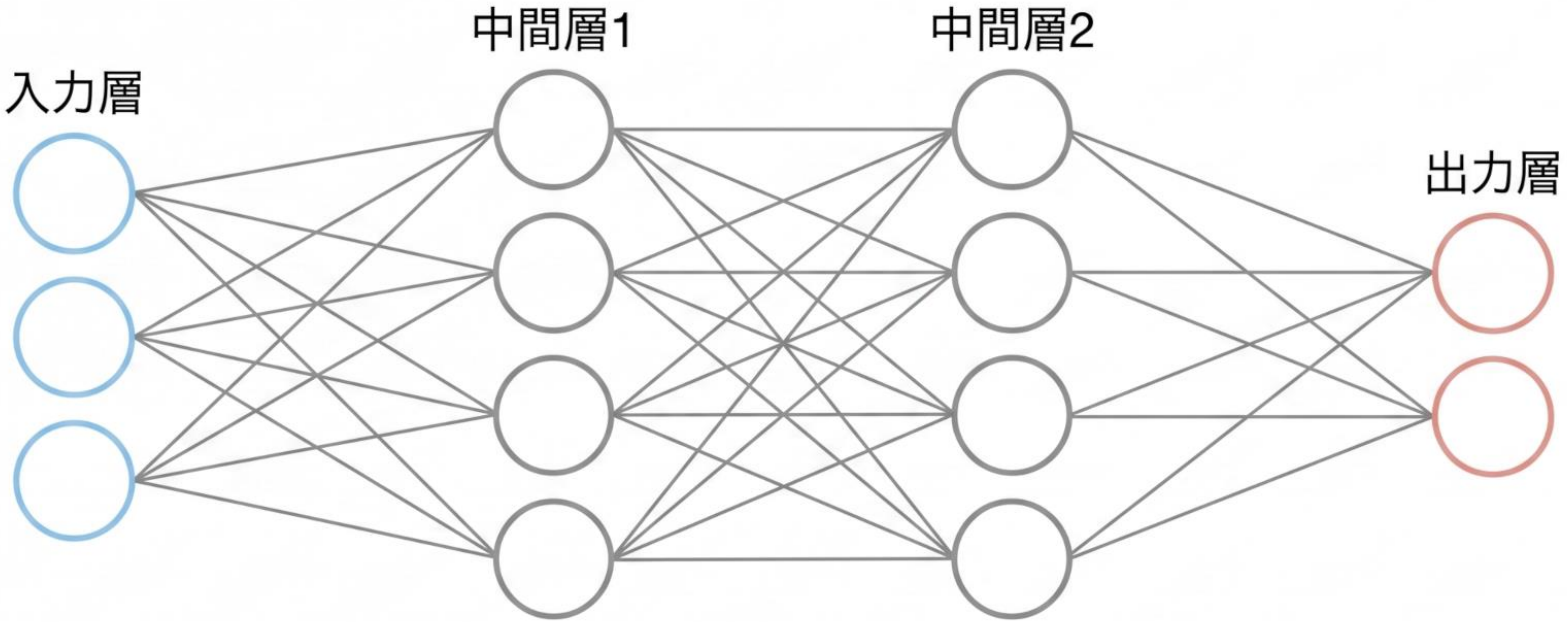


ニューロンが反応する「適合パターン群」

ニューロンが反応する「非適合パターン群」

ディープラーニング

中間層が複数ある**ニューラルネットワーク**を**ディープニューラルネットワーク**と呼び、その**学習**を**ディープラーニング（深層学習）**と呼ぶ。



入力層
役割: データ（画像、テキスト、数値など）を受け入れる。

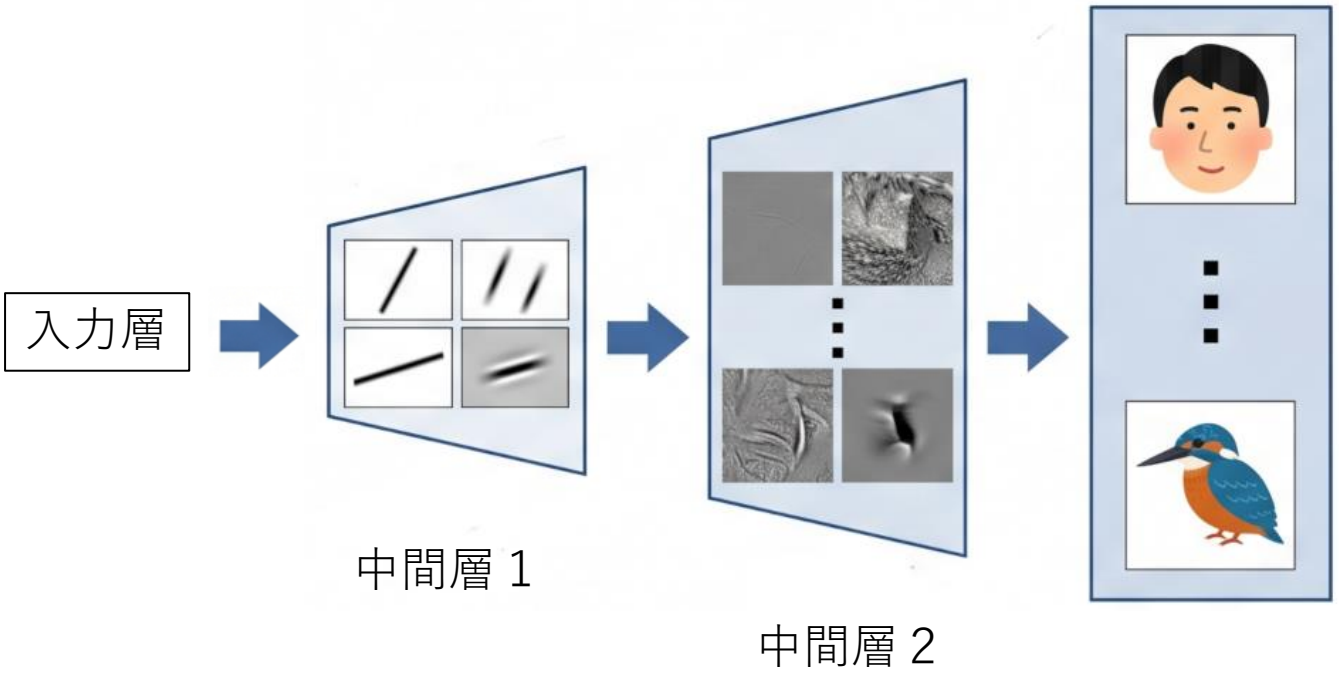
中間層1
役割: 基本的な特徴やパターンを抽出する。

中間層2
役割: さらに高度な特徴を抽出し、複雑な演算を行う。

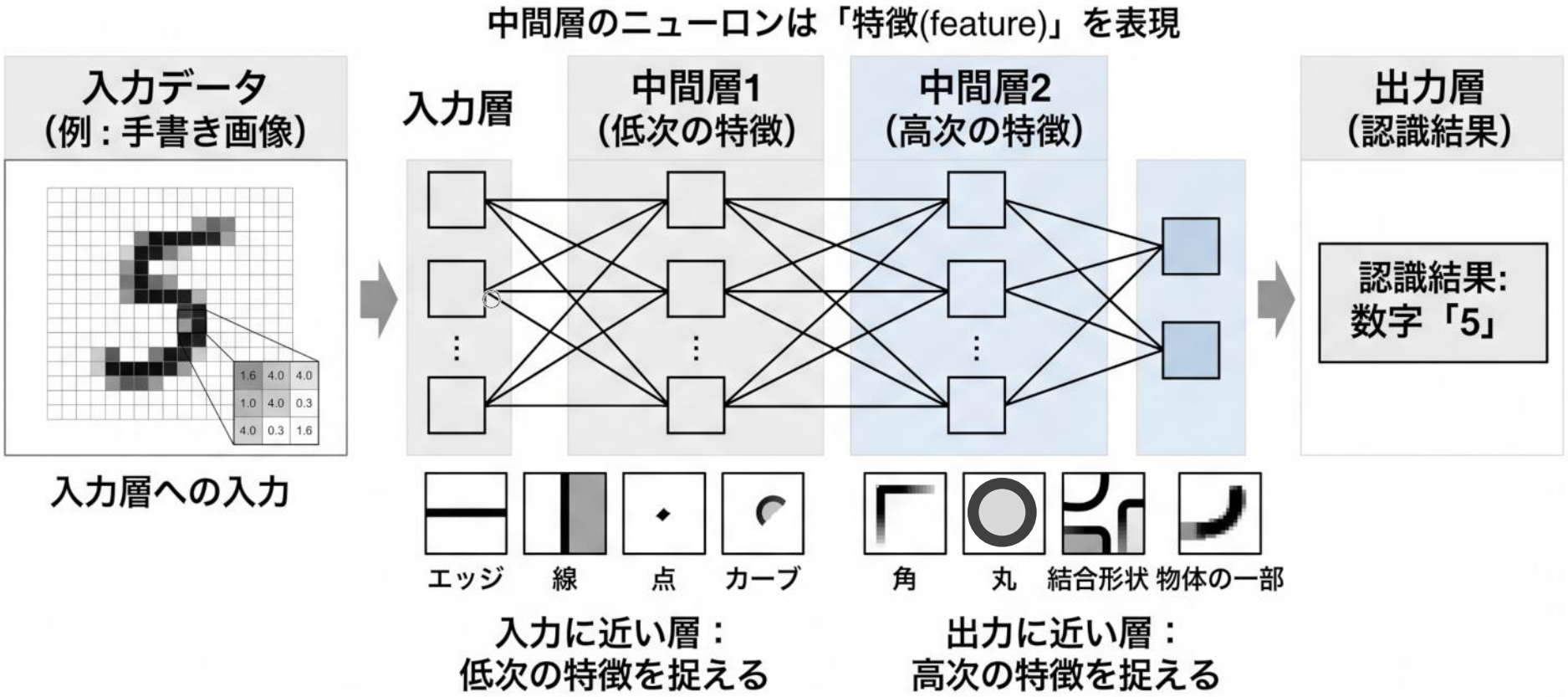
出力層
役割: 最終的な予測や分類結果を出力する。

ディープラーニング

層を深くすることで複雑なパターンの学習を目指す



ニューラルネットワークとパターン認識



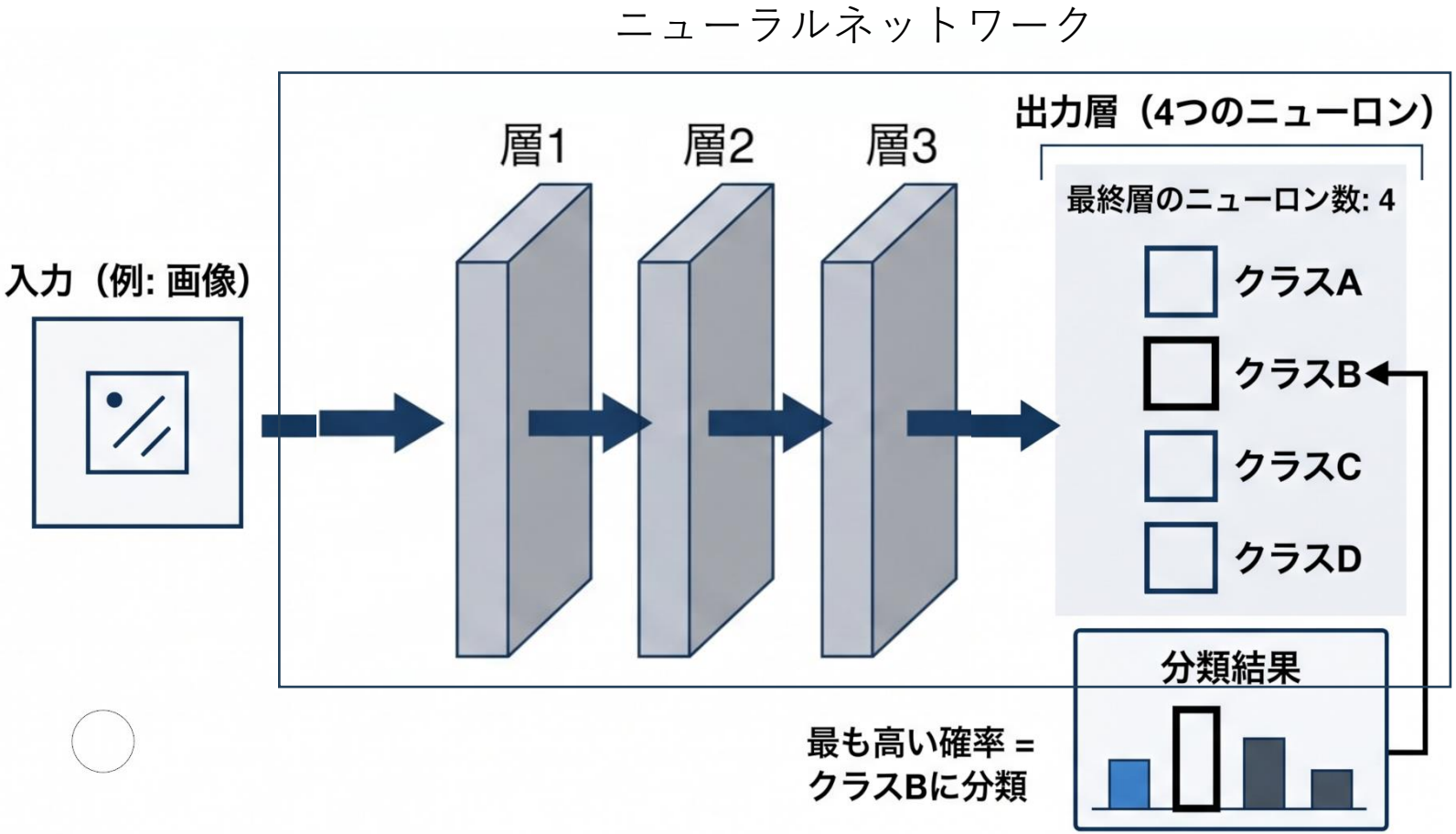
特徴量と特徴抽出と自動化

- データから【自動】で特徴抽出を行う
- 中間層のニューロンが段階的に特徴を捉える
- 層が進むにつれて低次から高次の特徴へと複雑化する

5-2. ニューラルネットワークを用いた分類

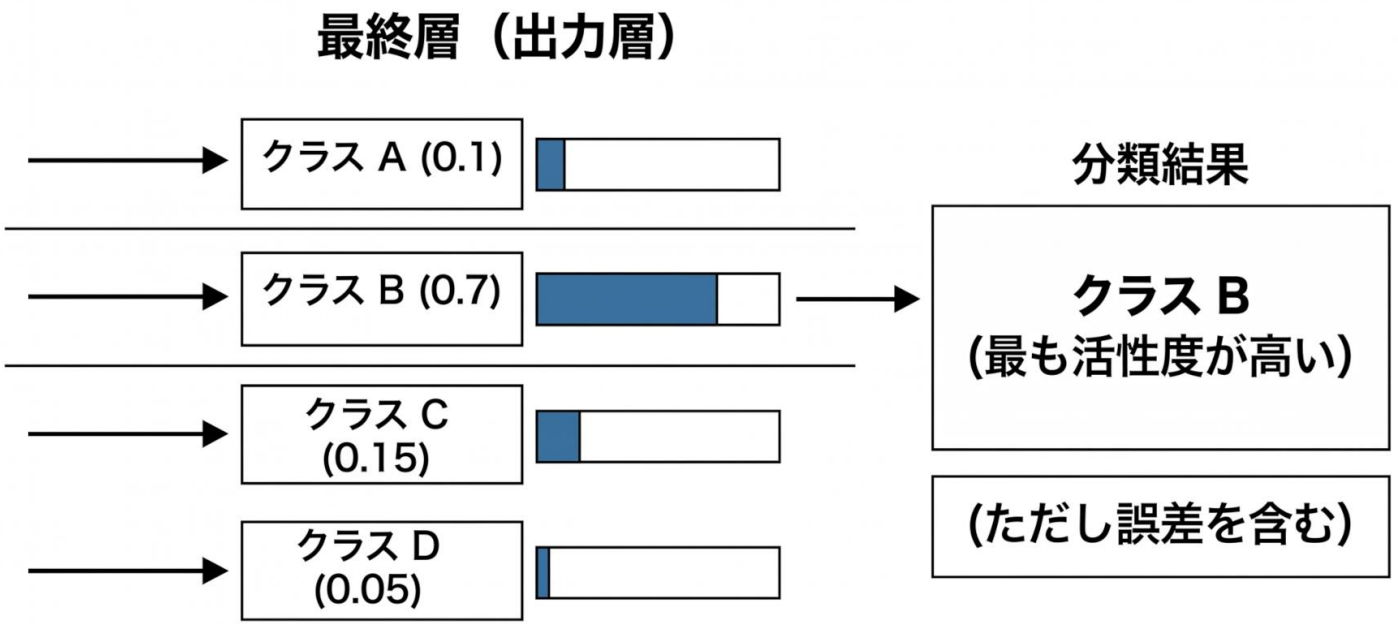
ニューラルネットワークを用いた分類

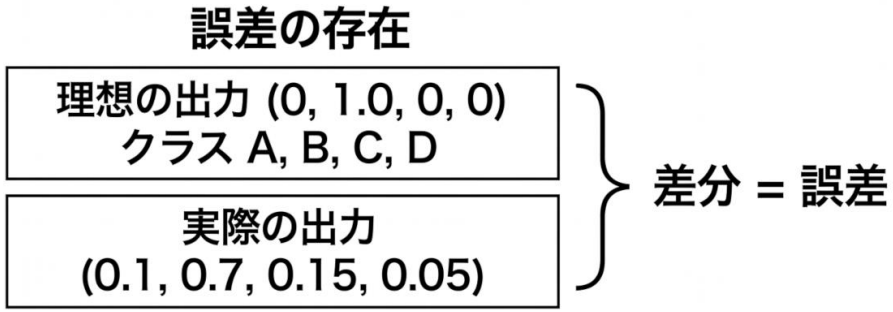
出力層：分類するクラスの数（ここでは４）と**同じ数のニューロン**



ニューラルネットワークを用いた分類と誤差

最終層のニューロンのうち**最も活性度が高いもの**に**対応するクラス**が**分類結果**となる。分類結果には誤差がある。

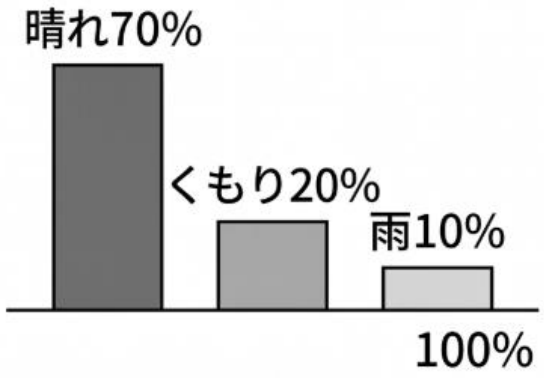




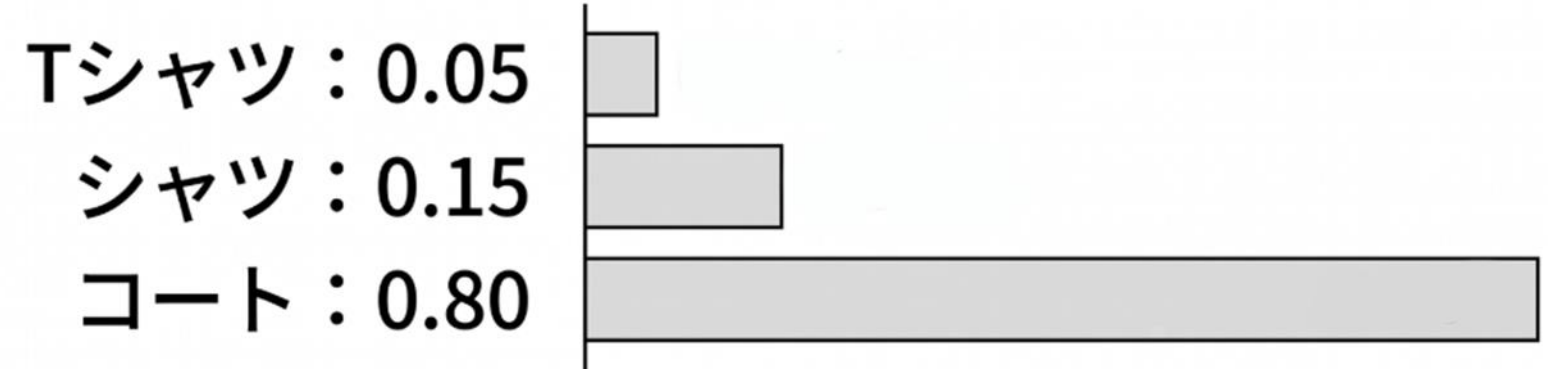
確率出力（結果と確信度を同時に得る）

- 天気予報の場合

晴れ70% / くもり20% / 雨10%



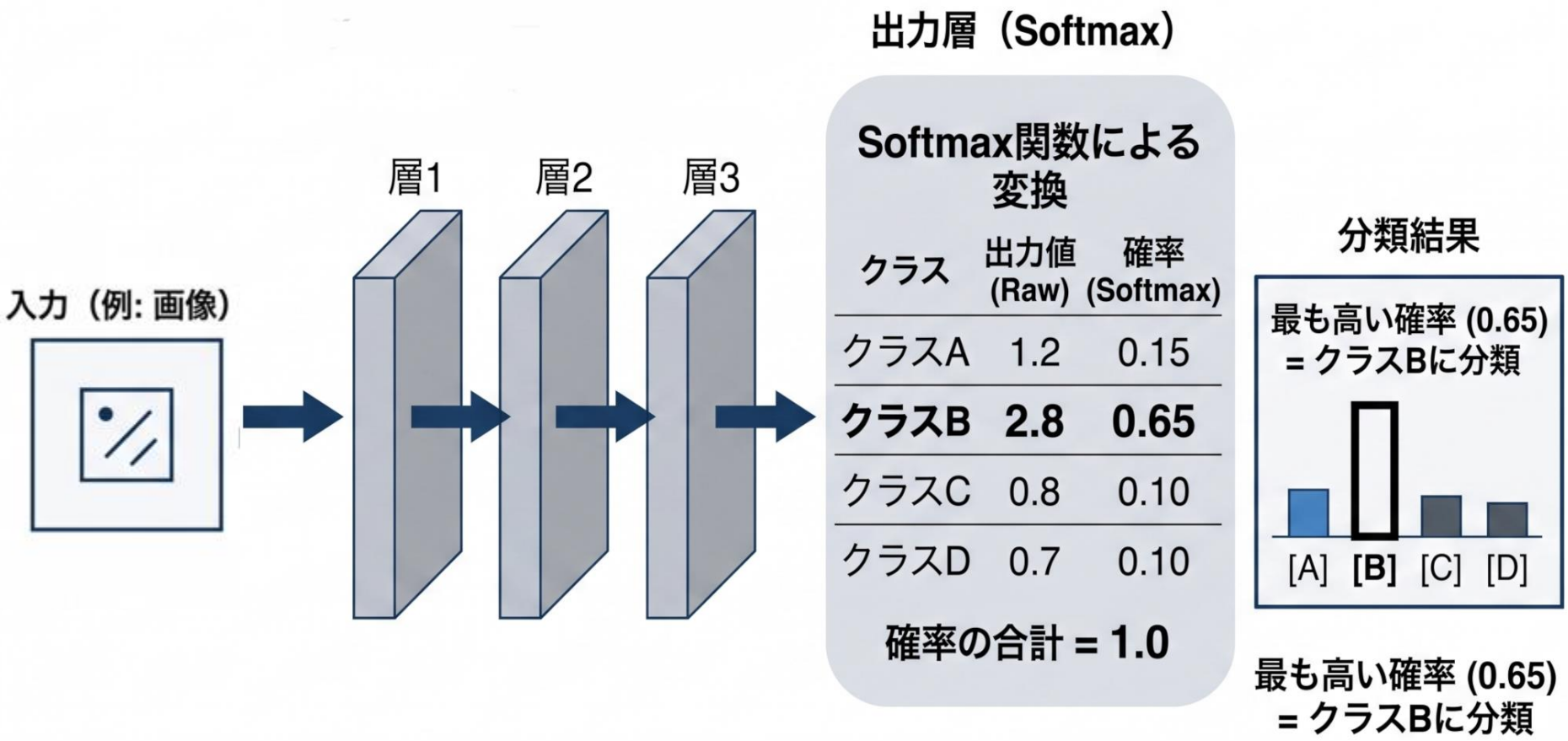
- AIによる画像分類の場合



それぞれの「確信度」が確率で分かる

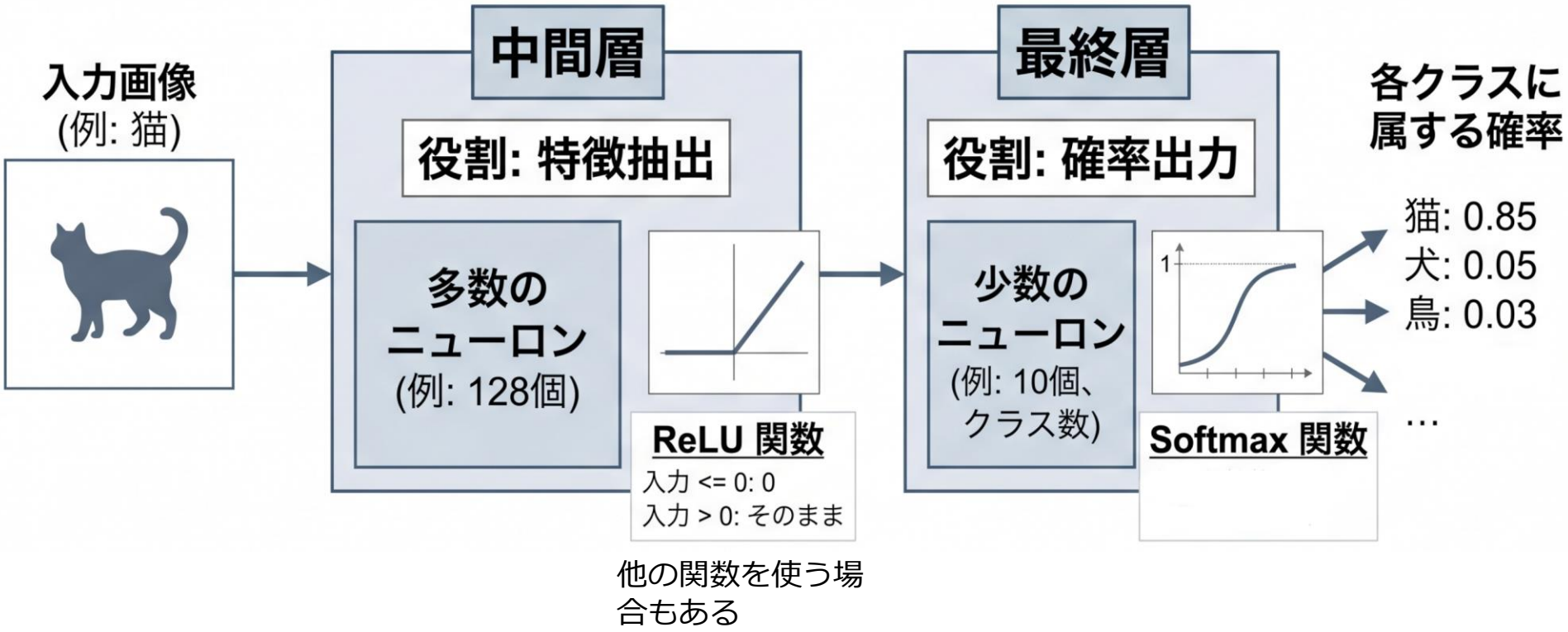
ソフトマックス (Softmax関数) の役割

出力を、**0から1の範囲の数値（確率）に変換**するソフトマックス (Softmax)



活性化関数の使い分け

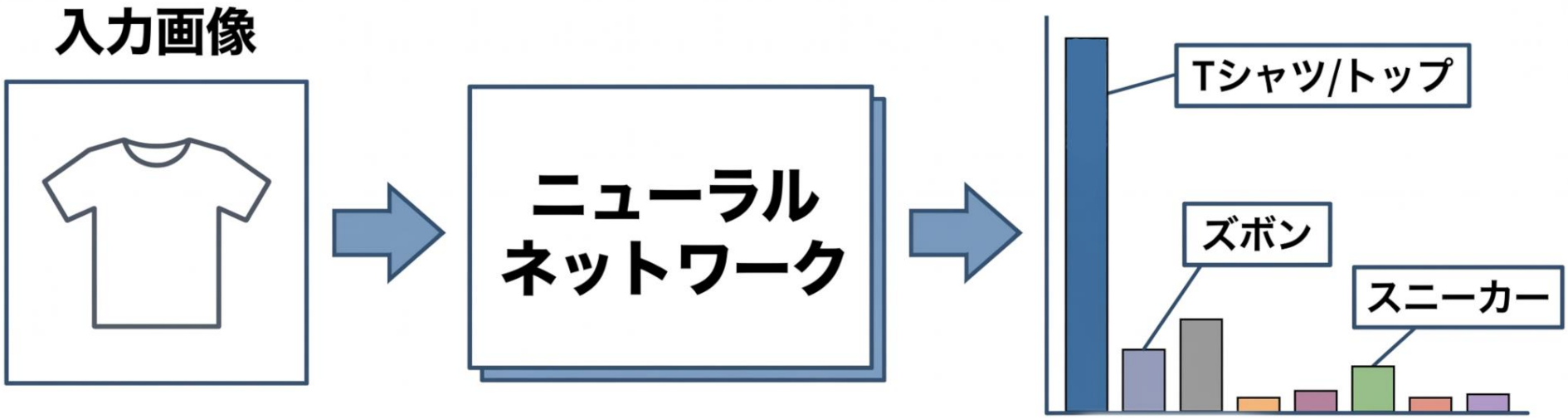
中間層と最終層で異なる種類の関数を用いる
（入力層はデータの受け取りのみで、もともと活性化関数がない）



画像分類

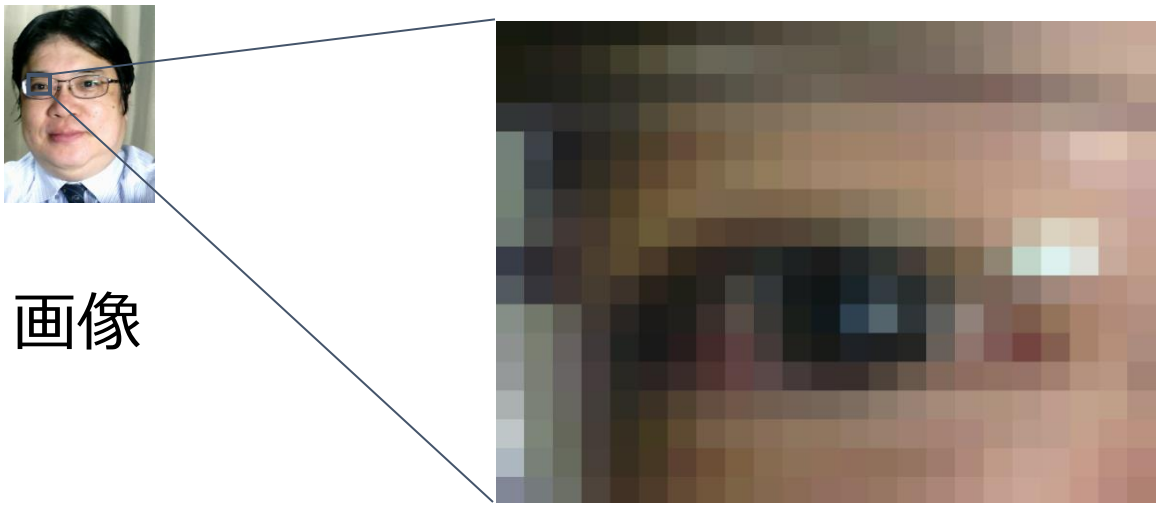
画像を所定の種類に自動分類

出力：クラスごとの確率



デジタル画像の構造：画像と画素

- 画像を拡大すると，個々の画素が確認できる.

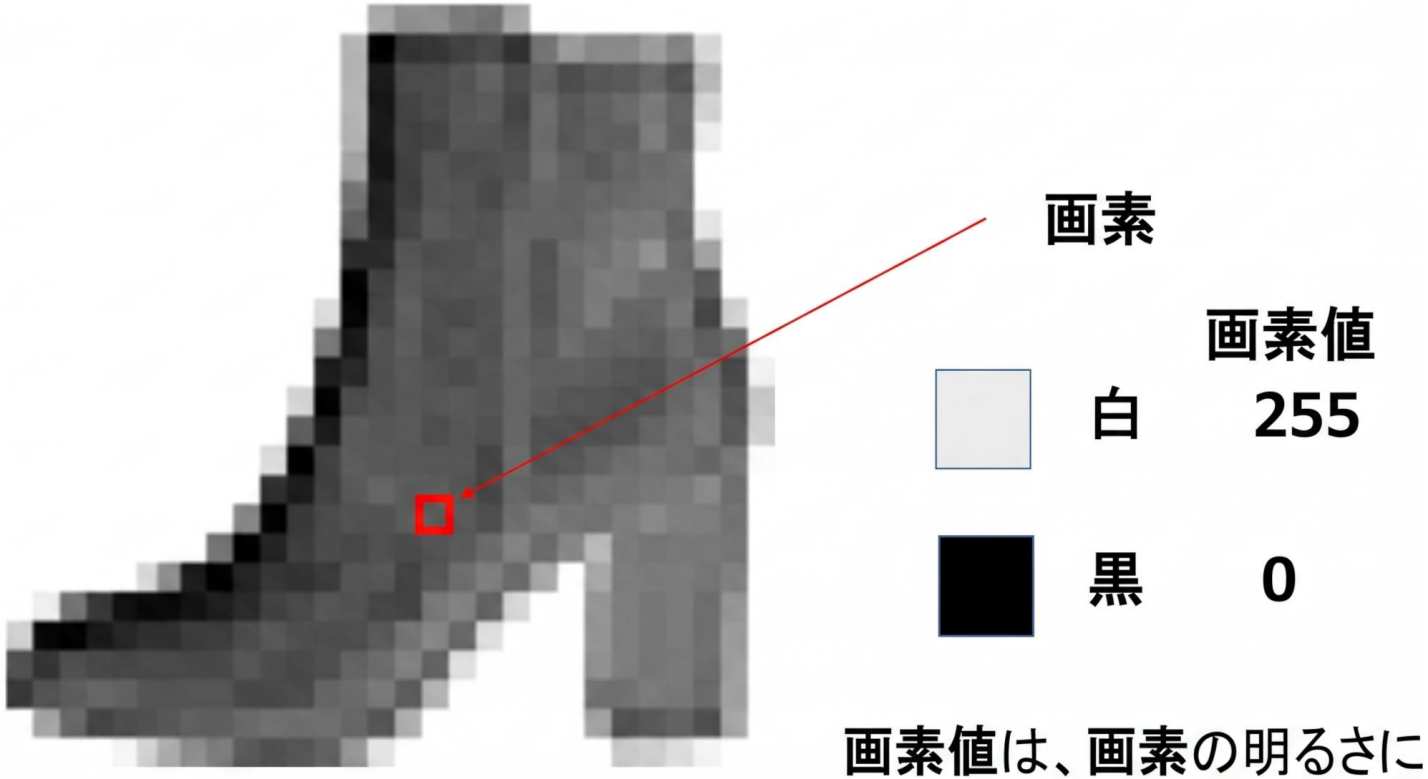


画像

それぞれの格子が画素

画像と画素

画像は**格子状に並んだ画素**（画像を構成する最小の要素）から構成される.

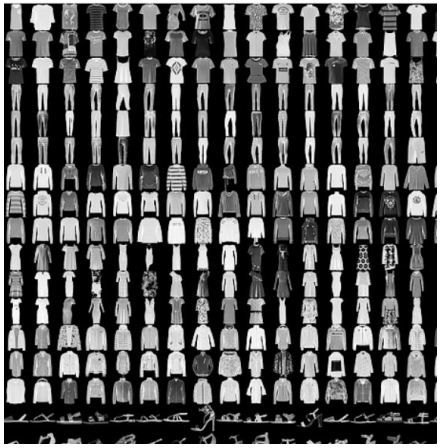


画素値は、**画素**の明るさに応じた **0** から **255** の数値

Fashion MNISTデータセット（衣類の画像データセットで、濃淡画像）

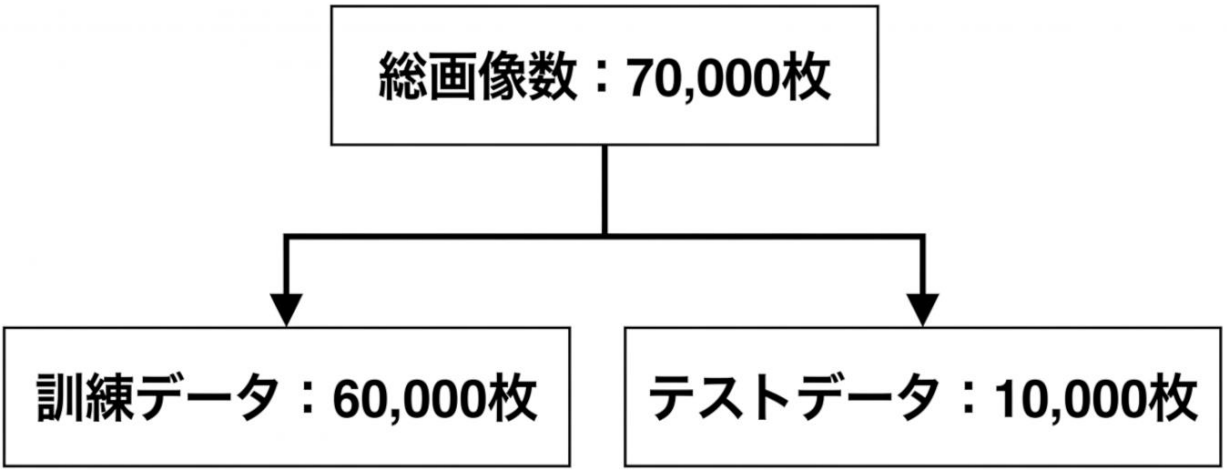
画像サイズ: **28 x 28**

Fashion MNIST データセット

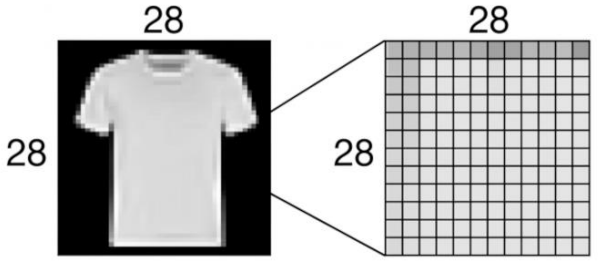


Zalando Research が 2017年に公開した画像分類用のデータセット。衣料品・履物・バッグの計10クラス

データ構成



画像仕様



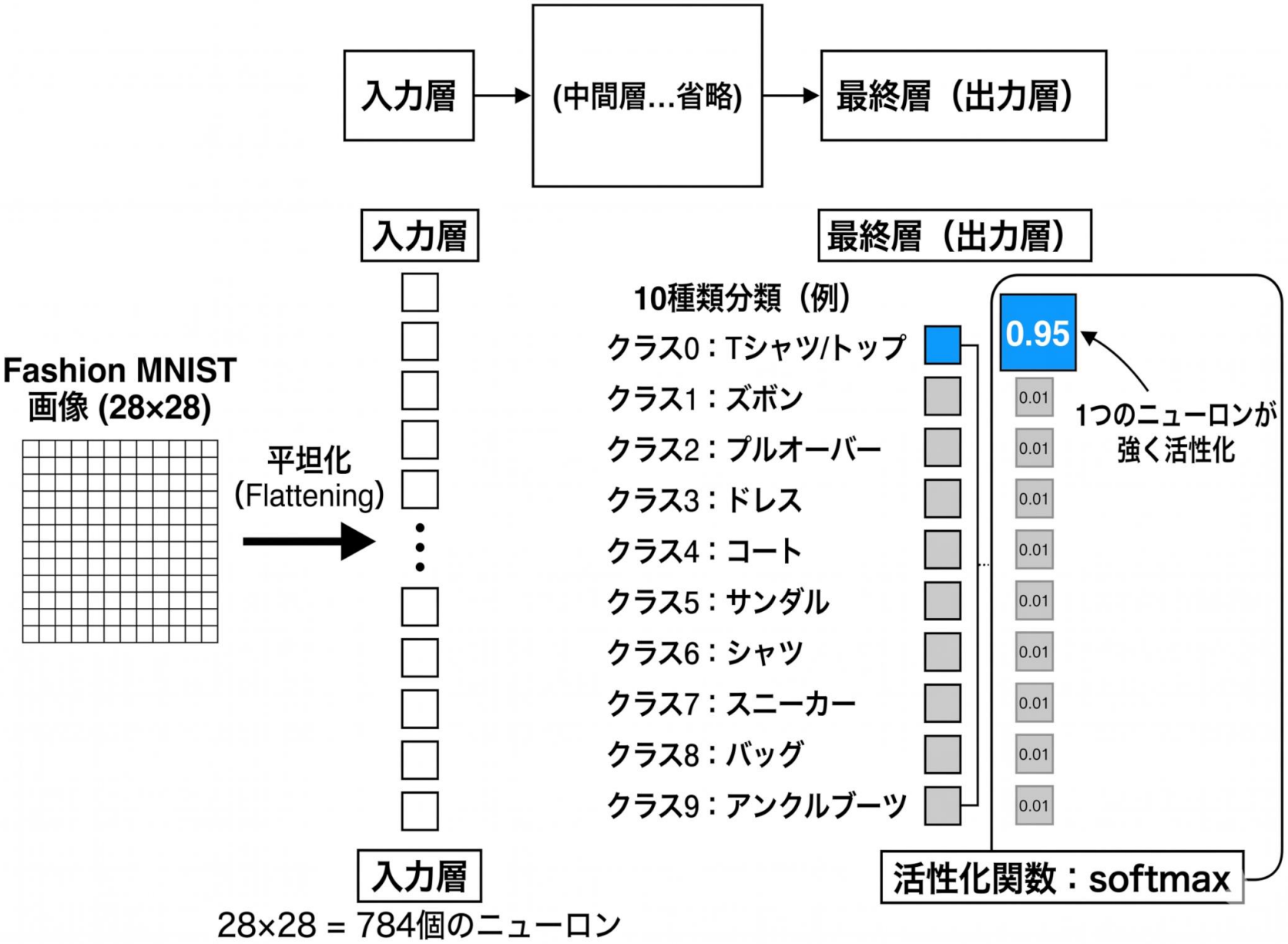
サイズ：28 × 28 ピクセル

色：グレースケール
(8ビット, 画素値 0〜255)

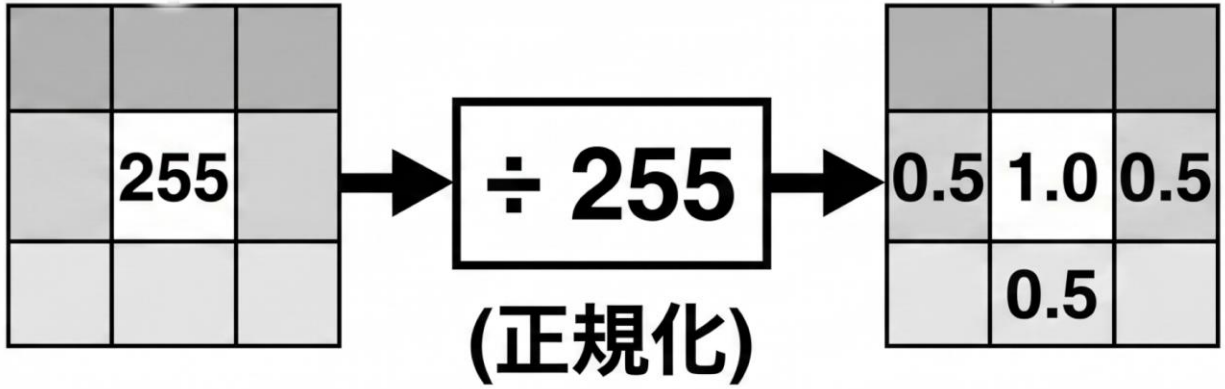
Fashion MNIST データセットの 10 クラス

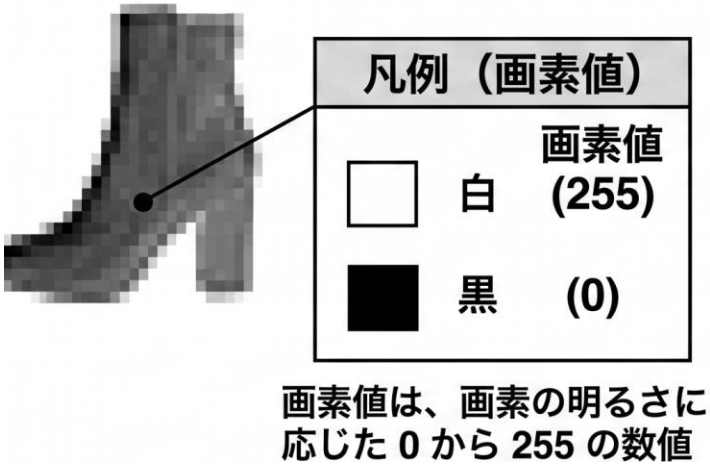
Class 0 **T-shirt/top** **(Tシャツ)**	Class 1 **Trouser** **(ズボン)**	Class 2 **Pullover** **(プルオーバー)**	Class 3 **Dress** **(ドレス)**	Class 4 **Coat (コート)**
Class 5 **Sandal** **(サンダル)**	Class 6 **Shirt** **(シャツ)**	Class 7 **Sneaker** **(スニーカー)**	Class 8 **Bag** **(バッグ)**	Class 9 **Ankle boot** **(アンクルブーツ)**

Fashion MNIST 分類用ニューラルネットワークの構成



画像データの前処理（正規化）





演習

演習① Neural Network Visualizer による画像分類の学習体験

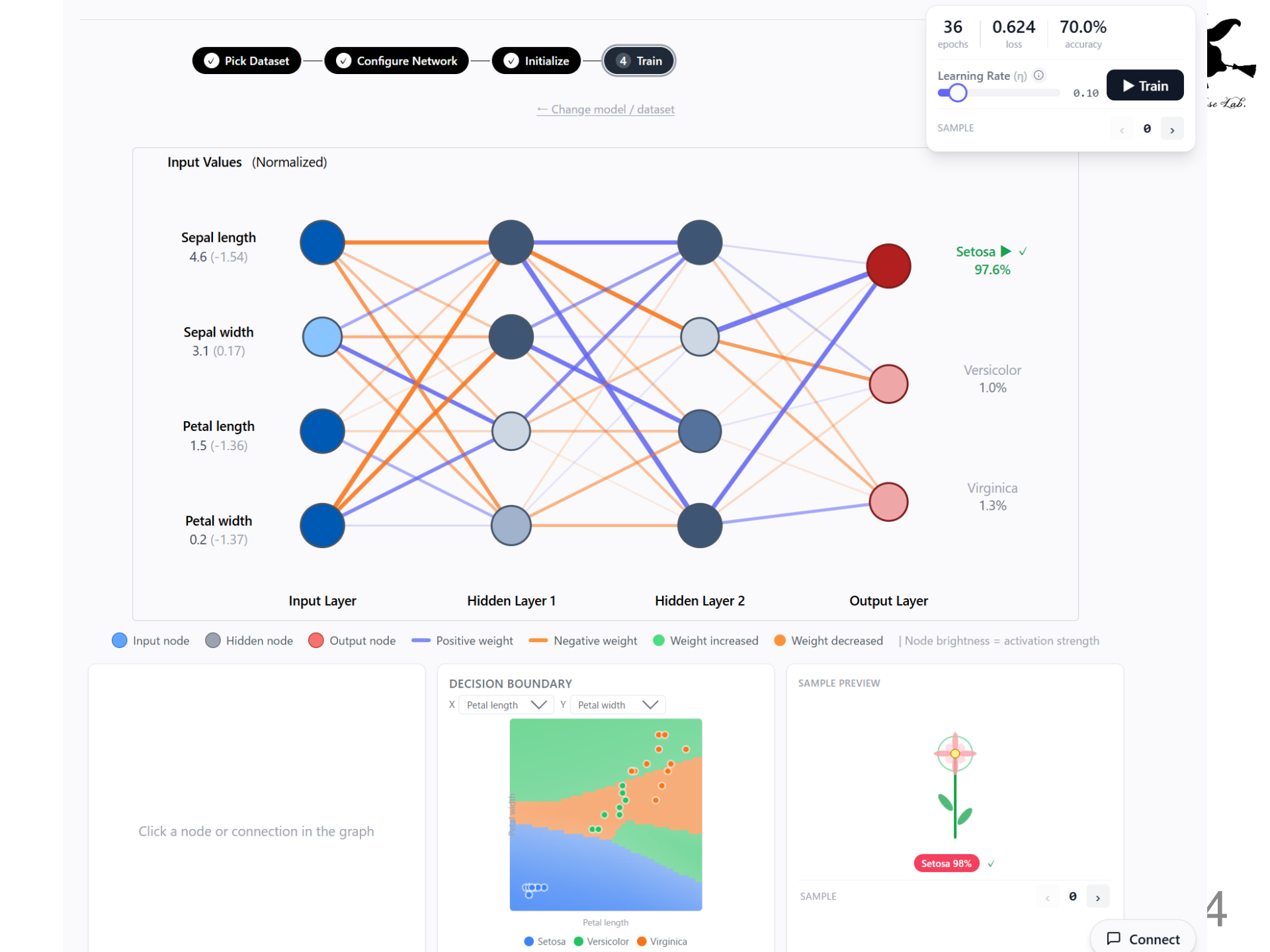
- ニューラルネットワークの構造と学習過程を視覚的に観察する
- **層の構造，活性度，損失と正解率**の意味の理解

https://www.nn-visual.com/ にアクセス

Iris を選択

Initializeを実行。Train を**繰り返し実行**

- **学習の繰り返しごとに損失（誤差）が減少**し，**アヤメの花（Iris）の分類**ができる
- **中間層の数やニューロン数を変更**し，**学習の変化を比較**できる


Pick Dataset
Configure Network
Initialize
4 Train
36
epochs
0.624
loss
70.0%
accuracy
Learning Rate (η)
0.10
Train
← Change model / dataset
SAMPLE
0
Input Values (Normalized)
Sepal length
4.6 (-1.54)
Sepal width
3.1 (0.17)
Petal length
1.5 (-1.36)
Petal width
0.2 (-1.37)
Setosa ▶ ✓
97.6%
Versicolor
1.0%
Virginica
1.3%
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer
Input node
Hidden node
Output node
Positive weight
Negative weight
Weight increased
Weight decreased
| Node brightness = activation strength
Click a node or connection in the graph
DECISION BOUNDARY
X Petal length
Y Petal width
Petal width
Petal length
Setosa
Versicolor
Virginica
SAMPLE PREVIEW
Setosa 98%
✓
SAMPLE
0
Connect

演習②ReLU（Pythonで実現）

- **活性化関数 ReLU** の入出力の**挙動**を確認する
- **ReLU** の定義，**条件分岐 if/else** によるプログラミング

https://trinket.io/python/61c6503fcada にアクセス

実行結果を確認

- **負の入力に対し 0，0以上の入力に対し入力値そのまま**を得る
- **入力値のリストを書き替えて，異なる値での挙動**を試せる

main.py

```python
# ReLU（Rectified Linear Unit）関数の実装と動作確認のプログラムです。
# 入力値xが0未満の場合は0を返し、0以上の場合はxをそのまま返します。
# -2から2までの異なる入力値でReLU関数を呼び出し、その変換結果を表示
# することで、活性化関数としてのReLUの基本的な挙動を示しています。

def relu(x):
  if x < 0:
    return 0
  else:
    return x

x = -2
print("x = ", x, "relu(x) =", relu(x))
x = -1
print("x = ", x, "relu(x) =", relu(x))
x = 0
print("x = ", x, "relu(x) =", relu(x))
x = 1
print("x = ", x, "relu(x) =", relu(x))
x = 2
print("x = ", x, "relu(x) =", relu(x))
```

Result

Powered by trinket

```
('x = ', -2, 'relu(x) =', 0)
('x = ', -1, 'relu(x) =', 0)
('x = ', 0, 'relu(x) =', 0)
('x = ', 1, 'relu(x) =', 1)
('x = ', 2, 'relu(x) =', 2)
```

演習③入力の重みづけ，合計とバイアス，活性化関数の適用

- **1つのニューロン**で行われる**計算の流れ**を理解する
- **重み・バイアス・活性化関数の役割と連携**の理解

https://trinket.io/python/1c339b660ee1 にアクセス

実行結果を確認

- **4つの入力 (0,0), (0,1), (1,0), (1,1)** に対する**活性度**を得る
- **重みやバイアスの値を書き替え，活性度の変化を予想**して**確認**できる

```
# 1つのニューロンnを持つシンプルなニューラルネットワークです．2つ
# の入力(x1, x2)に対して，重みはどちらも1，バイアスは-0.5に設定さ
# れています．ニューロンはReLU関数を活性化関数として使用し，4つの
# 異なる入力パターンに対する活性度を計算して出力します．

def relu(x):
  if x < 0:
    return 0
  else:
    return x

def n(x1, x2):
  # 重みは 1, 1, バイアスは -0.5
  s = 1 * x1 + 1 * x2 - 0.5
  return relu(s)

print("入力は (0, 0), 活性度 =", n(0, 0))
print("入力は (0, 1), 活性度 =", n(0, 1))
print("入力は (1, 0), 活性度 =", n(1, 0))
print("入力は (1, 1), 活性度 =", n(1, 1))
```

Result

Powered by trinket

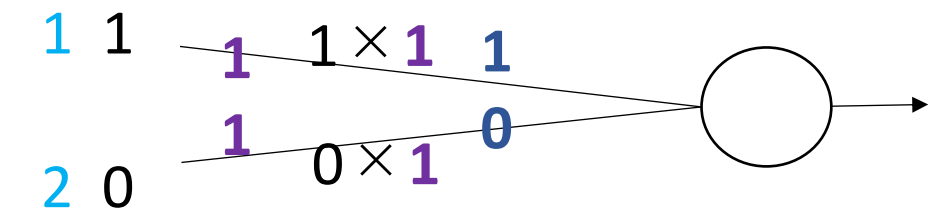
```
('入力は (0, 0), 活性度 =', 0)
('入力は (0, 1), 活性度 =', 0.5)
('入力は (1, 0), 活性度 =', 0.5)
('入力は (1, 1), 活性度 =', 1.5)
```

【入力が１０のとき】

1 と 0 の合計である 1 に
バイアス -0.5を加算

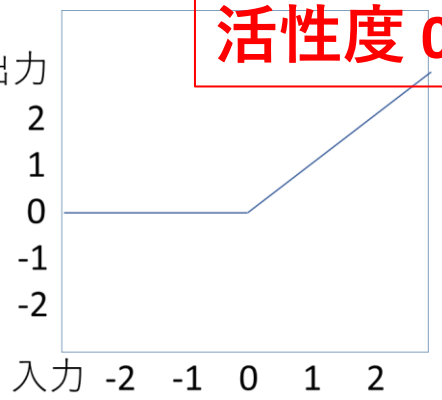
重みとの
掛け算

入力 重み



結果 0.5

ReLUを適用
活性度 0.5



ReLU

1 2

入力

白を 1
黒を 0 とする

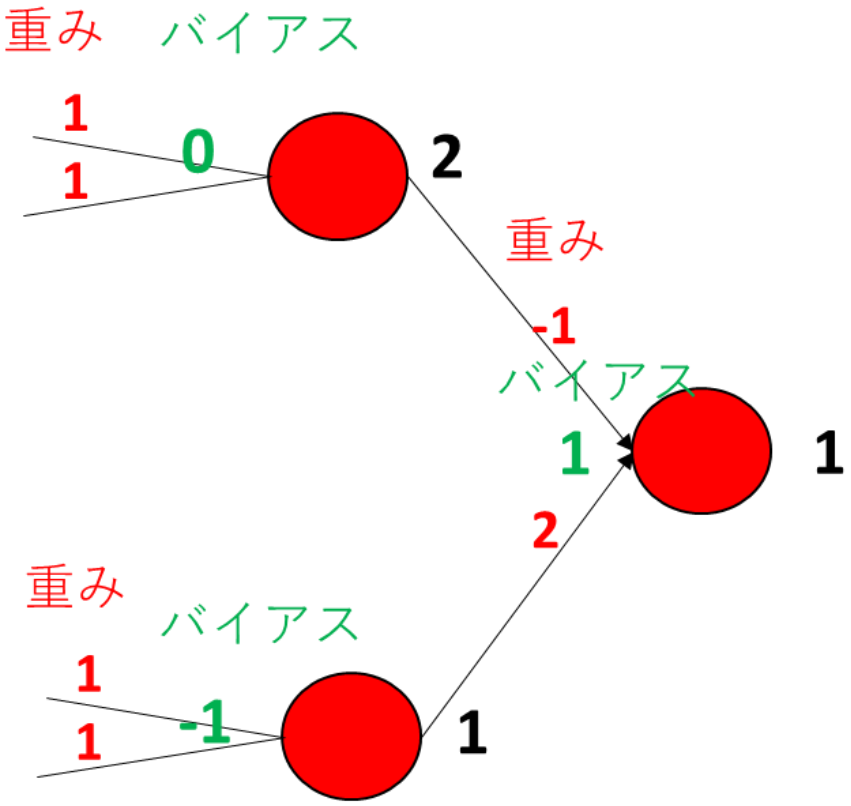
演習④ニューラルネットワーク

- **複数のニューロン**を層として**組み合わせ**た処理を体験する
- **2層構造（n1・n2 と n3）**による処理の基礎の理解

https://trinket.io/python/3cbc3f3ed057 にアクセス

実行結果を確認

- **4つの入力に対する活性度**を得て，**1層との違い**を観察する
- **各ニューロンの重み・バイアスを書き替え，比較**できる



main.py

```
# 3つのニューロン(n1, n2, n3)を組み合わせた単純なニューラル
# ネットワークです．各ニューロンは重みとバイアスを持ち，入力
# 値(x1, x2)に対してReLU関数を適用して活性度を計算します．
# 最終的なニューロンnは，n1とn2の出力を入力として受け取り，
# n3を通じて最終的な活性度を計算します．プログラムは4つの異
# なる入力パターンに対する活性度を表示します．

def relu(x):
  if x < 0:
    return 0
  else:
    return x

def n1(x1, x2):
# 重みは 1, 1, バイアスは 0
  s = 1 * x1 + 1 * x2 + 0
  return relu(s)
def n2(x1, x2):
# 重みは 1, 1, バイアスは -1
  s = 1 * x1 + 1 * x2 - 1
  return relu(s)
def n3(x1, x2):
# 重みは -1, 2, バイアスは 1
  s = -1 * x1 + 2 * x2 + 1
  return relu(s)
def n(x1, x2):
  return n3(n1(x1, x2), n2(x1, x2))

print("入力は (0, 0), 活性度 =", n(0, 0))
print("入力は (0, 1), 活性度 =", n(0, 1))
print("入力は (1, 0), 活性度 =", n(1, 0))
print("入力は (1, 1), 活性度 =", n(1, 1))
```

Result

Powered by trinket

```
('入力は (0, 0), 活性度 =', 1)
('入力は (0, 1), 活性度 =', 0)
('入力は (1, 0), 活性度 =', 0)
('入力は (1, 1), 活性度 =', 1)
```

演習⑤ Fashion MNIST 画像分類のコード読解

- **画像分類プログラムの構成**と**学習過程**をコードから読み取る
- **データ前処理，モデル構築，学習，画像分類（分類結果の予測）**の一連の流れ

https://www.tensorflow.org/tutorials/keras/classification を閲覧

- **学習の繰り返し**ごとに**損失が減少**し**正解率が上昇**する様子を確認
- **ログイン不要．コード中の数値（128，10，epochs 等）の意味を考察**できる

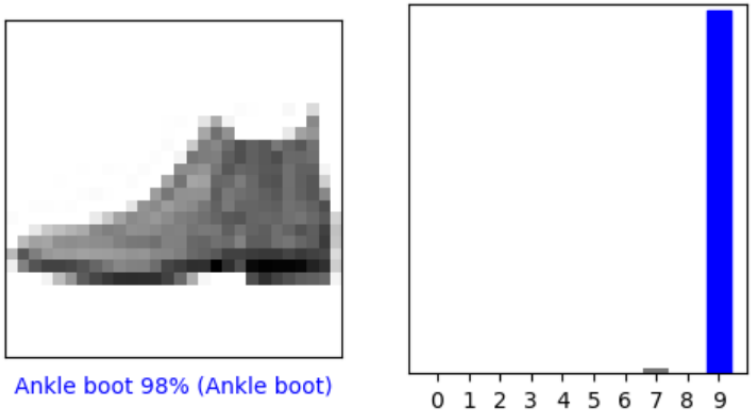
学習過程の観察

同じ訓練データを繰り返し用いて学習を行う。

訓練データ
 - 学習に用いるデータである。
繰り返し回数
 - 学習を繰り返す回数である（例：5回）。
損失（loss）と正解率（accuracy）
 - 学習の繰り返しごとに損失が減少し、同時に正解率が上昇する。

```
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3157 - accuracy: 0.8838
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2996 - accuracy: 0.8889
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2831 - accuracy: 0.8948
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2700 - accuracy: 0.8989
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2583 - accuracy: 0.9038
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2489 - accuracy: 0.9079
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2407 - accuracy: 0.9096
<keras.callbacks.History at 0x7f659e338ee0>
```

学習過程



10種類のどれに分類されたかを棒グラフで表示。
 - 青は正解、赤や黒は不正解を表す。