

# ae-5.ニューラルネットワーク とディープラーニング入門：基 礎理論から実践まで

(AI 演習) (全15回)

<https://www.kkaneko.jp/ai/ae/index.html>

金子邦彦



# 人工知能（AI）の学習のための指針



1. **実践重視**： AIツールを実際に使用し、機能に慣れる
2. **エラーを恐れない**： 実行においては、エラーの発生の可能性はある。エラーを恐れず、むしろ学習の一部として捉えるポジティブさが大切。
3. **段階的学習**： 基礎から応用へと段階的に学習を進め、AIの可能性を前向きに捉える



# アウトライン

1. イントロダクション（ニューロンと脳の構造）
2. ニューラルネットワークの基本構造
3. 活性化関数とその役割
4. フォワードプロパゲーション
5. バックプロパゲーション
6. ディープラーニングの特徴と応用

- Trinket は**オンライン**の Python、HTML 等の**学習サイト**
- ブラウザで動作
- 有料の機能と無料の機能がある
- **自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能**（そのとき、書き替えて実行も可能）
- **Python の標準機能**を登載、その他、次の外部ライブラリがインストール済み

matplotlib.pyplot, numpy, processing, pygal

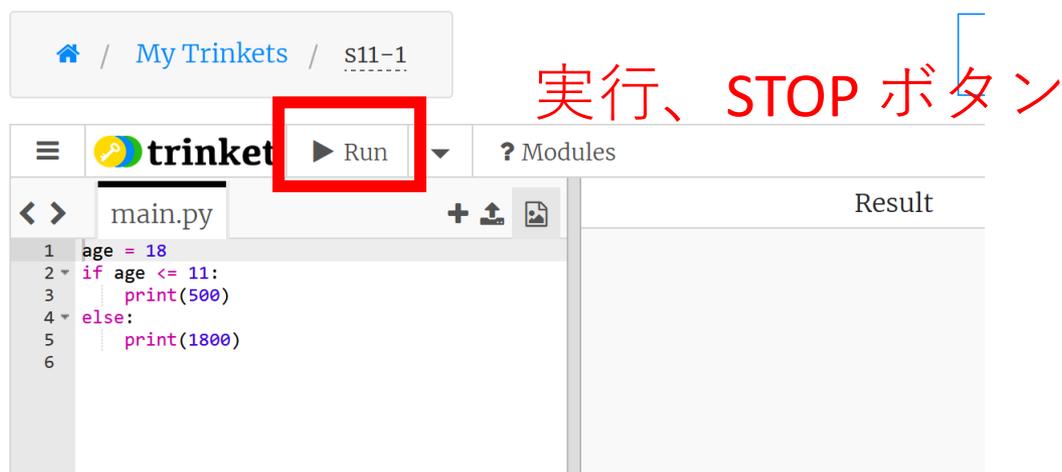


# trinket でのプログラム実行

- trinket は Python, HTML などのプログラムを書き実行できるサイト

- <https://trinket.io/python/0fd59392c8>

のように、違うプログラムには違う URL が割り当てられる



ソースコードの  
メイン画面

実行結果

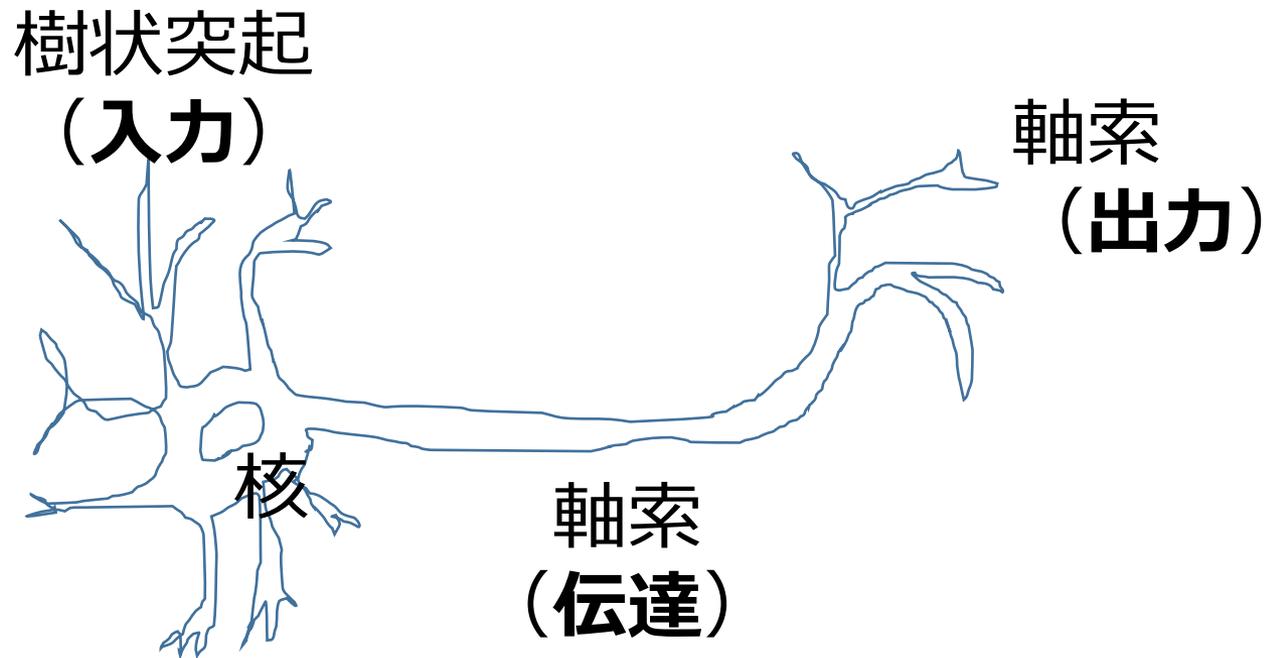
- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを書き替えて再度実行することも可能

# 5-1. イントロダクション (ニューロンと脳の構造)

# ニューロン



- **ニューロン**は、**脳や神経系**において、**情報伝達**、**記憶**、**情報処理**を行う基本要素.
- **ニューロンは樹状突起で入力を受け取り、軸索で信号を伝達する**. 核を中心に、入力部（樹状突起）、情報処理部（細胞体）、出力部（軸索）から構成される



# 脳のニューロン



## 脳のニューロンの数 (推定値)

- カタツムリ 11,000
- ロブスター 100,000
- アリ 250,000
- カエル 16,000,000
- ハツカネズミ 71,000,000
- タコ 500,000,000
- ネコ 760,000,000
- ヒト **86,000,000,000**  
~ 100,000,000,000
- アフリカゾウ 257,000,000,000

- 生物の脳には、種によって異なる数のニューロンが存在
- 人間の脳には**約860億～1000億個**のニューロンが存在
- ニューロンが複雑なネットワークを形成している

Wikipedia の記事:

<https://ja.wikipedia.org/wiki/%E5%8B%95%E7%89%A9%E3%81%AE%E3%83%8B%E3%83%A5%E3%83%BC%E3%83%AD%E3%83%B3%E3%81%AE%E6%95%B0%E3%81%AE%E4%B8%80%E8%A6%A7> より

# 脳や神経系の研究の進展



1900年頃: ニューロンによる情報伝達の理解.

1980年頃: 脳機能マッピングによる脳の機能解明 (どの部分がどの機能を持つか). 生きた脳を対象とした測定が行われるようになってきた.

2010年頃: 異なる脳領域の連携の観察

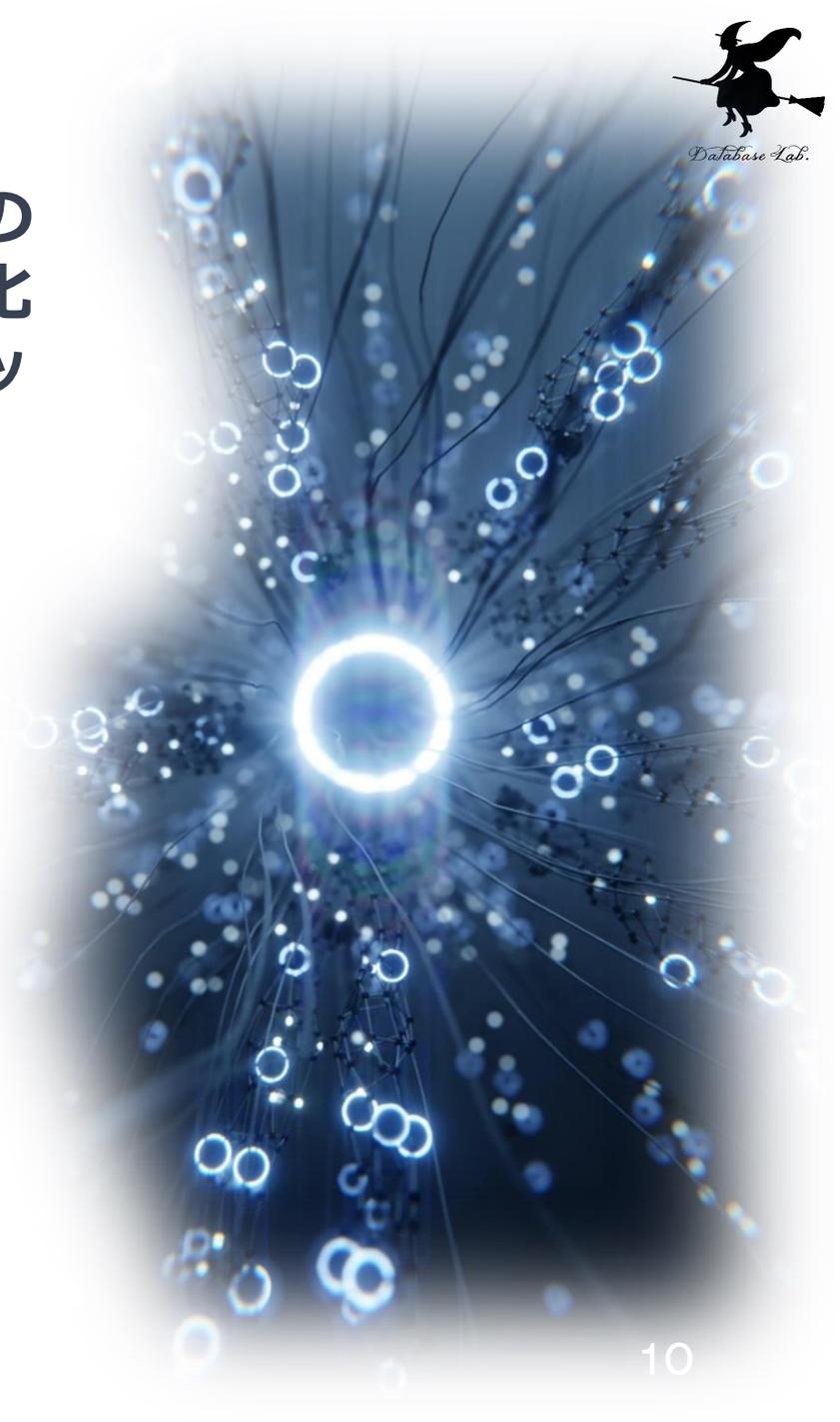
脳の仕組みと機能について洞察が得られるように

# ニューラルネットワーク



ニューラルネットワークは、入力の重みづけ、合計とバイアス、活性化関数の適用を行うニューロンがネットワークを形成する

各ニューロンは複数の入力を受け取り、重み付け、合計とバイアスを適用した後、活性化関数によって出力を決定



# ニューラルネットワークの歴史



1940年代：ニューラルネットワークの誕生

1960年代：バックプロパゲーションの誕生（ニューラルネットワークの学習の基本的な仕組み）

1986年：Rumelhartらによるバックプロパゲーションの再発見

2010年代：多層化，正規化，ReLU，ドロップアウト  
など，ニューラルネットワークの**新技術の登場**  
(コンピュータで扱えるニューロン数は年々増加)

# ニューラルネットワークの進展傾向

年	コンピュータで扱えるニューロン数の規模
2010年	100,000個
2020年	2,000,000個
2030年	50,000,000個
2040年	1,000,000,000個
2050年	20,000,000,000個

## 【将来傾向】

- **規模は増加傾向**
- GPU（プロセッサの一種）、クラウドコンピューティング（大規模計算の基盤）の**発展**
- 人間の脳の**約860～1000億個**のニューロンを模倣することも、あながち不可能ではないかも

所説あります

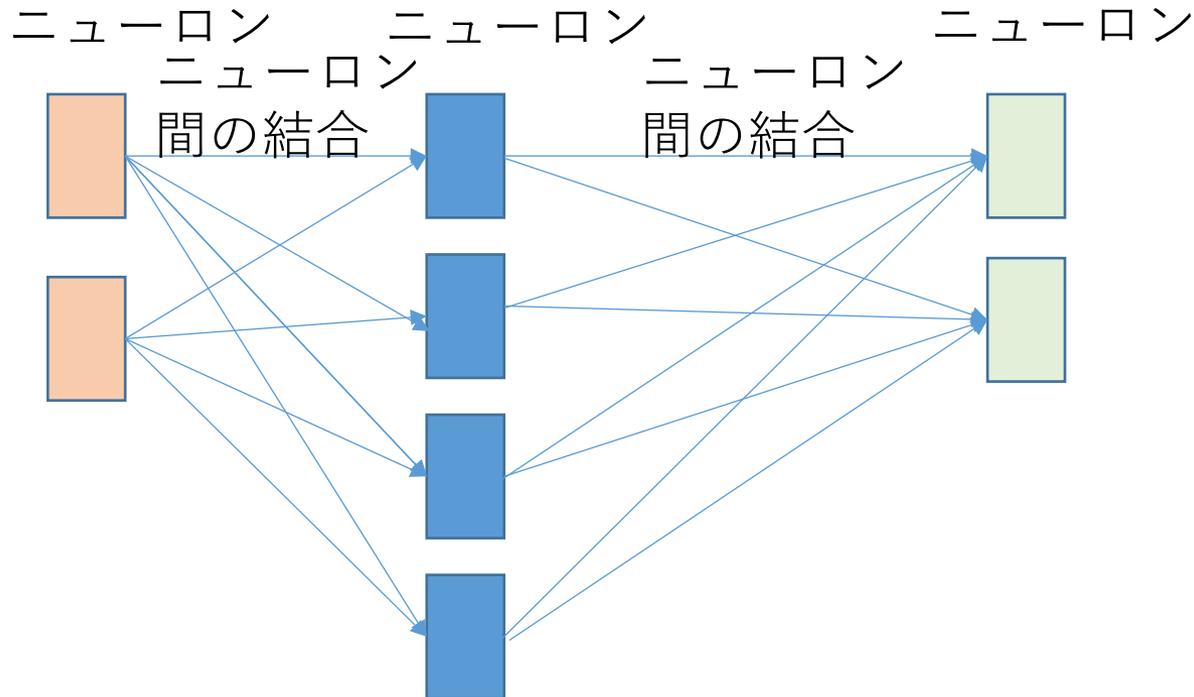
2055年ごろには1000億を超えるかも？

## 5-2. 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仕組み



ニューラルネットワークは、入力の重みづけ、合計とバイアス、活性化関数の適用を行うニューロンがネットワークを形成する



# 入力の重みづけ, 合計とバイアス, 活性化関数の適用



- 1. 入力の重みづけ:** 各入力に対応する重みを掛ける. 例:  
 $0.3 \times 0.1$   $-0.5 \times 0.8$   $0.2 \times -0.5$
- 2. 合計とバイアス:** 重みづけした値の合計にバイアス (数値) を加える. 例  $0.03$   $-0.4$   $-0.1$  の合計は  $-0.47$ . これに  $0.2$  を加える
- 3. 活性化関数の適用:** 結果に, 活性化関数を適用し, 活性度を決定 (活性度が高いほどニューロンが強く反応していることを示す→次のニューロンの入力になる)

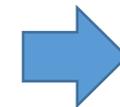
$$0.3 \times 0.1 \Rightarrow 0.03$$

$$-0.5 \times 0.8 \Rightarrow -0.4$$

$$0.2 \times -0.5 \Rightarrow -0.1$$

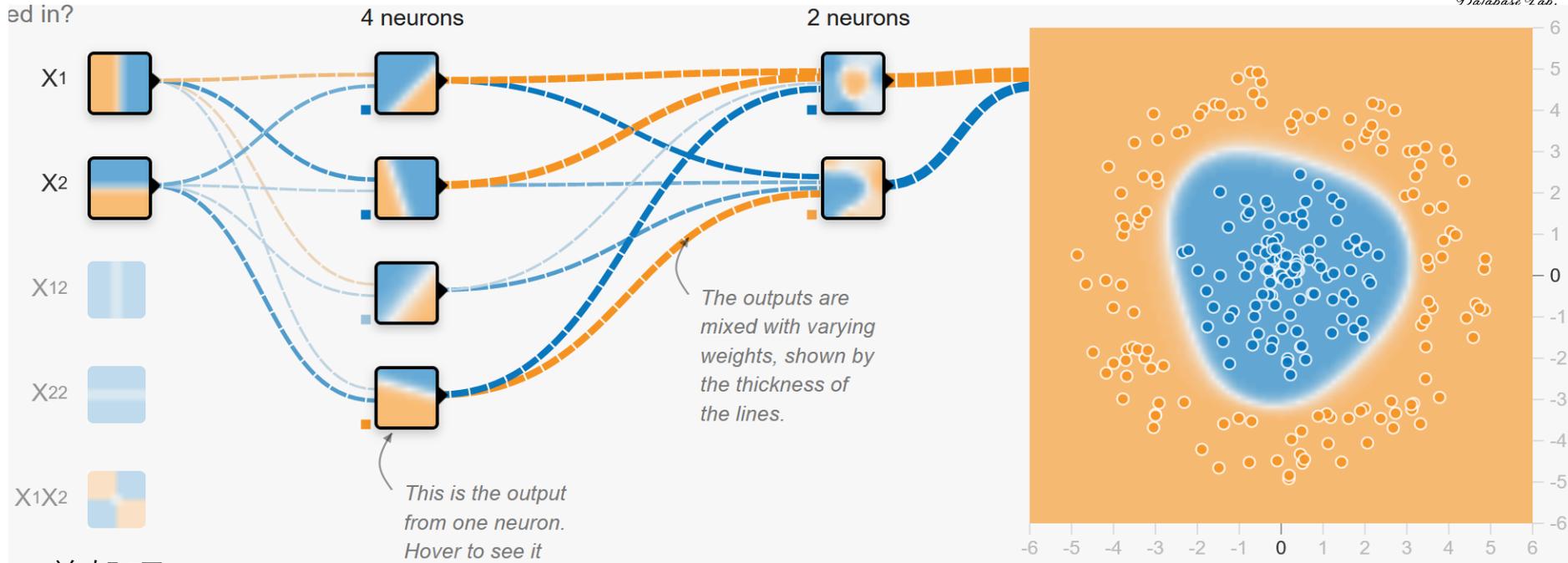
入力 重み

合計  
 $-0.47$



$-0.27$  に  
活性化関数  
を適用

バイアス  $0.2$



前処理

(データが青い部分にあれば活性化)

データが中央にあれば活性化

→結合→ 1層目      →結合→ 2層目

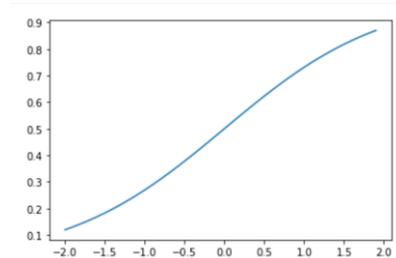
ニューラルネットワーク

## 5-3. 活性化関数とその役割

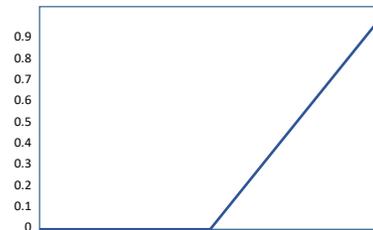
# 活性化関数



- ニューロンの活性度を決定する関数
- 入力の重みづけ, 合計とバイアスの結果に, 活性化関数を適用し, ニューロンの活性度を決定
- ReLUやシグモイドなど, さまざまな種類

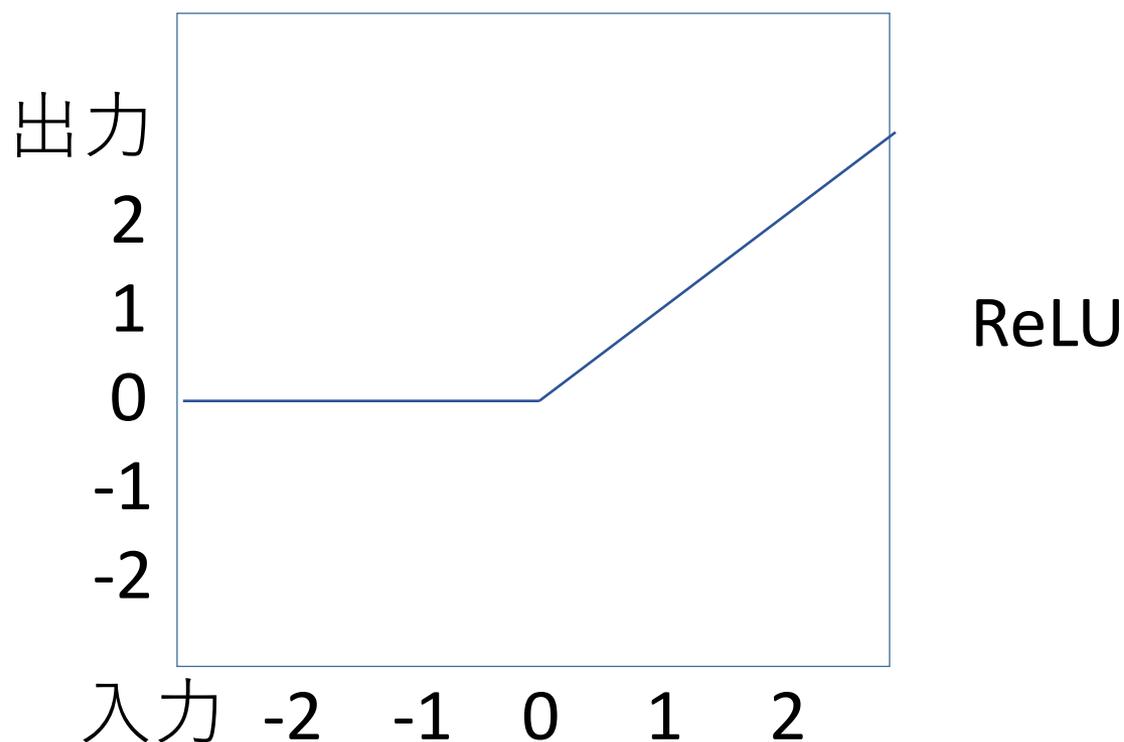


シグモイド



ReLU

- 2011年に提案された活性化関数
- 入力が0以下の場合は0を返し，0より大きい場合はその値をそのまま返す特性を持つ
- シンプルな構造ながら高い性能を示し，現代のディープラーニングで広く使用されている。



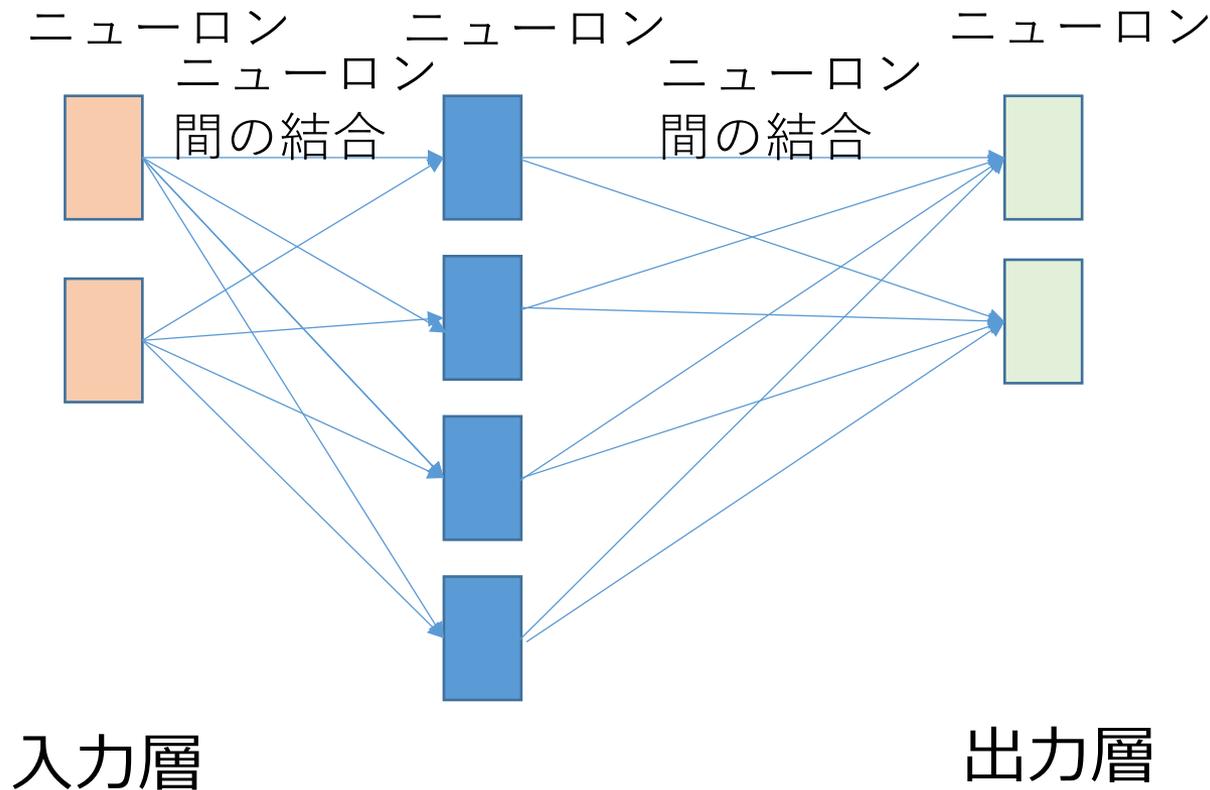


## 5-4. フォワードプロパ ゲーション

# フォワードプロパゲーション

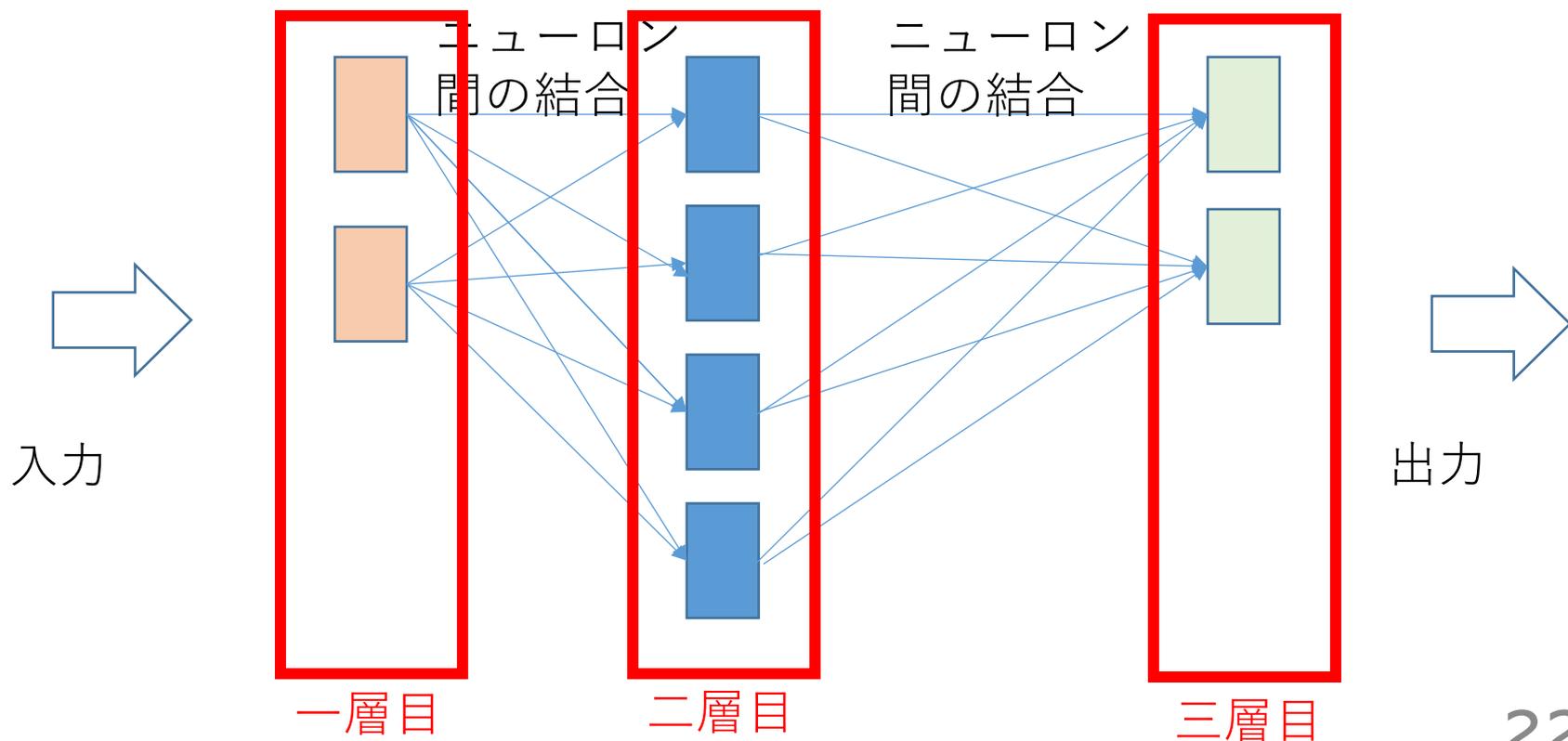


- フォワードプロパゲーションは、データが入力層から出力層へと順方向に伝播する基本的な処理過程である。
- 各層のニューロンは、前層からの入力を処理し、その結果を次層に伝達する。



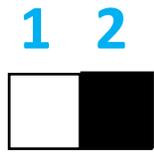
# ニューラルネットワークの層構造

ニューラルネットワークの基本構造であり，複数の層から成る．データは入力から出力への一方向に流れ，各層は同じ種類のニューロンで構成される．



# ①入力

数値, 複数可      例 : 1 0

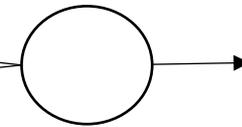


入力

入力

1 1

2 0



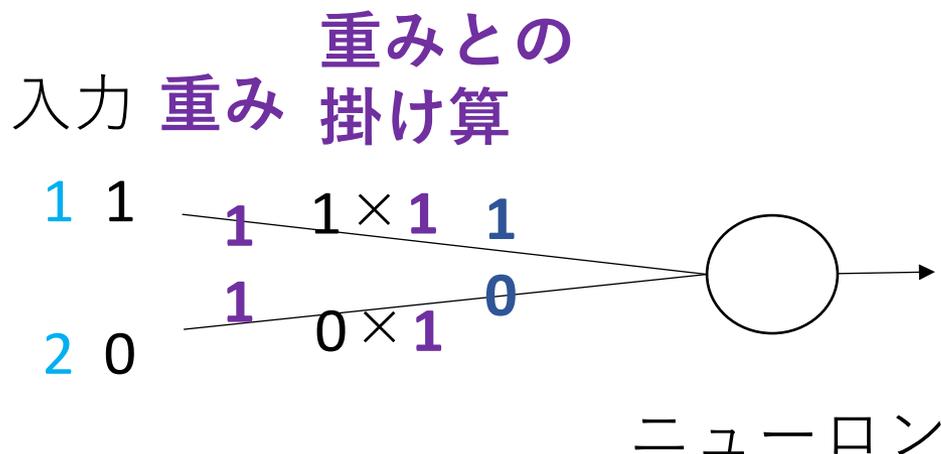
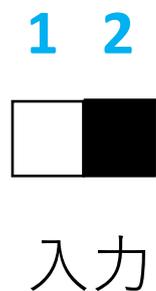
ニューロン

白を 1  
黒を 0 とする

## ② 入力の重みづけ

- ニューロン間の結合には**重み**が設定され、**信号の伝達強度**を決定する。
- 各入力値に対して、それぞれの対応する**重みを掛ける**

例：  $1 \times 1$   $0 \times 1$

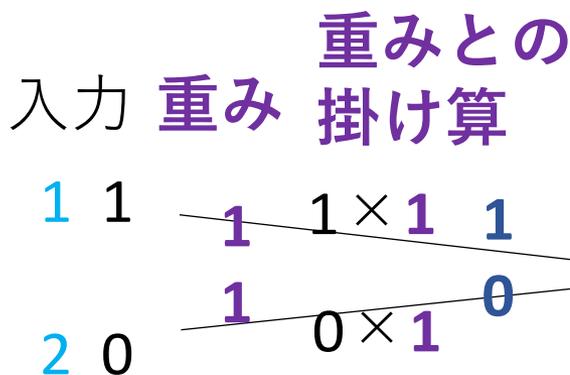
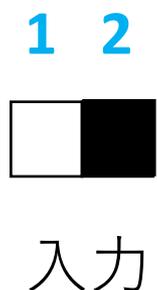


白を 1  
黒を 0 とする

### ③ バイアス

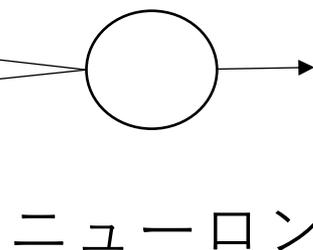
- ・ バイアスはニューロンの活性化のしやすさを調整する.
- ・ 重みづけした値の合計にバイアス (数値) を加える.

例 10 の合計は 1. これに -0.5 を加える



1 と 0 の合計である 1 に  
バイアス -0.5 を加算

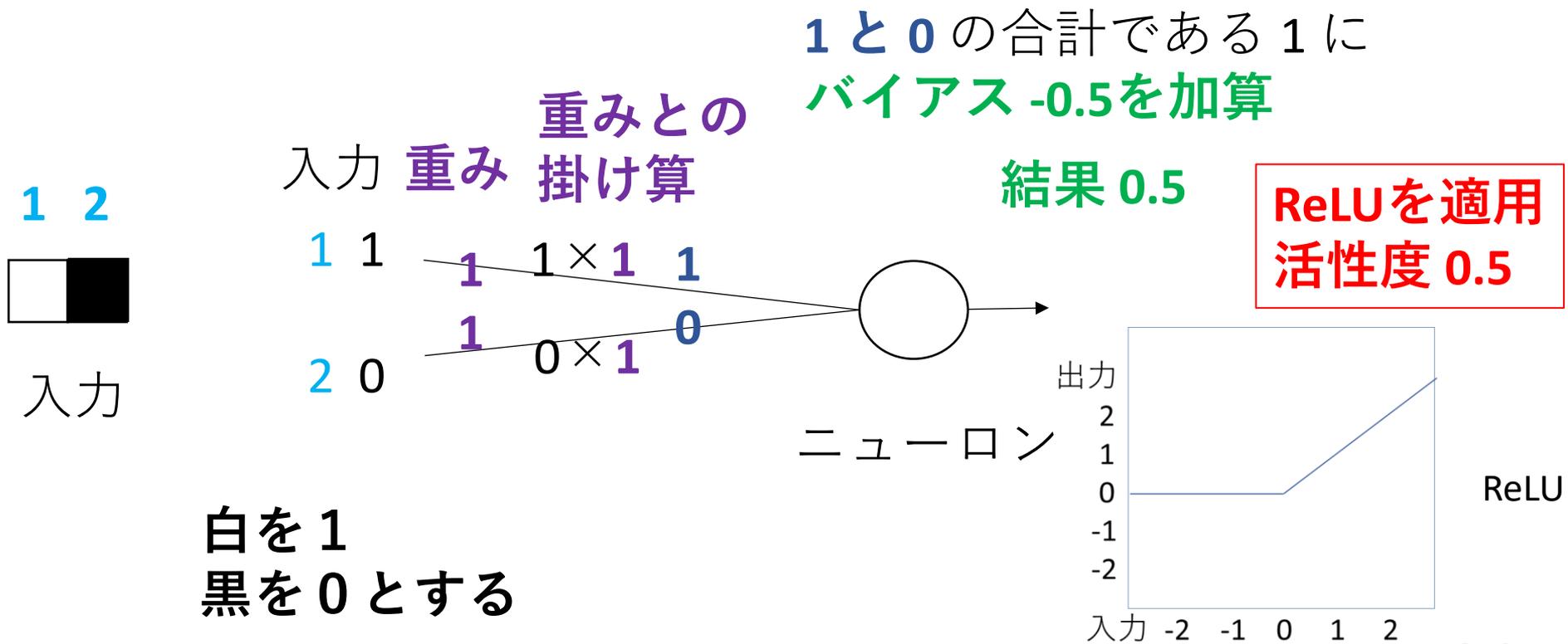
結果 0.5



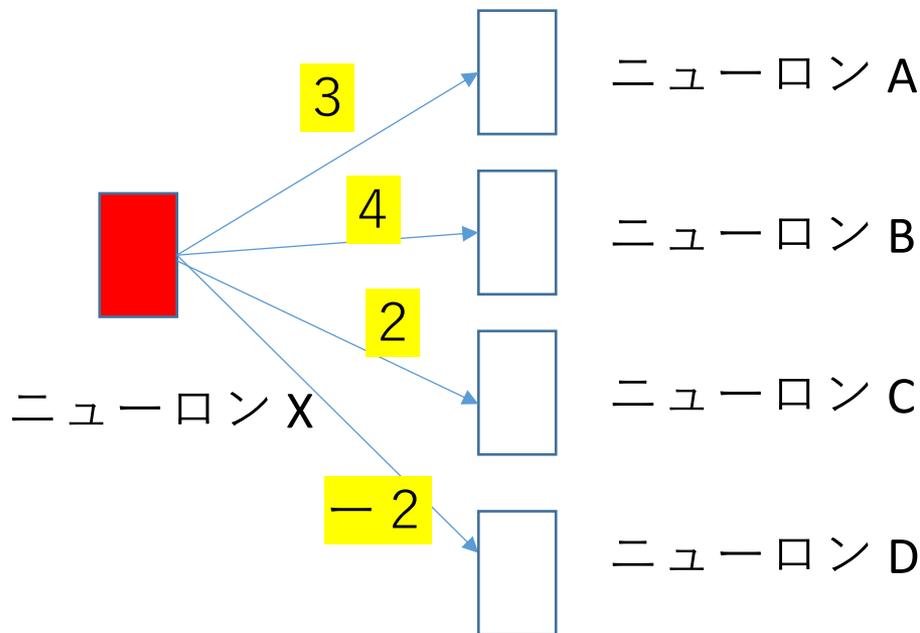
白を 1  
黒を 0 とする

## ④活性化関数の適用

- 結果に、**活性化関数**を適用。ニューロンの**活性度**を得る  
(→次のニューロンの入力になる)
- 値が大きいほどニューロンが活性化していることを示す**



活性度（数値）は、次のニューロンの入力になる



**次のニューロンの入力**  
= **前のニューロンの出力**  
× **結合の重み**

**結合の重みは、**  
**結合ごとに違う値**

# 演習 1 ReLU



# 演習の狙い

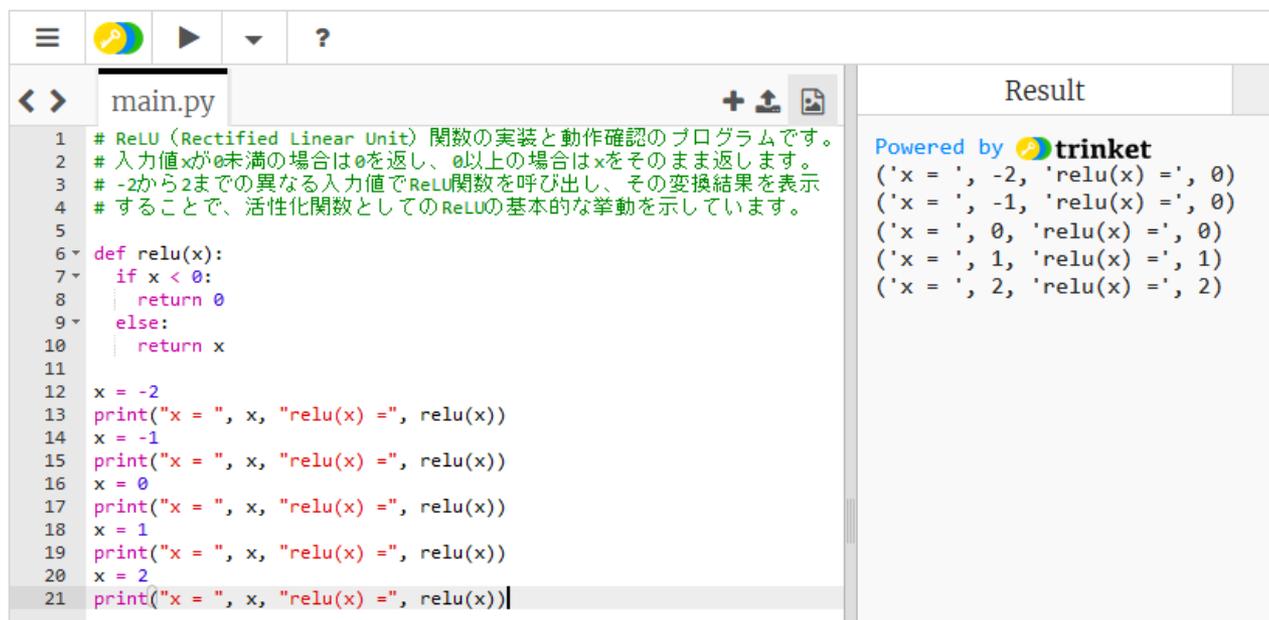


- **ReLU関数の特徴**（負の入力を0に変換、正の入力はそのま  
ま出力）を**直感的に理解**
- **異なる入力値（負数、0、正数）**で実験して**結果を確認す**  
る体験
- **if と else を使うプログラミング**（条件分岐）

## ① trinket の次のページを開く

<https://trinket.io/python/61c6503fcada>

## ② 実行結果が、次のように表示されることを確認



The screenshot shows the Trinket Python IDE interface. On the left, a code editor displays a Python script named 'main.py'. The script defines a 'relu(x)' function and tests it with values -2, -1, 0, 1, and 2. On the right, the 'Result' panel shows the output of the program, which is a list of tuples containing the input 'x' and the corresponding 'relu(x)' value.

```
1 # ReLU (Rectified Linear Unit) 関数の実装と動作確認のプログラムです。
2 # 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。
3 # -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
4 # 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。
5
6 def relu(x):
7     if x < 0:
8         return 0
9     else:
10        return x
11
12 x = -2
13 print("x = ", x, "relu(x) =", relu(x))
14 x = -1
15 print("x = ", x, "relu(x) =", relu(x))
16 x = 0
17 print("x = ", x, "relu(x) =", relu(x))
18 x = 1
19 print("x = ", x, "relu(x) =", relu(x))
20 x = 2
21 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)
```

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

# 演習 2 入力の重みづけ, 合計とバイアス, 活性化関数の適用

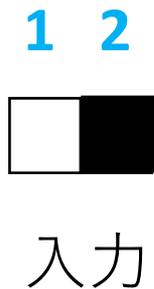


# 演習の狙い

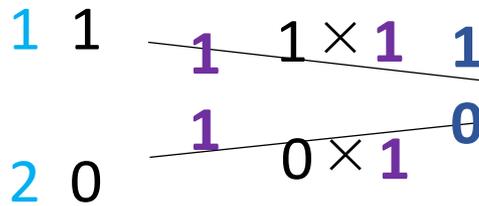


- **重み, バイアス, 活性化関数 (ReLU)** という3つの基本要素の役割と連携を具体的に理解できる.
- これは複雑なニューラルネットワークを理解するための重要な第一歩となる.

## 【入力が10のとき】

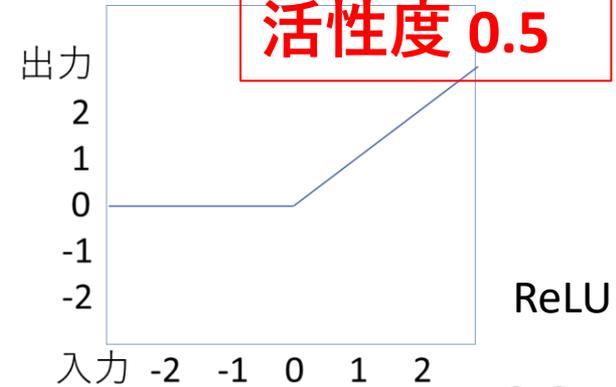
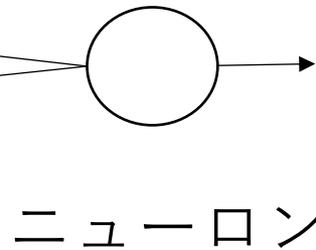


入力 重み 重みとの  
掛け算



1と0の合計である1に  
バイアス -0.5を加算

結果 0.5

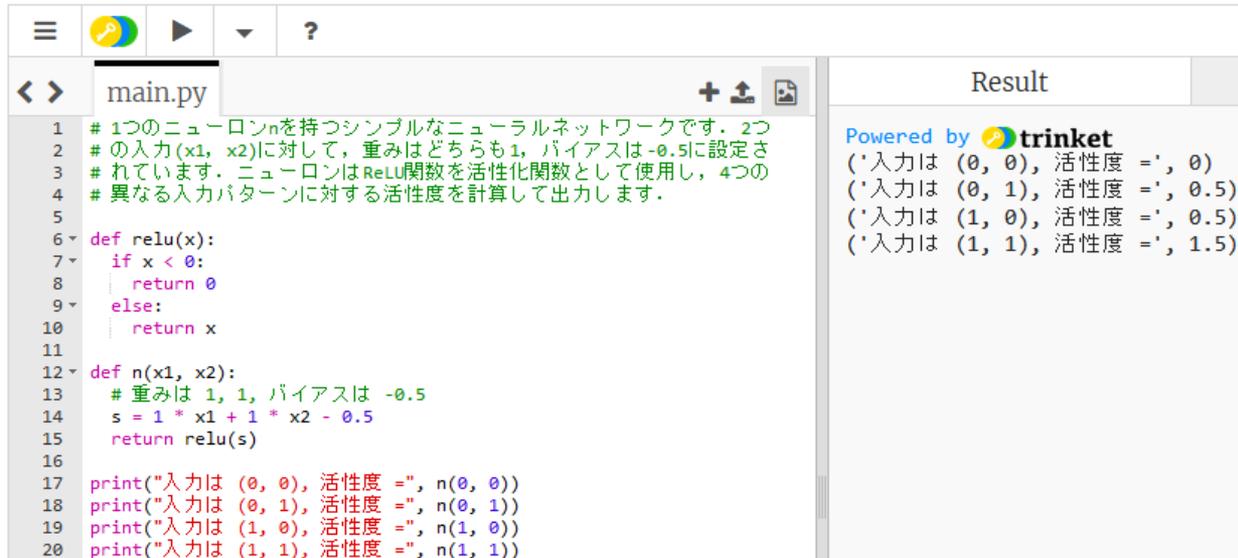


白を1  
黒を0とする

## ① trinket の次のページを開く

<https://trinket.io/python/1c339b660ee1>

## ② 実行結果が、次のように表示されることを確認



```
main.py
1 # 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
2 # 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
3 # 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
4 # 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。
5
6 def relu(x):
7     if x < 0:
8         return 0
9     else:
10        return x
11
12 def n(x1, x2):
13     # 重みは 1, 1, バイアスは -0.5
14     s = 1 * x1 + 1 * x2 - 0.5
15     return relu(s)
16
17 print("入力は (0, 0), 活性度 =", n(0, 0))
18 print("入力は (0, 1), 活性度 =", n(0, 1))
19 print("入力は (1, 0), 活性度 =", n(1, 0))
20 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)

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

# ニューラルネットワークの基本まとめ

- 各ニューロンは、前の層からの**入力**を受け取る
- **入力**には**重み**が関連付けられており、各データがニューロンの活性化に与える影響を調整する
- ニューロンは**バイアス**を持ち、**入力**に**重み**をかけた**総和**に**バイアス**を加算する
- **活性化関数**による**処理**を行い、その結果を**次の層のニューロン**に伝える
- ReLUなどの**活性化関数**がある。
- ニューロンは**特定のパターン**の**入力**に対して**活性化**し、その**活性度**は**入力**、**重み**、**バイアス**、**活性化関数**によって決まる

# ニューラルネットワークによるパターン認識の例



## 1. 入力の重みづけ:

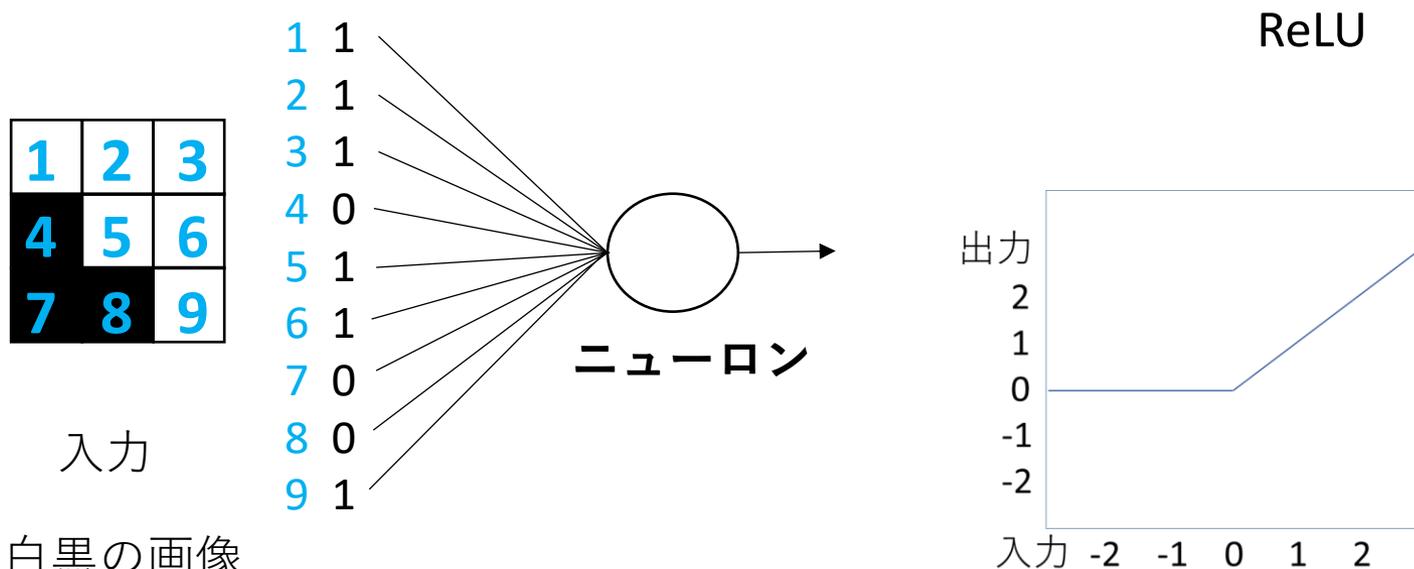
$1 \times 1, 1 \times 1, 1 \times 1, 0 \times 0, 1 \times 1, 1 \times 1, 0 \times 0, 0 \times 0, 1 \times 1$

## 2. 合計とバイアス:

合計 6 にバイアス -5 を足す. 結果 1

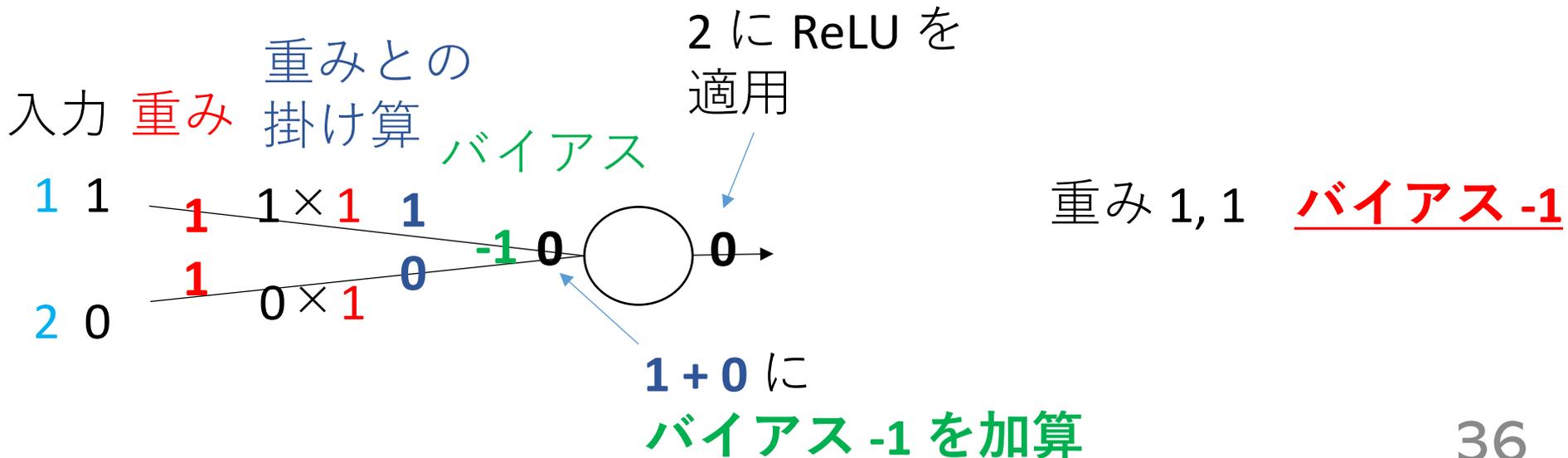
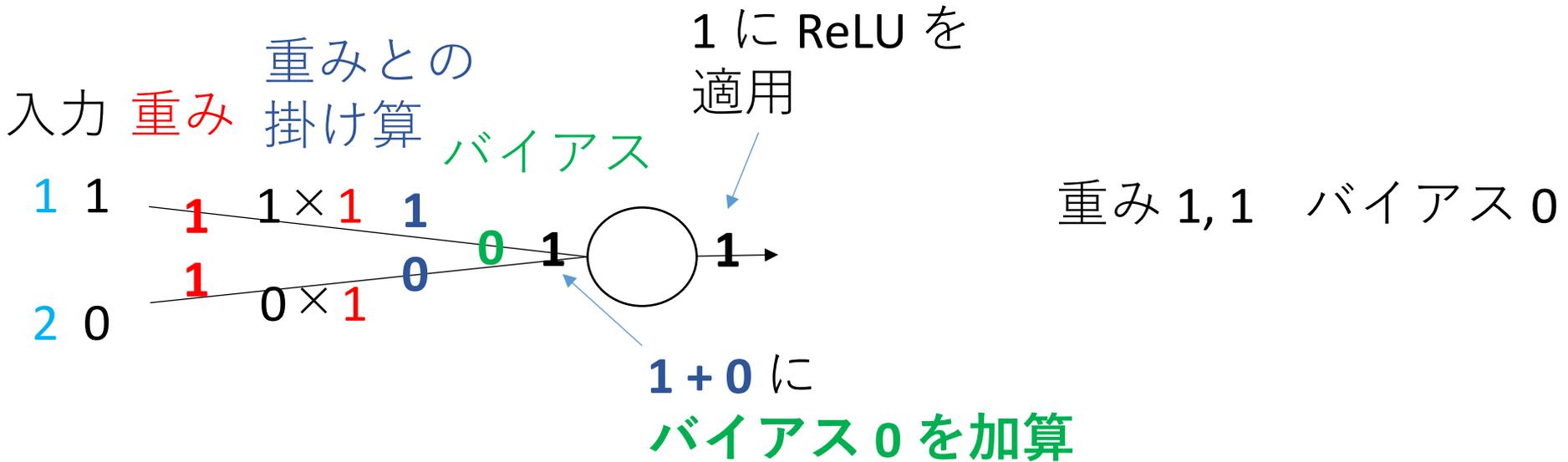
## 3. 活性化関数の適用

結果 1 に ReLU を適用した場合は, 活性度 1

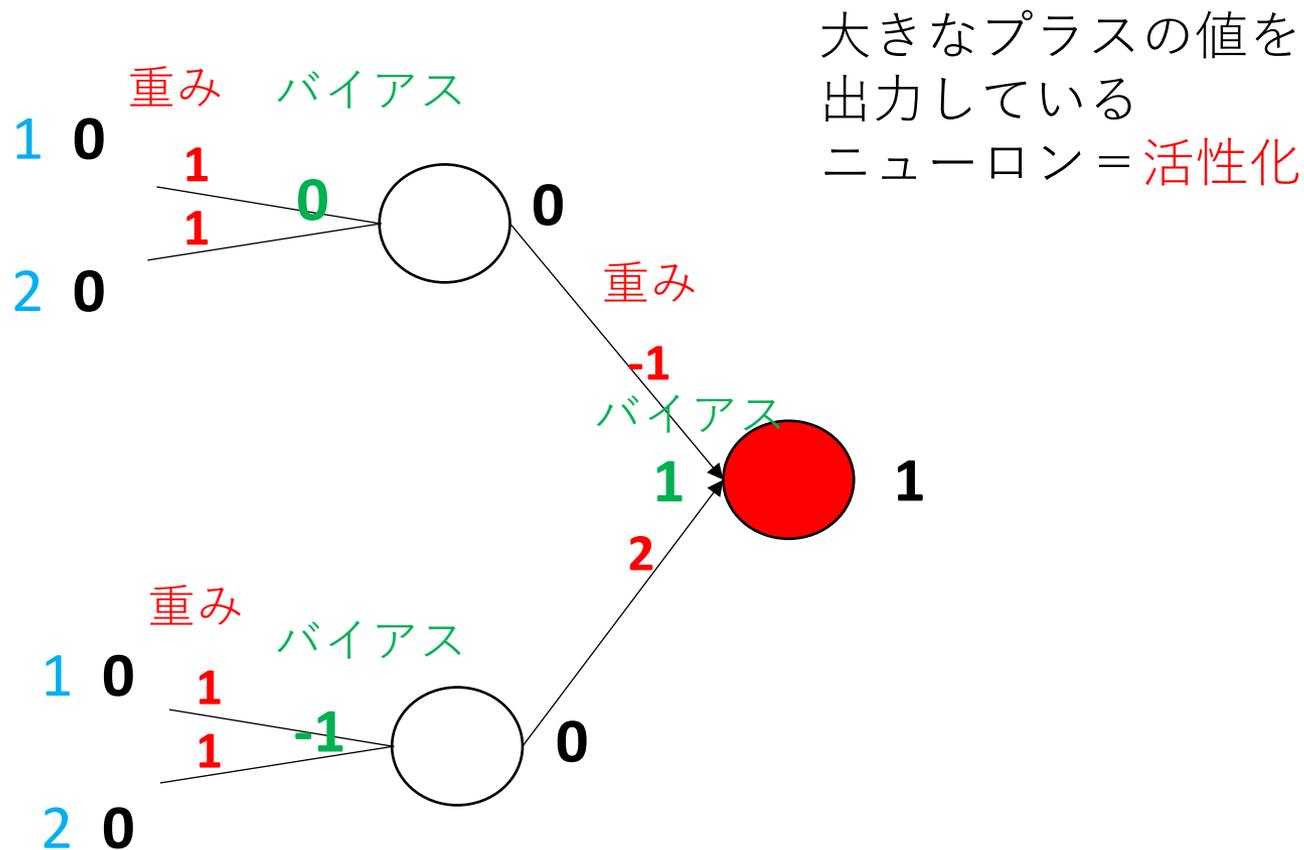


白黒の画像  
(画素は 0 または 1)

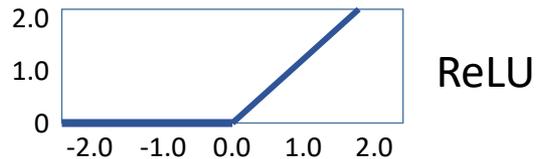
# パラメータ（バイアス）の変化による出力の変化



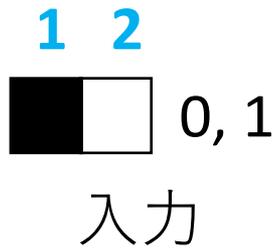
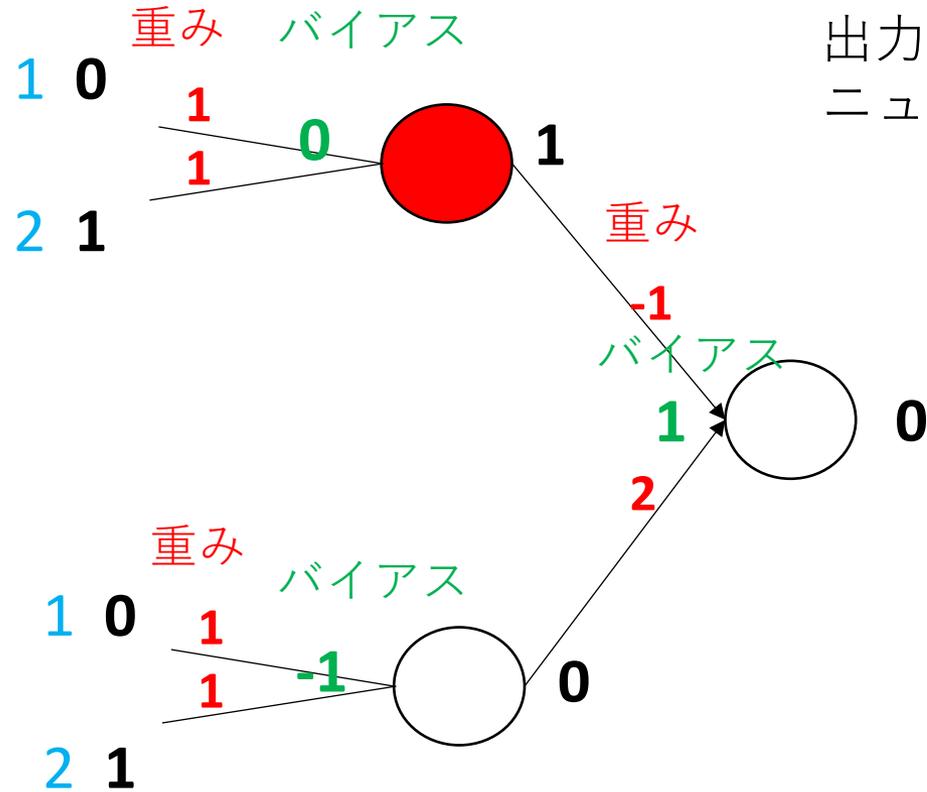
1 2  
0,0  
入力



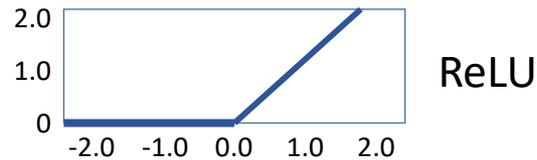
3つのニューロンの活性化関数はすべて ReLU



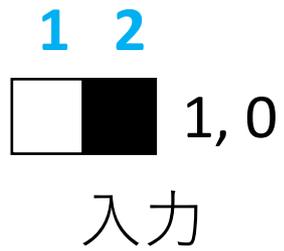
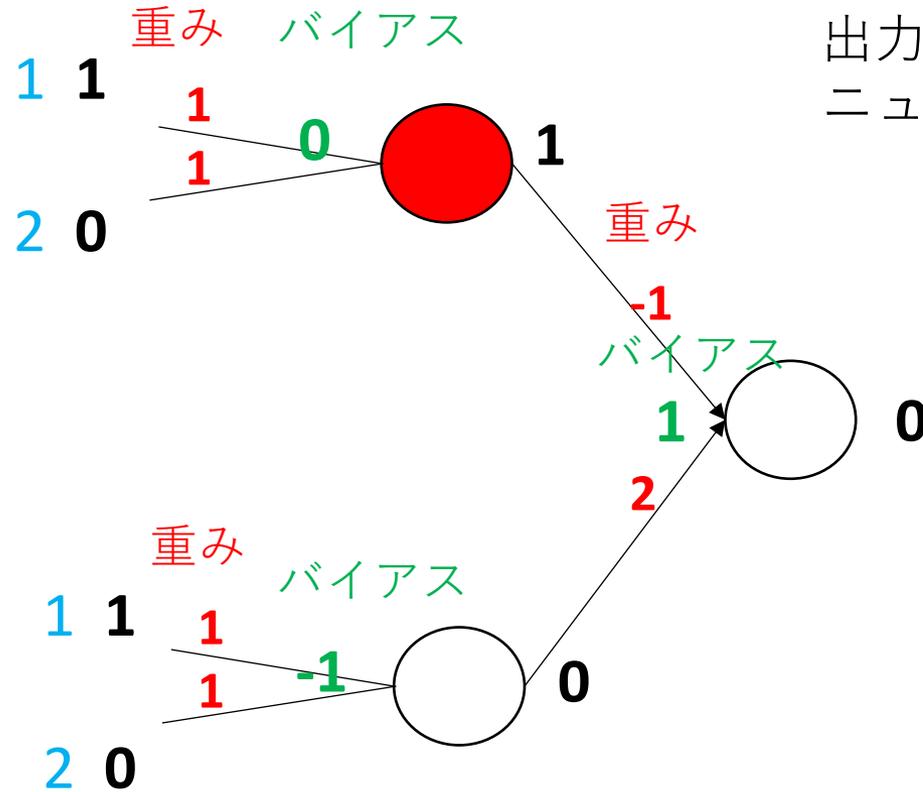
大きなプラスの値を  
出力している  
ニューロン = 活性化



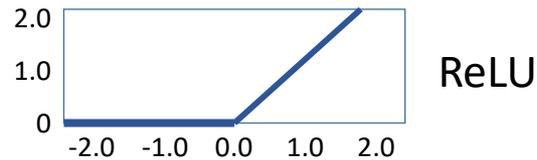
3つのニューロンの活性化関数はすべて ReLU



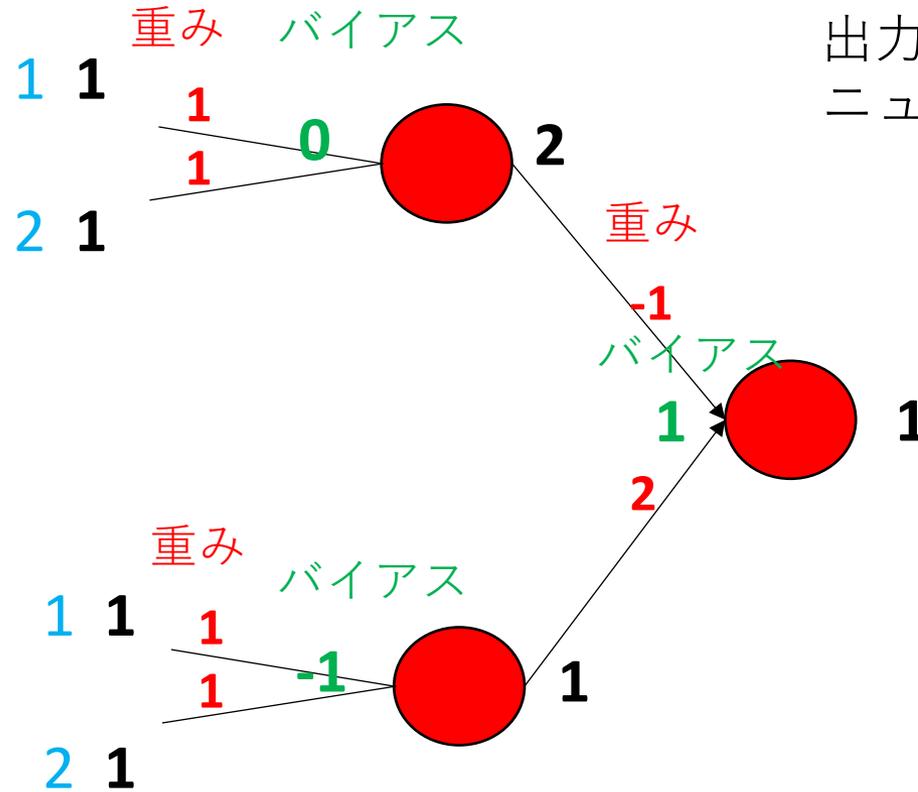
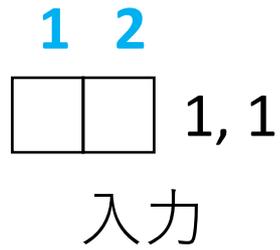
大きなプラスの値を  
出力している  
ニューロン = 活性化



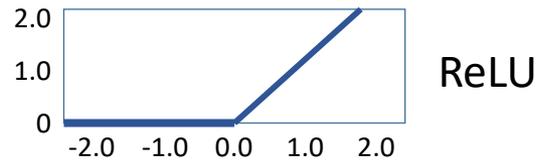
3つのニューロンの活性化関数はすべて ReLU



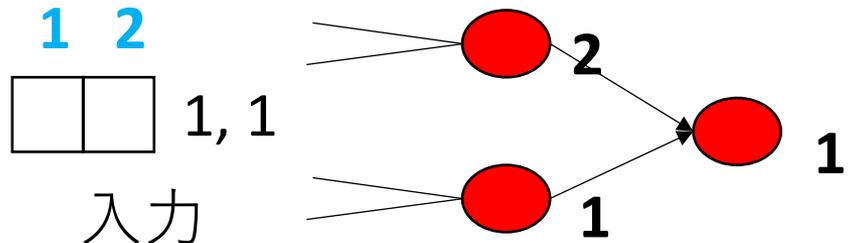
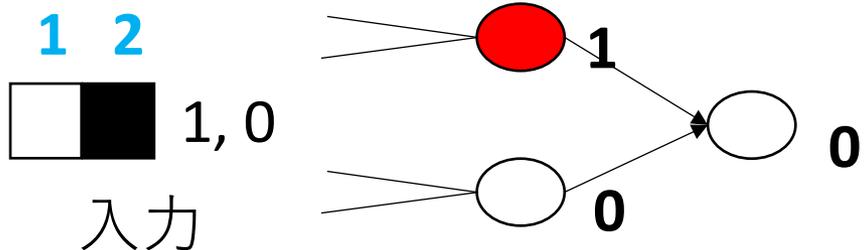
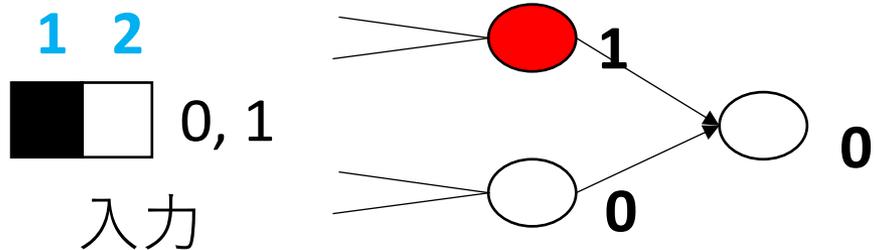
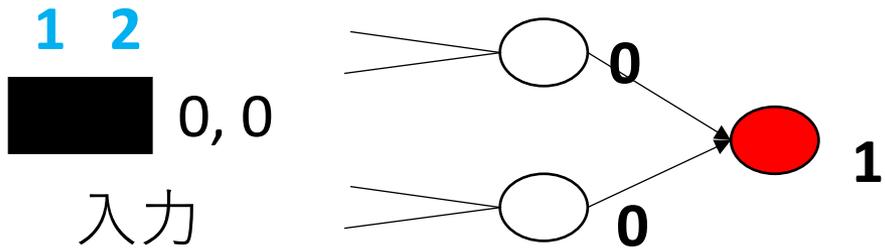
大きなプラスの値を  
出力している  
ニューロン = 活性化



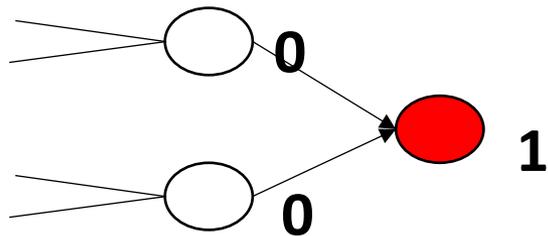
3つのニューロンの活性化関数はすべて ReLU



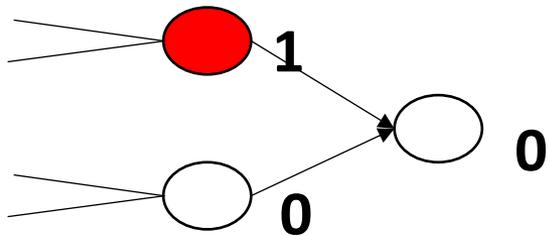
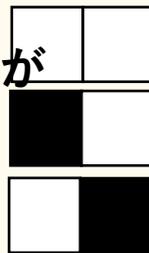
# 入力の変化による活性度の変化



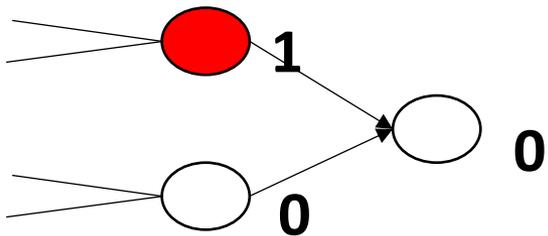
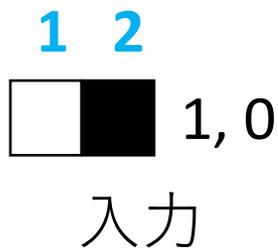
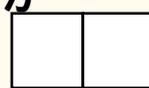
# ニューロンは特定のパターンに応じて活性化される



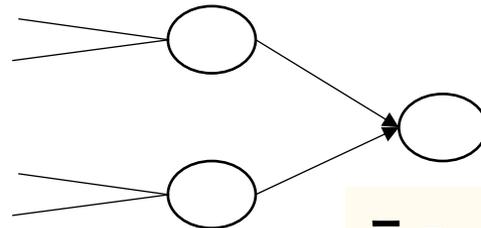
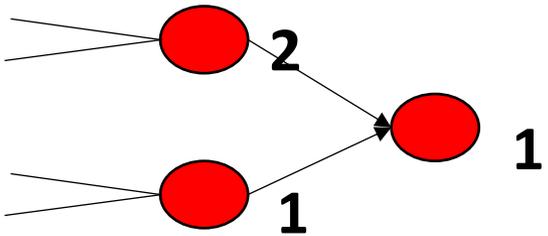
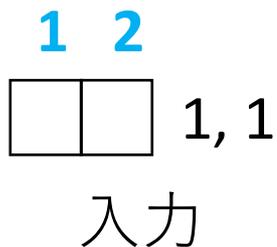
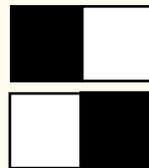
ニューロンが  
識別する  
パターン



ニューロンが  
識別する  
パターン



ニューロンが  
識別する  
パターン



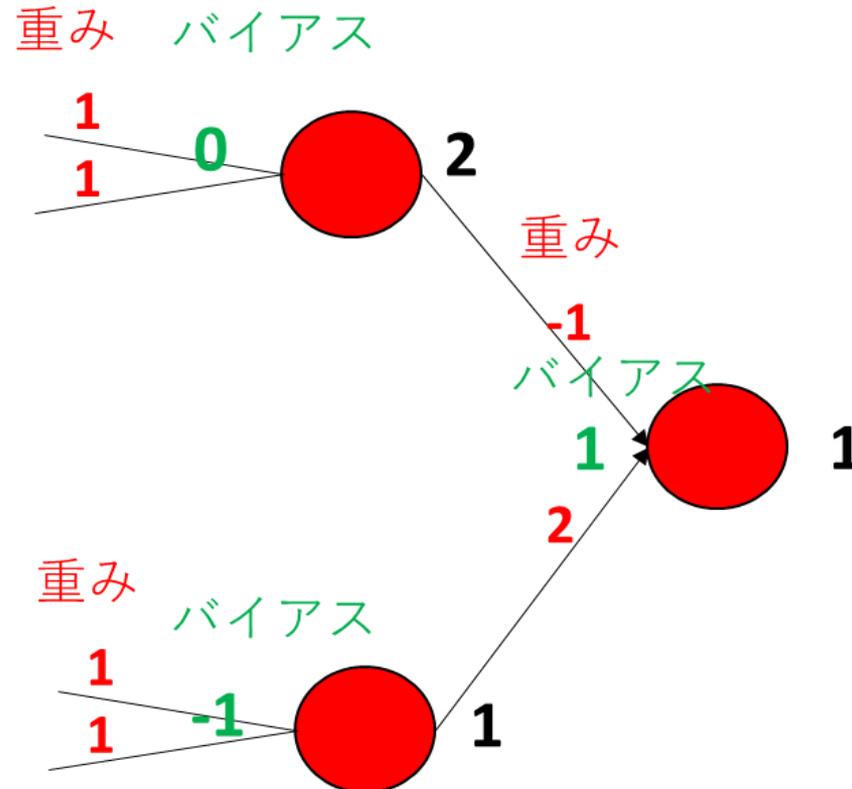
# 演習3 ニューラルネットワーク ワーク



# 演習の狙い



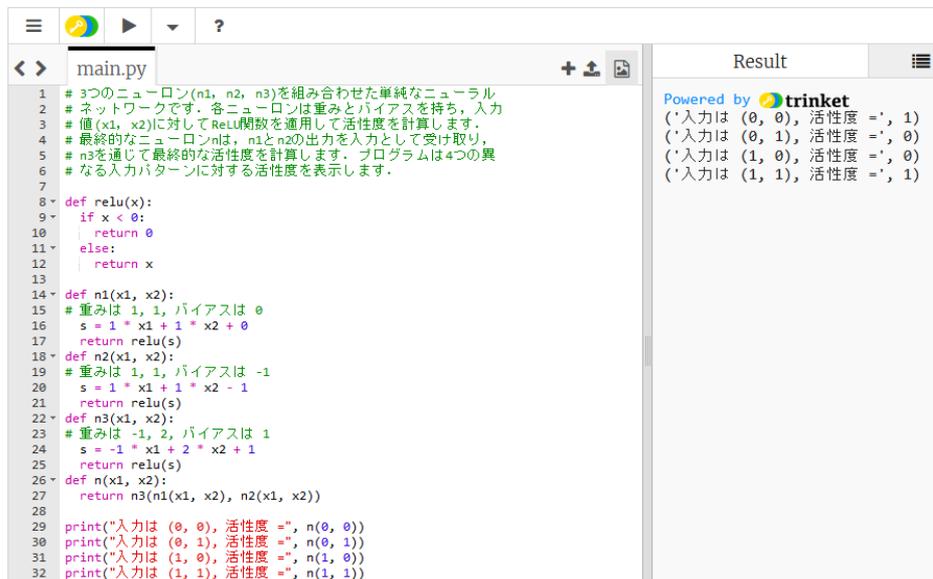
- 複数のニューロンが層を形成
- 1層の場合より複雑な判断が可能になる
- ニューラルネットワークによる高度な情報処理の基礎を体験



## ① trinket の次のページを開く

<https://trinket.io/python/3cbc3f3ed057>

## ② 実行結果が、次のように表示されることを確認



```
main.py
1 # 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
2 # 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
3 # 値(x1, x2)に対してrelu関数を通して活性化を計算します。
4 # 最終的なニューロンn3は、n1とn2の出力を入力として受け取り、
5 # n3を通じて最終的な活性化を計算します。プログラムは4つの異
6 # なる入力パターンに対する活性化を表示します。
7
8 def relu(x):
9     if x < 0:
10        | return 0
11    else:
12        | return x
13
14 def n1(x1, x2):
15     # 重みは 1, 1, バイアスは 0
16     s = 1 * x1 + 1 * x2 + 0
17     return relu(s)
18
19 def n2(x1, x2):
20     # 重みは 1, 1, バイアスは -1
21     s = 1 * x1 + 1 * x2 - 1
22     return relu(s)
23
24 def n3(x1, x2):
25     # 重みは -1, 2, バイアスは 1
26     s = -1 * x1 + 2 * x2 + 1
27     return relu(s)
28
29 def n(x1, x2):
30     return n3(n1(x1, x2), n2(x1, x2))
31
32 print("入力は (0, 0), 活性化度 =", n(0, 0))
33 print("入力は (0, 1), 活性化度 =", n(0, 1))
34 print("入力は (1, 0), 活性化度 =", n(1, 0))
35 print("入力は (1, 1), 活性化度 =", n(1, 1))
```

Result

Powered by  trinket

```
('入力は (0, 0), 活性化度 =', 1)
('入力は (0, 1), 活性化度 =', 0)
('入力は (1, 0), 活性化度 =', 0)
('入力は (1, 1), 活性化度 =', 1)
```

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能



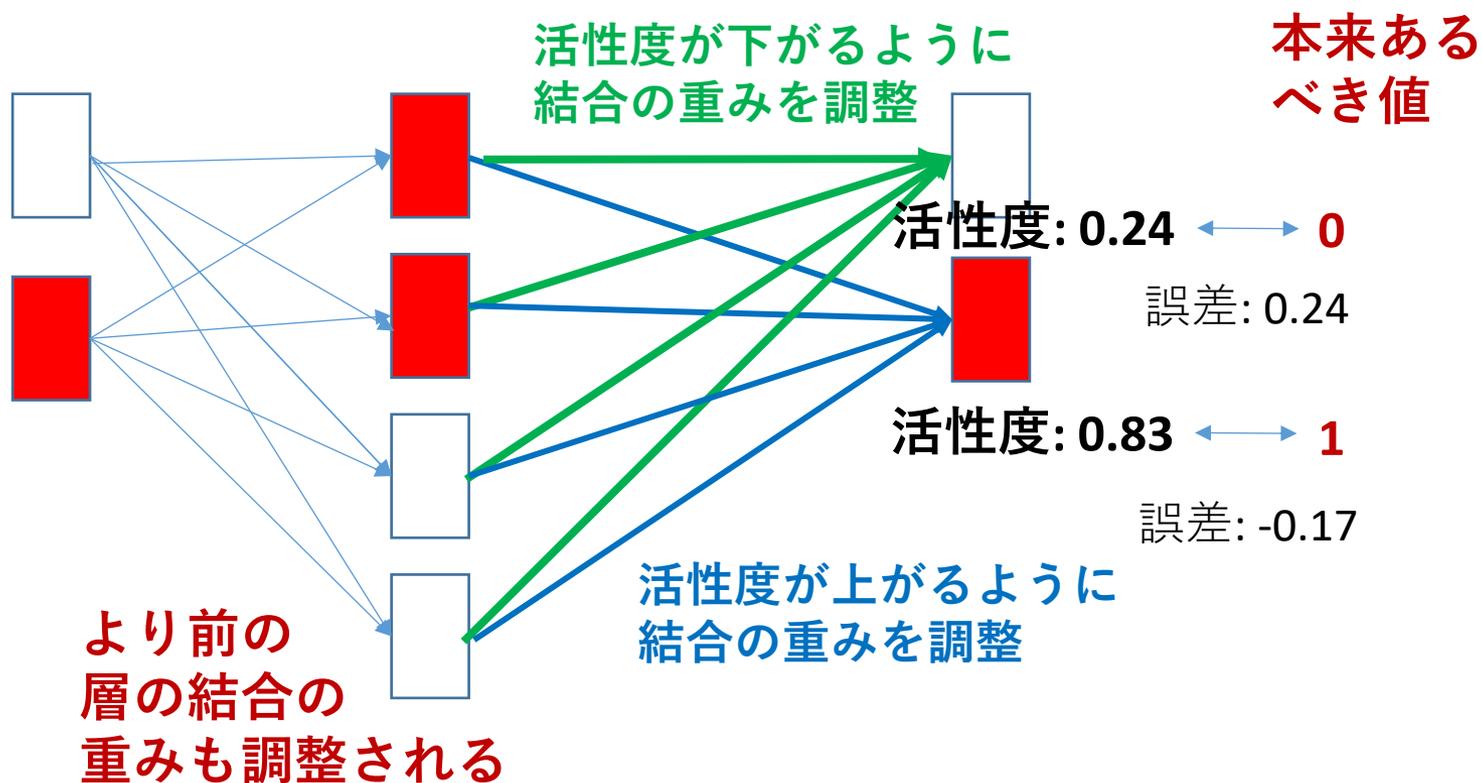
## 5-5. バックプロパゲーション

# バックプロパゲーション



- **バックプロパゲーション**は、**出力の誤差をもとに出力層から入力層へ向かって結合の重みとバイアスを調整する学習**の仕組み.
- 誤差を最小化するように**結合の重みとバイアスを自動調整**する
- 1986年にRumelhartらによって効果的な学習手法として確立

# ニューラルネットワークの動作イメージ



# 正解と誤差



正解は 2  
であるとする

活性度: 0.0068



誤差: 0.0068



あるべき値: 0

活性度: 0.1592



誤差: 0.1592



あるべき値: 0

活性度: 0.8340



誤差: - 0.1760



あるべき値: 1

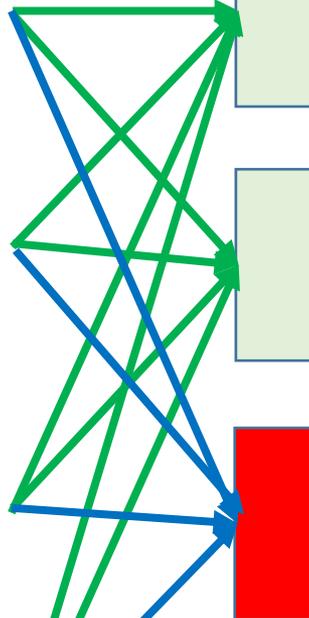
誤差をもとに、結合の重みを自動調節

# ニューラルネットワークの学習

正解は2  
であるとする



活性度が下がるように  
結合の重みを調整



誤差: 0.0068

←→ あるべき値: 0  
活性度: 0.0068

誤差: 0.1592

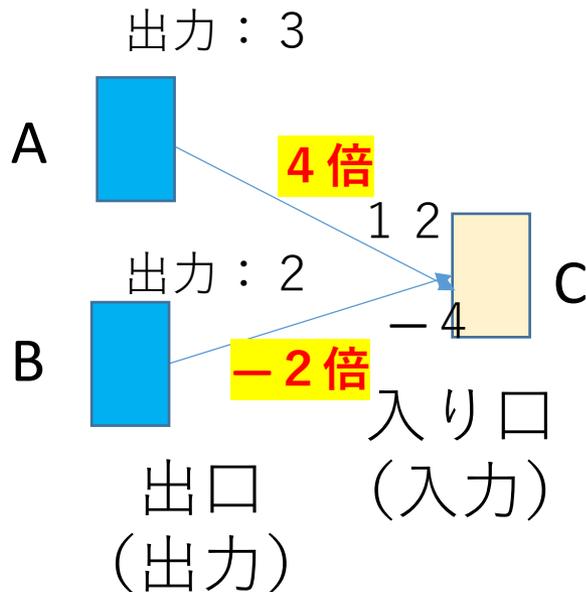
←→ あるべき値: 0  
活性度: 0.1592

誤差: - 0.1760

←→ あるべき値: 1  
活性度: 0.8340

活性度が上がるように  
結合の重みを調整

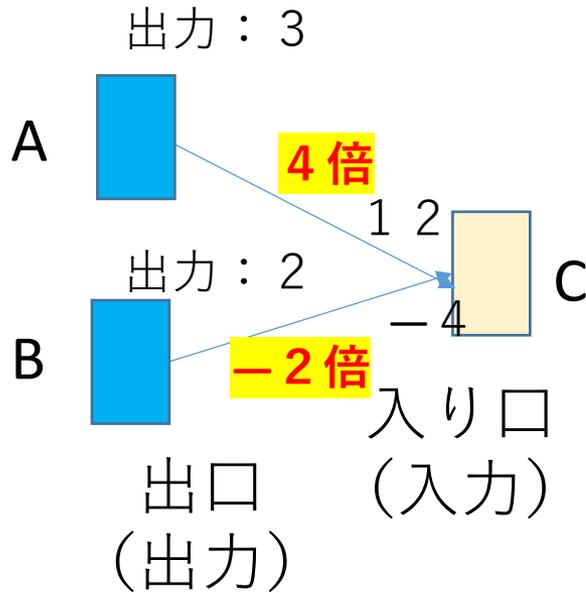
# 結合の重みの調整



ニューロン間の結合の  
「**〇〇倍**」  
は、学習の途中で**変化**する

ニューラルネットワークの  
学習は、  
望み通りの出力が得られる  
ように、「**〇〇倍**」の  
ところ（**結合の重み**）を**調整**  
すること

# 確認問題



次のような**ニューロン**がある

- ・ 入力の合計が **0以上** のとき、**活性化** し、 **1** を出力する
- ・ 入力の合計が **0未満** のとき、**非活性化** し、 **0** を出力する

ニューロンAの出力は3であるとする。

ニューロンBの出力が**2以下**のとき

ニューロンCは**活性化**

ニューロンBの出力が少しでも**2を超えた**とき

ニューロンCは**非活性化**

にしたい。結合の重みをどう調整するか？

(答え) 「**-2倍**」を「**-6倍**」へ

URL: <https://trinket.io/python/b1f04a9758>

```
main.py
1 def relu(x):
2     if x < 0:
3         return 0
4     else:
5         return x
6
7 def n(x1, x2):
8     s = 4 * x1 + -2 * x2
9     return relu(s)
10
11 print("n(3, 0) =", n(3, 0))
12 print("n(3, 1) =", n(3, 1))
13 print("n(3, 2) =", n(3, 2))
14 print("n(3, 3) =", n(3, 3))
15 print("n(3, 4) =", n(3, 4))
16 print("n(3, 5) =", n(3, 5))
17 print("n(3, 6) =", n(3, 6))
18 print("n(3, 7) =", n(3, 7))
19 print("n(3, 8) =", n(3, 8))
```

Result

Powered by  trinket

```
('n(3, 0) =', 12)
('n(3, 1) =', 10)
('n(3, 2) =', 8)
('n(3, 3) =', 6)
('n(3, 4) =', 4)
('n(3, 5) =', 2)
('n(3, 6) =', 0)
('n(3, 7) =', 0)
('n(3, 8) =', 0)
```

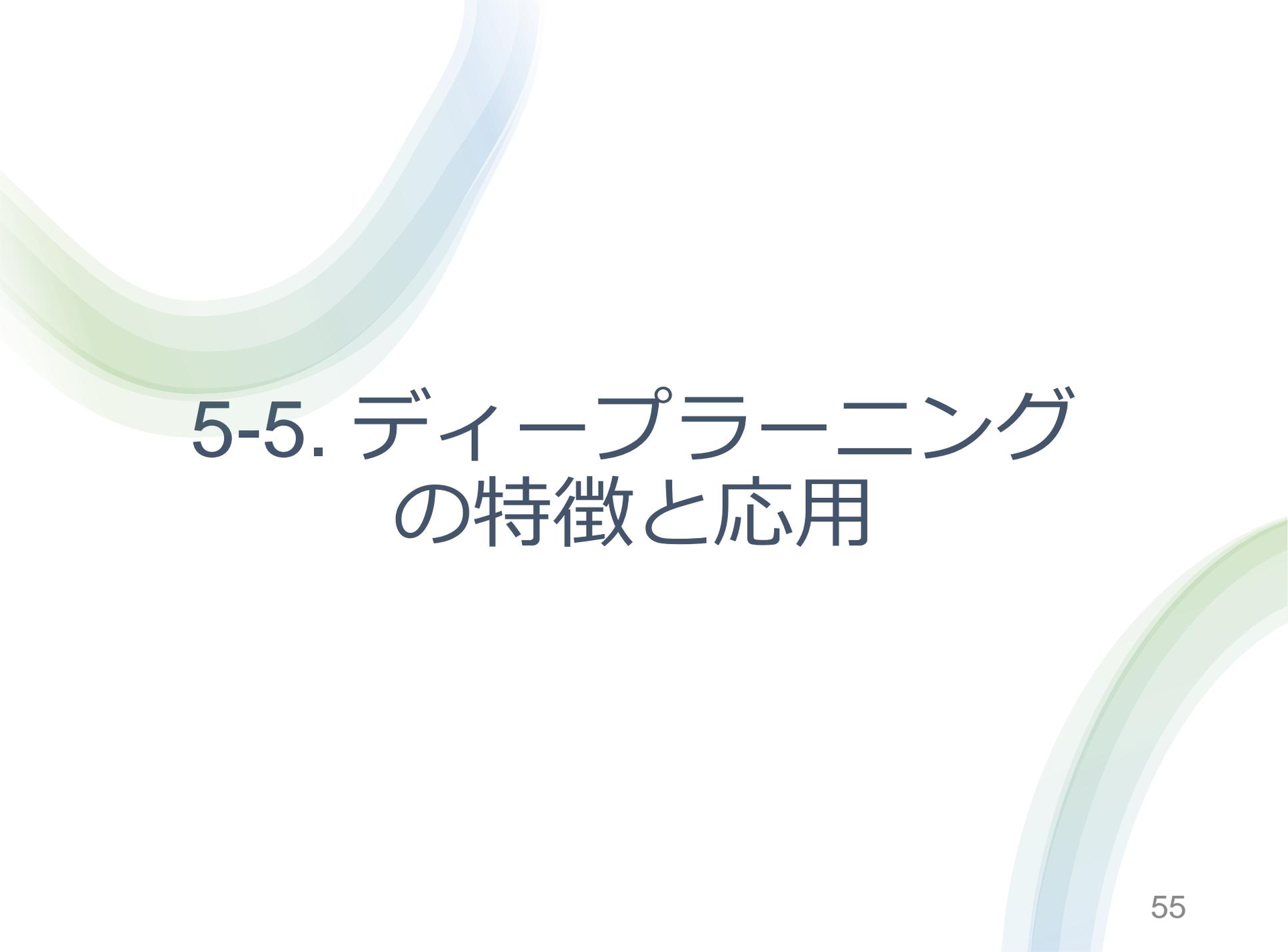
-2 を -6 に変えて、変化を見る

# ニューラルネットワークのまとめ



- ニューラルネットワークは、単純な原理で動く
- ニューロンは複数の入力を受け取る。中では、**活性化関数**を用いた**値の変換**を行う。
- **層構造**と**全結合**のニューラルネットワークが最もシンプルである。入力から出力の方向へ一方向にデータが流れる。2つの層の間のニューロンは**すべて結合**。
- ニューラルネットワークは、**結合の重みとバイアスを調整**して、**望みの出力を得る**。

ニューラルネットワークの**学習**では、**データを利用して、望みの出力が得られるように、結合の重みとバイアスの調整**が行われる。画像認識や自然言語処理などに広く利用されている。



## 5-5. ディープラーニング の特徴と応用

# 人工知能

知的なITシステム

## 機械学習

データから**学習**し、知的能力を向上

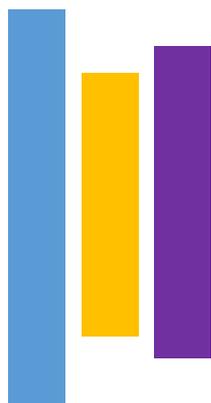
## ディープラーニング

データから**学習**し、複雑なタスクを実行。**多層のニューラルネットワーク**を使用

# ディープニューラルネットワーク



ディープラーニングは多層のニューラルネットワークを用いた機械学習



層の数が少ない（浅い）

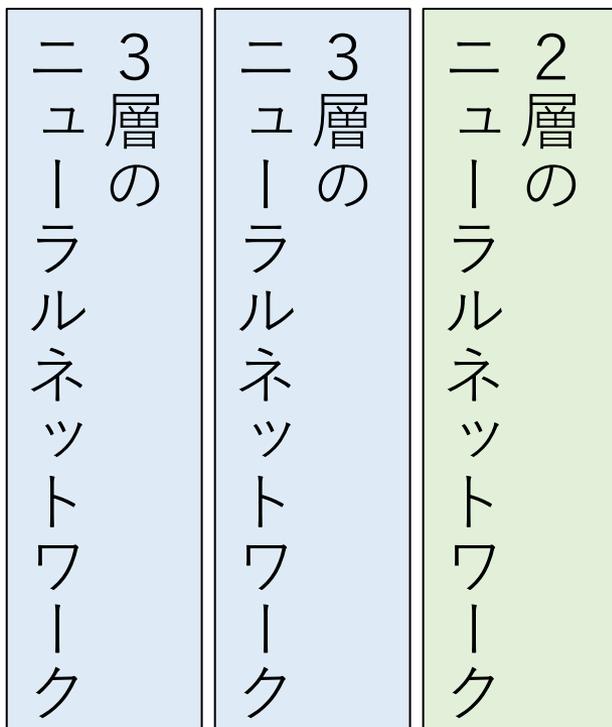


層の数が多し（深い）

# ディープニューラルネットワーク



- 層が浅い（層の数が少ない）ニューラルネットワークを組み合わせることもある



合計で8層

（さまざまな組み合わせが  
ありえる）

# ディープラーニングが広く利用されている理由



- **多様なデータへの対応**

例：画像、テキスト、音声、動画など

- **広範な応用分野**

例：画像認識、自然言語処理、音声認識など

- **優れたパターン抽出能力**

複雑なパターンを抽出する能力に優れる。

- **タスクの自動実行能力**

パターン抽出ならびにタスク実行は自動で行われる。

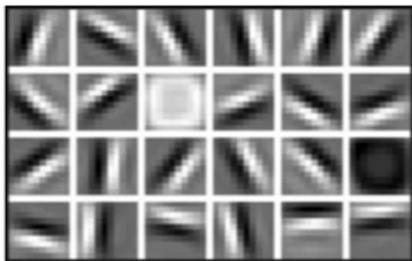
## さまざまなレベルのパターン

### Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features



Eyes & Nose & Ears

High Level Features



Facial Structure



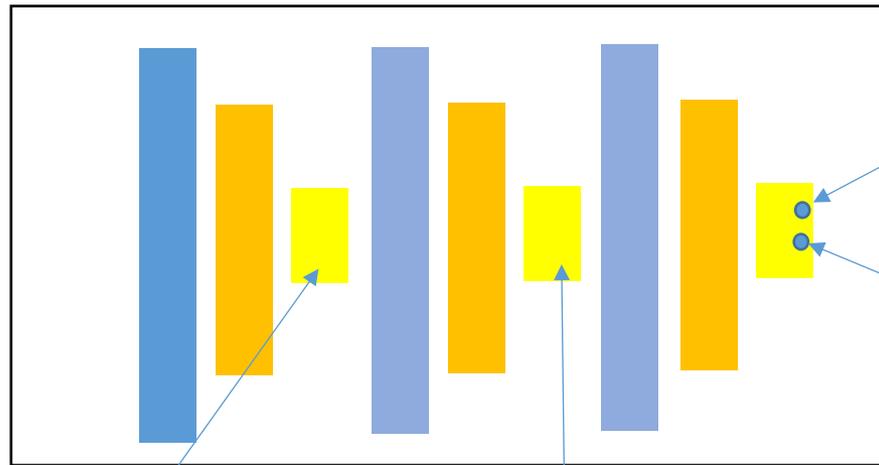
線や点のレベル 目, 鼻, 耳のレベル 顔の構造のレベル

MIT Introduction to Deep Learning | 6.S191,  
[https://www.youtube.com/watch?v=5tvmMX8r\\_OM](https://www.youtube.com/watch?v=5tvmMX8r_OM)  
の「Why Deep Learning」のページ

# 教師なし学習のニュース (2011年)

- **教師なし学習** (この画像が「人の画像である」, 「猫である」という**正解がない**)
- 訓練データ: YouTube から**ランダム**に選ばれた画像 **1000万枚**
- **1000台のマシンで, 3日間の学習**
- **9層のニューラルネットワーク**を使用

高次のパターンを認識  
できる能力を獲得



人の顔のみに  
反応するニューロン

猫の顔のみに  
反応するニューロン

特定の線や点に  
反応するニューロン

目や鼻や口に  
反応するニューロン

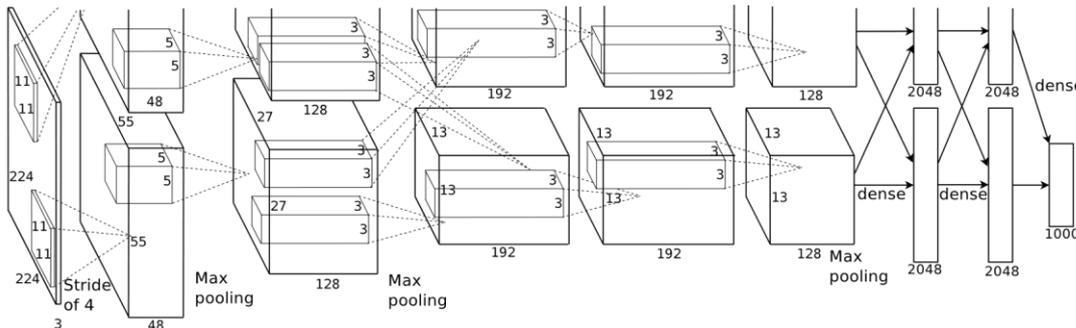
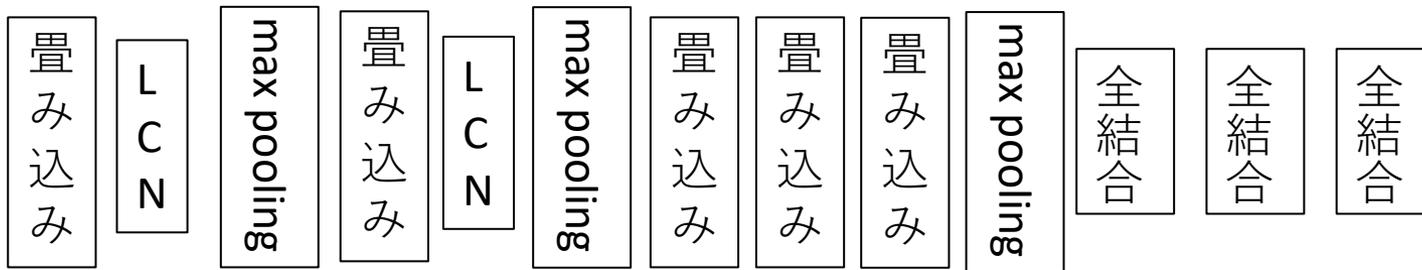
文献: Building high-level features using large scale unsupervised learning

Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, Andrew Y. Ng,  
arXiv 1112.6209, 2011, last revised 2012.

# 教師あり学習のニュース (2012年)

- 教師あり学習の AlexNet で画像分類を行う
- 訓練データ: 画像約 100万枚以上 (ImageNet データセット, 22000種類に分類済み)
- ILSVRCコンペティション: 画像を 1000 種類に分類
- ディープニューラルネットワークを使用

畳み込み, max pooling, 正規化(LCN), softmax, ReLU, ドロップアウト



文献: ImageNet classification with deep convolutional neural networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, NIPS'12, 2012.

# ディープラーニングへの期待



さまざまなレベルのパターンを抽出・認識できるようにする」という考える場合も

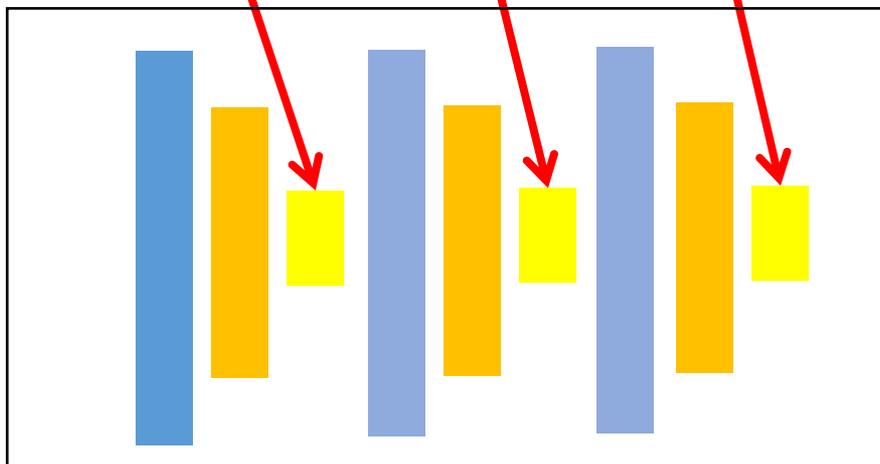
### Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice  
Can we learn the **underlying features** directly from data?

Low Level Features	Mid Level Features	High Level Features
Lines & Edges	Eyes & Nose & Ears	Facial Structure

Massachusetts Institute of Technology  
6.S191 Introduction to Deep Learning  
introtodeeplearning.com @MITDeepLearning  
1/18/21

IntroToDeepLearning.com



「より出力に近い層では、より高次のパターンを認識したい」という考え方も

# 画像分類の精度の向上



GT: horse cart  
1: horse cart  
2: minibus  
3: oxcart  
4: stretcher  
5: half track



GT: birdhouse  
1: birdhouse  
2: sliding door  
3: window screen  
4: mailbox  
5: pot



GT: forklift  
1: forklift  
2: garbage truck  
3: tow truck  
4: trailer truck  
5: go-kart



GT: coucal  
1: coucal  
2: indigo bunting  
3: lorikeet  
4: walking stick  
5: custard apple



GT: komondor  
1: komondor  
2: patio  
3: llama  
4: mobile home  
5: Old English sheepdog



GT: yellow lady's slipper  
1: yellow lady's slipper  
2: slug  
3: hen-of-the-woods  
4: stinkhorn  
5: coral fungus

- **ディープラーニング**の進展
- 画像分類は、場合によっては、AIが人間と同等の精度とも考えられるように

画像分類の誤り率 (top 5 error)

人間: 5.1 %

PReLU による画像分類: 4.9 %

(2015年発表)

ImageNet データセット  
の画像分類の結果

文献: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun,  
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification  
arXiv:1502.01852, 2015.

# ディープラーニングの応用と未来



## ディープラーニングの研究は急速に進展

- 2011年：教師なし学習による顔認識の成功（YouTube画像1000万枚使用）
- 2012年：AlexNetによる画像分類の革新
- 2015年：人間の精度を上回る画像認識の達成（誤り率4.9%）

## ディープラーニングを支える要素

- **活性化関数**（ReLUなど）
- **ドロップアウト**や**初期化手法**の開発
- **大規模データ**の活用
- **高性能コンピュータ**の発展
- **データ拡張技術**の進歩

# ディープラーニングの応用と特徴



- **ディープラーニング**は**機械学習**の一種であり、人工ニューラルネットワークを使用して**データから学習**し、**複雑なタスクを実行**する技術
- 「ディープ」の名前は、**多層のニューラルネットワーク**を使用することに由来
- ディープラーニングが広く利用される理由は、**多様なデータに適用**でき、**さまざまなタスク**で高性能を発揮するため。例えば、**画像認識**、**自然言語処理**、**音声認識**など。



## 自習 1. 「1. 活性化関数 ReLU」について

目的: この自習問題の目的は、**活性化関数 ReLU**がどのように動作するかを理解し、プログラム内で実際に実行してみることです。

- プログラムを読んで、それが何をしているのか理解してください。
- **relu 関数がどのように動作するか**を特に注意して理解しましょう。
- **プログラムを書き換えて、-3 や 3 のときの ReLU の値を計算**させてみましょう。

ヒント:

- relu 関数は、与えられた  $x$  が負の場合は 0 を返し、それ以外の場合には  $x$  自体を返します。
- $x$  の値を変更して、異なる  $x$  の値に対する ReLU の値を計算させてみてください。-3 や 3 などを試してみましょう。

ReLU 関数は、ニューラルネットワークで広く使用されます。そのため重要です。

自習は提出する必要はありません。

## 自習 2. 「2. ニューロン、重み、バイアス、活性化関数、ニューロンの活性化度」について



目的:ニューラルネットワークのニューロンがどのように入力に対して活性化度を計算し、ReLU 活性化関数を使用して活性化度を処理するかを理解することです。

- プログラムのコードを読んで、それが何をしているのか理解してください。n 関数は、入力、重み、バイアス、活性化関数を用いて、活性化度を計算するためのものです。
- **プログラムを書き換えて、異なる入力値 (0.1, 0.8, -0.5)、(5.1, 0.8, -0.5)、(10.1, 0.8, -0.5)、(15.1, 0.8, -0.5) を試してみましょう。**

ヒント:

- n 関数内で、s は重み付きの入力とバイアスを用いて計算された値です。この値を relu 関数に渡して、活性化度を求めています。

ニューロンの活性化度の計算は、ニューラルネットワークの重要な概念です

自習は提出する必要はありません。