数学月間秋の企画公演（第１１回）

なぜ深層学習を
思いつかなかったか

2022/09/24

株式会社モーション
中西 達夫

自己紹介

Motion

中西　達夫

- 株式会社モーション　システム・エンジニア
- データ分析システム構築、気付けば２３年
- 週１回だけ非常勤講師

近年話題の
AIについて、
思うところを
お話させてください。

我が人生最大の過ち

ディープラーニングを作り出せなかった

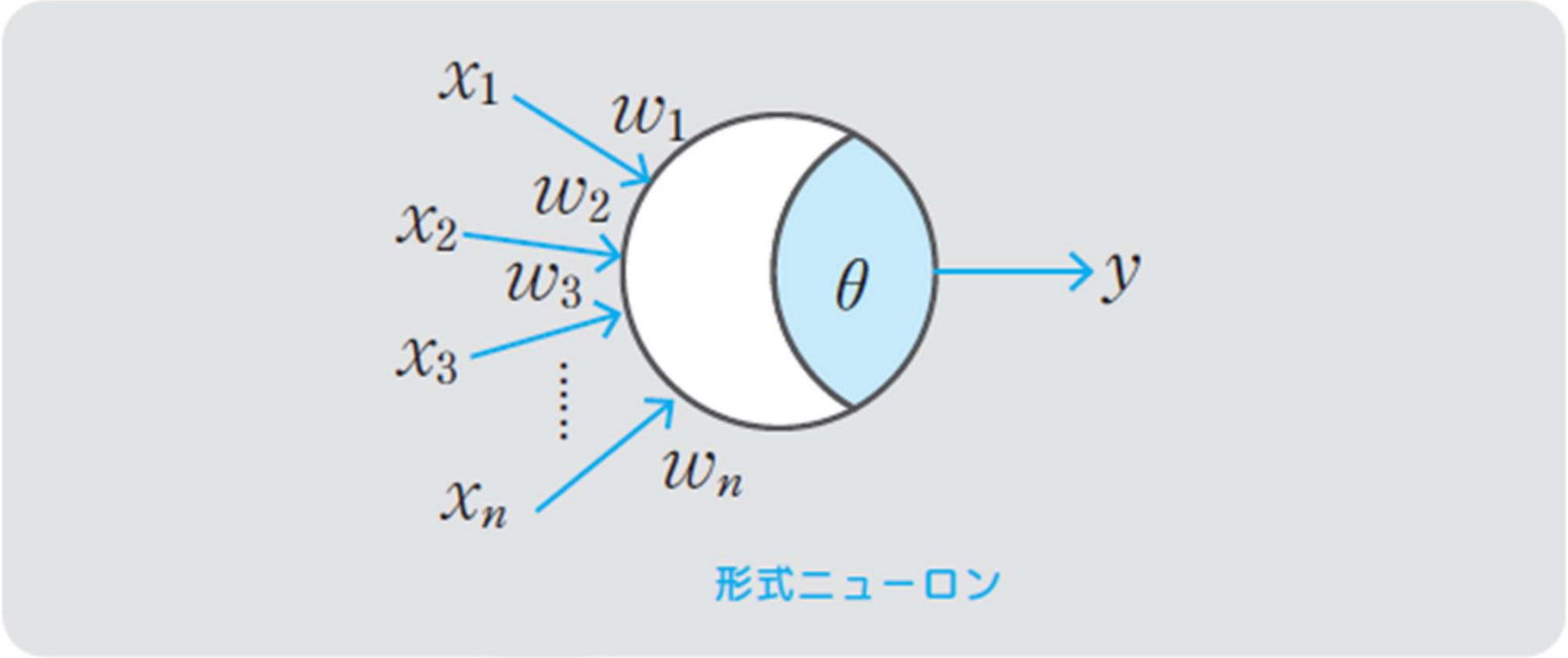
今から振り返ってみれば、
あらゆる条件が揃っていながら、
最後の一歩に至らなかった。

ディープラーニングへの突破口は何か？

「ReLU関数による勾配消失問題の解決」

・・・というのが一般的な見解

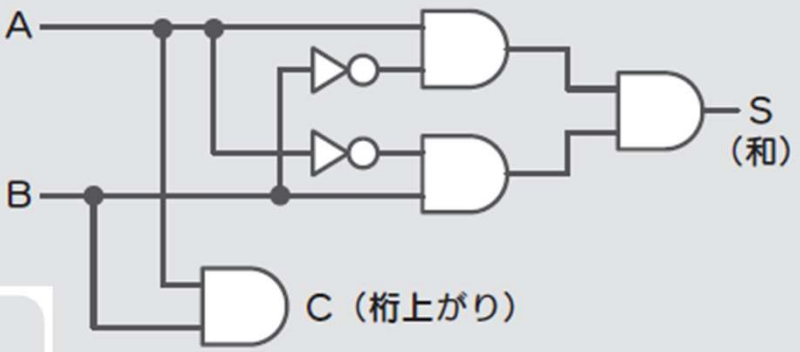
短くて長いAIの歴史



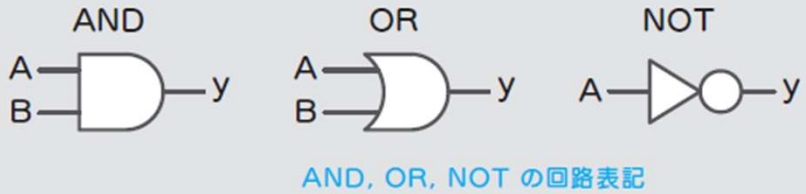
$$\Sigma\, w_i \cdot x_i \geqq \Theta \text{ then } y=1 \text{ else } 0$$

形式ニューロンモデル
McCulloch & Pitts (1943)

デジタル回路



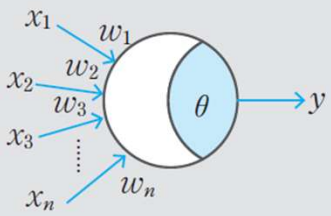
デジタル回路の一例、ハーフアダー



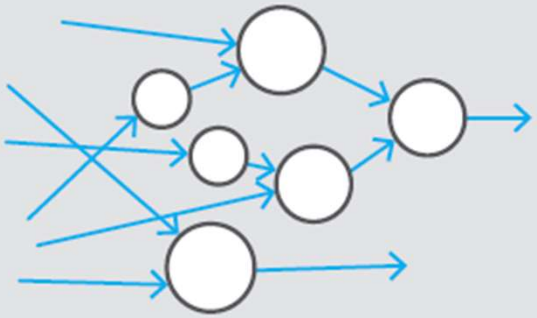
AND, OR, NOT の回路表記

この2つは似ている！

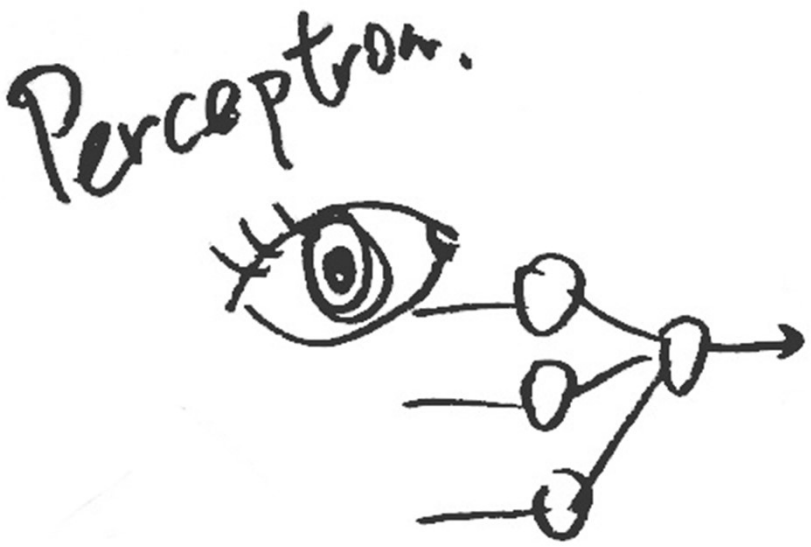
神経回路

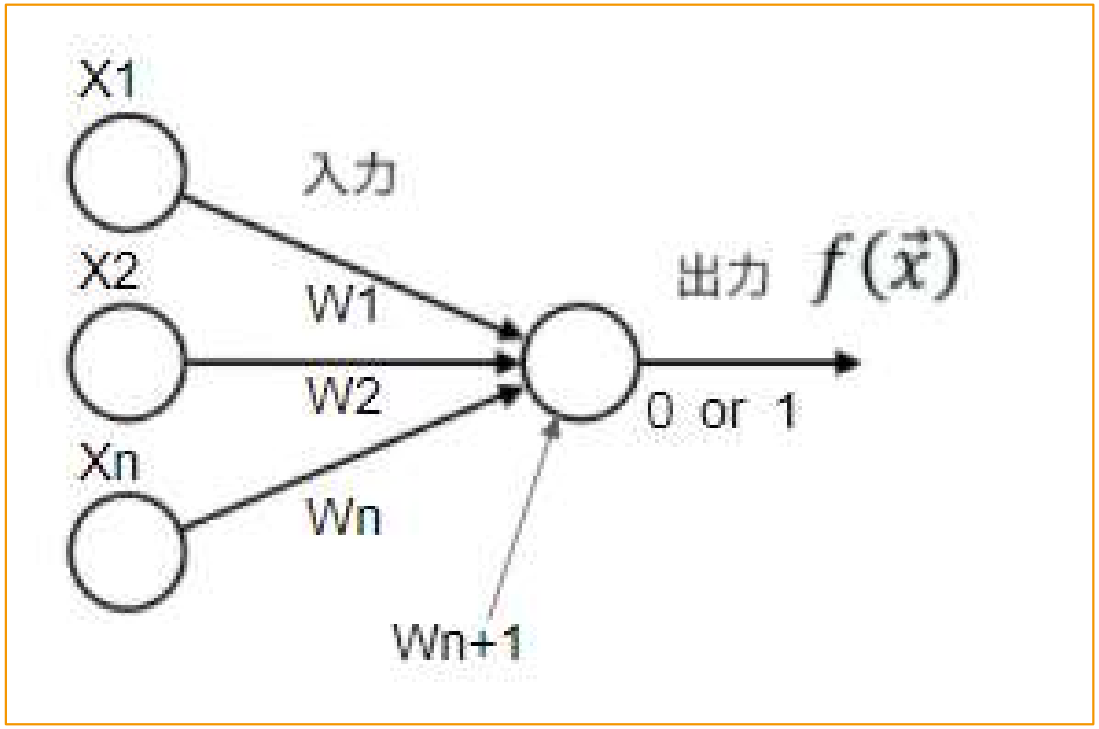


形式ニューロン

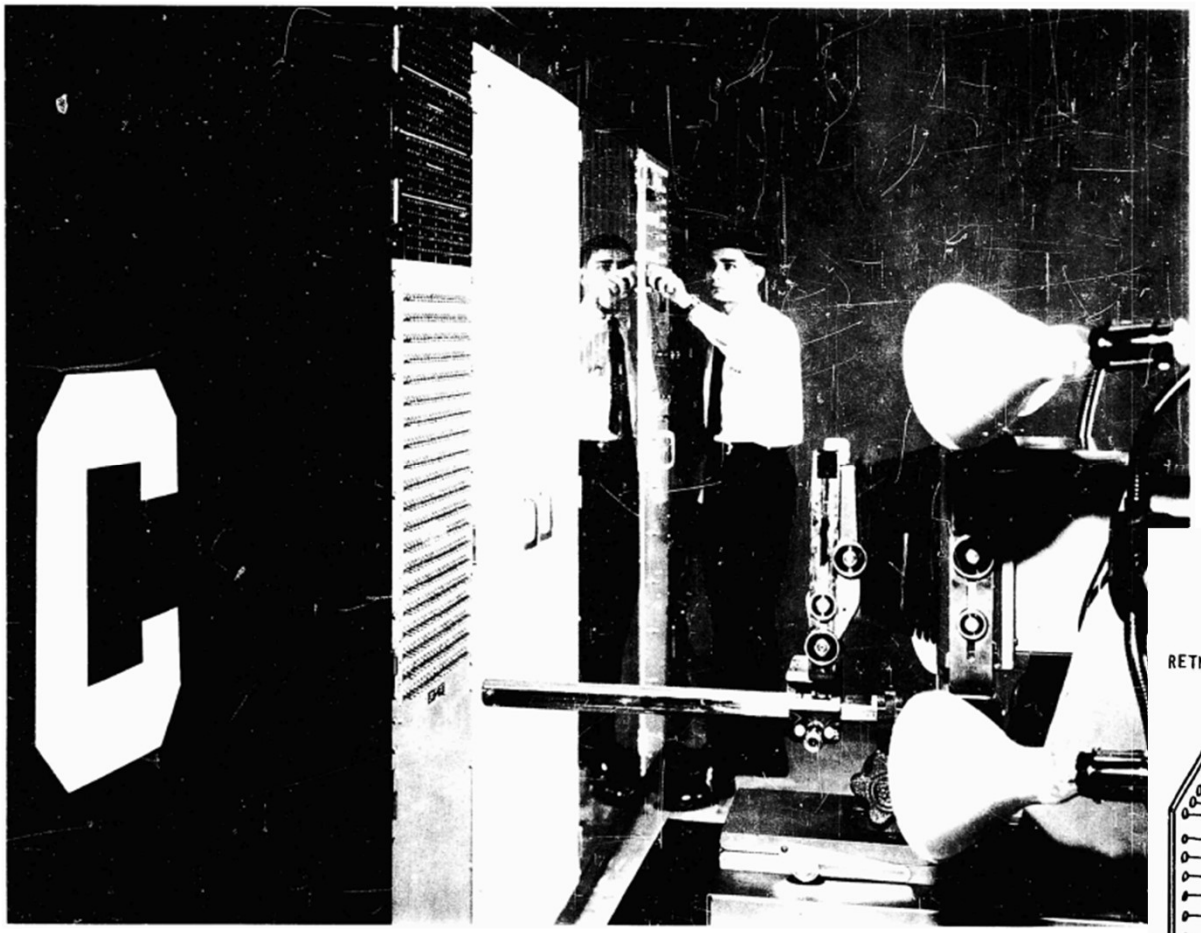


ニューロンが集まって回路をつくっている





単純パーセプトロン
F. Rosenblatt (1958)



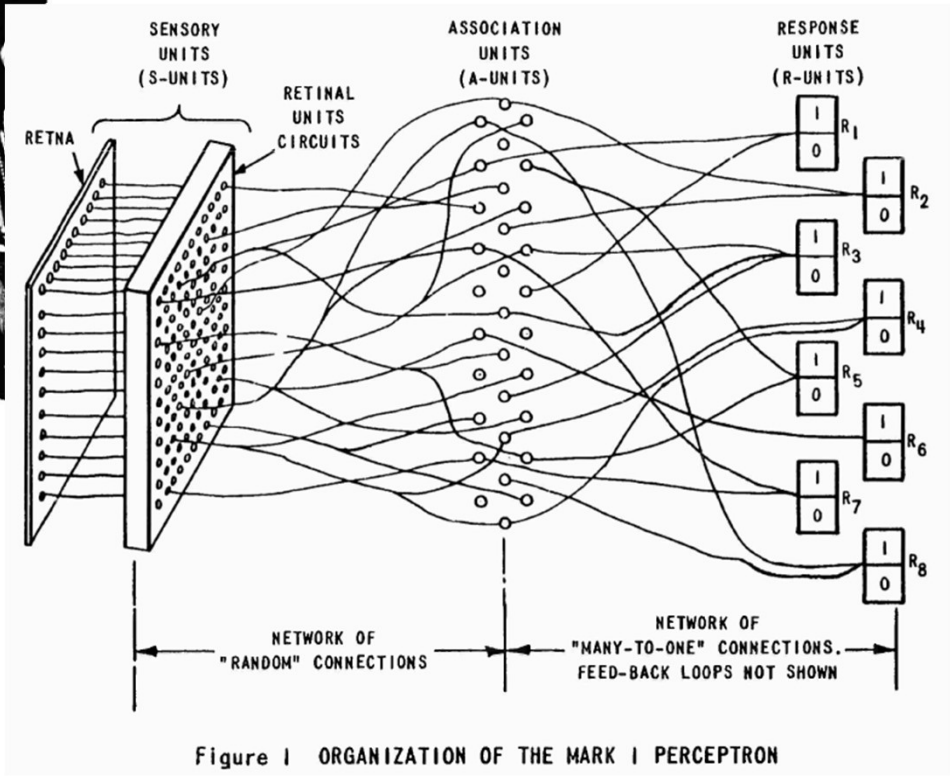
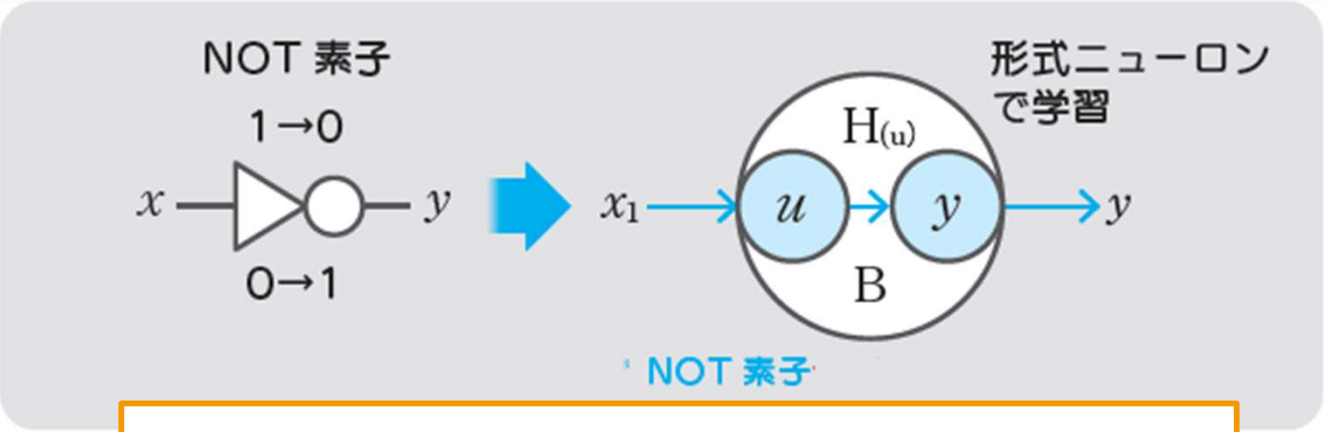


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

MARK1 PARCEPTRON OPERATOR'S MANUAL (1960) より引用.

NOT素子の学習



$$u = W \cdot X + B$$

$$Y = \text{if } u > 0 \text{ then } 1 \text{ else } 0$$

① 最初、W と B には適当な（デタラメな）値を入れておく。

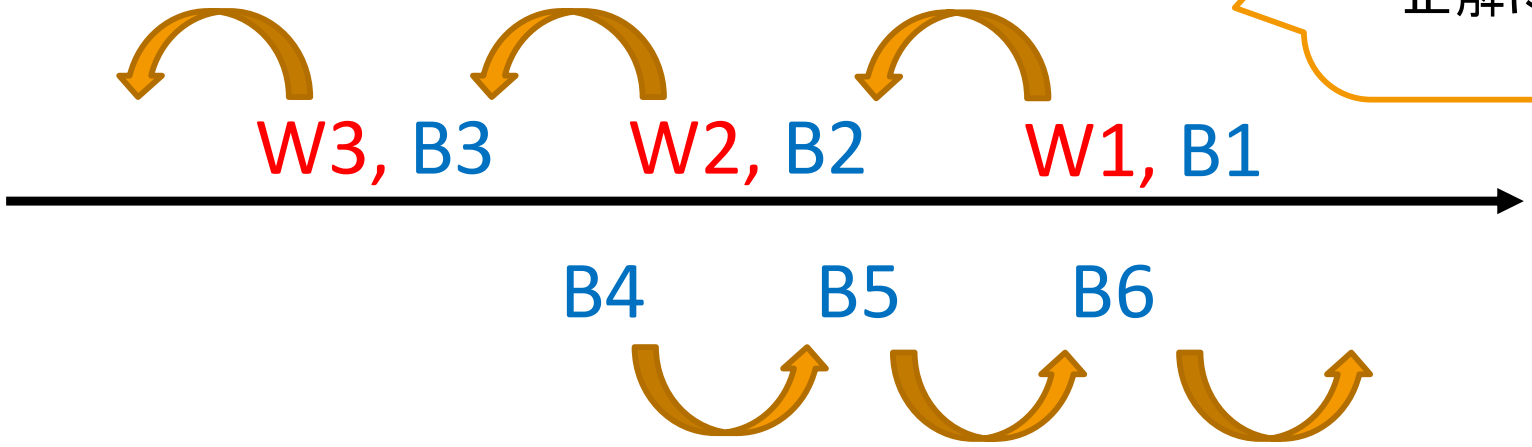
② X に 1 か 0 を適当に入力し、
Y が間違っていたら、正しい答に近づくように
W と B を少しずつ（たとえば0.1ずつ）修正する。

③ 正しい答を出力するようになるまで、修正を繰り返す。

パーセプトロンの学習則

X=1 → Y=1 だったら・・・
　　正解 Y=0 に向けて、W と B を両方小さくする。
X=0 → Y=0 だったら・・・
　　正解 Y=1 に向けて、B だけを大きくする。
X=1 → Y=0 だったら・・・
　　正しい動作なので、何もしない。
X=0 → Y=1 だったら・・・
　　正しい動作なので、何もしない。

いずれは
Wが負で、
Bが正という
正解にたどり着く



パーセプトロンの学習則

$$w \leftarrow w + \eta\ (\text{teach} - \text{output})\, x$$

$$\mathrm{B} \leftarrow \mathrm{B} + \eta\ (\text{teach} - \text{output})$$

x : 入力データ

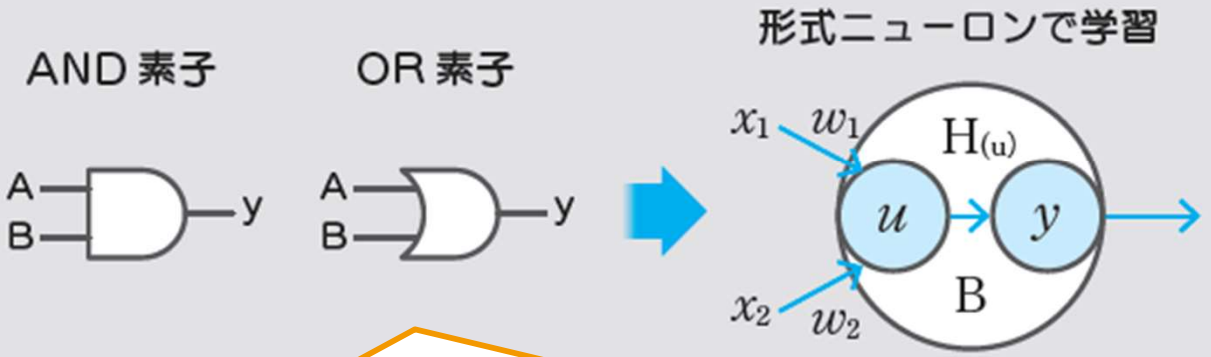
teach : 教師データ、正解

output : 予測データ、間違っているかもしれない

(teach - output) : 教師データと予測データの差、つまり誤差

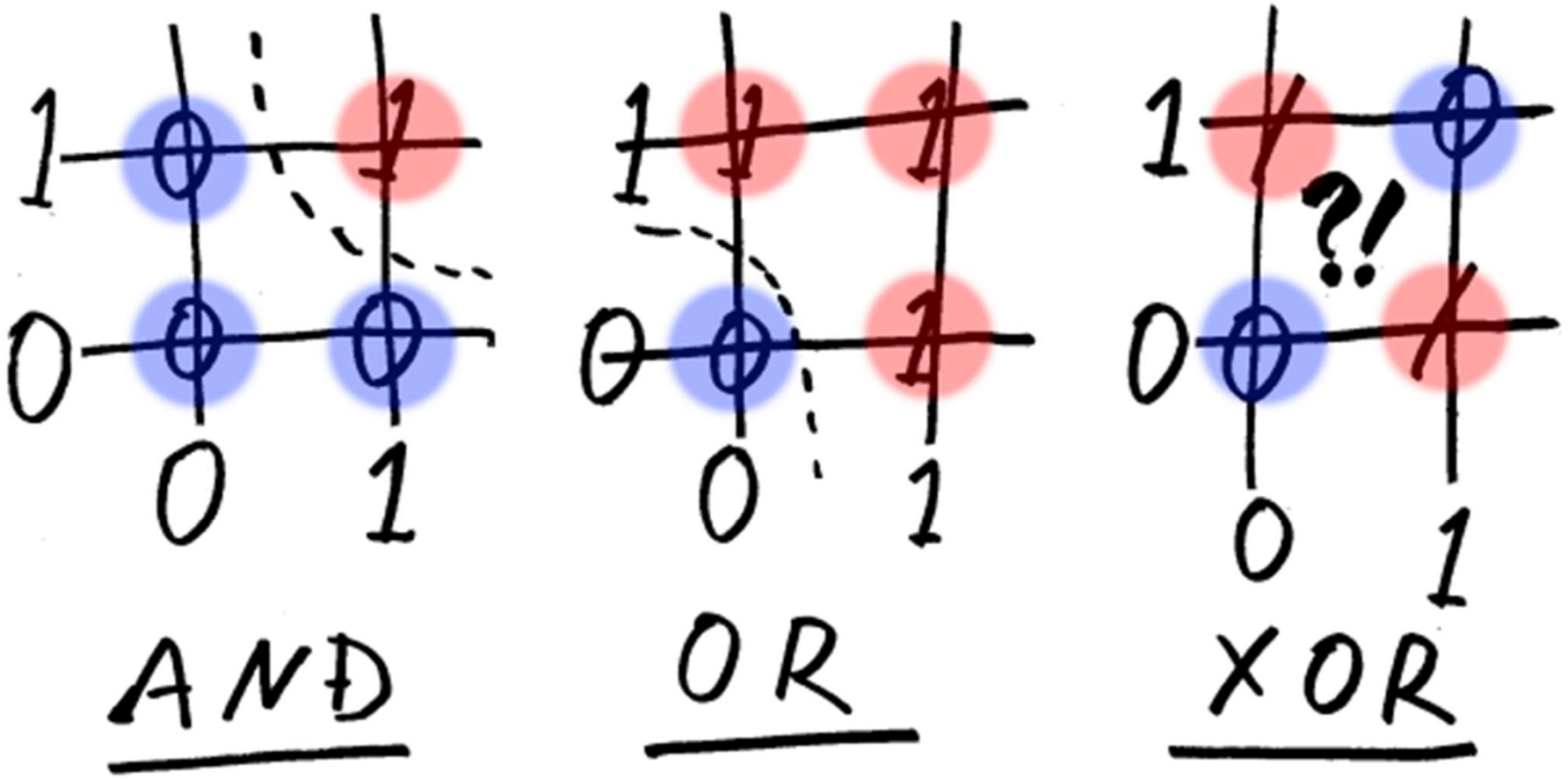
x が 1 のときは重み w に影響がある。

x が 0 のときは重み w に影響が無い。



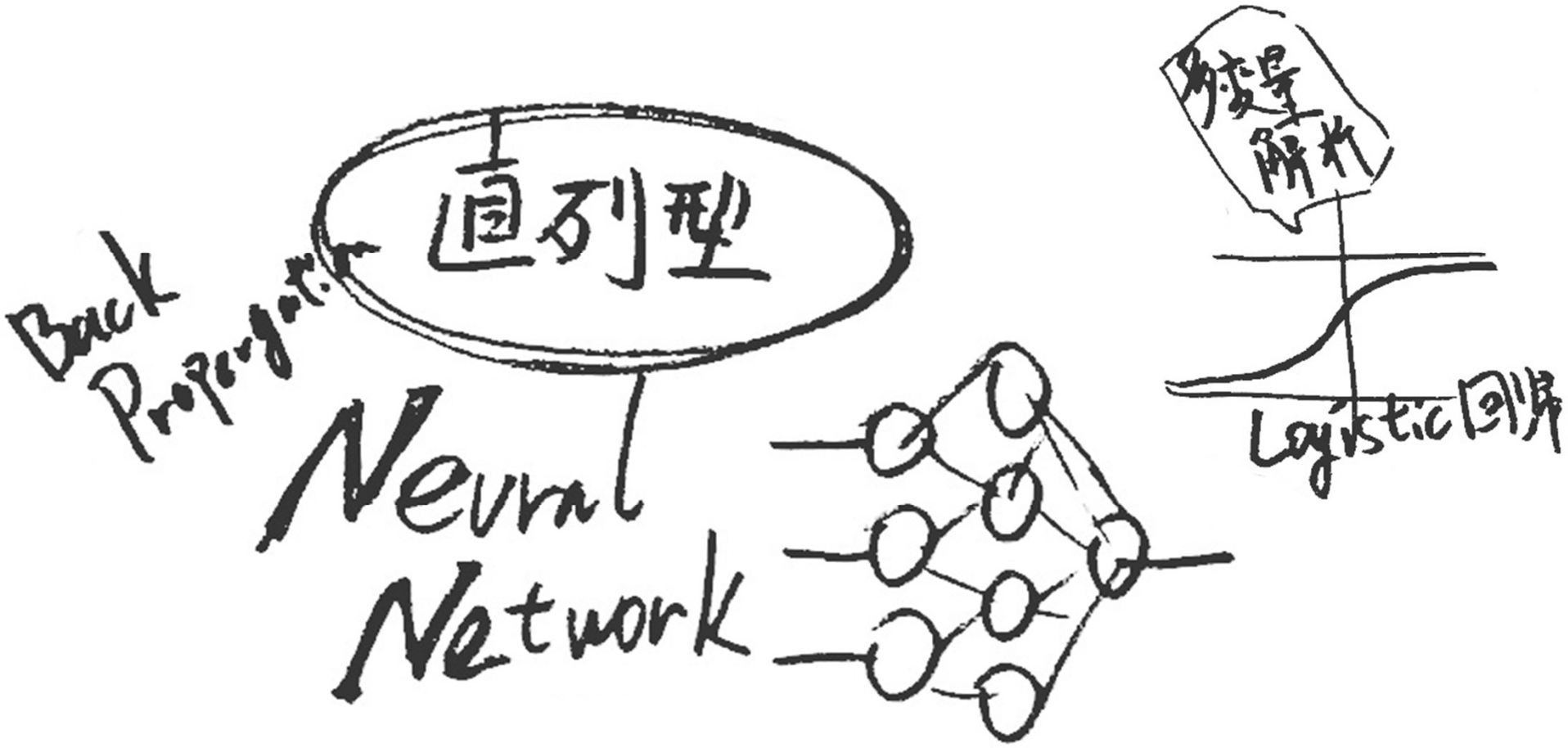
「間違ったら正す」を繰り返すことで、
NOT, AND, OR が学習できた！

線形分離可能・不能

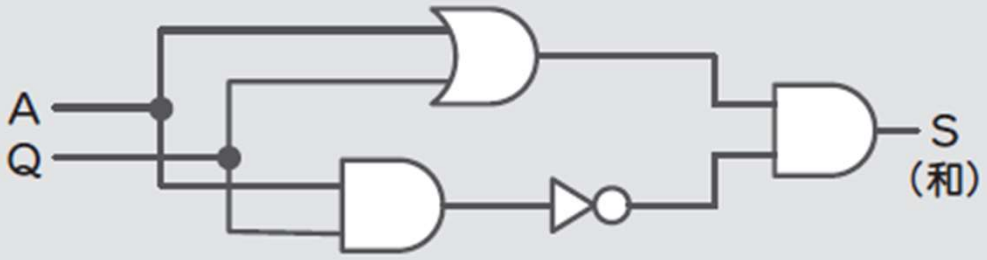


パーセプトロンの限界
M. Minsky (1969)

1969～80 AIの冬第1期



Back Propagation
Rumelhart & McClelland (1986)

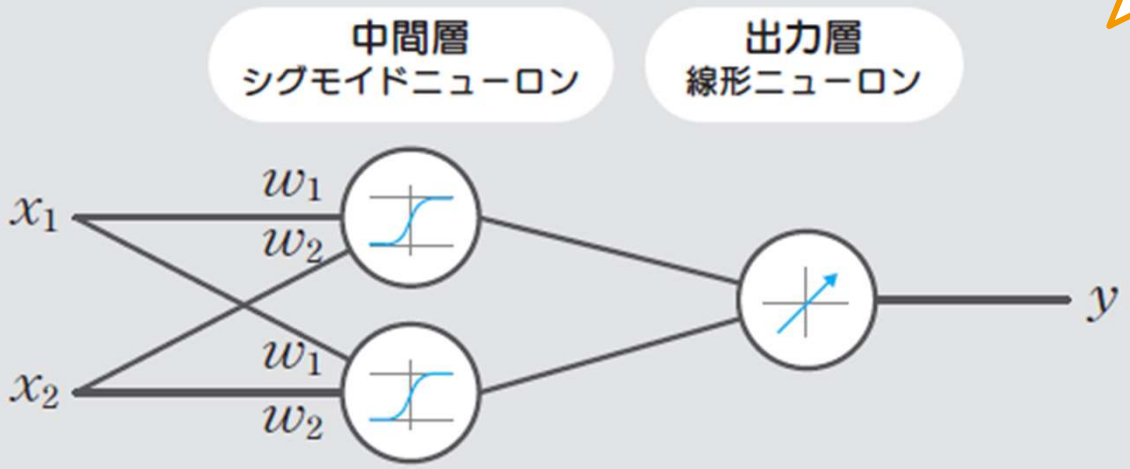


組合せてつくったXORの回路図

すでに私たちは、1個のニューロンで、AND, OR, NOT ができることは確認済みです。

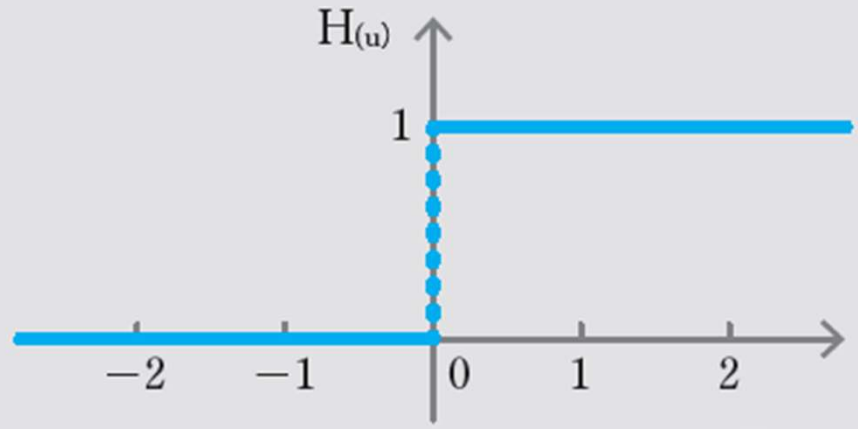
そうであれば、ニューロンを3個組み合わせれば、XORがつくれるはずです。

微分の概念



2層ニューラルネットワーク

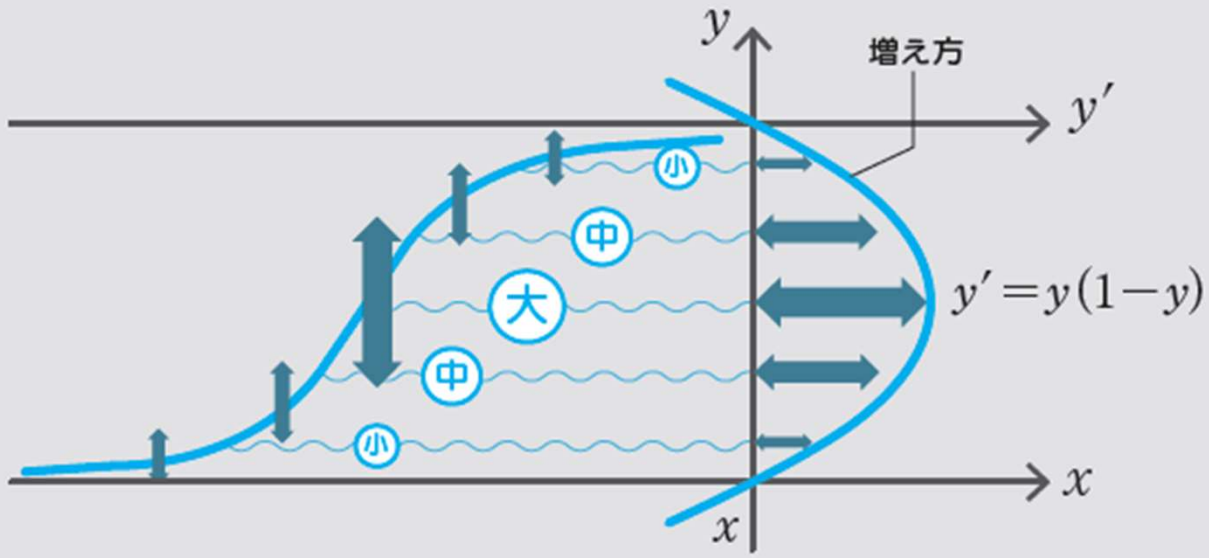
活性化関数の進化



ヘヴィサイドの階段関数

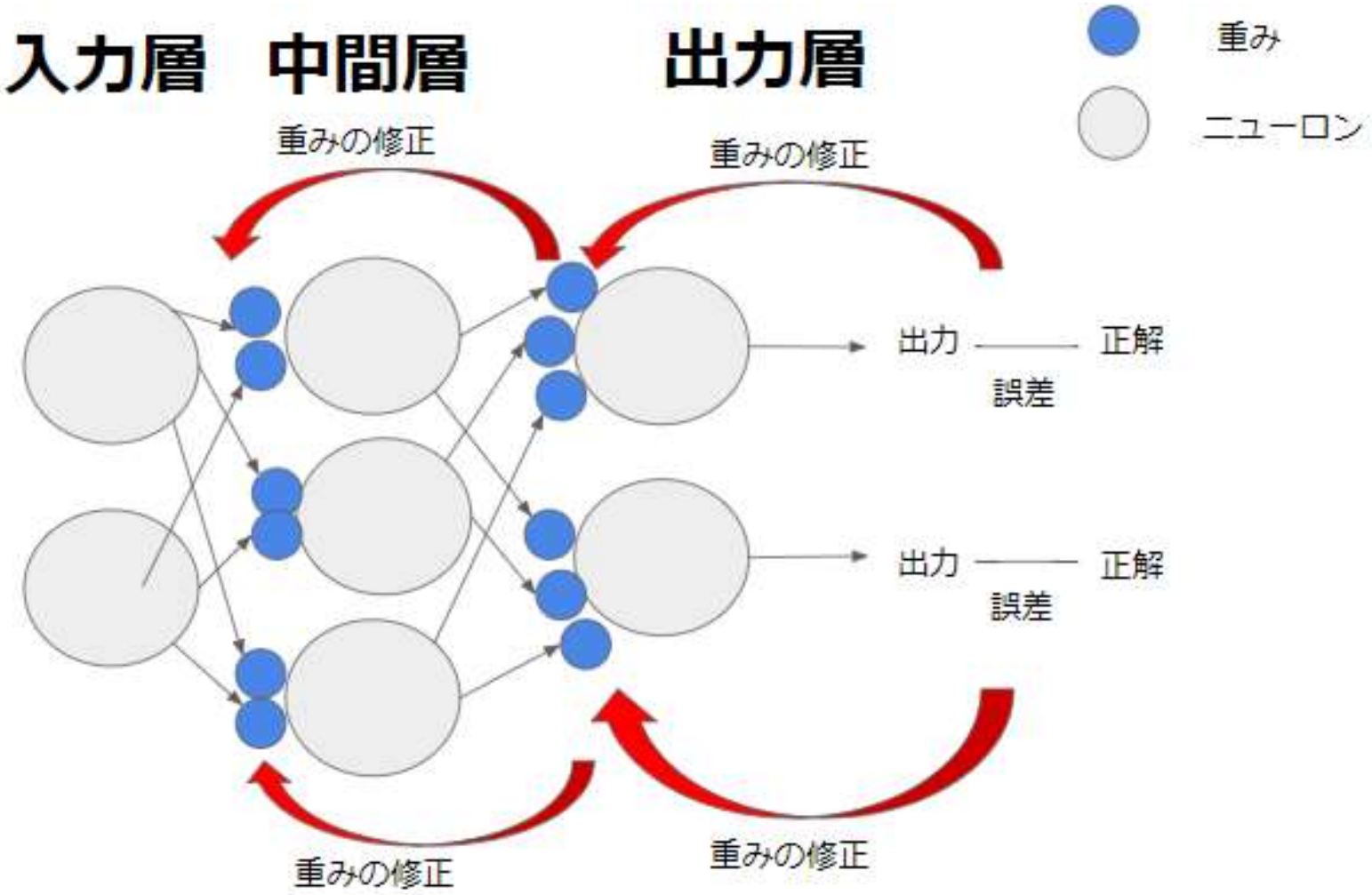
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



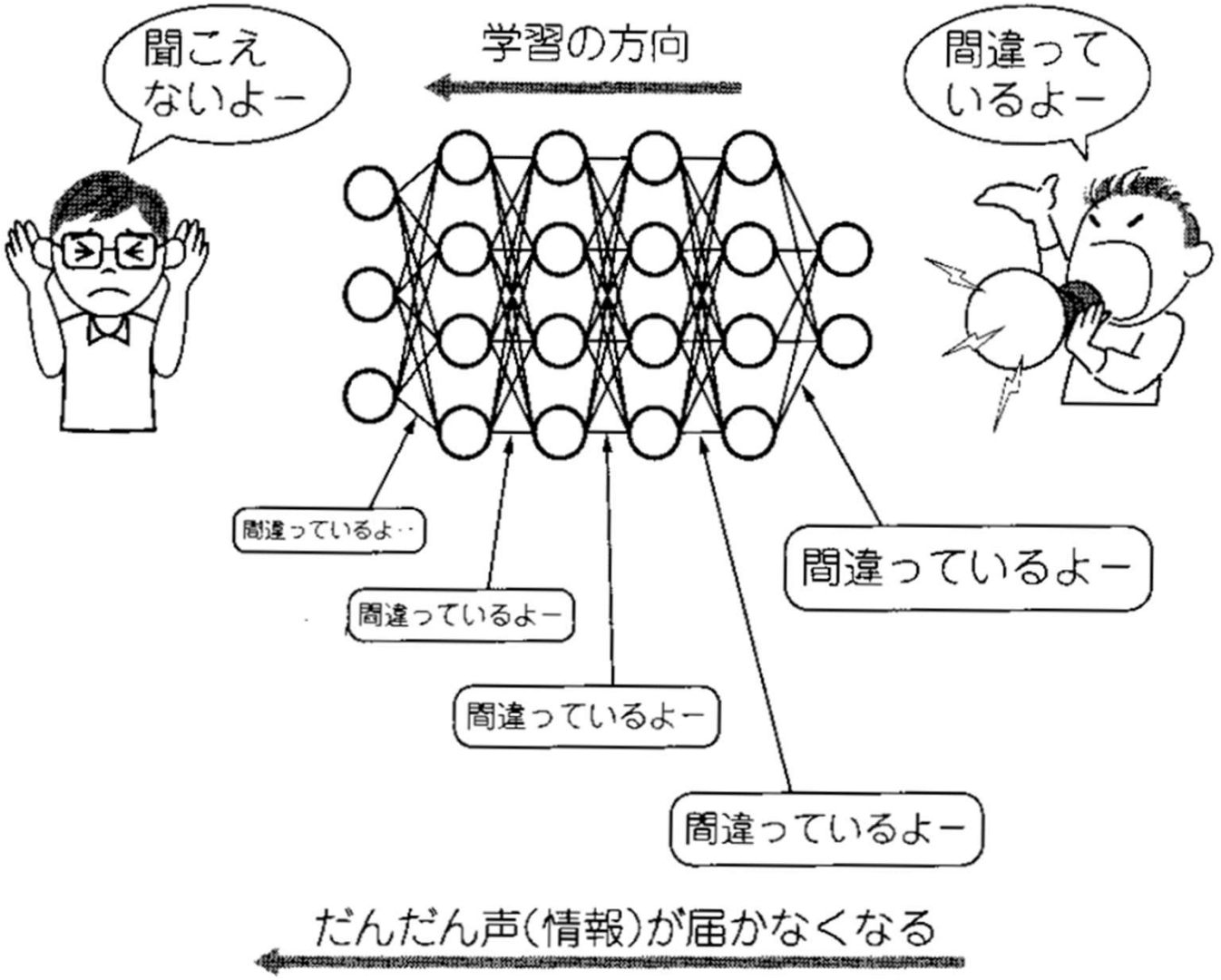
ロジスティック曲線、微分方程式の図

バックプロパゲーション（逆誤差伝搬法）



引用元:初心者のためのAI人工知能テクノロジーブログ
https://newtechnologylifestyle.net/**今さら聞けないバックプロパゲーションとは/**

勾配消失問題



図C　昔のニューラル・ネットワークは「出力データと教師データの差」が入力側に伝搬するに従って小さくなっていた

引用元: インターフェイス 2017年8月号.

1990後半～2012 AIの冬第2期

The pioneers of deep learning have won the Turing Award

「深層学習の父」ら3人にチューリング賞

https://www.technologyreview.jp/nl/the-pioneers-of-deep-learning-have-won-the-turing-award/

大規模人工ニューラル・ネットワークの学習能力の研究によって、人工知能（AI）革命のきっかけを作った3人の科学者に、コンピューター科学の分野でもっとも権威ある賞、チューリング賞が授与された。

ジェフリー・ヒントン博士、ヤン・ルカン博士、ヨシュア・ベンジオ博士の3人は、AIコミュニティがこの分野を将来性のない行き止まりだとみなし、シンボリック・アプローチ（学習ではなく手入力によってコード化された論理規則を用いる手法）に焦点を当てていた時期に、数十年に渡って辛抱強くニューラル・ネットワークの研究に取り組んだ。

2012年、突如、多層ニューラル・ネットワークが画像認識に驚くほど優れた威力を発揮することが証明された。その鍵となったのは、ニューラル・ネットワークに膨大な量の訓練データを与え、要求される並列演算処理に適した強力な画像処理チップを使って画像認識を実行することだった。

AlexNet

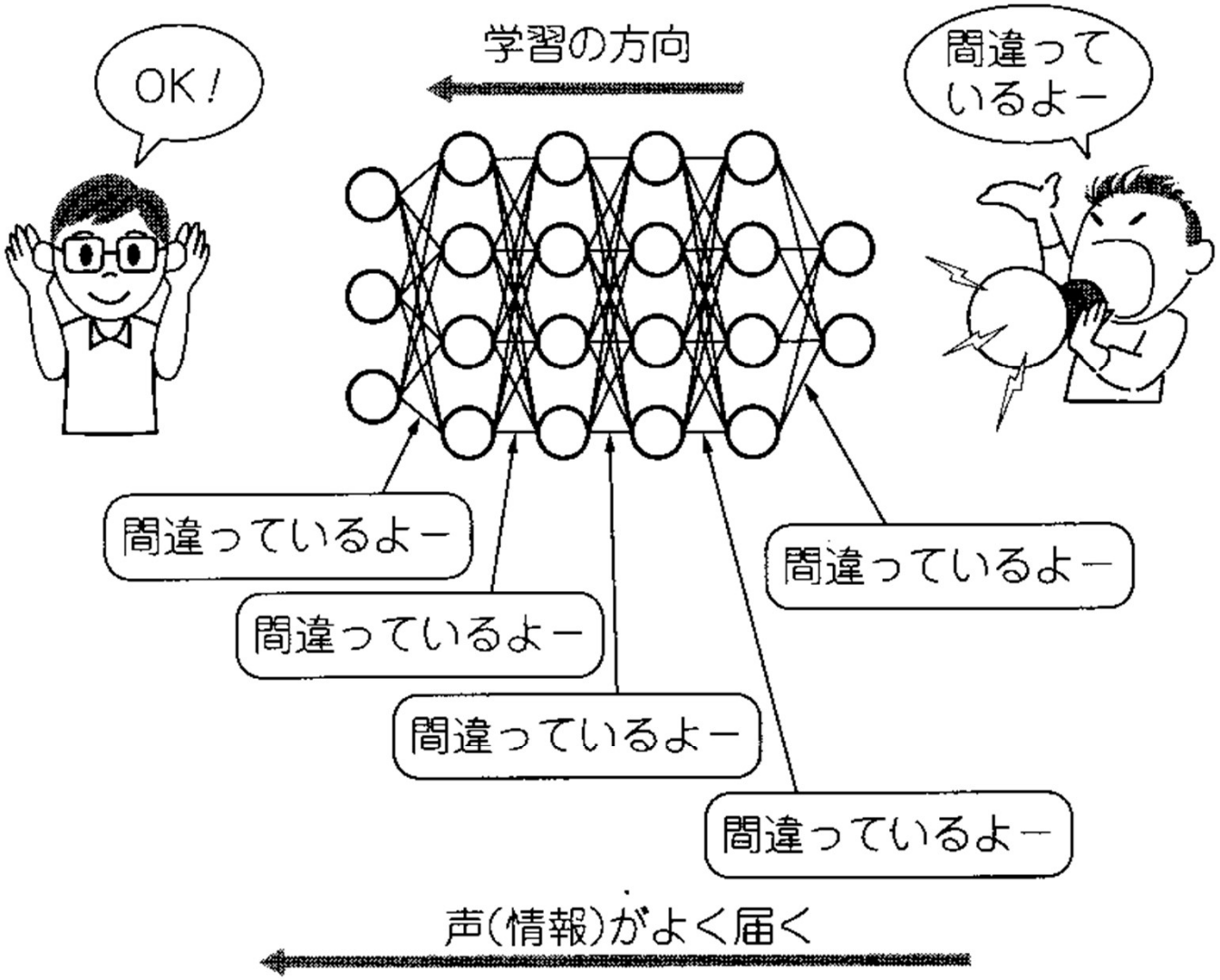
ページ　ノート

出典: フリー百科事典『ウィキペディア（Wikipedia）』

AlexNet は畳み込みニューラル ネットワーク（CNN）のアーキテクチャの名前であり、Alex Krizhevsky が博士課程の指導教官である Ilya Sutskever および ジェフェリー・ヒントン と共同で設計した[1] [2]。

AlexNet は、2012 年 9 月 30 日に開催された ILSVRC 2012[3] に参加した。AlexNet はエラー率 15.3% で優勝し、次点よりも 10.8% 以上低かった。この論文の主な内容は、モデルの深さが高性能には不可欠であるというもので、計算コストは高くなるものの、GPU を用いて学習することで実現した[2]。

ディープラーニングの嚆矢とされるモデルの1つ



図D　現在のニューラル・ネットワークは「出力データと教師データの差」の情報が入力側まで戻る

引用元: インターフェイス 2017年8月号.

ReLU関数への進化

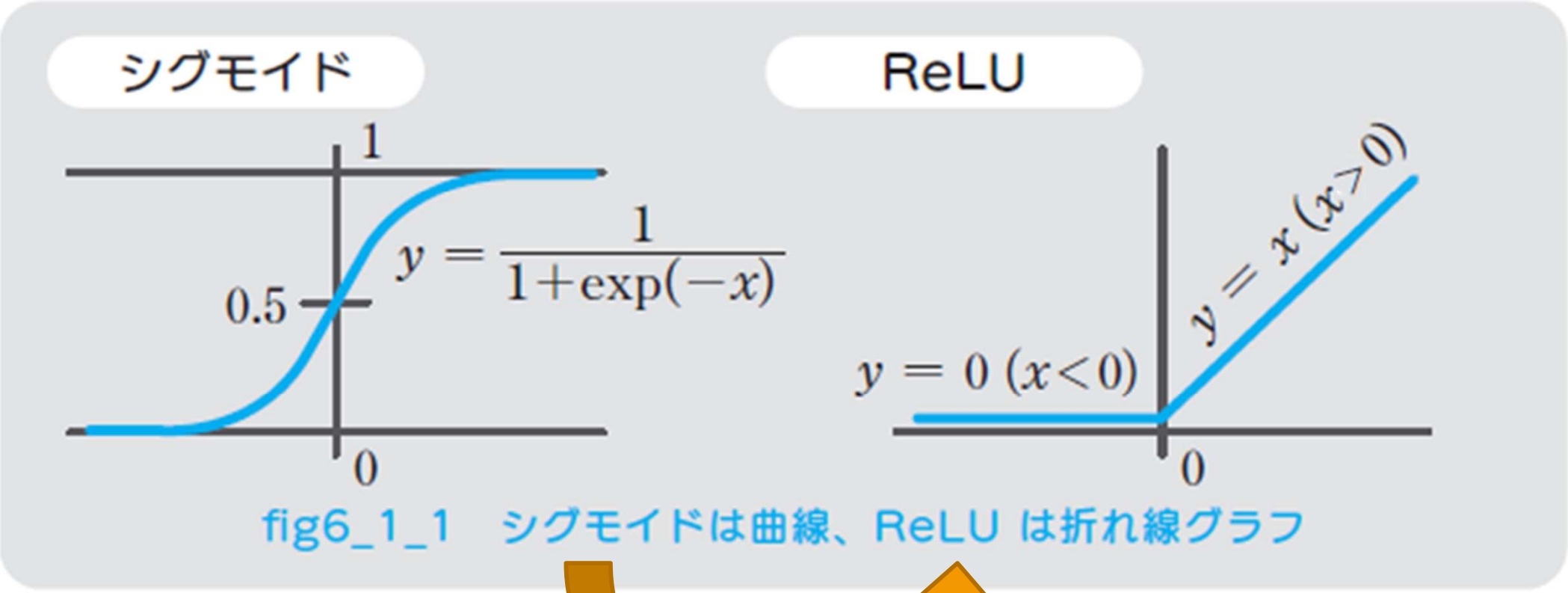


fig6_1_1　シグモイドは曲線、ReLU は折れ線グラフ

3.1 ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

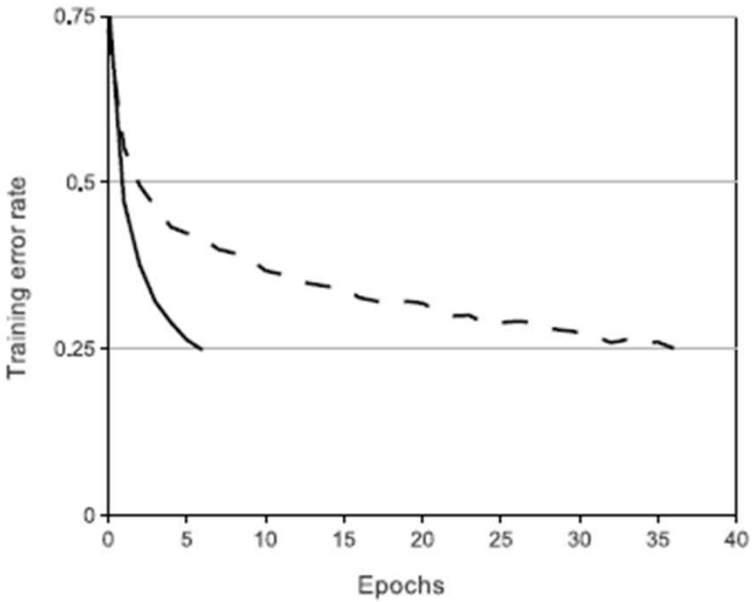
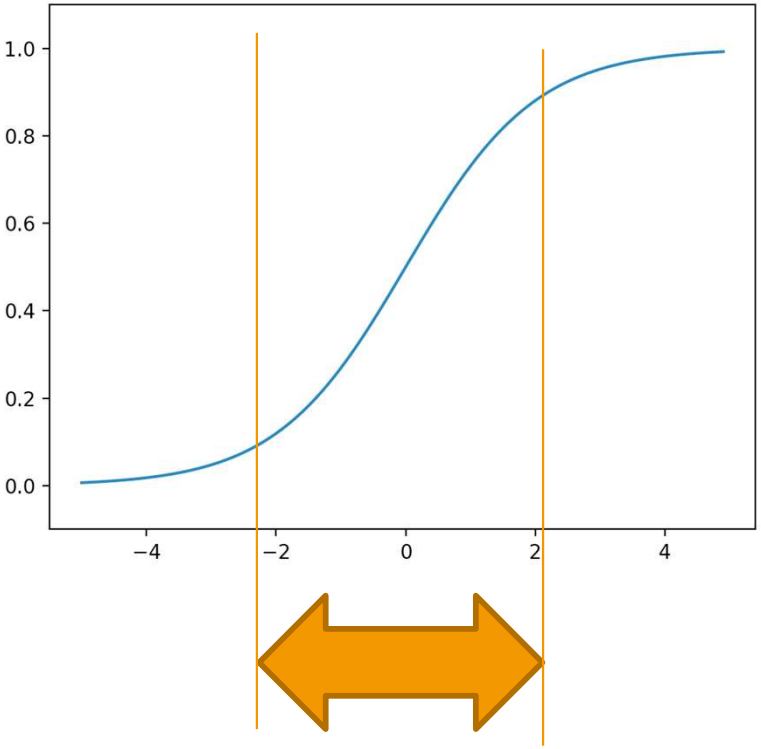


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

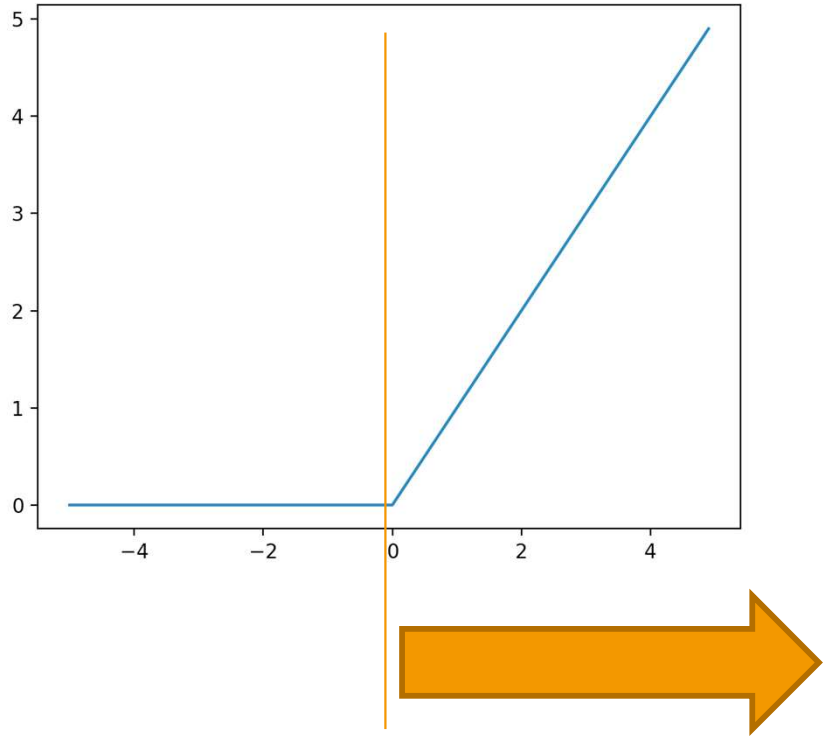
ImageNet Classification with Deep Convolutional Neural Networks (2012) より引用.

なぜ、ReLU関数が良いのか？

一見するとシグモイド関数の方が
高度に思えるのだが？



狭い有限区間



開かれた無限区間

“スイートスポット”の広さの違い

非線形の本質

下図のようにReLU2個を組み合わせると、Z字型のカーブがつくれます。
このZ字型のカーブは、シグモイドのS字型のカーブと似ているので、
ReLU2個でシグモイド1個分と、ほぼ同等の働きが実現できるのです。

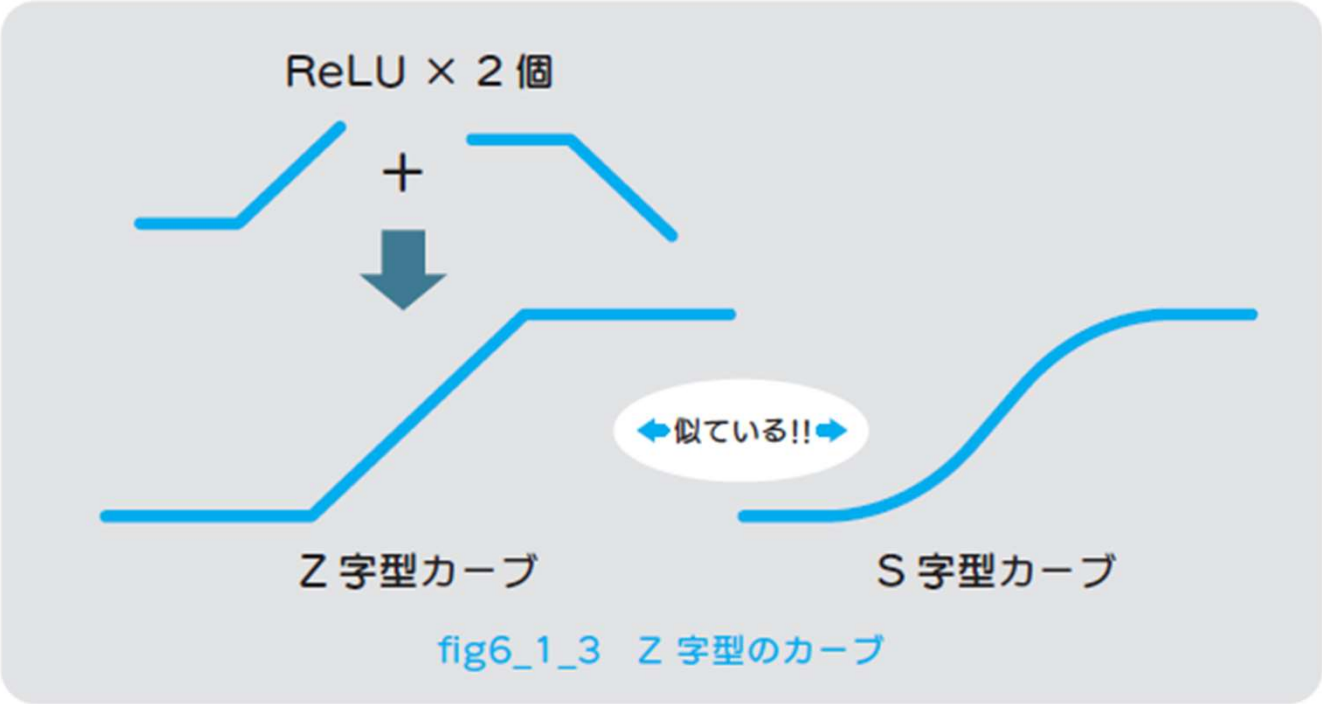


fig6_1_3　Z 字型のカーブ

ReLU関数
非線形を構成する最も基本的な要素

引用元:ディープラーニングの動きを理解するための数式入門

ＸＯＲ素子の構成

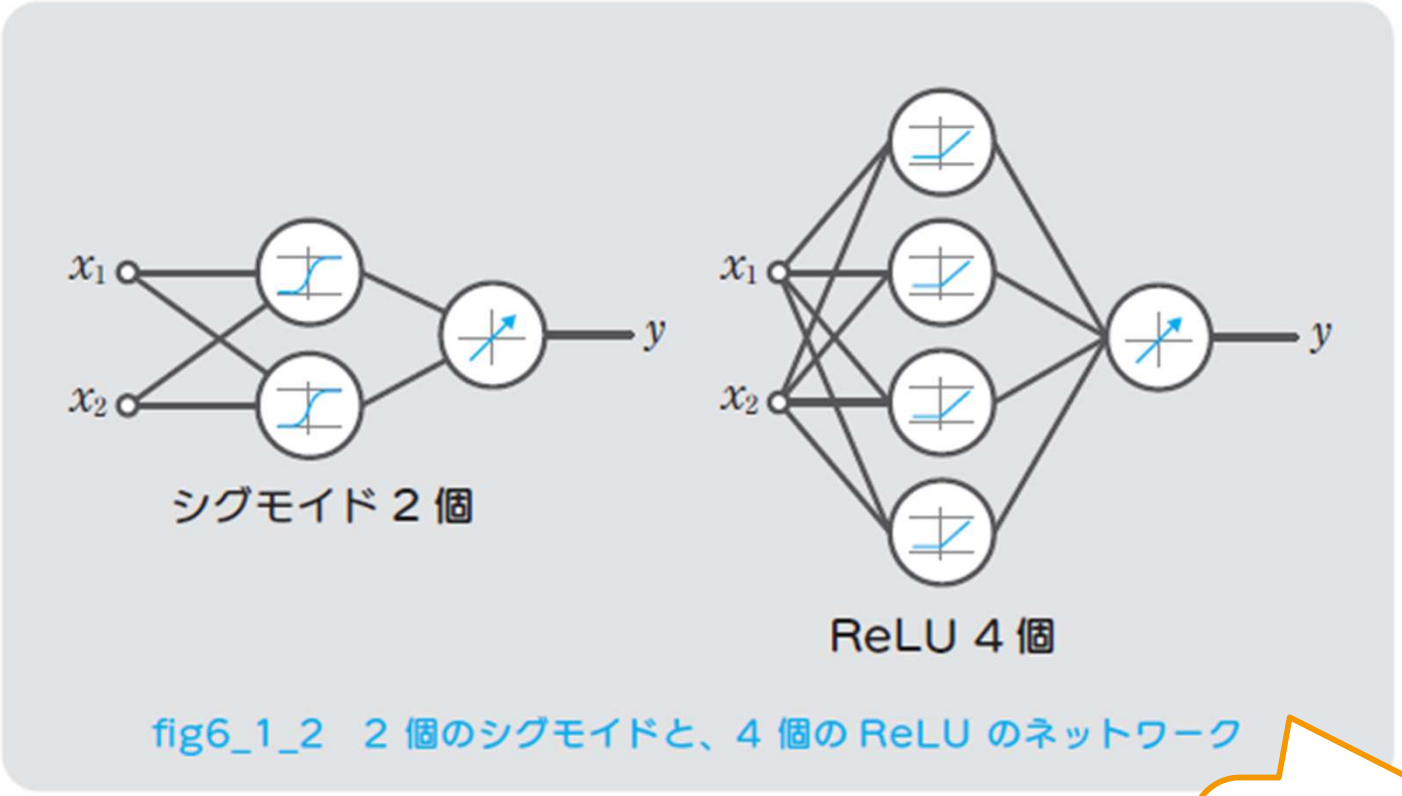


fig6_1_2　２個のシグモイドと、４個の ReLU のネットワーク

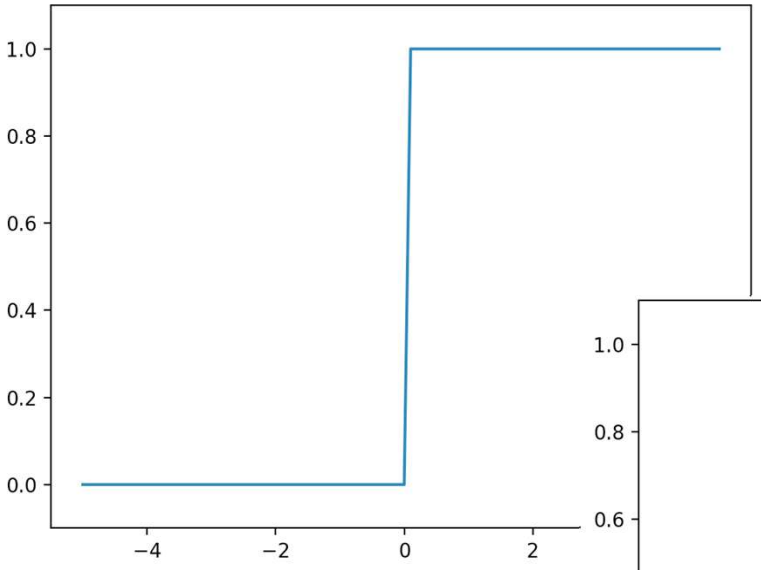
“数で勝負”
戦略

シグモイドなら2個
ReLUなら4個

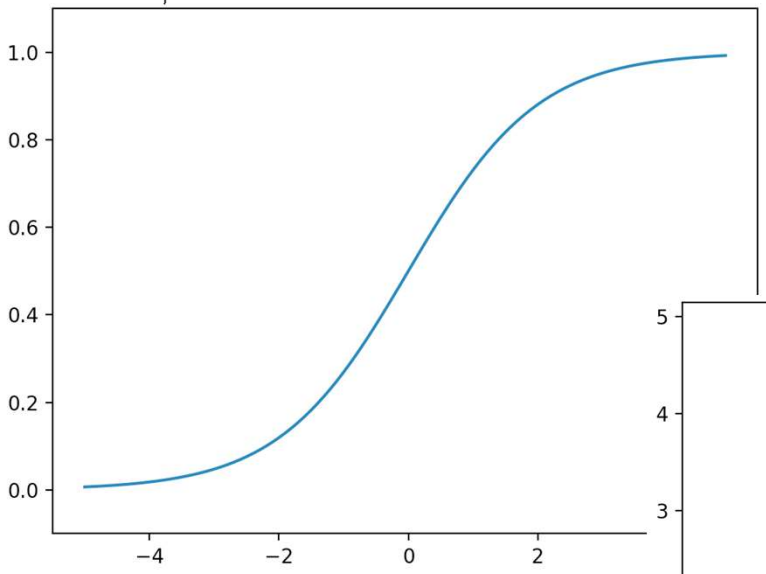
引用元:ディープラーニングの動きを理解するための数式入門

活性化関数の変遷

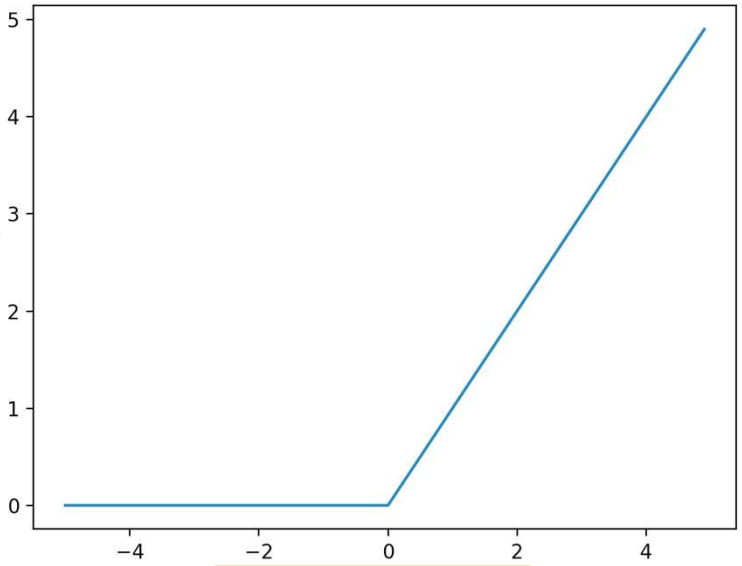
引用元: 活性化関数のまとめ（ステップ、シグモイド、ReLU、ソフトマックス、恒等関数）
https://qiita.com/namitop/items/d3d5091c7d0ab669195f



ステップ(閾値)関数



シグモイド関数



ReLU関数