

深層学習（1）ニューラルネットワークの基礎

（高校生・大学初年次向け）

ニューロンのモデルから多層ネットワークへ

Planned and edited by Yoshio Okamoto / Written by Claude Opus 5.5 (Anthropic)

Ver. 0.2 October 5, 2026

※ 以下の内容はAIが生成したものであり、一応編者がチェックしていますが、間違いや思い込みが含まれている場合があることはご注意ください。

© 2026 Yoshio Okamoto. 教育目的での利用は自由です（出典を明記してください）。

はじめに

写真に写っているのが犬か猫かを見分ける、手書きの数字を読み取る、文章の続きを書く——こうしたことをこなす AI の中身は、**ニューラルネットワーク**と呼ばれる「数を入れると数が出てくる計算の仕組み」です。その部品はとても単純で、同じ部品を何段にも積み重ねることで、驚くほど複雑なことができるようになります。

前の資料「教師あり学習と教師なし学習」では、線形回帰とロジスティック回帰を例に、教師あり学習の枠組み（モデル → 損失関数 → 最小化）を整理しました。今回から4回に分けて、その延長にある**深層学習**（ディープラーニング, deep learning）を扱います。

回	題目	主な内容
第1回	ニューラルネットワークの基礎	ニューロンのモデル, 1層の限界, 多層化, 活性化関数, 表現力
第2回	誤差逆伝播法と学習の実際	連鎖律, 逆伝播の導出, 確率的勾配降下法, 勾配消失, 過学習対策
第3回	畳み込みニューラルネットワーク	畳み込み, 重み共有, プーリング, 画像認識
第4回	系列データと Transformer	RNN, Attention, Transformer

第1回では、**ニューラルネットワークとはどういう形の関数なのか**を扱います。重みの求め方（学習）は第2回で扱うので、今回は、重みが与えられたときにネットワークが何を計算するかに集中します。

読み方 本文は、高校の数学（1次関数と不等式、指数関数、数列の和 Σ 、微分）を使って読めるように書きました。大学で学ぶ道具（行列とベクトルによる書き方、誤差の評価式）が必要になる部分は、**脚注**と**巻末の付録**にまとめました。付録は読み飛ばしても本文の流れは追えます。例題の数値はすべて実際に計算した値です（付録Cのプログラム）。

記号の約束

・ $x_1, x_2, \dots$ は入力, $w_1, w_2, \dots$ は重み, b はバイアスです。

- $\sum_{j=1}^d w_j x_j$ は $w_1 x_1 + w_2 x_2 + \dots + w_d x_d$ のことです.
- $\hat{y}$ (ハット) はネットワークの出力 (予測値) を表します.

1 ニューロンのモデル

1.1 人工ニューロン

脳の神経細胞 (ニューロン, neuron) は, ほかの多数のニューロンから信号を受け取り, その合計がある程度を超えると自分も信号を出します. これを極端に単純化した数理モデルが人工ニューロンです (図1) .

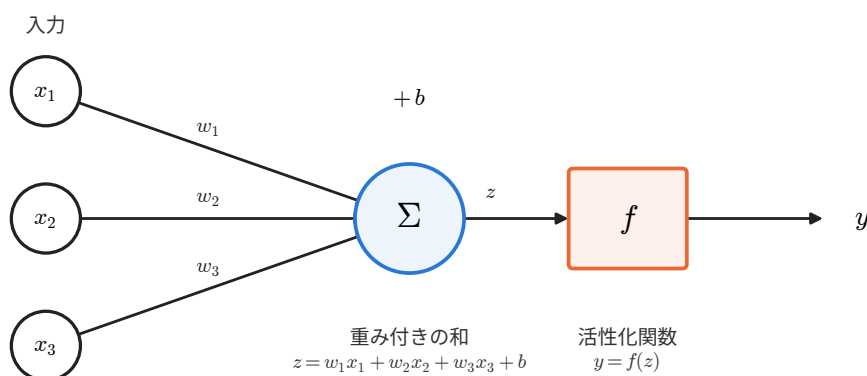


図1 人工ニューロン. 入力 x_i に重み w_i を掛けて足し, バイアス b を加えた値 z を, 活性化関数 f に通して出力 y を得る.

入力が $x_1, \dots, x_d$ の d 個, それぞれの重みが $w_1, \dots, w_d$ のとき, ニューロンは

$$z = \sum_{j=1}^d w_j x_j + b = w_1 x_1 + \dots + w_d x_d + b, \quad y = f(z)$$

を計算します¹. b は**バイアス** (bias, しきい値の役割), f は**活性化関数** (activation function) と呼ばれます.

入試の可否判定にたとえてみます. 英語・数学・理科の点数 (入力) に, 学部ごとに「どの教科をどれだけ重視するか」の倍率 (**重み**) を掛けて合計し, 合格ライン (**バイアス**) と比べて可否 (出力) を決めます. 理学部なら理科の倍率を大きく, 文学部なら英語の倍率を大きく——重みを変えれば, 同じ仕組みで違う判定ができます. 人工ニューロンがしているのは, これと同じ計算です.

- f が**ステップ関数** ($z \geq 0$ なら1, $z < 0$ なら0) のものを**パーセプトロン** (perceptron) といいます (Rosenblatt, 1958年). 合格なら1, 不合格なら0です.
- f が**シグモイド関数** $\sigma(z) = 1/(1 + e^{-z})$ のものは, 前の資料で学んだ**ロジスティック回帰** そのものです. ロジスティック回帰は, 入力の重み付きの和をシグモイド関数に通して「合格する確率」を出すモデルでした.

¹大学では, 入力と重みをベクトル $\mathbf{x} = (x_1, \dots, x_d)^T$, $\mathbf{w} = (w_1, \dots, w_d)^T$ にまとめ, $z = \mathbf{w}^T \mathbf{x} + b$ と書きます ($\mathbf{w}^T \mathbf{x}$ は内積. 付録A) .

つまり、前の資料で学んだロジスティック回帰は「ニューロン1個からなるニューラルネットワーク」だったのです。

1.2 ニューロン1個で論理演算を作る

ステップ関数のニューロンで、2入力の論理演算を作ってみます。入力 x_1, x_2 は0か1です。

x_1	x_2	AND	OR	NAND
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

AND は「両方1のときだけ1」，OR は「どちらか一方でも1なら1」，NAND は AND の反対 (Not AND) です。電気回路にたとえると，AND は2つのスイッチを**直列**につないだ回路（両方入れたときだけ電球がつく），OR は**並列**につないだ回路（どちらか一方を入れればつく）です。

重みとバイアスを次のように選べば，それぞれの論理演算になります。

演算	w_1	w_2	b	$z = w_1x_1 + w_2x_2 + b$
AND	1	1	-1.5	両方1のときだけ $z = 0.5 \geq 0$
OR	1	1	-0.5	どちらか1なら $z \geq 0.5$
NAND	-1	-1	1.5	AND の符号を反転

たとえば AND で $(x_1, x_2) = (1, 0)$ なら $z = 1 - 1.5 = -0.5 < 0$ なので出力0， $(1, 1)$ なら $z = 2 - 1.5 = 0.5 \geq 0$ なので出力1です。AND と OR は重みが同じで，違うのは合格ライン（バイアス）だけです。ラインが厳しい（-1.5）と「2つとも1」が必要になり，ゆるい（-0.5）と「1つでも1」で通ります。

1.3 図で見る：直線で平面を2つに分ける

ニューロンの出力が切り替わる境目は $z = w_1x_1 + w_2x_2 + b = 0$ ，つまり**平面上の直線**です。たとえば AND の境目は $x_1 + x_2 - 1.5 = 0$ ，すなわち直線 $x_2 = -x_1 + 1.5$ です。

ニューロン1個は，入力の平面に**定規で1本の直線を引き**，片側に1，反対側に0を出力します（図2）。この性質を「ニューロン1個は**線形分離**しかできない」と表現します²。

2 ニューロン1個の限界：XOR 問題

2.1 XOR は1本の直線で分けられない

排他的論理和 XOR は，入力のどちらか一方だけが1のときに1を出力します。

²入力が3個なら境目 $w_1x_1 + w_2x_2 + w_3x_3 + b = 0$ は3次元空間の平面，一般に d 個なら d 次元空間の**超平面**になります。

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

身近な XOR の例が、**階段の照明**です。階段の下と上にスイッチがあり、どちらのスイッチを切り替えても電灯の点灯・消灯が入れ替わります。2つのスイッチの状態が「違う」とき点灯、「同じ」とき消灯——これがちょうど XOR です。

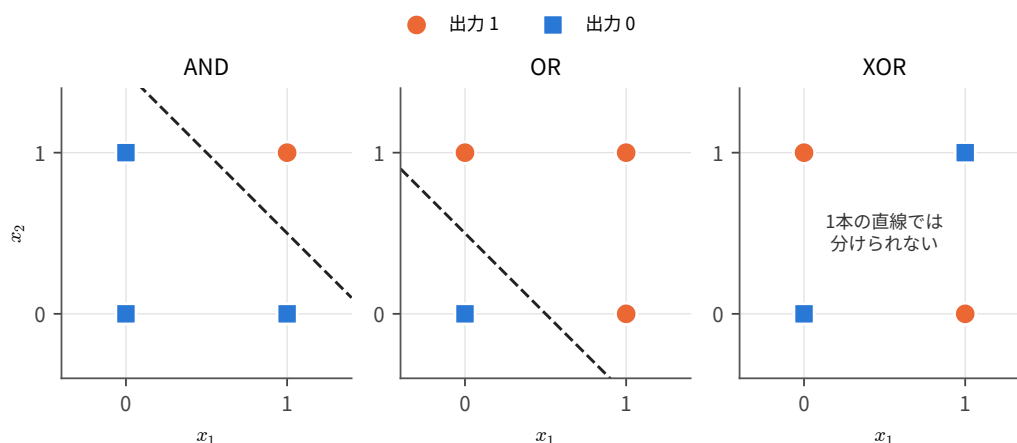


図2 AND, OR, XOR の入力と出力。丸が出力1, 四角が出力0, 破線はニューロンの境界線 $z = 0$. AND と OR は1本の直線で分けられるが, XOR は分けられない。

図2の右のように、出力1の点 $((0,1)$ と $(1,0)$) と出力0の点 $((0,0)$ と $(1,1)$) が、チェス盤の白と黒のように対角に並ぶため、**定規1本ではどうしても分けられません**。

確かめてみよう ステップ関数のニューロンで XOR が作れたとすると、4つの入力について

$$\begin{aligned}
 (0,0) : & b < 0 \\
 (0,1) : & w_2 + b \geq 0 \\
 (1,0) : & w_1 + b \geq 0 \\
 (1,1) : & w_1 + w_2 + b < 0
 \end{aligned}$$

がすべて成り立つ必要があります。2番目と3番目の式を足すと $w_1 + w_2 + 2b \geq 0$, つまり $w_1 + w_2 + b \geq -b$ です。1番目の式から $-b > 0$ なので $w_1 + w_2 + b > 0$ となり、4番目の式と矛盾します。したがって、**どんな w_1, w_2, b を選んでも XOR は作れません**。シグモイド関数などを使っても、出力を0.5で区切る限り境目は直線なので、結論は同じです。

この限界は1969年に Minsky と Papert によって指摘され、ニューラルネットワークの研究が一時停滞する原因の一つになりました。

2.2 解決策：ニューロンを重ねる

XOR は、AND・OR・NAND を組み合わせると作れます。

$$\text{XOR}(x_1, x_2) = \text{AND}(\text{OR}(x_1, x_2), \text{NAND}(x_1, x_2))$$

ことばで言えば「少なくとも1つは1（OR）で、しかも両方1ではない（NAND）」です。つまり、1段目でORとNANDを計算し、その2つの結果を2段目のANDに入れるという2層の構造にすればよいのです（図3(a））。

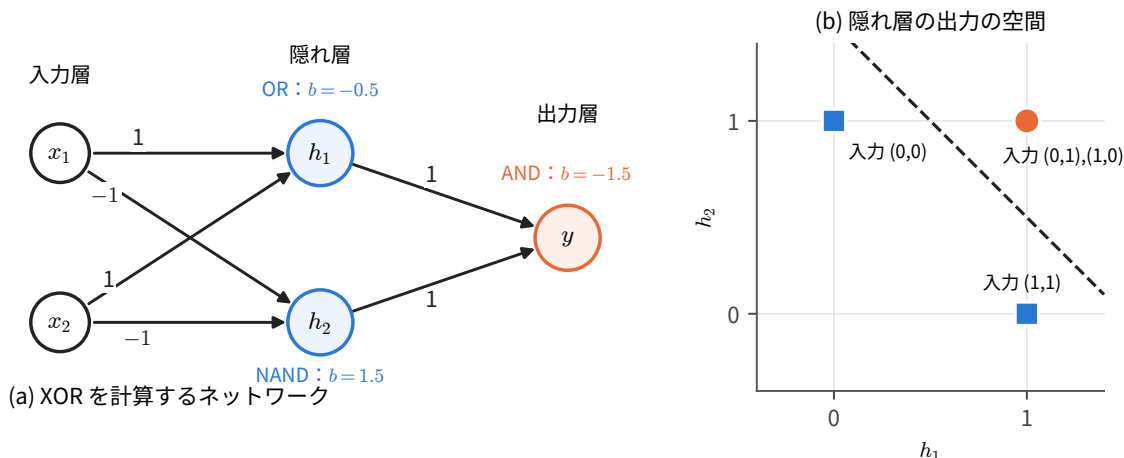


図3 (a) XOR を計算する2層のネットワーク。矢印の数字が重み、 b がバイアス（活性化関数はステップ関数）。(b) 隠れ層の出力 (h_1, h_2) の空間。4つの入力が3つの点に移され、1本の直線（破線）で分けられるようになる。

各入力に対する計算は次のとおりです（付録C.1のプログラム）。

x_1	x_2	$h_1 = \text{OR}$	$h_2 = \text{NAND}$	$y = \text{AND}(h_1, h_2)$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

2.3 隠れ層は「表現」を変換している

なぜ2層にすると解けるのでしょうか。図3(b) は、入力 (x_1, x_2) を1層目の出力 (h_1, h_2) に置き換えた図です。

- ・ 入力 $(0, 0) \rightarrow (h_1, h_2) = (0, 1)$
- ・ 入力 $(0, 1)$ と $(1, 0) \rightarrow$ どちらも $(1, 1)$
- ・ 入力 $(1, 1) \rightarrow (1, 0)$

元の平面では対角に並んでいて分けられなかった点ですが、 (h_1, h_2) の平面では1本の直線 $h_1 + h_2 = 1.5$ で分けられる配置に移されています。2層目のニューロンは、この新しい平面で線形分離をしているだけです。

1層目は、データを「2層目が直線で分けられる形」に変換している。

紙に描いたチェス盤の4マスで、定規1本で白と黒に分けることはできません。しかし、2つの黒マスを結ぶ対角線で紙を折ると、2つの白マスがぴったり重なり、黒2マスは折り目の上に並びます。こうなれば、折り目に平行な1本の線で白と黒を分けられます。隠れ層がしているのは、この「紙を折りたたむ」ことにあたります。

この「入力を、問題を解きやすい別の表現に変換する」ことこそが、多層のニューラルネットワークの本質です。入力層と出力層の間にある層は、外から直接見えないので**隠れ層**（中間層、hidden layer）と呼ばれます。

ここでは重みを人が考えて決めましたが、実際のニューラルネットワークでは、**どんな変換をすればよいかも含めて、重みをデータから学習します**（第2回）。人が特徴の取り出し方を設計しなくてよい点が、深層学習の大きな特徴です。

3 多層ニューラルネットワーク

3.1 層を重ねる

隠れ層のニューロンが m 個あるとき、各ニューロン k ($k = 1, \dots, m$) は、同じ入力 $x_1, \dots, x_d$ を受け取り、それぞれ自分の重みとバイアスで

$$h_k = f\left(\sum_{j=1}^d W_{kj} x_j + b_k\right)$$

を計算します。 W_{kj} は「入力 j からニューロン k への重み」で、添字が2つあるのは「どのニューロンの」「どの入力に対する」重みかを区別するためです。 m 個のニューロンの出力 $h_1, \dots, h_m$ が、次の層への入力になります。

層を L 段重ねたネットワークでは、

入力 $\rightarrow$ 1層目 $\rightarrow$ 2層目 $\rightarrow \dots \rightarrow L$ 層目（出力 $\hat{y}$ ）

と、前の層の出力を次の層の入力にして、順に計算を進めます。

どの層でも計算は同じで、「**重み付きの和をとり、バイアスを足し、活性化関数に通す**」操作の繰り返しです。入力から順に計算を進めることを**順伝播**（forward propagation）といいます。

工場の流れ作業にたとえると、各層は1つの工程です。前の工程から届いた部品（数）を組み合わせて少し加工し、次の工程に送ります。1つひとつの工程は単純でも、工程をいくつも重ねると複雑な製品ができあがります。

大学では、1つの層の計算を行列とベクトルを使って $\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$ と1つの式で書きます。図3(a)のXORネットワークを行列で書いた例も含めて、付録Aにまとめました。最後の層（出力層）の関数については6節で扱います。

3.2 「深層」とは

隠れ層を何層も重ねたネットワークを**深層ニューラルネットワーク**（deep neural network）と呼び、その学習を**深層学習**といいます。「深い」とは層の数が多い（工程が多い）という意味です。

3.3 パラメータの数

たとえば、 $28 \times 28 = 784$ 画素の手書き数字の画像を入力とし、隠れ層100個、出力10個（数字0～9）のネットワークを考えます。隠れ層の100個のニューロンは、それぞれ784個の入力すべてに重みをもち、バイアスも1個ずつもちます。

層	重みの数	バイアスの数	計算
入力 → 隠れ層	78,400	100	784×100
隠れ層 → 出力	1,000	10	100×10
合計	79,510		

重みやバイアスの1つひとつを、ラジオのつまみだと思ってください。この小さなネットワークでも、約8万個のつまみがあります。大規模言語モデル（LLM）では、つまみの数（パラメータの数）が数十億～数千億個にもなります³。このつまみをどう合わせるかが、第2回の「学習」です。

4 活性化関数

4.1 代表的な活性化関数

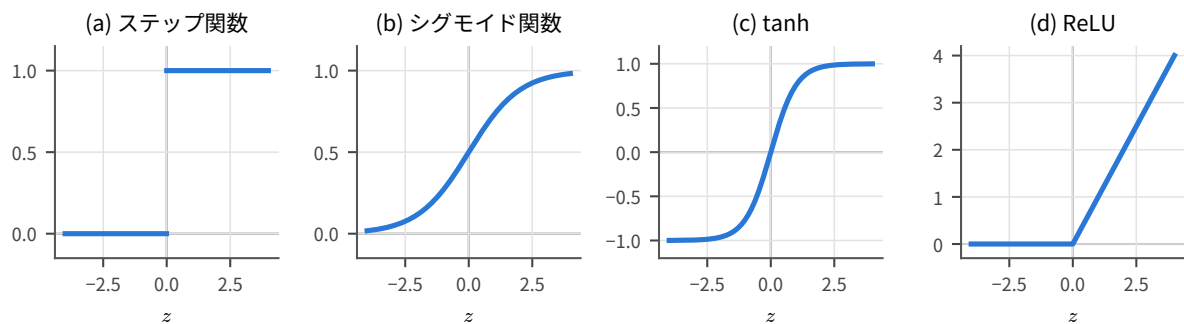


図4 代表的な活性化関数.

名前	式	値の範囲	導関数
ステップ関数	$1 (z \geq 0), 0 (z < 0)$	0 または 1	0 ($z = 0$ で不連続)
シグモイド関数	$\sigma(z) = \frac{1}{1 + e^{-z}}$	$0 < \sigma < 1$	$\sigma(z)(1 - \sigma(z))$
tanh (双曲線正接)	$\tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$-1 < \tanh z < 1$	$1 - \tanh^2 z$
ReLU	$\max(0, z)$	0 以上	$1 (z > 0), 0 (z < 0)$

$\max(0, z)$ は「0 と z のうち大きいほう」という意味で、 $z > 0$ なら z 、 $z \leq 0$ なら 0 です。

³たとえば「8Bモデル」と呼ばれる LLM のパラメータは約80億個（B は billion, 10億）で、この例の10万倍の規模です。

- **ステップ関数**は、導関数が ($z = 0$ 以外では) いつも0なので、勾配降下法が使いません⁴。真っ平らな台地の上で「下り坂の方向」を探しても見つからないのと同じです。学習させるネットワークでは使われません。
- **シグモイド関数**と **tanh** は滑らかで、古くから使われてきました⁵。ステップ関数の角を丸めて、「急な段差」を「なだらかな坂」に変えたものです。
- **ReLU** (Rectified Linear Unit, 正規化線形関数) は、計算が簡単で、 $z > 0$ で導関数がいっぱい1なので学習が進みやすく、現在の深層学習で最も広く使われています。その理由は第2回 (勾配消失) で説明します。

ReLU の rectified は「整流した」という意味です。電気回路のダイオードによる半波整流が、電流を一方向にだけ通し、逆向きは0にするのと同じ形をしています。正の値はそのまま通し、負の値は0にするのです。

4.2 なぜ非線形な活性化関数が必要か

活性化関数をなくして ($f(z) = z$ として) 層を重ねるとどうなるでしょうか。入力1個、隠れ層のニューロン1個の簡単な場合を考えます。

$$h = w_1x + b_1, \quad \hat{y} = w_2h + b_2$$

1つ目の式を2つ目に代入すると

$$\hat{y} = w_2(w_1x + b_1) + b_2 = (w_2w_1)x + (w_2b_1 + b_2)$$

となり、傾き w_2w_1 、切片 $w_2b_1 + b_2$ の1次関数、つまり**1層のネットワークと同じ**になってしまいます。何層重ねても、1次関数の合成は1次関数です (入力やニューロンが多い場合も同じです。付録A.3)。

コピー機で200%拡大し、それをさらに150%拡大しても、結果は「300%拡大」という1回の拡大と同じです。拡大・縮小 (1次関数) を何回重ねても、できるのは1回の拡大・縮小だけで、まっすぐな線を曲げることはできません。
非線形な活性化関数をはさむことで、はじめて層を重ねる意味が生まれる。

XOR が解けたのも、ステップ関数という非線形な関数をはさんだからです。先ほどの「紙を折りたたむ」操作は、1次関数 (拡大・縮小や平行移動) ではできません。折り目をつけるのが、非線形な活性化関数の役割です。

5 ニューラルネットワークの表現力

5.1 どんな関数でも表せるか

隠れ層が1層だけでも、隠れ層のニューロンを十分多くすれば、(有限な区間の上の) どんな連続関数でも好きな精度で近似できることが知られています。これを**万能近似定理** (universal approximation theorem) といいます (Cybenko, 1989年ほか)。

⁴勾配降下法は「損失の傾きと反対の向きに重みを少し動かす」方法でした。傾きがどこでも0では、どちらに動かせばよいかわかりません。

⁵tanh は、シグモイド関数を縦に2倍して1下げた形をしています ($\tanh z = 2\sigma(2z) - 1$)。

5.2 ReLU の重ね合わせで曲線を作る

万能近似定理の感覚をつかむために、ReLU を使って $y = \sin 2\pi x$ ($0 \leq x \leq 1$) を近似してみます。

ReLU を横にずらした関数 $\max(0, x - c)$ は、 $x < c$ では0、 $x > c$ では傾き1の直線で、 $x = c$ で折れ曲がる「折れ線」です。これを重みを付けて足し合わせると、好きな位置で好きな角度だけ折れ曲がる折れ線が作れます。

$$\hat{y}(x) = b_0 + \sum_{k=0}^{S-1} a_k \max(0, x - c_k)$$

これは、隠れ層に ReLU のニューロンが S 個ある1入力・1出力のネットワークにほかなりません（ニューロン k の重みは1、バイアスは $-c_k$ 、出力層への重みは a_k ）。

重みの決め方 区間 $[0, 1]$ を S 等分し、分点 $c_k = k/S$ での関数値を $y_k = \sin(2\pi k/S)$ とします。区間 k ($c_k \leq x \leq c_{k+1}$) で折れ線の傾きを $s_k = (y_{k+1} - y_k) \div \frac{1}{S}$ にしたいとします。 $\max(0, x - c_k)$ は $x > c_k$ で傾き1なので、 x が c_k を越えるたびに、折れ線の傾きが a_k だけ増えます。したがって

$$b_0 = y_0, \quad a_0 = s_0, \quad a_k = s_k - s_{k-1} \quad (k = 1, \dots, S-1)$$

とすれば、各区間で傾きがちょうど s_k になり、折れ線が分点をすべて通ります。

こうして作った折れ線は図5のようになります（付録C.2のプログラム）。

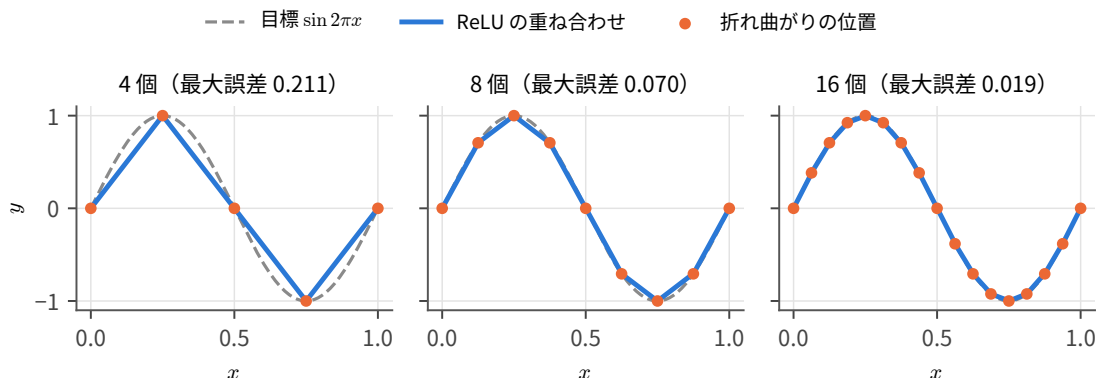


図5 ReLU の重ね合わせによる $\sin 2\pi x$ の近似。見出しの数字は隠れ層のユニット（ニューロン）の数。灰色の破線が目標の関数、青線がネットワークの出力、点は折れ曲がりの位置。

隠れユニットの数 S	4	8	16
最大誤差	0.211	0.070	0.019

ユニットを2倍にすると誤差はおよそ $1/4$ になり、いくらでも精度を上げられることがわかります⁶。

円を正多角形で近似するのと同じです。半径1の円に内接する正6角形では、辺と円のすき間は最大0.134ありますが、正12角形では0.034、正24角形では0.0086と、辺の数を2倍にするたびに約 $1/4$ になります。アルキメデスが円周率を求めたときの考え方です。まっす

ぐな部品でも、数を増やせば曲線にいくらでも近づけます。

5.3 では、なぜ「深く」するのか

万能近似定理は「隠れ層1層で十分」と言っているように見えます。しかし、この定理は**必要なユニット数がどれだけになるか**については何も保証しません。

実際には、複雑な関数を1層で表そうとするとユニット数が膨大になることが多く、層を重ねたほうがはるかに少ないパラメータで同じ関数を表せる場合があることがわかっています。

直観的には、2節の XOR の例のように、各層が「前の層の出力を組み合わせ、少しずつ複雑な特徴を作る」ことができるからです。文字を組み合わせで単語を、単語を組み合わせで文を作るように、部品の組み合わせを段階的に積み上げるほうが、すべてを一度に作るよりずっと効率的です。たとえば画像認識では、浅い層が輪郭や模様を、深い層が目や車輪のような部品を、さらに深い層が物体全体を表すようになることが観察されています（第3回）。

また、万能近似定理は「近似できる重みが**存在する**」ことを言っているだけで、その重みを**データから見つけられる**かどうかは別の問題です。宝物がどこかに埋まっていると保証されても、掘り当てる方法がわからなければ手に入らないのと同じです。これが第2回の主題です。

6 出力層と損失関数

最後の層（出力層）の関数と損失関数は、課題に応じて前の資料と同じ組み合わせを使います。

課題	出力層の関数	損失関数	前の資料の対応する手法
回帰	そのまま ($g(z) = z$)	2乗誤差	線形回帰
2クラス分類	シグモイド関数	交差エントロピー	ロジスティック回帰
多クラス分類	ソフトマックス関数	交差エントロピー	多クラスのロジスティック回帰

つまり、ニューラルネットワークは、**前の資料の線形回帰やロジスティック回帰の手前に、入力を変換する隠れ層を何層も付け加えたもの**と見ることができます。隠れ層がデータを扱いやすい表現に変換し（紙を折りたたみ）、最後の層がその表現を使って回帰や分類を行う（定規で線を引く）のです。

7 まとめ

- 人工ニューロンは、入力の重み付きの和にバイアスを足し、活性化関数に通した値 $y = f(w_1x_1 + \dots + w_dx_d + b)$ を計算する。シグモイド関数を使えばロジスティック回帰と同じである。
- ニューロン1個の境目は直線（一般には超平面）なので、XOR のような線形分離でき

⁶誤差は $1/S^2$ に比例して小さくなります。その理由（折れ線近似の誤差の評価）は付録Bにあります。正多角形のすき間は $1 - \cos(\pi/n)$ で、同じ理由で $1/n^2$ に比例します。

ない問題は解けない。

3. 層を重ねると、隠れ層が入力を別の表現に変換し、線形分離できない問題も解けるようになる。
4. 多層ネットワークは「重み付きの和をとり、バイアスを足し、非線形な活性化関数に通す」ことの繰り返しである。非線形性がなければ何層重ねても1層と同じになる。
5. 隠れ層のユニットを増やせば、どんな連続関数でも近似できる（万能近似定理）。ただし、その重みをどう見つけるかは別の問題である。

次回予告 今回は重みを人が決めました。しかし実際のネットワークには数万～数十億個の重み（つまみ）があり、人が決めることはできません。第2回では、損失関数の勾配を効率よく計算して重みを学習する**誤差逆伝播法**を扱います。

付録A 行列とベクトルで書く

この付録では、行列とベクトル（大学初年次の線形代数）を使います。

記号 ベクトルは太字の小文字（ $\mathbf{x}$, $\mathbf{h}$ ），行列は大文字（ W ）で書きます。ベクトルは縦ベクトルとします。右肩の T は**転置**（transpose）を表し、べき乗ではありません。転置とは行列の行と列を入れ替える操作で、横に並べた $(x_1, \dots, x_d)$ に付けると縦ベクトルになります。行列 A の転置 A^T は、 (i, j) 成分が A_{ji} の行列です。 $\mathbf{w}^T \mathbf{x} = w_1 x_1 + \dots + w_d x_d$ は内積（横ベクトル $\times$ 縦ベクトル = 1個の数）です。 $W^{(1)}$, $\mathbf{h}^{(2)}$ のように右肩に**かっこ付き**で書いた数字は**層の番号**で、べき乗ではありません。 $f(\mathbf{z})$ のように関数をベクトルに作用させるときは、各成分に別々に作用させる（ $f(\mathbf{z}) = (f(z_1), \dots, f(z_m))^T$ ）ものとします。

A.1 1つのニューロン

入力を $\mathbf{x} = (x_1, \dots, x_d)^T$ ，重みを $\mathbf{w} = (w_1, \dots, w_d)^T$ とすると、

$$z = \sum_{j=1}^d w_j x_j + b = \mathbf{w}^T \mathbf{x} + b, \quad y = f(z)$$

です。境目 $\mathbf{w}^T \mathbf{x} + b = 0$ は d 次元空間の**超平面**で、 $\mathbf{w}$ はこの超平面に垂直なベクトル（法線ベクトル）です。

A.2 層を行列で書く

隠れ層のニューロンが m 個あるとき、3.1節の

$$h_k = f\left(\sum_{j=1}^d W_{kj} x_j + b_k\right), \quad k = 1, \dots, m$$

をまとめて行列で書くと

$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$$

となります。 W は $m \times d$ 行列で、その k 行 j 列の成分 W_{kj} が「入力 j からニューロン k への重み」です。 $\mathbf{b}$ は m 次元ベクトルで、 f はベクトルの各成分に別々に作用させます。

層を L 段重ねたネットワークは、

$$\mathbf{h}^{(1)} = f(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \quad \mathbf{h}^{(2)} = f(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}), \quad \dots, \quad \hat{\mathbf{y}} = g(W^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)})$$

と、「行列を掛けてバイアスを足し、活性化関数に通す」操作の繰り返しで表されます (g は出力層の関数で、6節で扱いました)。

図3(a) の XOR ネットワークを行列で書くと、

$$W^{(1)} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, \quad \mathbf{b}^{(1)} = \begin{pmatrix} -0.5 \\ 1.5 \end{pmatrix}, \quad W^{(2)} = \begin{pmatrix} 1 & 1 \end{pmatrix}, \quad b^{(2)} = -1.5$$

です。 $W^{(1)}$ の1行目が OR, 2行目が NAND の重みです。3.3節のパラメータ数の例では、 $W^{(1)}$ は 100×784 行列、 $W^{(2)}$ は 10×100 行列です。

A.3 線形な層を重ねても1層と同じ

活性化関数をなくして ($f(\mathbf{z}) = \mathbf{z}$ として) 2層重ねると、

$$\mathbf{h} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \hat{\mathbf{y}} = W^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

を代入して

$$\hat{\mathbf{y}} = W^{(2)}W^{(1)}\mathbf{x} + (W^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}) = W'\mathbf{x} + \mathbf{b}'$$

となり、行列 $W' = W^{(2)}W^{(1)}$ とベクトル $\mathbf{b}'$ をもつ1層のネットワークと同じになります。何層重ねても、線形な変換の合成は線形な変換です。4.2節の1入力の例は、この式で行列がすべて1行1列の場合にあたります。

付録B 折れ線近似の誤差

5.2節の折れ線近似で、誤差がユニット数の2乗に反比例して小さくなる理由を説明します。

関数 $y(x)$ を、幅 h の区間ごとに両端を結ぶ線分で近似するとき、その誤差は、区間の幅 h と関数の2階導関数の絶対値の最大値 M_2 を使って

$$|\text{誤差}| \leq \frac{h^2}{8} M_2$$

以下になることが知られています⁷。

$y = \sin 2\pi x$ では $y'' = -4\pi^2 \sin 2\pi x$ なので $M_2 = 4\pi^2$ です。区間の幅は $h = 1/S$ なので、誤差の上限は

$$\frac{1}{8S^2} \times 4\pi^2 = \frac{\pi^2}{2S^2}$$

です。 $S = 4, 8, 16$ ではそれぞれ 0.31, 0.077, 0.019 で、表の実際の誤差 (0.211, 0.070, 0.019) はこれを下回っています。

⁷区間 $[\alpha, \beta]$ で両端を結ぶ線分を $\ell(x)$ とすると、 $e(x) = y(x) - \ell(x)$ は両端で0になります。ロルの定理 (平均値の定理の特別な場合) を繰り返し使うと $e(x) = -\frac{1}{2}y''(\xi)(x - \alpha)(\beta - x)$ となる ξ が区間内にあり、 $(x - \alpha)(\beta - x)$ の最大値は区間の midpoint の $h^2/4$ なので、 $|e(x)| \leq \frac{h^2}{8} M_2$ が得られます。

誤差の上限が $1/S^2$ に比例するので、ユニットを2倍にすると誤差は $1/4$ になります。

本文の正多角形の例も同じ構造です。半径1の円に内接する正 n 角形の辺と円の最大のすき間は $1 - \cos(\pi/n)$ で、 $\cos \theta \approx 1 - \theta^2/2$ (θ が小さいとき) より約 $\pi^2/(2n^2)$ となり、やはり $1/n^2$ に比例します。

付録C Python コード（標準ライブラリのみ）

本文の例題を計算するプログラムです。外部ライブラリは使わず、Python 3 の標準機能だけで書いてあります。

C.1 論理ゲートと XOR ネットワーク（第1～2節）

```
# 1個のニューロン（ステップ関数）で論理ゲートを作り、
# 2層にして XOR を計算する
def step(z):
    return 1 if z >= 0 else 0

def neuron(x, w, b):
    z = sum(wi * xi for wi, xi in zip(w, x)) + b
    return step(z)

AND = lambda x: neuron(x, [1, 1], -1.5)
OR = lambda x: neuron(x, [1, 1], -0.5)
NAND = lambda x: neuron(x, [-1, -1], 1.5)

def XOR(x):
    h = [OR(x), NAND(x)]          # 隠れ層
    return AND(h), h              # 出力層

print(" x1 x2 | AND OR NAND | h1 h2 | XOR")
for x in [(0, 0), (0, 1), (1, 0), (1, 1)]:
    y, h = XOR(x)
    print(f" {x[0]} {x[1]} | {AND(x)} {OR(x)} {NAND(x)} | {h[0]} {h[1]} | {y}")
```

実行結果：

x1	x2	AND	OR	NAND	h1	h2	XOR
0	0	0	0	1	0	1	0
0	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1
1	1	1	1	0	1	0	0

C.2 ReLU の重ね合わせによる関数近似（第5節）

```
# ReLU を重ね合わせて  $\sin(2\pi x)$  を折れ線で近似する
import math

def relu(z):
    return max(0.0, z)

def build(S):
    """区間 [0,1] を S 等分し、折れ線近似のための重みを求める"""
```

```

knots = [k / S for k in range(S + 1)]
ys = [math.sin(2 * math.pi * t) for t in knots]
slopes = [(ys[k + 1] - ys[k]) * S for k in range(S)]
# 隠れユニット k: relu(x - knots[k]), 出力側の重み: 傾きの変化量
a = [slopes[0]] + [slopes[k] - slopes[k - 1] for k in range(1, S)]
return knots[:S], a, ys[0]

def net(x, c, a, b0):
    return b0 + sum(ak * relu(x - ck) for ck, ak in zip(c, a))

for S in (4, 8, 16):
    c, a, b0 = build(S)
    err = max(abs(net(i / 10000, c, a, b0) - math.sin(2 * math.pi * i / 10000))
              for i in range(10001))
    print(f"隠れユニット {S:2d} 個: 最大誤差 = {err:.4f}")

```

実行結果：

```

隠れユニット  4 個: 最大誤差 = 0.2105
隠れユニット  8 個: 最大誤差 = 0.0704
隠れユニット 16 個: 最大誤差 = 0.0188

```