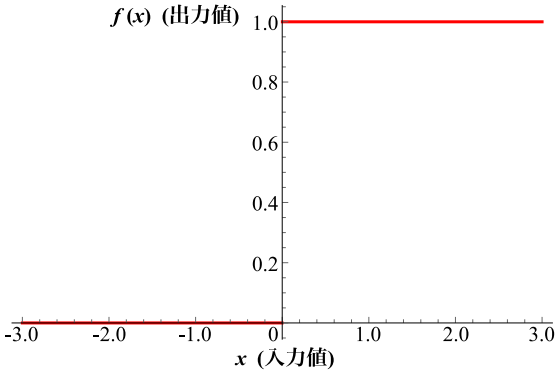
Step 関数

step (ステップ) 関数は、次の式で定義される関数です。

$$f(x) = \begin{cases} 1 & (x \geqq 0) \\ 0 & (x < 0) \end{cases}$$

つまり、入力 x が 0 を超えたら 1 を、それ以外ならば 0 を出力する関数となっています。$x = 0$ のところで値が 0 から 1 へ飛び移りますが、それ以外のところではずっと 0 か 1 で一定、というとても単純な関数です。



step 関数

グラフにすると、上の図のように $x = 0$ のところで階段 (step) 状のグラフとなるのが関数の名前の由来です。また、値が 0 か 1 しかないことから、binary step (バイナリステップ) 関数と呼ばれることもあります。

♣ step 関数の歴史

1943 年に McCulloch (マカロック) と Pitts (ピッツ) により、神経細胞の入出力モデルとして、初めてこの step 関数が導入されました。

♣ Heaviside の階段関数との関係

step 関数とよく似た関数として、Heaviside (ヘヴィサイド) の階段関数 $H(x)$ が知られています。

$$H(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x < 0) \end{cases}$$

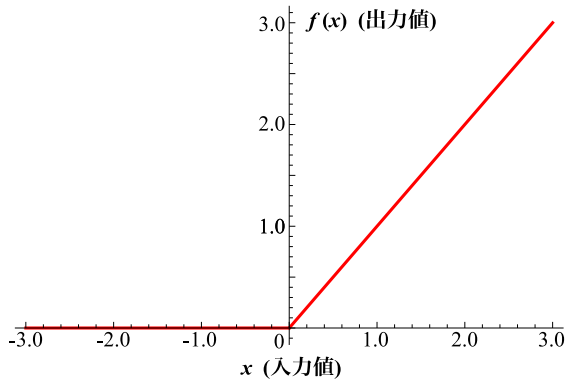
step 関数とほとんど同じですが、x がピッタリ 0 の時の扱いが異なっています。step 関数では $f(0) = 1$ と決まっていますが、Heaviside の階段関数では $x = 0$ の値が決まっていません。実際に使われる時は、$x = 0$ での値を 0, 1, $\dfrac{1}{2}$ など、必要に応じて都合のよい値が選ばれます。

ReLU 関数

ReLU(レルー) 関数[1]は、次の式で定義される関数です。

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leqq 0) \end{cases}$$

つまり、入力 x が 0 以下の場合には 0 を、入力 x が 0 より大きい場合には入力値と同じ x を出力する関数となっています。上限はなく、入力 x が大きくなると出力も大きくなり続けます。



ReLU 関数

グラフにすると、$x = 0$ のところからランプ (高速道路にのるための上り坂のような坂のこと) になっているように見えるため「ramp (ランプ) 関数」とも呼ばれますが、特にニューラルネットワークの文脈では ReLU 関数と呼ばれることの方が多くなっています。

♣ より簡潔な表記

ReLU 関数は、

$$f(x) = \max(0, x)$$

と表現されることもあります。ここで、$\max(a, b)$ は a と b、大きい方の値を返す関数です。つまり、$x < 0$ の時は $x = 0$ の方が大きいため $f(x) = 0$ となり、$x > 0$ では x の方が大きくなるため $f(x) = x$ となります。

♣ ReLU 関数の歴史

2011 年に Xavier Glorot らによって、隠れ層の活性化関数として導入されました。とてもシンプルな関数であるにも関わらず、多くのケースで最善の関数であることが知られており、最近では多くの深層ニューラルネットワークでこの ReLU 関数やその亜種 (LeakyReLU 関数) が用いられています。

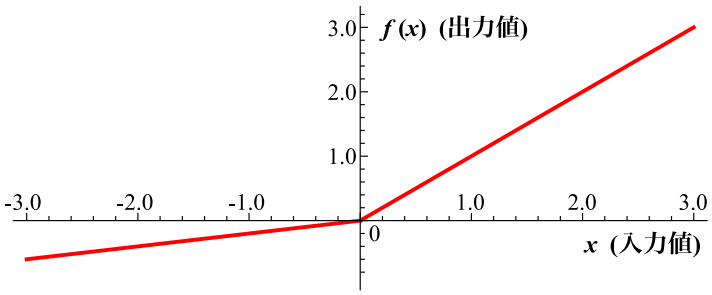
[1] 「ReLU」とは、Rectified Linear Unit の略で、「正規化線形関数」と日本語に訳されることが多いです。

LeakyReLU 関数

LeakyReLU (リーキーレルー) 関数は、次の式で定義される関数です。

$$f(x) = \begin{cases} x & (x > 0) \\ \alpha x & (x \leqq 0) \end{cases} \qquad [\alpha \text{は定数, 基本的には} \alpha = 0.01 \text{ とする}]$$

なんだか難しそうな名前の関数ですが、単に傾きの異なる 2 つの直線を $x = 0$ でくっつけただけの関数です。



LeakyReLU 関数 (見やすくするために、$\alpha = 0.2$ としています)

♣ 名前の由来

LeakyReLU 関数は、ReLU 関数の拡張版です。「Leaky (漏れている)」という言葉の通り、ReLU 関数では入力値 x が 0 以下の場合は出力値が常に 0 となりますが、LeakyReLU 関数では入力値が 0 より小さい場合、出力値は下に漏れ出す (leak) ように 0 より下の値となります。

♣ ReLU 関数との比較

現在のニューラルネットワークでは、隠れ層 (中間層) の活性化関数として ReLU 関数を用いることが一般的となっていますが、より良い結果を求めて考えられたのが、この LeakyReLU 関数です。いくつかの論文では、LeakyReLU 関数を用いた方が精度が良くなると報告されていますが、あまり精度の向上が見られないと結論づけている論文もあります。実際に使用する際には、最適な α 値 (基本的には 0.01) を試行錯誤しながら探しだすことで、精度を向上させることができると考えられます[2]。

[2] α 値を固定せず、学習で決定する場合は、特に PReLU (Parametric ReLU) 関数と呼ばれます。

Softmax関数

softmax (ソフトマックス) 関数は、次の式で定義される関数です。

$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \qquad (i = 1, 2, \cdots, n)$$

なんだかややこしい関数ですが、一言でいえば「n 個のごちゃごちゃなデータ $\{x_1, x_2, \cdots, x_n\}$ を、n 個全ての数の合計が100% (1.0) となったデータ $\{y_1, y_2, \cdots, y_n\}$ に変換する関数」です。以下では、この式に出てくる Σ や e の意味を一つ一つ確認していき、この式を理解していきます。

♣ 2の小数乗？マイナス乗？

softmax関数を理解するために、まず累乗について確認しましょう。中学校では累乗に関しては次のように習うと思います。

「a を n 回掛け算するとき、$a \times a \times \cdots \times a = a^n$ と表す」

具体的には、

$$2^3 = 2 \times 2 \times 2 = 8$$
$$(-2)^3 = (-2) \times (-2) \times (-2) = -8$$

といった具合です。この累乗の考え方は間違ってはいないのですが、次のような式が出てきたときに困ってしまいます。

$$2^0, \quad 2^{-3}, \quad 2^{0.5}$$

2を「0回」かける、2を「−3回」かける、2を「0.5回」かけた値と言われても、さっぱり分かりません。これらの値を考える際には、この「何回かける」という考え方を捨てて、次のようにして規則性を見出すのが良いでしょう。

$$\begin{aligned} 2^3 &= 8 \\ 2^2 &= 4 \\ 2^1 &= 2 \\ 2^0 &= 1 \end{aligned}$$

（各行の右側に $\times \frac{1}{2}$ の矢印）

3乗、2乗、1乗と小さくしていくと、値は $\frac{1}{2}$ 倍されていくことが分かります。この規則性を用いると、0乗は

$$2^0 = 1$$

となります[3]。さらに、 -1 乗、-2 乗、-3 乗と小さくしていくと、値は $\frac{1}{2}$ 倍されていくため、

$$2^{-1} = 1 \times \frac{1}{2} = \frac{1}{2^1} = \frac{1}{2}$$

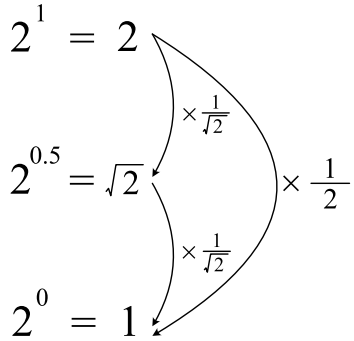
$$2^{-2} = \frac{1}{2} \times \frac{1}{2} = \frac{1}{2^2} = \frac{1}{4}$$

$$2^{-3} = \frac{1}{4} \times \frac{1}{2} = \frac{1}{2^3} = \frac{1}{8}$$

となることが分かります。ここでは 2 の累乗を具体的に考えましたが、一般に a の $-n$ 乗は、

$$a^{-n} = \frac{1}{a^n}$$

となっています。0.5 乗についても同様に規則性から考えましょう。



1 乗と 0 乗の間にあるため、$2^1 = 2$ を $\frac{1}{2}$ の「半分倍」すれば良さそうです。ただし、$\frac{1}{4}$ 倍ということではありません。2 回かけた時に $\frac{1}{2}$ 倍となっているので、$\frac{1}{\sqrt{2}}$ 倍することになります[4]。

$$2^{0.5} = 2 \times \frac{1}{\sqrt{2}} = \sqrt{2}$$

つまり、2 の 0.5 乗は $\sqrt{2}$ ということになります[5]。

ただ、「-3 乗とか 0.5 乗ってそもそもどういう意味？」と言われると、なかなか直感的で分かりやすいイメージはこれと言ってありません。このように計算すると決めておくと、矛盾が生じず都合が良い、と思っておけば十分です。累乗は、適切に計算のルールを決めることで、2 乗や 3 乗のような「正の整数乗」のみならず、様々な数の場合に拡張できる概念となっているのです。

---

[3] 0 乗は 0 でないことに注意しましょう。

[4] $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}} = \frac{1}{2}$

[5] 同じような考え方で、他の小数乗の値も考えることができます。

♣ 秀吉も驚いた指数関数

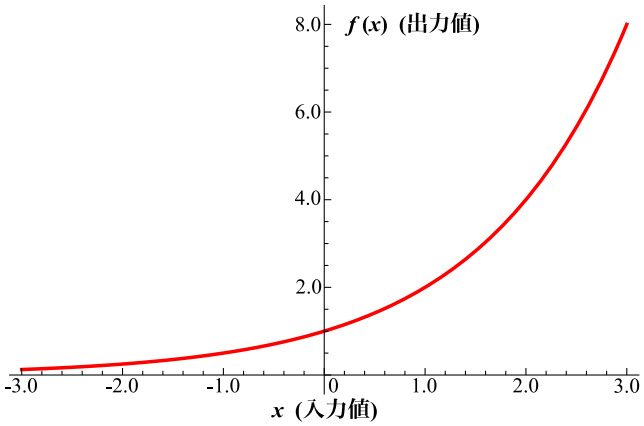
次に、指数関数と呼ばれる関数を確認しましょう。指数関数の「指数 (しすう)」というのは、累乗の際に右肩にのせる小さい数のことです。例えば

$$2^3 = 2 \times 2 \times 2 = 8$$

の場合、2 の右肩にのっている 3 が指数です。一方、2 のことは「底 (てい)」といいます。指数関数は、正の実数 a が底で指数が変数 x となっている関数で、一般に

$$f(x) = a^x$$

という式で表されます[6]。具体的に底 a が 2 の指数関数 $f(x) = 2^x$ のグラフを描いてみると、次のようになります。x が大きくなるにつれて値が大きくなる曲線であることが分かります。



指数関数 $f(x) = 2^x$

先ほど説明したように、累乗の考え方を拡張することで、x が負や小数の場合にもちゃんと値を持つようになっています。

この指数関数の最大の特徴は、最初はゆっくり増えていくが気付くとものすごい勢いで増えていく、という点です。この性質にまつわる有名な話で「曽呂利新左衛門（そろりしんざえもん）の逸話」が知られています。曽呂利新左衛門は豊臣秀吉に仕えていた人物です。ある日、秀吉に「好きな褒美をやろう」と言われた新左衛門は、次のようにお願いしました。

> 「米粒を頂きたいです。1 日目は 1 粒、2 日目は倍の 2 粒、3 日目はその倍の 4 粒といった具合で、次々に倍をしていって、100 日目まで米粒を頂きたいです。」

秀吉は「そんな小さな望みで良いのか」と思い、新左衛門の要求を快諾しました。最初は 1 粒、2 粒、4 粒と雀の涙ほどの量でした。そして 15 日目にやっと 16384 粒となり、茶わん 4 杯分くらいの米粒になりました。しかし、この辺から 1 日に貰える米粒の数が急激に増えてきます。20 日目には 524288 粒になり、約 10kg 分の米粒になりました。23 日目には、なんと 4194304 粒に

[6] 2^x は指数関数ですが、x^2 は 2 次関数です。変数 x が指数にあるものを指数関数という点に注意して下さい。

なり、米俵 2 俵ほどの量になります。驚いた秀吉は、100 日目に何粒になるか家来たちに計算させました。すると、

$$633825300114114700748351602688 \text{ 粒}$$

というとんでもない数になることが分かり、秀吉は新左衛門の賢さにしてやられた、という話です。n 日目に貰える米粒の数は 2^n 粒と指数関数になっており、指数関数の性質がよく分かる逸話です。

♣ ネイピア数 e

ネイピア数[7] e は、

$$e = 2.71828182845904523536028747135 2\cdots$$

という値の定数です[8]。円周率 π と同じ無理数 (＝小数点以下が無限に続く数) です。なぜこんな変な値を考えるか不思議でしょうが、ここでは単に 2.7 ぐらいの数とでも思っておけば十分です (数学を勉強していくと、このネイピア数 e は至る所で出てきて、この値が深い意味を持っていることが分かってきます)。因みに、ネイピア数という名前は、ジョン・ネイピア (John Napier: 1550〜1617) というスコットランドの数学者に由来しています[9]。

♣ 便利な総和記号

何か 5 つのデータがあったとして、それらを $\{x_1, x_2, x_3, x_4, x_5\}$ とします。このデータの合計は、

$$x_1 + x_2 + x_3 + x_4 + x_5$$

と表せます。では、100 個のデータ $\{x_1, x_2, \cdots, x_{100}\}$ の場合にはどうなるでしょうか。流石に 100 個の足し算を書くわけにはいかないので、次のように「$\cdots$」を使って途中をサボるでしょう。

$$x_1 + x_2 + \cdots + x_{100}$$

もちろん、この「$\cdots$」を使った書き方は分かりやすくて良いのですが、総和記号 Σ を用いると、次のようにコンパクトに表現することができます。

$$x_1 + x_2 + \cdots + x_{100} = \sum_{i=1}^{100} x_i$$

記号の Σ はギリシャ文字のシグマの大文字です[10]。上の式で i のことを「添え字」といいます。Σ の下側に、この添え字の初期値 (上の式では $i = 1$) を書き、Σ の上側に添え字がいくつにな

[7] オイラー数と呼ばれることもあります
[8] [語呂合わせ] 鮒一鉢二鉢一鉢二鉢至極惜しい (ふなひとはちふたはちひとはちふたはちしごくおしい)
[9] ネイピアは、物理学者、天文学者、占星術師としても知られています
[10] シグマの小文字は σ で、標準偏差を表す文字として統計の分野でよく出てきます

るまで足し続けるか (上の式では $i = 100$) を書きます [11]。例えば、100 個のデータのうち、5 番目から 10 番目までのデータの合計は、

$$\sum_{i=5}^{10} x_i = x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$$

と表すことができます。なお、添え字は i でなくても構いません。

$$\sum_{i=5}^{10} x_i = \sum_{j=5}^{10} x_j = \sum_{k=5}^{10} x_k = \sum_{\star=5}^{10} x_\star$$

上の式は全て $x_5 + x_6 + \cdots + x_{10}$ を表していて、全く同じ意味の式となっています。添え字の文字自体はあまり重要ではなく、「どこからどこまで添え字が動くか」が重要なのです [12]。

慣れてしまえば非常に便利な記号なのですが、慣れるまでは扱いにくい記号かもしれません。分からない時は、「$\cdots$」を使って具体的にどのような足し算になっているか確認するようにすると (この作業を「書き下す」と言います)、理解が深まるはずです。

♣ もう 1 度 softmax 関数の式を見る

以上で、softmax 関数を理解するために必要な知識が揃いましたので、もう 1 度式を見てみます。

$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \qquad (i = 1, 2, \cdots, n)$$

まず、分母を見てみましょう。総和記号を書き下してみると、

$$\sum_{k=1}^{n} e^{x_k} = e^{x_1} + e^{x_2} + \cdots + e^{x_n}$$

となります。ここで、e^x は底がネイピア数 e の指数関数です [13]。つまり、分母は入力値 $\{x_1, x_2, \cdots, x_n\}$ を指数関数 e^x に入力した時の、出力値の合計となっていることが分かります。あとは、この分母の値を用いて、softmax 関数の n 個の出力値 $\{y_1, y_2, \cdots, y_n\}$ が、

$$\begin{aligned}
y_1 &= \frac{e^{x_1}}{\sum_{k=1}^{n} e^{x_k}} = \frac{e^{x_1}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} \\
y_2 &= \frac{e^{x_2}}{\sum_{k=1}^{n} e^{x_k}} = \frac{e^{x_2}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} \\
&\vdots \\
y_n &= \frac{e^{x_n}}{\sum_{k=1}^{n} e^{x_k}} = \frac{e^{x_n}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}}
\end{aligned}$$

[11] 文章中では、省スペース化のために $\sum_{i=1}^{100} x_i$ のように右下と右上に書いて表すこともあります。
[12] 添え字の動く範囲を表す時に、$\sum_{i=5}^{10}$ の場合には「添え字 i は 5 から 10 まで走る」とよく言います。
[13] 特にネイピア数が底の指数関数を「自然指数関数」といいます。

といった具合で求まります。この n 本の式をまとめて表したのが、最初の式という訳です。因みに、n 個の出力値 $\{y_1, y_2, \cdots, y_n\}$ を全て足してみると[14]、

$$y_1 + y_2 + \cdots y_n = \frac{e^{x_1}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} + \frac{e^{x_2}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} + \cdots + \frac{e^{x_n}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}}$$

$$= \frac{e^{x_1} + e^{x_2} + \cdots + e^{x_n}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} = 1$$

と合計が 1 になっていることが確認できます。最初に述べたように、softmax 関数は n 個の入力値 $\{x_1, x_2, \cdots, x_n\}$ を合計が 1 となるように変換して出力する関数になっていることが分かります。

♣ 具体例

最後に、具体的なデータに対してこの softmax 関数を使ってみましょう[15]。例として、次の 3 つの入力データ $(n = 3)$

$$\{x_1, x_2, x_3\} = \{1.0,\ 2.0,\ 3.0\}$$

を考えましょう。この入力データを softmax 関数を用いて変換すると、出力値は

$$y_1 = \frac{e^{1.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = 0.09003057\cdots$$

$$y_2 = \frac{e^{2.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = 0.24472847\cdots$$

$$y_3 = \frac{e^{3.0}}{e^{1.0} + e^{2.0} + e^{3.0}} = 0.66524096\cdots$$

となります。3 つを足すと、$y_1 = y_2 = y_3 = 1$ になることが確認できます。この性質から、出力値をある種の確率値と解釈できるようになるのが、softmax 関数を用いる利点となります。

[14] 分母は揃っているので、単に分子を足せば良いだけです。

[15] e^1, e^2, e^3 の値は、関数電卓 (iPhone の計算機アプリでも OK) を使うとすぐに分かります。

Sigmoid 関数

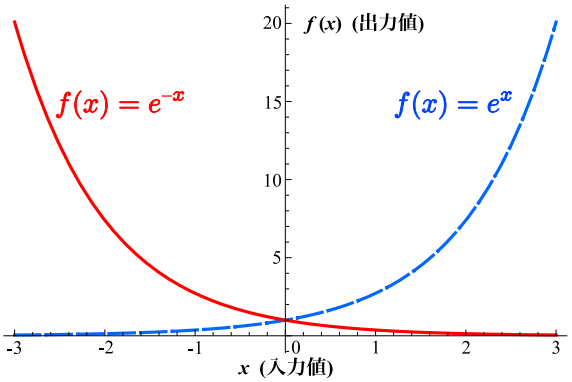
sigmoid (シグモイド) 関数は、次の式で定義される関数です。

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

式の中で出てくる e は、ネイピア数です[16]。

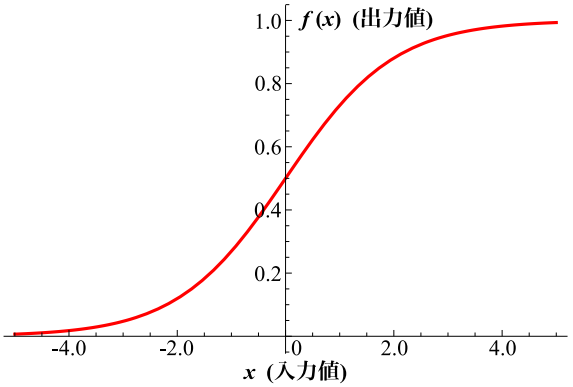
♣ sigmoid 関数のグラフ

指数関数 $e^{-x} = \dfrac{1}{e^x}$ については、softmax 関数のところで解説しました。



指数関数 $f(x) = e^x$ と $f(x) = e^{-x}$

上にグラフを描きましたが、x が大きくなるとどんどん 0 に近づいていき、x が小さくなるとどんどん大きくなっていきます[17]。これより、sigmoid 関数は、x が大きくなっていくと分母 $1+e^{-x}$ がほとんど 1 となり、関数全体としては $f(x) = \dfrac{1}{1} = 1$ に近づきます。一方、x が小さくなると分母がどんどん大きくなっていき、関数全体としては 0 に近づきます。すなわち、sigmoid 関数 $f(x)$ は、x が大きくなるにつれて 0 から 1 まで連続的に変化していく振る舞いとなります。



sigmoid 関数

[16] ネイピア数については、softmax 関数の解説を参照してください。
[17] x が小さくなると、e^{-x} は上限なく大きくなり続けます (=発散する)。

グラフの形状をみると、$x = 0$ の部分を基点として、点対称なS字型のカーブとなっていることが分かります。sigmoid関数の名前は、このグラフの形状に由来しています。sigmoidという英単語を辞書で調べてみると、「S字形、S字の」という意味になっています。つまり、sigmoid関数は「S字の関数」ということでグラフの形状をそのまま表現したネーミングとなっています。

♣ より一般的なsigmoid関数

sigmoid関数をより一般化した関数として、logistic (ロジスティック) 関数が知られています。

$$f(x) = \frac{L}{1 + e^{-\alpha(x - x_0)}}$$

この関数に含まれる各パラメータは、次の働きをしています。

- L: 関数 $f(x)$ の最大値を決めるパラメータ。関数 $f(x)$ は0から L の範囲の値を取る。
- x_0: S字カーブをしている曲線の真ん中の点[18]の位置を決めるパラメータ。
- α: 最小値から最大値に至る増加の緩急をコントロールするパラメータ [19]。

最初のsigmoid関数の式は、これらのパラメータを、

$$L = 1, \quad x_0 = 0, \quad \alpha = 1$$

と設定したものになっています。なお、このsigmoid関数は特に「標準sigmoid関数」と呼ばれます。一般には、α が1以外の場合も含めてsigmoid関数と呼ばれることがあります[20]。

♣ softmax関数との関係

入力データが2個 $\{x_1, x_2\}$ の場合、softmax関数の出力値 y_1 は次のように変形することができます[21]。

$$y_1 = \frac{e^{x_1}}{e^{x_1} + e^{x_2}} = \frac{1}{1 + e^{-(x_1 - x_2)}}$$

この形はsigmoid関数とよく似ています。つまり、$n = 2$ のsoftmax関数はsigmoid関数となり、逆にsigmoid関数を $n = 2$ 以外の場合まで一般化したものがsoftmax関数だと解釈することができます。このことから、機械学習の分類問題において、2値分類ではsigmoid関数、多値分類（3値以上）ではsoftmax関数が、出力層の活性化関数として用いられることが多くなっています。

[18]正確には「変曲点」と言います。
[19]α 値が大きくなるほど急激な増加となり、step関数のような形状に近づいていきます。
[20]ただしニューラルネットワークの文脈では、sigmoid関数は通常、標準sigmoid関数のことを指します。
[21]分母と分子を両方 e^{x_1} で割っています。

tanh 関数

tanh(ハイパボリックタンジェント[22]) 関数は、次の式で定義される関数です。

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

式の中で出てくる e は、ネイピア数です[23]。

♣ tanh 関数のグラフ

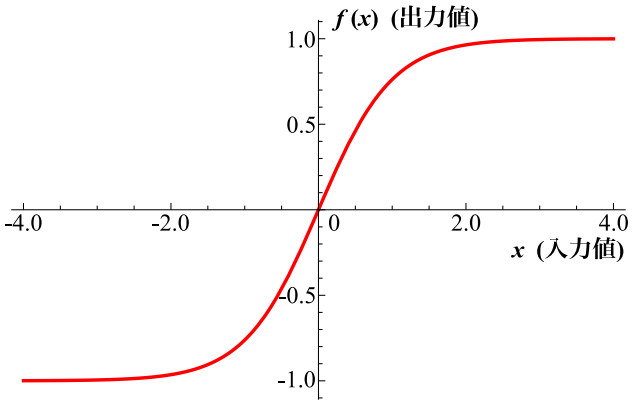
グラフの大まかな形状を理解するために、入力値 x が大きい場合と小さい場合の関数 $f(x)$ の値を考えてみましょう。まず、x が大きい場合を考えます。この場合、e^x はものすごく大きい値で e^{-x} はほとんど 0 であるため[24]、

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

となります。一方、x が小さい (大きな負の数) 場合を考えます。この場合は逆に、e^x はほとんど 0 で e^{-x} がものすごく大きな値となるため、

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

となります。すなわち、tanh 関数 $f(x)$ の値は、入力値 x が大きくなるにつれて -1 から 1 まで連続的に変化していく振る舞いとなります。



tanh 関数

グラフの形状的には sigmoid 関数とよく似ています。ただ sigmoid 関数の出力値は 0.0 ～ 1.0 の範囲であるのに対し、tanh 関数は -1.0 ～ 1.0 の範囲の値を返す関数になっています。

[22]略して「タンエイチ」や「タンチ」と呼ばれることもあります。
[23]ネイピア数について、またその指数関数については、softmax 関数と sigmoid 関数の解説を参照
[24]≃ はニアイコール (=だいたい同じの意味) です。$1 + 0.00000001 \simeq 1$ みたいな感じです。

♣ tanh 関数は双曲線関数の 1 つ

tanh 関数は、「双曲線関数」と呼ばれる関数の 1 つです。双曲線関数には他に

$$\sinh = \frac{e^x - e^{-x}}{2}, \qquad \cosh = \frac{e^x + e^{-x}}{2}$$

で定義されるハイパボリックサイン/コサイン関数があります。sinh, cosh, tanh の間には

$$\tanh x = \frac{\sinh x}{\cosh x}$$

の関係式が成り立っています。これらの関数の関係は、高校の数学で出てくる三角関数 sin (サイン), cos (コサイン), tan (タンジェント) とよく似ています。そのため、この三角関数とよく似た記号が用いられているようです。双曲線関数についている「h」は hyperbolic (=双曲的な) という形容詞から来ています[25]。

[25] 双曲線 $x^2 - y^2 = 1$ の媒介変数表示 (パラメータ表示) になっているため、双曲的という形容詞が含まれているのですが、この詳細はややこしい (かつ本題ではない) のでここでは省略します。