機械学習講習会 第四回

-「ニューラルネットワークの構造」

traP アルゴリズム班 Kaggle部

2023/xx/xx

はじめる前に

- 普通の関数をベクトルや行列に適用するときは各要素に適用する
  例) $f(x) = x^2$としたとき、$f((1,2,3)) = (1,4,9)$
- スペースの都合上ときどき総和の添え字が省略されています
- ベクトルは太字、行列は大文字で表します
- 今日はちょっとむずかしいです
- がんばりましょう

第四回: ニューラルネットワークの構造

振り返りタイム

第一回 「学習」
第二回 「勾配降下法」
第三回 「自動微分」

1. 予測をするには、「モデル」を作る必要があった
2. モデルのパラメータを決めるために、パラメータの関数である損失関数を導入した
3. 損失関数を最小にするパラメータを求めるために勾配降下法を導入した
4. 自動微分によって手で微分する必要がなくなった [← 今ココ！]

われわれができるようになったこと

データさえあれば...誤差を小さくするパラメータを

- 例え複雑な式でも
- 例え自分で導関数を見つけられなくても

求められるようになった！
(== **学習ができるようになった！**)

線形回帰からの飛躍

ここまでは$f(x) = ax + b$のかたちを仮定してきた

⇨ われわれの学習手法はこの仮定に依存しているか？

結論: 依存していない

我々の手法(自動微分と勾配降下法による学習)で満たすべき条件は、

$L(a, b)$がa, bについて微分可能である

ことのみ！

⇨ この条件を満たす関数なら文字通り**どんなものでも**学習できる！

今日の話

今日のおはなし

損失関数

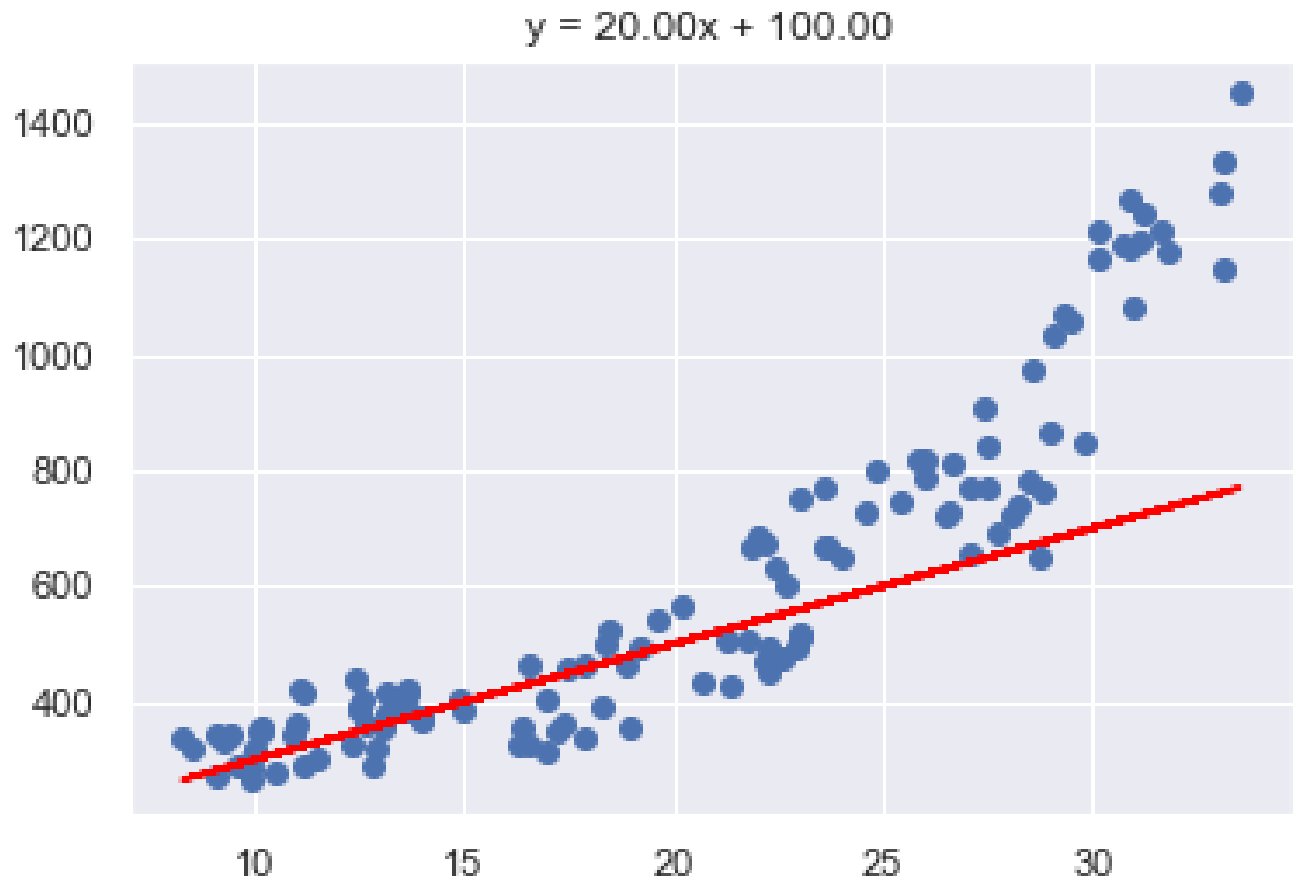
$$L(a,b) = \sum_{i=0}^{n-1} (y_i - \underline{f(x_i)})^2$$

のfを変えよう

線形回帰からの飛躍

$f(x) = ax + b$　は、a, bをどんなに変えても常に直線

⇨ 直線以外の関係を表現できない



どんな関数をつかうべきか?

$f(x) = ax^2 + bx + c$ でも動く

$f(x) = \sin(ax + b)$ でも動く

$f(x) = e^{ax+b}$ でも動く

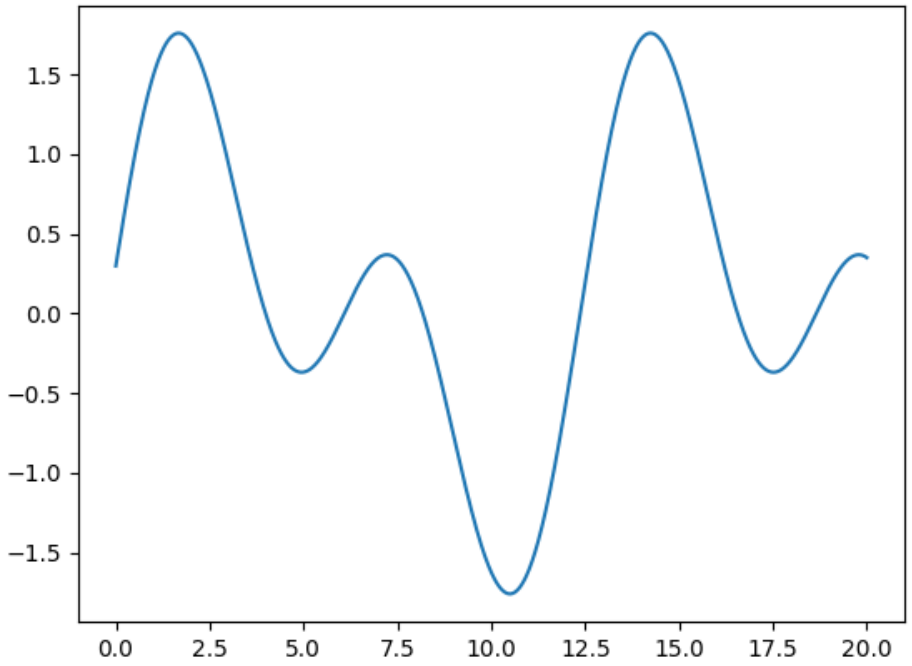
⇨ 直線以外を表現することはできるが

- 二次曲線
- sin
- 指数関数

しか表現できない...

複雑な関数を表現する方法を考えよう！

⇨ これらのパラメータどんなにいじっても

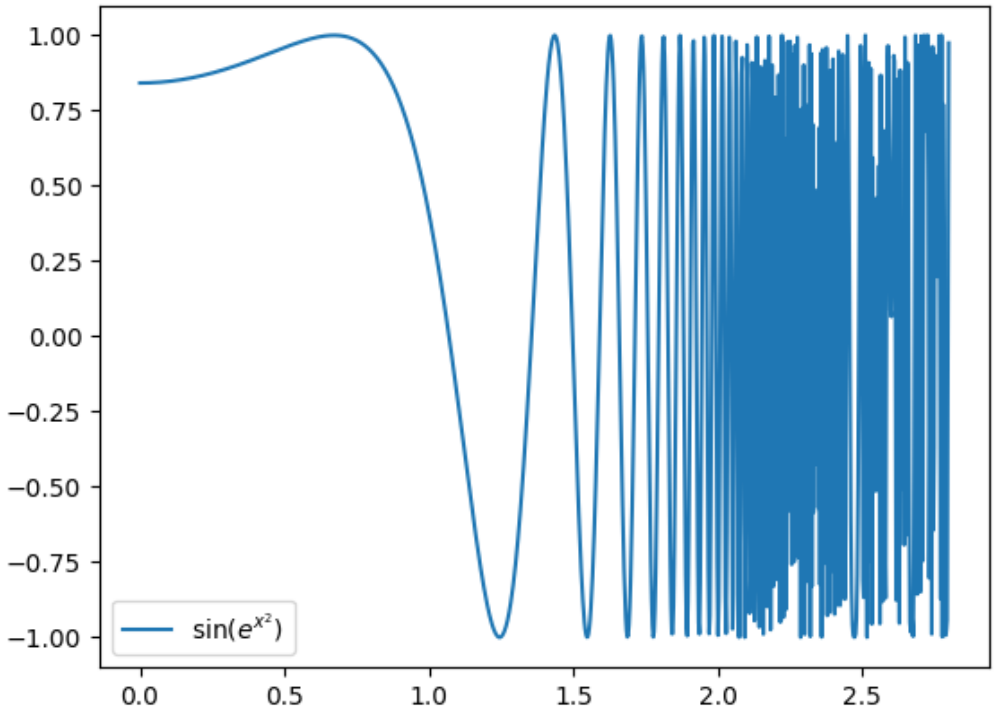


みたいなのは表現できない

複雑な関数の表現の仕方を考える

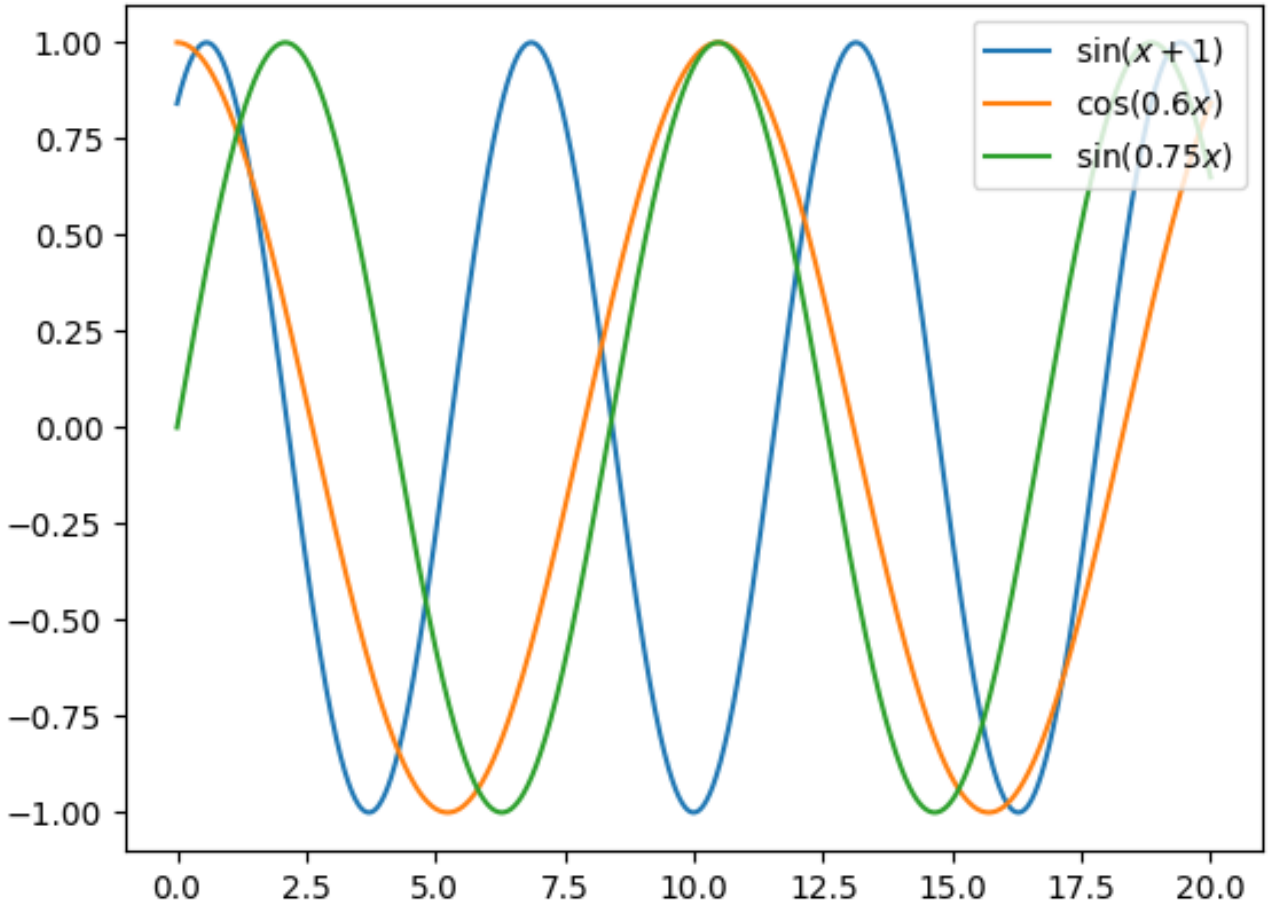
✅ アイデア1: 関数を合成する

$\sin(x), \cos(x), x^2 + x, e^x \cdots$ はそれぞれ単体だと単純な関数だが、
合成してみると....？



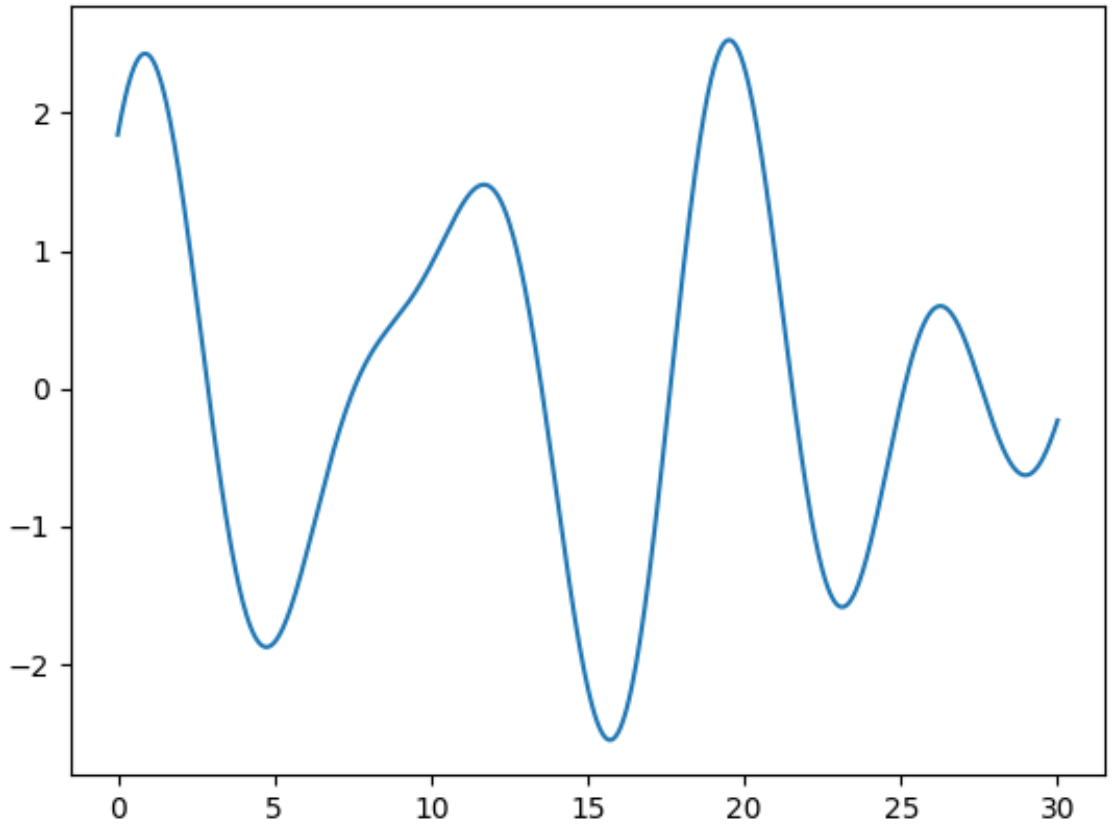
複雑な関数の表現を考える

✅ アイデア2: 和をとる



ぐにゃっとした関数の表現のしかた

⇨ ベースにした三角関数の和をとると...



 < 実質ASMR

✅ わりと簡単めの関数の

1. 合成

2. 和

を考えたら結構複雑なやつも表現できる

パラメータを変えることによって幅広い表現が得られる確認

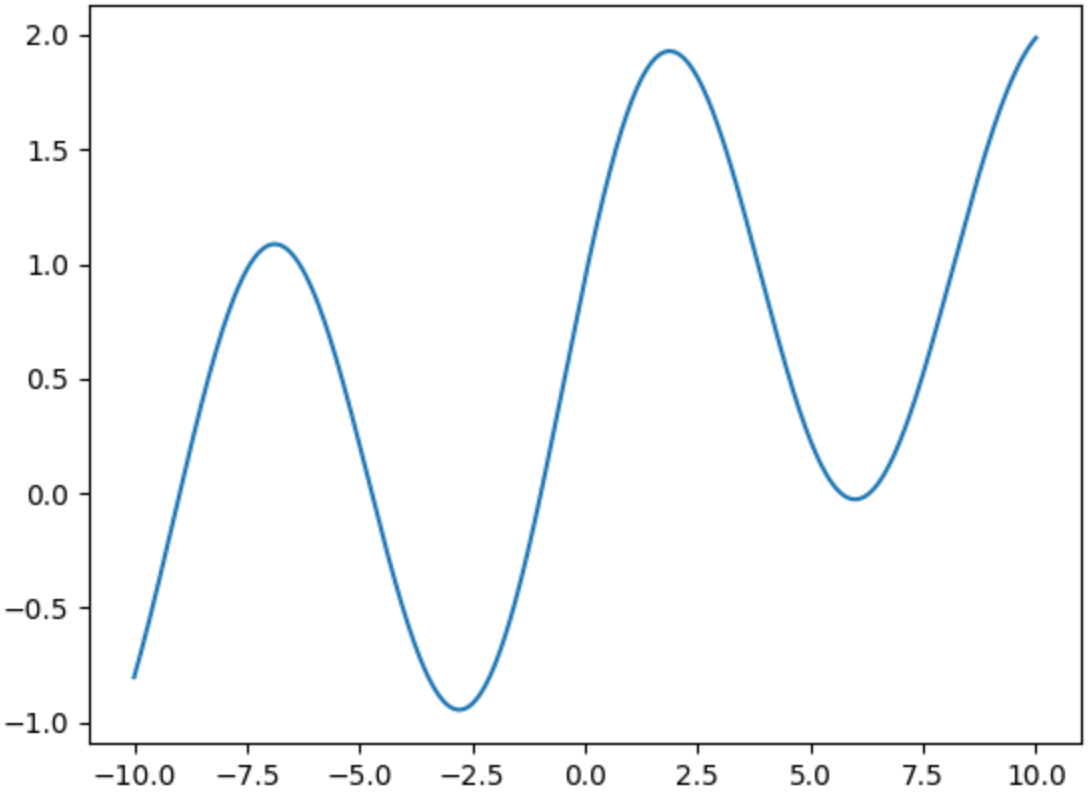
パラメータとして
$\boldsymbol{a} = (a_1, a_2, a_3, a_4, a_5)$,
$\boldsymbol{b} = (b_1, b_2, b_3, b_4, b_5)$,
$\boldsymbol{c} = (c_1, c_2, c_3, c_4, c_5)$
をもつ

$$f(x) = \sum_{i=1}^{5} a_i \sin(b_i x + c_i)$$

⇩

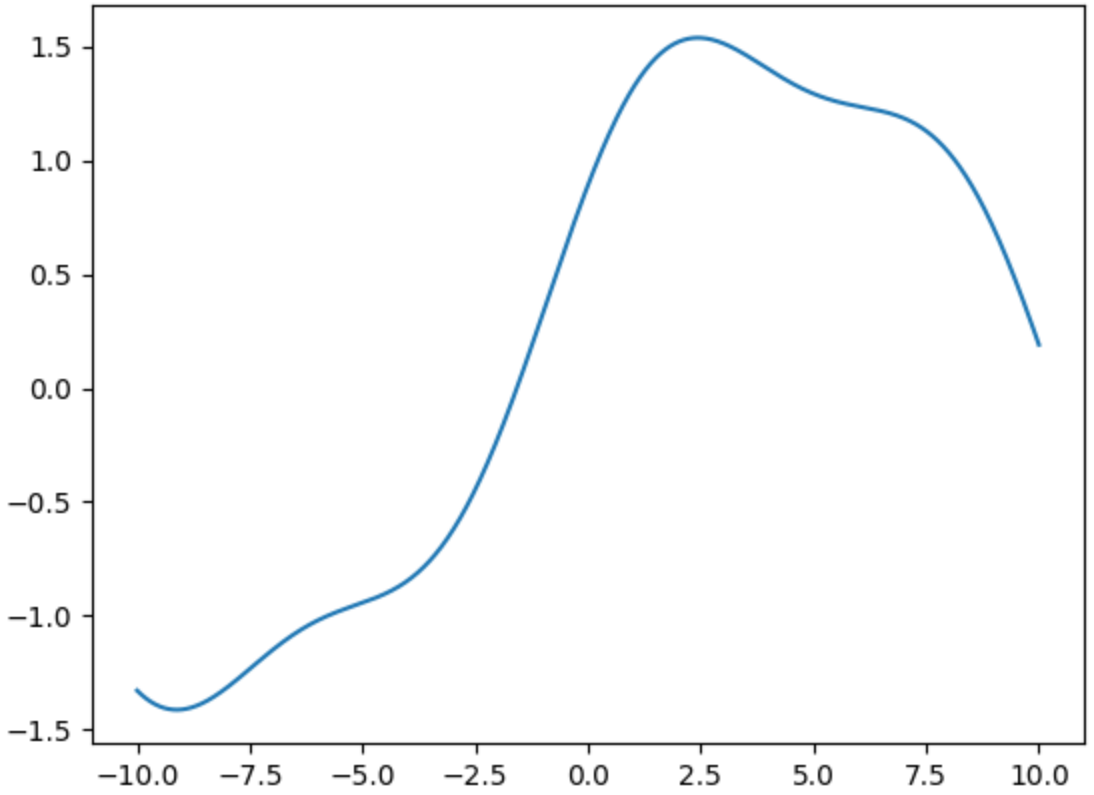
パラメータを変えることによって幅広い表現が得られる確認

$a = (0.04, 0.78, 0.09, 0.63, 0.01)$,
$b = (0.94, 0.43, 0.25, 0.19, 0.41)$,
$c = (0.01, 0.65, 0.87, 0.61, 0.81)$ のとき



パラメータを変えることによって幅広い表現が得られる確認

$a = (0.83, 0.27, 0.84, 0.28, 0.14),$
$b = (0.71, 0.47, 0.56, 0.39, 0.94),$
$c = (0.08, 0.92, 0.16, 0.44, 0.21)$　のとき



「基になる関数」はどう選ぶべきか？

「基になる関数」にどのような関数を選ぶべきか？

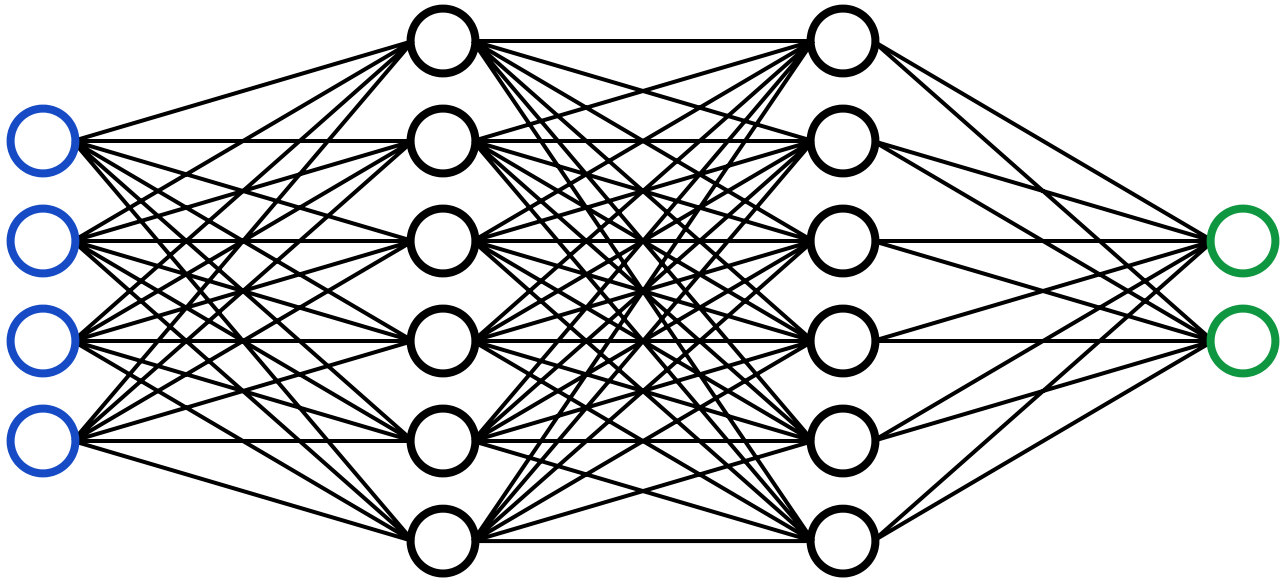
- 三角関数?
- 多項式関数?
- 指数関数?
- もっと別の関数?

  …

これまでの我々のアプローチを思い出すと、
「変化させるのが可能なところはパラメータにして、学習で求める」

「基になる関数」もパラメータに
よって変化するようにしよう

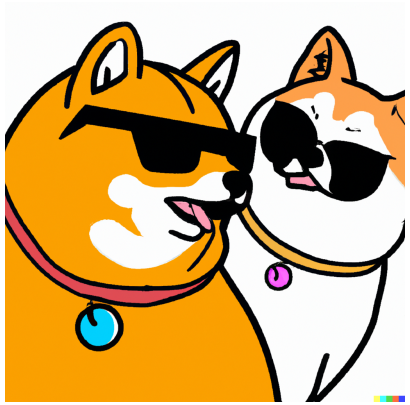
ニューラルネットワーク



ニューラルネットワーク

- 最近流行りの機械学習モデルは基本的にニューラルネットワークをつかっている
- ある程度以上複雑な問題では、たいていの場合ニューラルネットワークが最も精度が出やすい

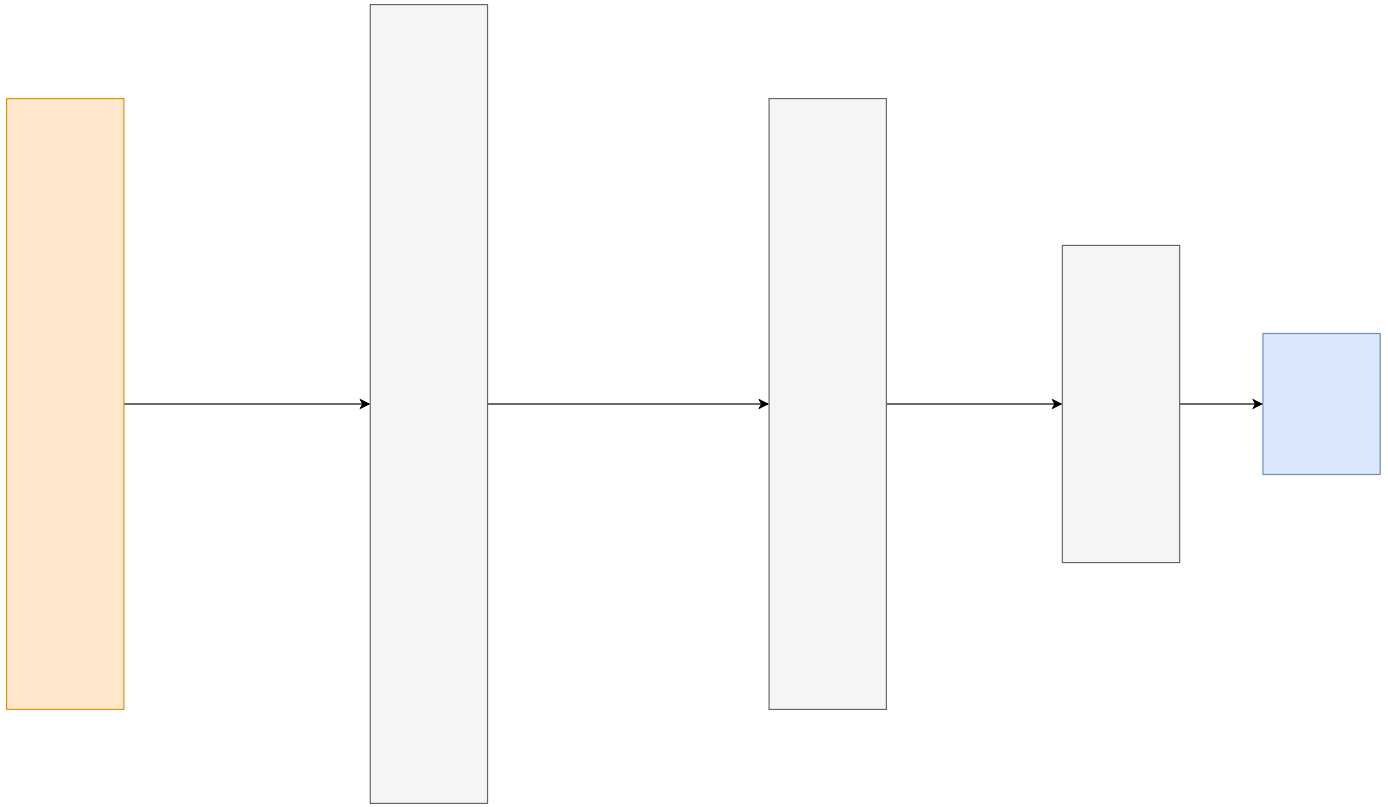
例) 画像分類, 音声分類, 画像生成, 対話 ...



ニューラルネットワークの構造

基本単位: レイヤー

ニューラルネットワークは、「レイヤー(層)」と呼ばれる関数の合成によって構成されるモデル



ニューラルネットワークの構造

基本単位: レイヤー

- 入力層
  入力を受け取るレイヤー
- 出力層
  出力を出力するレイヤー
- 中間層(隠れ層, hidden layer)
  それ以外のレイヤー

データの流れは、
入力層　→ 中間層　→ 中間層　→ ...　→ 出力層

もっとも普遍的・基本のレイヤー

$W \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ と (m, nは適当に定めた自然数)
全結合層が固有にもつ「活性化関数」を用いて

入力として$\boldsymbol{x} \in \mathbb{R}^n$を受け取り、

$\sigma\left(W\boldsymbol{x} + \boldsymbol{b}\right)$ を出力する。

全結合層(Linear, Dense層)

< は？

全結合層がしていること

1. n個の入力を受け取り、m個出力する
2. 複雑な関数を表現するアイデア...
    - アイデア1. 合成
    - アイデア2. 和をとる

    をする

全結合層がしていること

1. n個の入力を受け取り、m個出力する

「$W \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ と (m, nは適当に定めた自然数)
全結合層が固有にもつ「活性化関数」を用いて

入力として$\boldsymbol{x} \in \mathbb{R}^n$を受け取り、
$\sigma\left(W\boldsymbol{x} + \boldsymbol{b}\right)$ を出力する」

全結合層がしていること

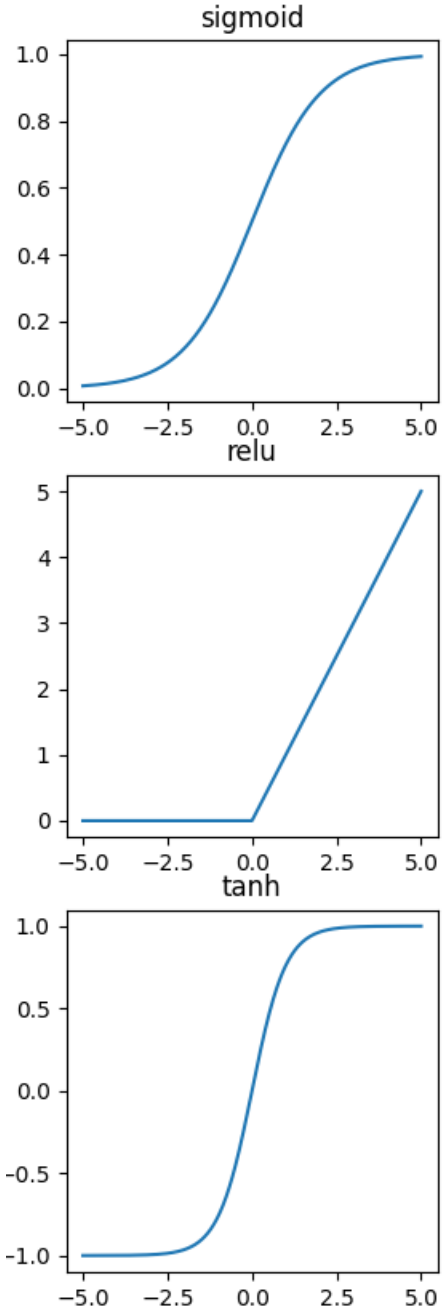
2. 複雑な関数を表現するアイデア...
    - アイデア1. 合成
    - アイデア2. 和をとる

    をする

活性化関数

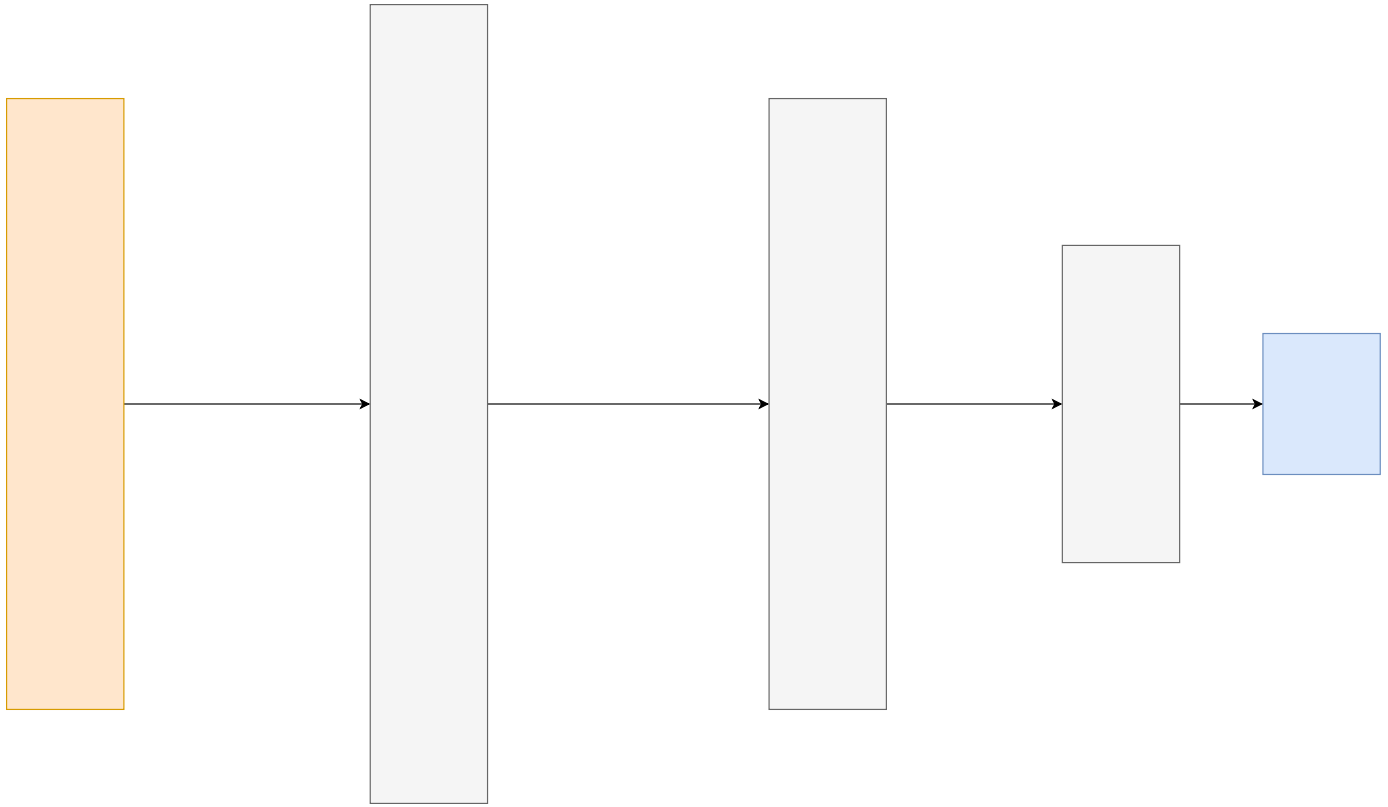
非線形関数

- シグモイド関数
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
- ReLU関数
$$\mathrm{relu}(x) = \max(0, x)$$
- tanh関数
$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



アイデア1. 合成

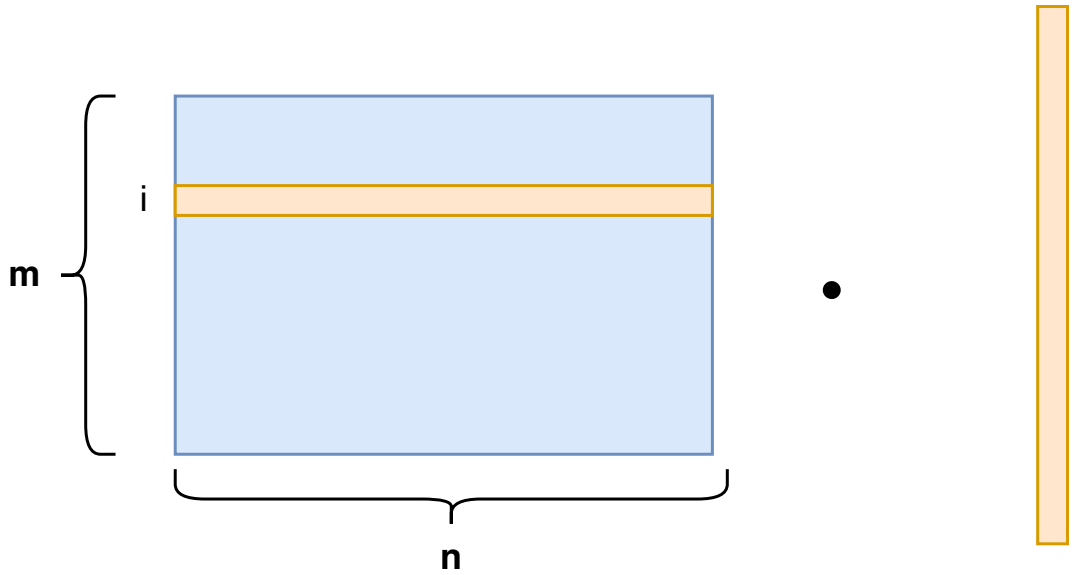
合成を繰り返す
⇨ 複雑な関数を表現



分解して考える

m個の出力のひとつに注目してみる

$$\boldsymbol{y} = \sigma\left(W\boldsymbol{x} + \boldsymbol{b}\right) \Rightarrow y_i = \sigma\left(\sum_j W_{ij}x_j + b_i\right)$$



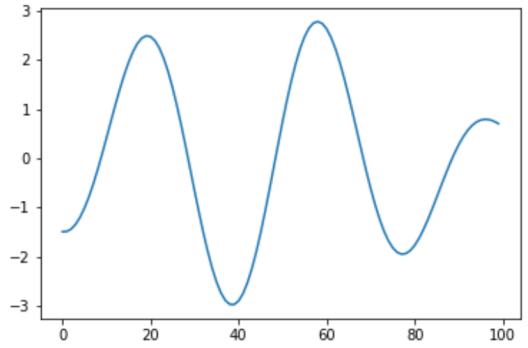
分解して考える ... 隠れ層

$$y_i = \sigma\left(\sum_j W_{ij}x_j + b_i\right)$$

は、

ぐにゃっとした関数の表現のしかた

2. ぐにゃっとした関数の和を考える



< 結構複雑になった 13/25

とおなじことをしている

分解して考える ... 隠れ層

$$y_i = \sigma\left(\sum_j W_{ij} x_j + b_i\right)$$

x_jはそれまでの層でσを通した、非線形な関数

$$\sigma\left(\sum_j W_{ij} x_j + b_i\right)$$

➡ 非線形関数の重みつき和
➡ 複雑な非線形関数を表現できる！ + さらにそれを非線形関数に通す

合成

演算をd 回繰り返す
(n 次元ベクトル $\to m_1, \to m_2, \to \cdots, \to m_d$ 次元ベクトルへと
変換されながら計算が進んでいく)

$$\boldsymbol{y}^{(1)} = \sigma\left(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{y}^{(2)} = \sigma\left(W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\cdots$$

$$\boldsymbol{y}^{(d)} = \sigma\left(W^{(d)}\boldsymbol{y}^{(d-1)} + \boldsymbol{b}^{(n)}\right)$$

出力層

$$\boldsymbol{y} = \mathrm{id}\left(W^{(o)}\boldsymbol{y}^{(d)} + \boldsymbol{b}^{(o)}\right)$$

⬇

$$y = \sum_j w_j^{(o)} y_j^{(d)} + b^{(d)}$$

ここで、

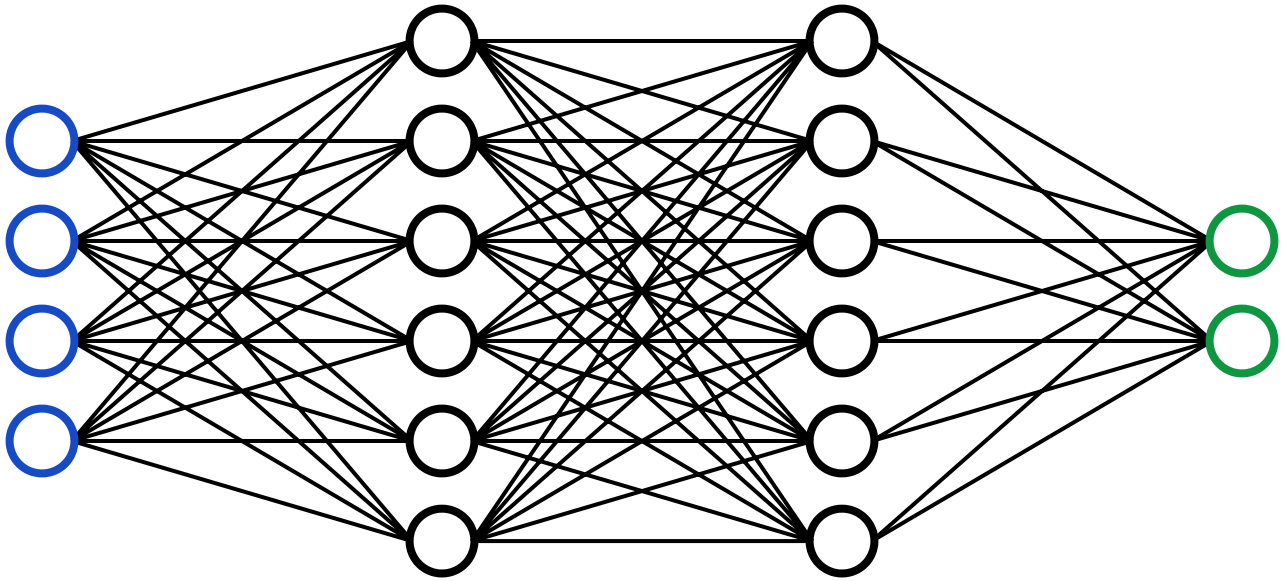
$\boldsymbol{y}^{(d)}$ は非常に複雑なxの非線形な関数

+ これはパラメータによって変化する

「基になる関数」もパラメータに
よって変化するようにしよう

MLP

とくに、全結合層のみからなるニューラルネットワークを
多層パーセプトロン (Multi Layer Perceptron, MLP) という



$f(x) = ax + b$ ⇨ ニューラルネットへ

 < どれくらい表現能力が変わったのか？

結論

直線 ⇨ 任意の関数

(注) 正確な表現ではありません！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！
詳しくは次のページの補足を見てください.

ニューラルネットワークの万能近似

ニューラルネットワークの万能近似定理(普遍性定理)

隠れ層を一つ持つニューラルネットワークは、
任意の連続関数を表現できる

ここではステートメント自体もかなり平易な表現に直しています。このあたりをきちんと論じようと思うと位相空間論や測度論、関数解析などの知識が必要になります。
(https://qiita.com/mochimochidog/items/ca04bf3df7071041561a) などに詳しくまとまっているので、興味がある人は読んでみてください。なおこの後発展的話題として直感的な証明を行います。

第5回 ニューラルネットワークの学習と評価

- He, Xavierの初期化
- 確率的勾配降下法
- 損失関数
- オプティマイザ
- バリデーションと性能評価
- ハイパーパラメータ

Next ⇨ 第6回　ニューラルネットワークの実装

発展的話題:万能近似の(直感的な)証明

- ニューラルネットワークがなんでも表現できるぜ！！という主張は条件を変えていろいろある
- ここでは一番有名(たぶん)なもの[1]について直感的な証明をする

[1]https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.441.7873&rep=rep1&type=pdf

主張

$$\sigma(x) \to \begin{cases} 0 & (x \to -\infty) \\ 1 & (x \to \infty) \end{cases}$$

を満たす関数を「シグモイド型関数」と呼ぶことにし、
$I = [0,1]^d$として、CをΩ上の連続関数の集合とする。

シグモイド型関数の重ね合わせによる近似

「任意の$f \in C, \varepsilon > 0$に対して、ある$N \in \mathbb{N}$と$a_i, b_i \in \mathbb{R}^d, c_i \in \mathbb{R}$が存在して、

$$G(x) = \sum_{i=1}^{n} a_i \sigma(b_i x + c_i)$$

が

$$\forall x \in I, |f(x) - G(x)| < \varepsilon$$

を満たす。

証明ステップ1. シグモイド型関数をつかった矩形関数のつくりかた

$$G(x) = \sum_{i=1}^{n} a_i \sigma(b_i x + c_i)$$

$\sigma(b_i x + c_i)$について、σはシグモイド型関数

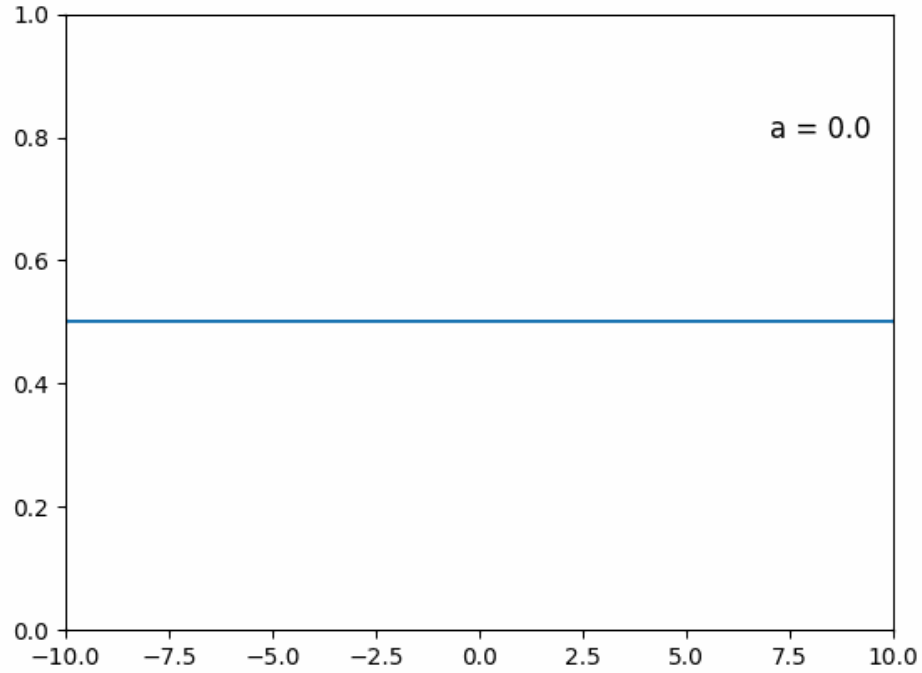
⇨ b_i をバカみたいに大きくするとどうなる？

$$\sigma(x) \to \begin{cases} 0 & (x \to -\infty) \\ 1 & (x \to \infty) \end{cases}$$

証明ステップ1. シグモイド型関数をつかった矩形関数のつくりかた

$\sigma(b_i x)$は$b_i = 9999999999999999999999999999999999999$

とすると、x_iがちょっともで正ならば1, そうでなければ0になる。



「直感的な」証明です(大声)

証明ステップ1. シグモイド型関数をつかった矩形関数のつくりかた

$\sigma(b_i x + c_i)$は$b_i = 9999999999999999999999999999999999999$

とすると、$x_i - \dfrac{c_i}{b_i}$ がちょっともで正ならば1, そうでなければ0になる。

⇨ c_iを適当に調整すれば、狙った点tで、

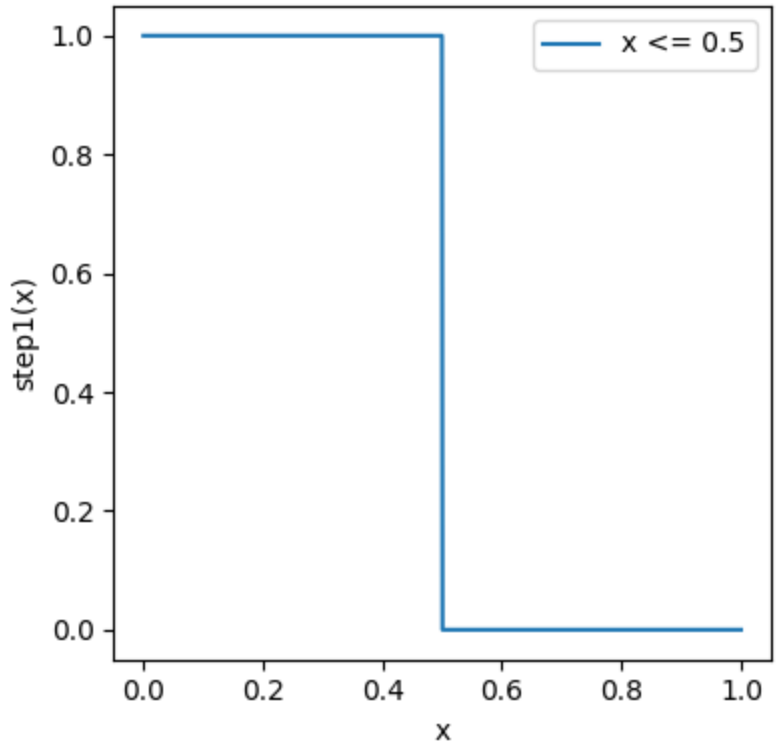
$$\sigma(b_i x + c_i) = \begin{cases} 1 & (x > t) \\ 0 & (x \leq t) \end{cases}$$

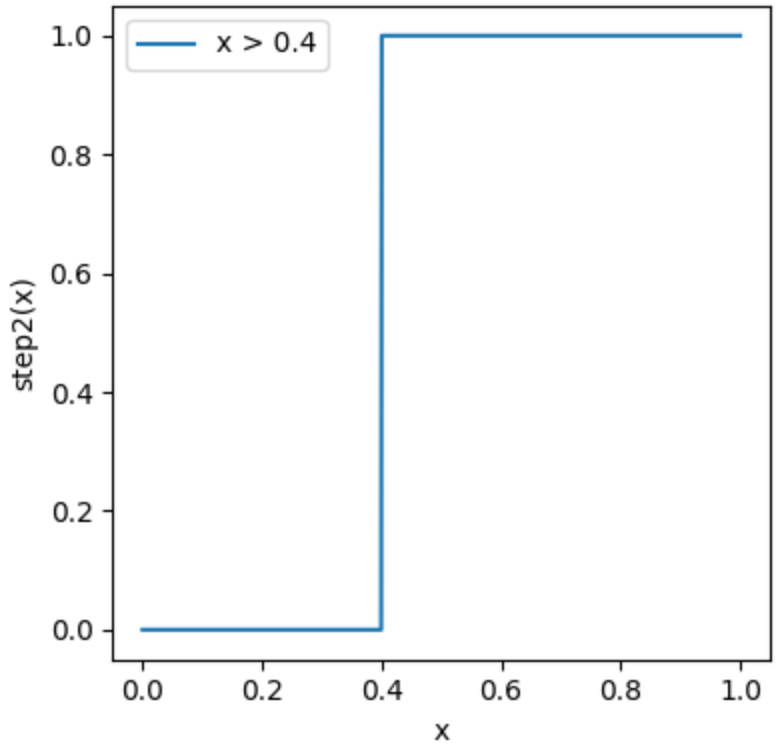
とすることができる． (例: $b_i = 10^{100}, c_i = 10^{99} \cdot 20$ なら $t = 2$)

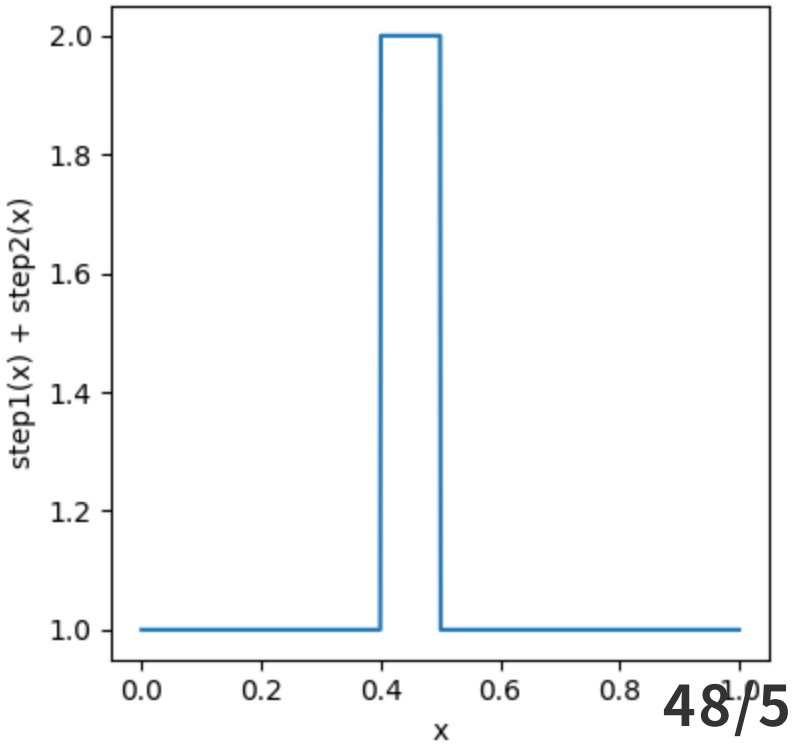
b_iを負のデカ数にすると、逆verもできる。

証明ステップ1. シグモイド型関数をつかった矩形関数のつくりかた

すると、
$\sigma(b_i x + c_i)$について、正の大きな数によってステップ関数にしたものと
負の大きな数によってステップ関数にしたものを足し合わせることで、
矩形関数を作ることができる。



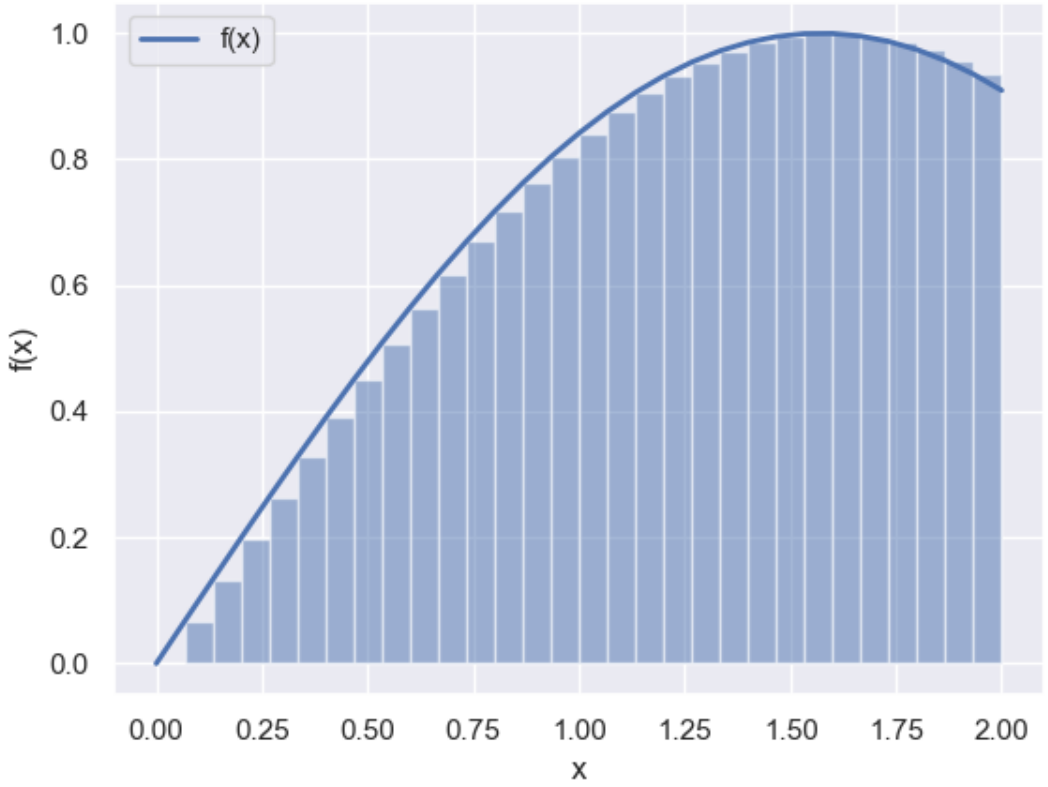




証明ステップ2. ウイニングラン

✅ これさえできればもう近似できる！

連続関数を全てこれの和としてみればよい



万能近似できるからいい？

任意の連続関数を近似できるのはニューラルネットワークだけ？

⇨ **NO**

❌「万能近似ができるからニューラルネットワークがよくつかわれる」

+ あくまでそのようなa_i, b_i, c_iが存在するという主張であって、
それを求める方法については何ら保証していない

⇨ ニューラルネットワークの優位性を考えるなら、もうちょい議論を進めていく必要がある
(例えば今回は連続関数だけを考えたが実際に得たい関数は常にそうか？
a_i, b_i, c_iの得やすさはどうか？)